ardor vs the competition









MAGAZINE

Ardor vs the Competition

Editor: apenzl
Writer: segfaultsteve
Design: apenzl, on behalf of Nxter Magazine

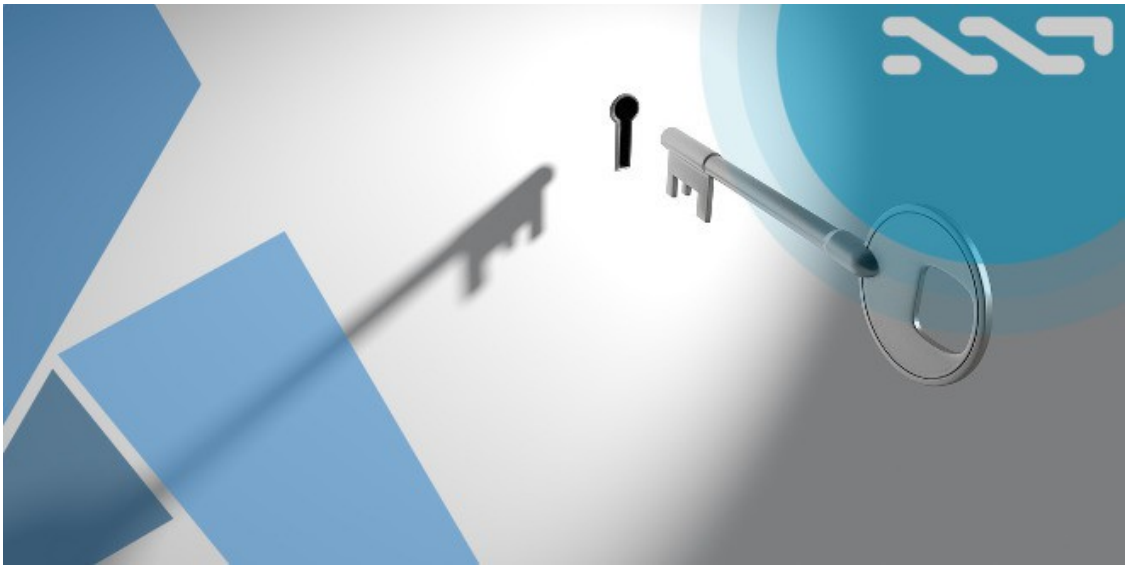
Published 24.11.2017

This publication is also available in Spanish, Chinese, Russian, and other languages.

Special thanks to: segfaultsteve, jose, rubenbc, fz1128, sergei, Nxt and Ardor Group, Jelurida.

TABLE OF CONTENTS

Preface



24.11.2017, Nxtville.

Congrat's, Nxt.

Wow, how time flies!

Since Nxt launched as the first 100% PoS cryptocurrency 2.0, four years ago, all described and elaborated in the book 'SNAPSHOT', Jelurida B.V. formed and registered, as a "software development company engaged in the creation of Nxt and Ardor blockchain technology". Jelurida now releases Nxt under a 'JPL' license, which is meant to make cloning fair for all.

Today, on Nxt's 4th anniversary, Jelurida has published the source code of Ardor.

They raised over $15,000,000 in their ICO for IGNIS, their child chain token on the Ardor Platform. Ignis inherits all the features of Nxt (token transferrals plus smart transactions triggering hardcoded smart contracts), and more. Jelurida says, "these funds will be used for the further development, maintenance, advancement and world-wide promotion of the Nxt and Ardor blockchain platforms, as well as protecting the intellectual property of the code base." Ignis will be released with the Ardor Genesis Block, to be forged on 0:00 UTC, Jan. 1st , 2018.

So what is Ardor? What is Nxt?

Non-nxters can START HERE, new nxters and developers should check out this whitepaper.

About three months ago, segfaultsteve joined /r/ardor, claimed to be ”no expert on Ardor”, but then immediately began answering questions about the Ardor blockchain platform, showing great knowledge and understanding of its design. Shortly thereafter, he had his first original post:

> *All the cool kids are talking about Plasma, so I figured I'd try to wrap my head around it too. I'm particularly interested in comparing and contrasting "child chains" in Plasma and Ardor. I'm having a hard time of it, though, because frankly, the Plasma paper is poorly written. There, I said it.*
>
> *I'm posting a brain-dump here so that y'all can correct me wherever I've misunderstood something. Hopefully you'll find it useful too. :)*

He starts his brain-dump,

> *In my understanding, the key difference between a Plasma chain on Ethereum and a child chain on Ardor is that the nodes securing the Ethereum network (i.e., miners) don't validate Plasma chain transactions directly, whereas on Ardor, all nodes on the Ardor network, including forgers, validate child chain transactions.*
>
> *All of the other differences between Plasma and Ardor--the bonds that validators post in smart contracts on the parent chain; the "fraud proofs" that force cheaters to forfeit their bonds; the "mass exit" mechanism that is supposed to protect against block-withholding attacks--flow from this one difference, in the sense that none of these things are necessary on Ardor because forging nodes directly validate child chain transactions.*
>
> *On Ardor, only accounts that hold ARDR forge, and every transaction that changes the ARDR balances is recorded directly on the Ardor chain. Because the entire network (including forgers) validates child chain transactions, one can check any block in the parent chain and verify that the network at the time came to a consensus about the child chain transactions, and therefore that the subsequent state of child chain accounts was valid. By induction, the current state of accounts is valid too. (I'm glossing over some details, but this is the gist of it, I think.)*
>
> *In Plasma, though, ETH miners don't see the child chain transactions. Rather, consensus on child chain transactions comes from "validators" on the child chains. These are accounts that are responsible for forging child chain blocks (e.g., using proof-of-stake) and reporting their hashes to the parent chain. These validators have posted bonds on the parent chain that are "listening" for proofs of fraud from users, and if a user can prove that a validator forged an invalid block, the validator forfeits the bond. In this way, validators are incentivized to reject fraudulent transactions. Ultimately, though, the consensus mechanism seems to be rooted in the willingness of users to audit child chain blocks and blow the whistle on validators who cheat.*

What if the validator (or a cartel of validators) on a child chain withholds new blocks, though? It wouldn't necessarily be possible to construct a proof of fraud, since some of the required information isn't available yet (though I admit I'm a little fuzzy on this part). To minimize the damage that could be caused in this case, Plasma describes a fairly elaborate "mass exit" mechanism that allows coalitions of users to withdraw their funds en masse if they detect that validators are withholding blocks.
One other difference between Plasma and Ardor that is probably worth mentioning is that Plasma chains can be organized in a tree structure, where each Plasma chain can have multiple children, and the children can have children, and so on. In some cases, this will facilitate scaling of large computations across a lot of child chains using an algorithm similar to MapReduce.

Maybe I've misunderstood, but I'm a little skeptical of this part. Large jobs only scale in cases where each node does a small subset of the computations. If I'm only validating/policing my part of the computation, though, and nobody is validating another piece, then the final reduced result will be wrong. It seems like the only way for me to guarantee against that possibility is to validate all of the computations (i.e., all transactions on all of the child chains involved), which means that MapReduce didn't achieve any scaling benefit.

So what to make of all of this?

In my opinion, the main similarity between Ardor and Plasma is that both reduce blockchain bloat by storing only hashes of child chain blocks on the parent chain. There are several important differences, though:

- *In Ardor, all nodes validate all child chain transactions, whereas in Plasma nodes only need to validate the transactions that affect them. This could greatly increase the total throughput of the network, since most nodes could ignore most computations, namely all those belonging to completely unrelated applications. A potentially similar feature that's on the Ardor roadmap would delegate the verification of child chain transactions to dedicated subnets, but there aren't many details in the white paper.*

- *The concept of "confirming" a transaction on Ardor is pretty straightforward. In contrast, there's a period of time on a Plasma chain where a block can be rolled back, even after a validator hashes it and adds it to the parent chain (I think), because it's possible somebody will present a proof of fraud. I'd like to understand this mechanism better.*

- *Ardor actually exists. :)*

In the past last months, segfaultsteve has read, analysed, and wrapped his head around blockchain platforms, compared them to Ardor, and blogged about it in Nxter Magazine.

Without further ado, dig into a well researched article series: ”Ardor vs the Competition” which takes a good look into Ardor, Ethereum, Lisk, IOTA, Waves, Stratis, NEM and Komodo Platform.

- apenzl

LISK

Ardor vs The Competition Pt 1

by segfaultsteve

I recently decided to start a series of posts that compare and contrast Ardor with other blockchain projects that appear to have similar goals or features. Roughly each week, I'll pick a project whose scope overlaps at least a little with Ardor's, study its technical documentation, and post a summary of my findings here for you to critique. This week, I've been reading about Lisk.

Lisk

In a nutshell, Lisk is a platform for developing decentralized applications (dapps) that run on sidechains anchored to the Lisk mainchain. It uses a delegated proof-of-stake (DPOS) consensus mechanism to secure the mainchain, while sidechains are each responsible for their own security (sort of, but see the description of the delegate marketplace below). The protocol uses a set of predefined transactions, rather like Nxt and Ardor, as opposed to a low-level scripting language like Bitcoin or Ethereum.

Before I get into the details, I should start by saying that Lisk is definitely in an early stage of development. The team is currently in the middle of rewriting the Lisk SDK, which will support sidechain development, and is continuously refactoring Lisk Core, which is the full node.

With the code in flux, some important architectural questions, particularly about sidechains and how they will interact with one another and with the mainchain, do not appear to have been settled yet. On the other hand, I had some difficulty finding a current, authoritative source of technical information about Lisk, so what I present here might be out of date. The best information I could find was in the wiki, this article by one of the co-founders, the roadmap, and these YouTube videos. None of the first three sources are recent, unfortunately, and even the videos don't go into much depth (though I admit I haven't watched all 6+ hours of them). If you've found better references, I'd be grateful if you could send them my way.

The marketing buzz surrounding Lisk seems to focus on the SDK, the goal of which is to make it easy to build, deploy, and secure a dapp running on a customizable blockchain. The devs wrote the SDK in JavaScript because they want to make Lisk accessible to as wide an audience as possible, and they also wrote the backend in JavaScript (Node.js) because...well, I guess I'll never understand why people insist on using JavaScript on the backend. :)

But clearly, ease of developing and deploying a custom blockchain is not the only goal of Lisk. If it were, then what purpose would the mainchain serve? You might as well clone Bitcoin or Nxt if all you want is a good starting point for building your own blockchain.

The mainchain/sidechain architecture is the real distinguishing feature of this platform. As far as I can tell, the mainchain serves at least three important functions:

1. The Lisk API will allow deposits of LSK on the mainchain to be transferred to and from sidechains. With two such transactions, it will be possible to send LSK from one sidechain through the mainchain and to another sidechain. Unfortunately, according to the article by one of the co-founders linked above, it sounds like transferring LSK onto a sidechain will require sending it *to the sidechain's owner*, which obviously requires some degree of trust. To avoid this problem, it will be possible to create sidechains that use their own forging tokens instead of LSK. This token would then need to be traded for LSK in order to transact through the mainchain with another sidechain. Alternatively, it might be possible for one sidechain to transact directly with another sidechain *without* going through the mainchain, but the developers are still researching how this would work.

2. Eventually, the team plans to build a "delegate marketplace" where delegates who are not securing the mainchain can offer to secure sidechains and are paid "either by the [sidechain] application owner or its users." Again, the details are a little fuzzy, but there seems to be a lot of value here: presumably the Lisk network is already far larger than a typical brand new blockchain network, and the delegate marketplace gives sidechains an "off-the-shelf" set of nodes that they can use to secure themselves in their infancy.

3. Some nodes on the network (not sure which ones) will periodically hash sidechains and store the hashes on the mainchain as a "basic validation of sidechain integrity." I haven't been able to find any details about how this mechanism will work, though.

Apart from these functions, and from the obvious role it plays in transferring LSK between accounts, the mainchain itself doesn't seem to have any other intended uses. All of the business activity is supposed to occur on the sidechains.

Compared to Ardor

How does this architecture compare with Ardor's parent chain and child chains?

Maybe the most obvious difference is that each sidechain must have its own set of nodes to secure it, whether these are provided by the sidechain creator, the users, or eventually the delegate marketplace.

With Ardor, in contrast, every node on the network validates child chain transactions, but only accounts holding ARDR forge. The fact that accounts holding child chain tokens don't forge with them means that it doesn't matter how small child chains are or how unequal the distribution of tokens on them is; they are all just as secure as the parent chain.

One additional note about Lisk is that, until the delegate marketplace opens, sidechain creators choose the nodes that forge on their chains, which seems to require that users place a great deal of trust in them. On the other hand, the team has also suggested that Lisk will be flexible enough to allow sidechains to use an entirely different consensus algorithm, like proof-of-work, so it seems that sidechain creators wouldn't determine which nodes secure the chain in that case.

There are also plans to allow existing sidechains to switch consensus mechanisms even after they launch, but again I haven't been able to find details.

Clearly, both Lisk and Ardor intend to offer scaling advantages over traditional blockchains. With Lisk, the computational scaling advantage is obvious, since each forging node validates only the transactions on a single blockchain, either the mainchain or a sidechain. The reduction in required storage space (i.e., blockchain bloat) is less clear, though. Compared to Ethereum, say, it's obvious that for a similar level of total activity, the many chains in the Lisk ecosystem will each grow more slowly than the single Ethereum chain, simply because sidechains will not store each other's data.

Compared to Ardor, though, the storage savings would be modest. Ardor's parent chain will grow at a similar rate to the Lisk mainchain--as both will store only hashes of sidechain or child chain data instead of the data itself--but on Ardor the child chain data will be pruned away, eliminating the blockchain bloat problem that Lisk will still have on each sidechain.

Conclusion

What, then, should we make of Lisk? Honestly--and I'm very disappointed to write this--I think it's simply too early to tell. Too many important details have yet to materialize:

- Will it be possible to convert one sidechain's token directly to another sidechain's token without converting to and from LSK? How?

- When the delegate marketplace opens, will it be possible for users to elect delegates using sidechain tokens? Or will they have to use LSK? Or will sidechain owners maintain control over which delegates forge?

- What will Lisk do with the hashes of sidechains that are stored on the mainchain? Will it be possible to roll back recent transactions on a sidechain to "restore" it to the state it had when it was hashed? If so, will there be some time after which this will *not* be possible, so that the sidechain can still be considered immutable?

- Will the Lisk SDK provide some clean mechanism for changing the consensus algorithm on an existing sidechain? I'm not sure what this would look like.

- What happens if a sidechain that uses LSK forks? Obviously, the LSK tokens on both resulting sidechains cannot be simultaneously backed by the same LSK reserves on the mainchain. I would assume the sidechain creator effectively gets to choose which chain is the "real" one, since he or she is the one holding the reserves on the mainchain, but I don't know for sure that this is correct.

- Depending on how Lisk will support transactions directly between sidechains, this same concern could require additional trust between sidechain creators. In particular, if sidechain creators must hold reserves of each other's tokens to enable cross-chain

transactions, which seems like one plausible way to do it, then a fork in one sidechain could give the other sidechain's creator some influence over which branch of the fork is honored. Moreover, if the forking sidechain transacts with *several* other sidechains, each of which hold reserves of the split token, then the situation could get ugly pretty quickly.

In my opinion, the most important advantage Lisk has over most blockchain platforms, including Ardor, is that it will accomplish a natural computational scaling by segregating each dapp onto its own blockchain. If, in addition, sidechains will be able to transact seamlessly and trustlessly with one another, then it seems like the design has immense potential.

If we're making the assumption that the Lisk team will successfully implement all the features required to make this happen, though, then we ought to grant Jelurida the same courtesy and assume that they'll be able to carry out their own scaling plans. In particular, one potential improvement on the Ardor roadmap is to confine child chain transaction processing to dedicated subnets of the Ardor network. It seems to me that this would accomplish a similar computational scaling to Lisk, while preserving Ardor's substantial advantage in reducing blockchain bloat.

In conclusion, Lisk's mainchain/sidechain architecture could potentially help it scale to accommodate a large number of dapps that could interact in interesting ways, but right now there seems to be a lot of uncertainty in the technical details. Ardor's approach is technically quite different but solves some of the same problems, namely blockchain bloat, potentially computational scaling, and the ability to transact easily between separate chains.

It will be very interesting to see how Lisk develops in the next two or three years, but then again, by that time Ardor will have been live for a long time already.

NEM/Mijin/Catapult

Ardor vs The Competition Pt 2

This week I studied NEM, a public blockchain similar to Nxt in many ways. As I'm primarily interested in each blockchain project's approach to scaling, I also researched Mijin, a version of NEM for private blockchains, and Catapult, a rewrite of Mijin which promises large performance gains and which will also be incorporated into future releases of NEM.

NEM

Although NEM's core developers abandoned their initial plan to start NEM as a fork of Nxt, choosing instead to start the project from scratch, NEM and Nxt are still fairly similar. Like Nxt, the NEM platform provides a predefined set of allowed transactions which applications can use as building blocks to create more complex features, as opposed to using a low-level scripting language to construct transactions, like Bitcoin or Ethereum.

Both platforms support a variety of "blockchain 2.0" features, like sending messages, creating and transfering assets, and sending transactions requiring the approval of multiple accounts (m-of-n multisig). And both platforms expose their functionality through HTTP-based APIs, so developers can use virtually any language to write applications for them.

Despite these similarities, NEM also has some notable differences compared to Nxt.

Perhaps the most fundamental one is its novel consensus algorithm, called proof-of-importance. This algorithm is similar to proof-of-stake, except the probability that an account may harvest (i.e., forge) the next block depends not only on its stake of XEM, which is the native coin on NEM, but also on how recently it has transacted with other accounts and how much XEM was exchanged. Accounts that hold a large stake of XEM and which transact frequently and in high volume harvest more blocks than accounts with less XEM or accounts which only rarely transact.

The authors of the NEM Technical Reference argue that, compared to proof-of-stake, the proof-of-importance algorithm gives somewhat less weight to the wealthiest accounts when determining the right to forge/harvest the next block (Section 7.8). Proof-of-importance is also central to NEM's spam filter, which requires that an attacker not only control a lot of accounts, which is easy to do, in order to spam the network with a large number of unconfirmed transactions, but also to hold a large stake in each account and transact frequently with other high-importance accounts.

In my view, another main difference between NEM and Nxt is the extent to which each platform's "blockchain 2.0" features are integrated directly into the API. For example, NEM's assets, called "mosaics," share several features with the Nxt Monetary System's currencies, but NEM does not have a built-in decentralized exchange for mosaics. (As a side note, the NEM Foundation has contracted with Blockchain Global to create a traditional, centralized exchange featuring mosaic-based ICO tokens.) Similarly, while you could certainly build a decentralized marketplace on top of NEM where users could buy and sell goods and services, NEM does not have such a marketplace built into its API the way that Nxt does.

Finally, one subtle but very important difference between NEM and most other blockchains, including Nxt, is the way that it handles multisignature transactions. Instead of allowing any account to generate a multisig transaction, NEM introduces the concept of a *multisig account* and requires that all multisig transactions originate from such accounts. Any co-signatory on the account can initiate a transaction from it, and the transaction is only executed if a sufficient number of the other co-signatories approve it.

At first this might appear to be a limitation, since it requires a separate multisig account for each set of co-signatories a user wants to cosign with, but it has two key advantages: the multisig account is a full account, capable of receiving payments, messages, and mosaics, for example; and co-signatories can be added and removed, so custody of the multisig account can be transferred. It is possible to create a "1-of-1" multisig account, i.e., an account with a single custodian who can transfer it to a different custodian if desired. In this way, multisig accounts on NEM can act like transferable containers for XEM, mosaics, and messages.

One particularly impressive application of this concept is a notary service built on NEM called Apostille. With Apostille, the process of notarizing a document looks like this:

1. Hash and sign the name of the document.
2. Create a multisig account for the document derived from the resulting signature.
3. Hash and sign the contents of the document.
4. Send a message containing the result to the document's multisig account.

Note that the last step also attaches a timestamp to the document, since the transaction that transfers the document's signed hash to the multisig account is recorded on the blockchain.

As an example of a potential application of Apostille, the authors of the white paper consider a case where the notarized document is a car title. Ownership of the car can be transferred by changing co-signatories on the multisig account that contains the title; messages describing maintenance and repairs can be sent to the multisig account to record the car's service history; and mosaics issued by governments or insurers could attest to payment of fees. In this way, the multisig account represents both the car itself and the history of other accounts' interactions with it.

Anyway, that's quite enough about NEM. Next, Mijin.

Mijin

At a high level, Mijin is a version of NEM that three of the core NEM developers and a company called Tech Bureau developed as a private, permissioned blockchain product. Like any private blockchain–and in contrast to NEM, which is public–a Mijin blockchain is owned and controlled by a central authority, such as a company.

This isn't the place for a full debate about the utility of private blockchains, but as Mijin and Catapult are an important part of the NEM ecosystem, please indulge me for a minute. In my opinion, the more "private" a private blockchain becomes, the less useful it is. While I can see a case to be made for "consortium" blockchains, where a handful of independent organizations who don't necessarily trust each other cooperate to secure the network against abuses by any one member of the group, I have trouble seeing the value in a blockchain controlled by a singleauthority. In my view, a blockchain without trustless consensus is basically just an extremely slow, extremely inefficient database.

I know there are plenty of people who disagree with me, though, so for the remainder of this post I'm going to assume private blockchains have value and that there is a market for them, especially in financial services, which seems to be the main industry that Tech Bureau intends for Mijin to serve.

There is not nearly as much information about Mijin available on the internet as there is about NEM, but I did learn some interesting facts that hint at its potential. For one thing, although Mijin and NEM are completely separate projects, Mijin does share the NEM API (or at least the two APIs overlap substantially), which suggests that it will be relatively easy for developers to write applications that run on either platform. The common API might also facilitate interactions between Mijin chains and the public NEM chain, but I haven't found any information about the details of those interactions.

Additionally, the Mijin website states that Mijin will support smart contracts, though the Catapult white paper seems to slightly contradict that statement when it says, “the approach here is to make the smart contract an external component, whether centralized (i.e., status quo with existing systems) or decentralized. The outputs of these smart contracts will then enter their transactions into the ledger through a secure transaction process.” To me, this implies that the contracts themselves will be neither stored on the blockchain nor executed by all nodes on the network.

Speaking of Catapult…

Catapult

Catapult is a rewrite of Mijin with a focus on increasing the rate at which transactions can be confirmed. Judging from the white paper (linked above), the first deployments of Catapult will be at banks and other financial institutions, where the author envisions it will replace patchworks of “disjointed monolithic systems” that he says are commonly used today. Eventually, the developers also plan to integrate Catapult into NEM to facilitate scaling the public blockchain as well.

Like Mijin, Catapult is currently closed-source and many technical details are not public. I was able to find some good information digging around the NEM blog, though, especially in this thread by one of the developers.

Catapult divides the work that the network does among three types of nodes:

- P2P nodes, which add new blocks to the blockchain and maintain consensus about its state;
- REST nodes, which present client applications with all the features they can use from the Catapult API; and
- API nodes, which, like P2P nodes, store the blockchain and can read directly from it (I think), but which do not add blocks to it. These nodes serve data to the REST nodes to fulfill client applications’ requests.

This breakdown appears to roughly correspond to the three-tier architecture commonly used for web applications, where the blockchain (P2P nodes) is the database, the REST nodes are the front-end, and the API nodes handle the business logic of interpreting and interacting with data in the database.

If this analogy is correct, then presumably the goal of this architecture is to allow each tier to scale independently. Especially for a private blockchain, the optimal number of P2P nodes used to establish consensus might be much smaller than the number of REST and API nodes required to handle all of the requests that applications send to the network. Delegating these responsibilities to separate nodes on the network should allow nodes of each type to be added or removed as needed to optimize performance.

Apart from this new architecture, Catapult also makes some other optimizations to improve performance. Whereas Mijin and NEM are written in Java and use HTTP for communicating with full nodes, Catapult is being written in C++, and communication between at least the API nodes and REST nodes uses full-duplex sockets (via ZeroMQ), potentially allowing for lower latency than HTTP.

A performance test of three Catapult nodes located in the same datacenter and configured to service requests from 10.8 million accounts showed that the network was able to process just over 3,000 transactions per second. It isn't completely clear from the press release, but it sounds like each of the three nodes in this test played all three roles: P2P, API, and REST. Confusingly, the accompanying diagram appears to refer to API nodes as "blockchain data ingestion servers" and to REST nodes as "API gateway" servers.

Compared to Ardor

How does NEM compare to Ardor, then?

Really, there are (at least) two separate questions: how do NEM's features compare to Ardor's features? And how does NEM's approach to scaling compare to Ardor's approach?

Since Ardor (the platform, not the parent chain) will support all of Nxt's current features, the comparisons I noted above between NEM and Nxt apply equally well to Ardor.

In particular, Ardor's child chains will have at their disposal a somewhat larger variety of built-in transaction types that support a richer set of features.

For example, NEM does not natively support a peer-to-peer exchange for mosaics, dividend payments to mosaic holders, transactions conditioned on votes by mosaic holders (or most of

Nxt's phased transaction types, for that matter), account properties, a decentralized marketplace, or anything like Nxt's shuffling and alias systems.

Ardor's parent-chain/child-chain architecture will add some extra functionality, too.

In particular, users will be able to exchange different child chain tokens for one another directly, without first converting to ARDR. This will be especially useful on pegged child chains, where users will be able to trade dollar-pegged coins directly for bitcoin-pegged coins (for example), whereas on NEM, somebody holding a dollar-pegged mosaic would have to sell it for XEM, then buy a bitcoin-pegged mosaic.

These differences notwithstanding, NEM still offers a rich set of features that application developers can use in interesting ways. Perhaps the best example is Apostille's creative use of NEM's unique multisig accounts. I'm not sure how easy it would be to replicate that kind of functionality on Ardor.

[EDIT: Lior Yaffe, core dev and co-founder of Jelurida, made the following comment:

> *With NXT this can be achieved by issuing a singleton asset for each license registration and sending it between accounts.*]

On the question of how to scale, the two platforms differ much more dramatically.

Catapult's approach, which NEM will eventually incorporate, is twofold: a new three-tier architecture to distribute the network's responsibilities among three specialized types of nodes; and a series of application-level optimizations, e.g., using C++ instead of Java. We will need to defer judgment of the latter approach until additional benchmarking tests are available, but we can still cautiously speculate about the implications of the new architecture.

The biggest advantage seems to be for private blockchains, where the owner can fine-tune the quantities of the three types of nodes and the topology of the network to optimize throughput. Moreover, in such a context, blockchain bloat isn't as severe a problem as it is for a public blockchain since companies can easily dedicate terabytes of storage on their servers to storing the blockchain.

The improvement in *NEM's* performance with this new architecture, on the other hand, is much harder to predict. It is not clear whether each peer on the network would have to run all three services (P2P, API, REST) or just one of the three. In the former case, the scaling advantage to the new architecture would presumably be lost. In the latter case, the classic trade-off between speed (fewer P2P nodes, more API and REST nodes) and security (greater fraction of P2P nodes) would remain. And since nobody could control the number of each type of node on a public network, the question of what the optimal balance is would be moot.

In contrast, Ardor's design does not try to achieve the highest possible throughput, at least initially. Rather, Ardor's main scaling goal is to greatly reduce the size and rate of growth of the blockchain. It does this using a unique parent-chain/child-chain architecture, where all nodes on

the network validate all transactions, but only those belonging to accounts holding the parent chain coin (ARDR) forge. Since the child chain coins *can't* be used to forge, the child chains' transaction history is irrelevant to the security of the network and can be pruned away.

It is worth noting, however, that computational scaling is on the Ardor roadmap.

Specifically, it is possible that child chain transaction processing will be delegated to separate subnets of the Ardor network in the future, allowing most nodes to ignore most transactions.

Conclusion

Ardor and NEM both offer rich, largely overlapping sets of features.

Overall, my impression is that developers will probably be able to build similarly complex applications on either blockchain with comparable ease. In that sense, the two platforms are direct competitors.
In their approaches to scaling, though, Ardor and NEM are quite different.

While Catapult will likely achieve a significant improvement in the rate that private blockchains can confirm transactions, I am somewhat more skeptical of the performance improvement that can be achieved on a public blockchain like NEM using the same approach.

Ardor, on the other hand, does not attempt to address the computational scaling problem (for now), but has found a very effective solution to the problem of blockchain bloat.

I suppose time will tell whether computational scaling or blockchain bloat is ultimately going to pose the biggest long-term problem for blockchain tech, and time will also tell whether either platform has found an adequate solution.

IOTA

Ardor vs The Competition Pt 3

This week I studied IOTA, a distributed ledger that doesn't use a blockchain.

Why Compare Ardor and IOTA?

At first blush, IOTA is about as different from Ardor as a distributed ledger can be. It uses a directed acyclic graph (DAG), which its developers call "the tangle," to represent the history of transactions, instead of storing transactions on a blockchain. It is intended to be used primarily for machine-to-machine microtransactions on the Internet of Things (IoT), a vision enabled by the fact that IOTA requires no transaction fees. And it doesn't (yet) support the "blockchain 2.0" features that form a core part of Ardor's appeal. On the surface, it doesn't really look like a competitor to Ardor.

So why include IOTA in a series entitled "Ardor vs. the Competition"?

As I've mentioned before, my main interest with this series is in exploring different distributed ledgers' approaches to scaling, and this is where the IOTA community has made some extraordinary claims. As I learned more about IOTA to better understand how it scales, I

eventually came to the conclusion that IOTA and Ardor offer complementary (or more bluntly, opposite) solutions to the scaling problem:
Ardor dramatically reduces blockchain bloat but requires all nodes of the network to agree about the strict ordering of transactions; whereas IOTA achieves potentially higher throughput by relaxing the consensus rules a bit, allowing temporary discrepancies between transactions, but faces a significant challenge in coping with the growth of the tangle. These tradeoffs, plus what I learned about the security of the tangle, seemed interesting enough to warrant a post in this series.

After this post, I plan to shift my focus away from scalability and towards features and market fit. Stratis, Ark, and Waves are on the agenda, but I'm not sure of the order, yet.

The Tangle

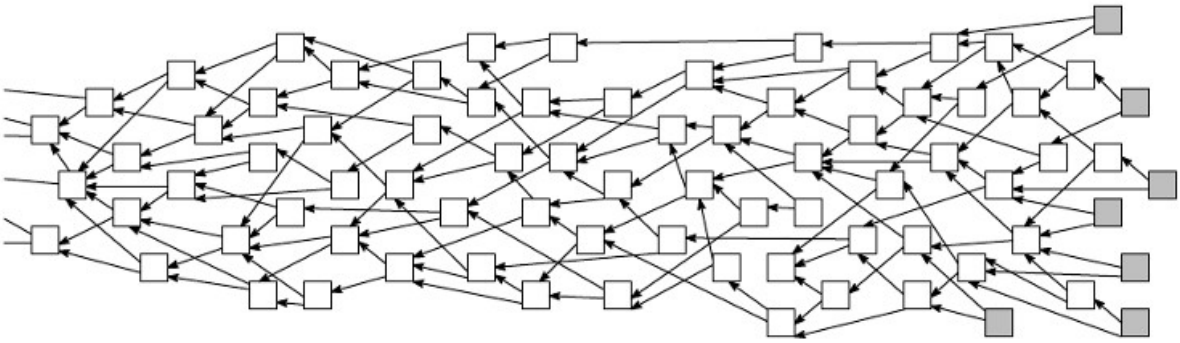
Without a doubt, the key distinguishing feature of IOTA is the tangle.

IOTA's other unique features, such as its lack of transaction fees, the fact that transactions are not strictly ordered but still eventually consistent, and the notion that (some) spam actually *increases* the throughput of the network, all stem directly from the way the tangle works.

For this reason, and also because I want to sidestep at least *some* of the recent controversy surrounding the IOTA project, I will try to focus primarily on understanding and evaluating the tangle itself, rather than picking apart the details of IOTA's specific implemetation of it.

The tangle is a directed acyclic graph whose vertices represent individual transactions, and whose edges represent "approvals" of previous transactions. Each time a node submits a new transaction to the network it must choose two previous transactions to validate, which it references in the new transaction it submits. As the new transaction permeates the network, each node adds it to its local copy of the tangle, with one edge pointed to each transaction that the new transaction approved.

I tried my best, but this description is probably confusing. This diagram should help. Each square represents a transaction, and the arrows that point from each transaction to two others represent that transaction's approval of the two earlier ones. The genesis transaction is somewhere far off the left side of the diagram, and the newest transactions, called "tips" in the white paper, are on the right side, shaded in gray.



What does it mean to validate, and hence approve, a transaction? Conceptually, the node doing the validation must start at the two transactions that it is validating and walk all paths back to the genesis transaction, ensuring that it never encounters a contradiction (e.g., double-spend, insufficient balance, or the like). If there *is* a contradiction, it chooses another pair of transactions to approve, knowing that no other node would ever approve the transaction it is submitting if it had approved a set of inconsistent transactions.

Notice that this means that each new transaction not only directly approves each of the two transactions it has chosen to validate, but also *indirectly* approves the transactions that those two approve, and the transactions that those transactions approve, and so on all the way back to the genesis. This is part of the basis for "eventual consensus" on the tangle.

In case you're wondering about the computational burden of doing this validation, in practice it can be optimized substantially. Notice from the figures on this page that as you walk the tangle from the tips (far right) towards the genesis, you eventually reach a point past which all transactions are (indirectly) approved by all tips. In these figures, transactions approved by all tips are colored green. You could, therefore, cut the tangle across arrows that point to green transactions, validate the paths from those particular green transactions to the genesis a single time, cache the results, and from that point forward only validate from your new transaction back to those green transactions. This optimization saves you the time of validating the entire tangle every time you submit a transaction, and also allows the tangle to be pruned. More on that below.

Consensus

One very interesting feature of a tangle-based ledger like IOTA is that nodes that receive new transactions from their peers *don't have to immediately validate them*. In fact, the tangle can temporarily contain contradictory transactions. Eventually, though, a node must decide which of the contradictory transactions to approve (possibly indirectly) as it adds a new transaction.

How does it choose between conflicting transactions? Assuming that each transaction is valid if considered separately, then the short answer is that a node could choose to approve either one. It has an incentive to approve the one that the rest of the network will build on, though, so that its own transaction will eventually be approved, too. Most of the nodes on the network are assumed to run the reference algorithm for selecting transactions to approve, so in the event of

a conflict, a node has an incentive to choose the transaction that the reference algorithm selects.

In order to understand the reference algorithm, it is important to first understand the concept of the *cumulative weight* of a transaction.

Each node that submits a new transaction must do some proof-of-work (PoW), which determines the "own weight" of the transaction. The *cumulative* weight of a transaction is then its own weight plus the own weights of all transactions that have directly or indirectly approved it. In a general tangle the node can decide how much work to do for a transaction, but in IOTA all transactions require the same PoW and thus have the same own weight. As a result, the cumulative weight of a transaction is proportional to the number of other transactions that directly or indirectly approve it.

What, then, is the reference algorithm? The author of the white paper calls it Markov-Chain Monte Carlo (MCMC, see section 4.1), which is a fancy way of saying that it is a random walk along the tangle that favors paths with greater cumulative weight. This post is already getting long, so I'll skip the details. Suffice it to say that, when there are conflicting transactions, the MCMC algorithm resolves the conflict by tending to choose whichever transaction has the greater cumulative weight behind it. Eventually, one subtangle becomes dominant and the other is orphaned. This is analogous to the mechanism that blockchains use to resolve forks, and the cumulative weight of a transaction in IOTA is a rough measure of its finality in the same way that adding blocks to a blockchain confirms previous transactions with greater and greater certainty.

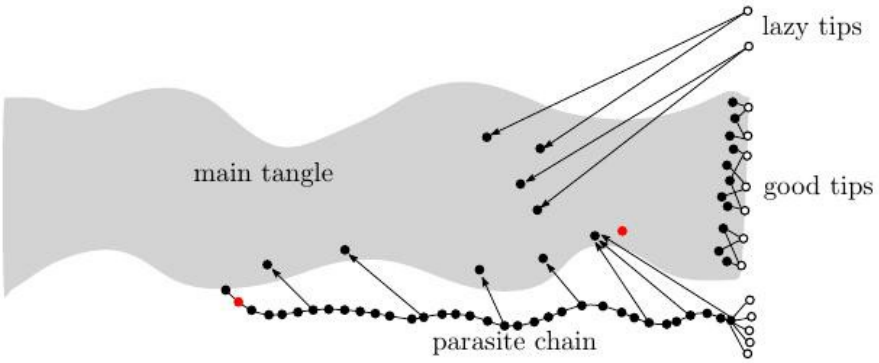
By the way, the fact that nodes don't immediately need to validate each new transaction received from their peers has big implications for performance. Each node does less work this way, validating transactions only when it submits a new transaction, and taking for granted that transactions that are indirectly approved by all tips have already been validated by the rest of the network. Also, validations run in parallel across the network, as different nodes choose different subsets of transactions to approve.

Security

So far I have mostly just regurgitated the information found in the IOTA white paper. The issue of the security of the tangle, on the other hand, is where things get a lot more interesting. While I definitely recommend reading the analysis in the white paper of different attacks on the tangle–and the rest of the white paper, for that matter, because it is very well written–I won't discuss most of that analysis here.

Instead, I want to focus on the most obvious threat, which is a 51% attack. The IOTA devs actually refer to it as a 34% attack, for reasons that I'm not sure I understand. I suspect it's because an attacker who waits for a fork to occur naturally only needs enough hashpower to out-compute the nodes on each branch of the fork–i.e., more than 50% of the *rest* of the

network's hashpower. Anyway, the exact number isn't important, and for the remainder of this article I will use the term "34% attack."



With IOTA, a 34% attack would look roughly like this. An attacker issues a transaction that spends some funds, represented by the rightmost red dot, then computes (or perhaps has precomputed) his own "parasitic" subtangle, which anchors to the main tangle somewhere upstream of his transaction and which contains a double-spend transaction, represented by the leftmost red dot. His goal is to add enough cumulative weight to his parasitic tangle to convince the MCMC algorithm to orphan the main tangle and follow the parasitic one.

Hopefully, the analogies to the blockchain are clear so far, because there is one more important one. Like a PoW blockchain, the tangle is secured by the current hashpower of the network, since this hashpower is what adds cumulative weight to the legitimate tangle. *Unlike* a PoW blockchain, though, nodes on IOTA only do PoW when they submit transactions. The security of the tangle, therefore, depends only on the transaction rate and the amount of PoW per transaction. Take a second to let that idea sink in because it is absolutely central to understanding the security of the tangle.

Because the IOTA network is currently small and the transaction rate is low, the IOTA team has established a single trusted node, called the Coordinator, that is ultimately responsible for deciding the current state of the tangle. Its purpose is to protect against 34% attacks, among other attacks. I'm not going to spend any more time on it, but I encourage you to read this critique and the devs' responses, and draw your own conclusions about whether IOTA can be called decentralized while running under the supervision of the Coordinator.

Let's see if we can come up with an order-of-magnitude estimate of how secure the network could be *without* the Coordinator. A recent stress test achieved well over 100 transactions per second (tps) on a small test network. The team suggested that 1,000 tps is achievable. To be generous, let's assume that IOTA will eventually scale to 10,000 tps. I don't know what the current PoW requirement on IOTA is, but let's suppose that the average IoT device is approximately a Raspberry Pi and it runs at 100% CPU for 10 seconds to do the required PoW. Again, I'm trying to be generous; many IoT devices are considerably less powerful than a Raspberry Pi, and pegging the CPU for 10 seconds for each transaction would probably be a dealbreaker.

With these assumptions, we conclude that the average computational power securing the network is roughly 10,000 x (# of computations by Raspberry Pi in 10 s) per second, or equivalently, 100,000 times the computational power of a single Raspberry Pi. There are a lot of nuances to properly benchmarking computers, but we're not concerned about factors of two or three–we're just going for an order-of-magnitude estimate–so we'll use some numbers I found on the internet.

A Raspberry Pi3 can achieve hundreds of MFLOPS (megaflops, or millions of floating-point operations per second), while high-end GPUs clock in at thousands of GFLOPS (gigaflops, or *billions* of FLOPS), a factor of 10,000 greater computing power. So in our hypothetical scenario, an attacker with ~10 GPUs could out-compute the entire network. Throw in another factor of 10 because I was being sloppy–maybe integer operations are a bit slower on the GPUs than floating-point operations, for example–and you still only need 100 GPUs to execute the attack.

I'm sure there are plenty of holes to poke in this analysis. Perhaps IOTA won't run on devices all the way at the edge of the network, for example. Instead, it might run on the gateways and routers that those IoT devices connect to, which are typically much more powerful.

Still, the point I'm trying to make is that PoW successfully secures blockchains like Bitcoin and Ethereum *because* it isn't tied to the transaction rate, or any other factor besides the economic value of the network. As the value of the mining reward (in fiat currency) increases with the price of Bitcoin, miners add more hardware and consume more electricity to mine it. The economic incentive to mine ensures that the amount of hashpower securing the network increases with the network's monetary value.

With IOTA, in contrast, there is no economic incentive to secure the network. Moreover, the hashpower securing the network is tied directly to the transaction rate, which naturally has some upper limit dependent on bandwidth and network topology.

On this last point, the IOTA developers have made a creative argument, not included in the white paper, that bandwidth limitations and network topology actually *improve* the security of the network. I haven't found an official statement of it anywhere, but after some digging I stumbled upon this Slack conversation, which is the most complete defense I could find.

Essentially, one of the IOTA developers (specifically Come-from-Beyond, a.k.a. Sergey Ivancheglo, possibly a.k.a. BCNext, also one of the original creators of Nxt), argues that the IOTA network will consist of IoT devices peered exclusively with their nearest neighbors in a meshnet topology, and that an attacker will not even have the option of peering with more than a very small number of devices on each such mesh. That is, the vast majority of devices will not be accessible from the internet or some other "backbone" of the network, and the only way to send messages to them will be through the mesh of other devices.

The general idea is that the mesh as a whole will be capable of achieving a high throughput, but each individual link in the mesh has a low enough bandwidth that an attacker would easily

saturate it by trying to add enough transactions to convince the network to follow his parasitic subtangle. Since the attacker only has a few entry points into the mesh, he saturates all of them before his parasitic tangle accumulates enough weight for his attack to succeed.

I'll let you draw your own conclusions about this argument. I personally don't think the IOTA team has made enough details public to thoroughly evaluate it.

Speaking of bandwidth limitations, let's talk about scaling.

Scalability

Because each node must validate two other transactions before submitting its own transaction, the IOTA team likes to point out that spam actually tends to make the network *more* efficient. Other members of the IOTA community get carried away with this point, sometimes even making the absurd claim that IOTA is "infinitely scalable."

Every node on the IOTA network must eventually receive every transaction in order to maintain a globally consistent tangle. Broadcasting transactions to remote nodes takes time, though, and if the transaction rate is high enough that a node receives a lot of transactions from nearby nodes before it receives the next transactions from distant nodes, the MCMC algorithm will continue to select tips submitted by nearby nodes. Eventually the tangle splits, with only nearby nodes transacting on the local copy of the tangle and remote nodes transacting on their own, divergent copy.

So bandwidth and network topology must place some limitations on the transaction rate of IOTA if the tangle is to be consistent across the entire network. We will have to wait for more stress tests to learn what these limitations are.

Additionally, like all distributed ledgers, IOTA must grapple with bloat. Each transaction on IOTA is approximately 1.6 kB in size, so a transaction rate of 100 tps would grow the tangle at a rate of 160 kB per second, or about 14 GB per day. Needless to say, that's an unrealistic storage requirement for an IoT device.

IOTA currently solves this problem by taking periodic snapshots of the tangle, which map its current state into a new genesis transaction, allowing the transaction history to be pruned away. In the limit of very frequent pruning, a node would only have to store enough of the tangle to be able to run the MCMC algorithm.

Syncing a new node with the network is a different story, though. Either the node must download the latest snapshot from a trusted peer, or it must start at the original genesis transaction and work its way forward through the entire tangle. There is no way to trustlessly *and* efficiently join the network.

Finally, it's worth noting that the IOTA team has proposed a type of horizontal partitioning of the tangle that they call a "swarm," where many nodes together store the complete tangle but no one node stores all of it. Unfortunately, there aren't many details yet on how this works.

Compared to Ardor

So what does any of this have to do with Ardor?

In my opinion, there are two main comparisons to draw, namely on the issues of security and scalability.

Regarding **security**, it isn't clear to me that IOTA could possibly reach a high enough transaction rate to be considered secure without the Coordinator, given the monetary value of even the current network, without choosing a very high PoW requirement.

Ardor, in contrast, has the advantage that its child chains are all secured by the single parent chain.

A "small" child chain wouldn't need a trusted node like IOTA's Coordinator to protect it because consensus is established by the entire network and recorded (via hashes of child chain blocks) by forgers on the parent chain.

On **scalability**, IOTA and Ardor both currently share the requirement that each node of the network process all transactions. With IOTA, this simply means adding transactions to the tangle, which is computationally cheap, whereas, with Ardor, every node must validate every transaction. Moreover, the clever design of the tangle ensures that the confirmation time for a transaction actually decreases as the network gets busier. I would not be surprised to see IOTA achieve higher throughput than Ardor as both networks grow.

On the other hand, IOTA faces a tremendous challenge in combating tangle bloat if it is ever to achieve hundreds of transactions per second, whereas Ardor has largely solved this problem.

Finally, it's worth noting that a proposal on the Ardor roadmap would delegate child chain transaction processing to dedicated subnets of the network. This would potentially achieve a computational gain similar to IOTA's "swarming" proposal, possibly allowing similarly high throughput.

Final Thoughts

If you've read this far (thank you!!) and were already familiar with IOTA, then you've undoubtedly noticed that I left out a *lot* of details, including its homebuilt hashing algorithm, the deliberate flaw in this algorithm that Come-from-Beyond included as a copy-protection mechanism, the use of ternary encoding, and the mysterious Jinn processor that will provide hardware support for IOTA in IoT devices. In the course of my research, I've formed fairly strong opinions on all of these things, but I was reluctant to share them here for two reasons.

First, I don't have sufficient information to make objective statements on these issues. I'm not a cryptographer, and I know next to nothing about ternary computing or Jinn. The best I could do would be to offer subjective judgments of the design decisions the IOTA team made, but that would have simultaneously weakened the focus of this article and opened it to criticism from people who have made different subjective judgments.

Secondly, and more importantly, I'm more interested in the fundamental concepts behind the tangle than IOTA's specific implementation of it. Regardless of whether IOTA succeeds or fails, the tangle is a beautiful idea and deserves all the attention we can muster.

So what *can* we say about the tangle, then? While I'm positively enamored with the elegance of its design and the nuances of its consensus mechanism, at the end of the day I'm afraid I'm quite skeptical of its suitability for the Internet of Things. Drop that aspect, increase the PoW requirement by several orders of magnitude, and find a way to tie the PoW threshold to the monetary value of the network without cutting ordinary users off from their funds, and I think the tangle has tremendous potential as a distributed ledger.

The last missing piece is how to cope trustlessly and efficiently with bloat, a problem that Ardor have solved extremely well. Perhaps somebody will find a way to combine the best elements of both designs at some point in the future. A lot could happen by then, especially in cryptoland.

Waves

Ardor vs The Competition Pt 4

Until now, one of my main goals with this series has been to survey different approaches to scaling a distributed ledger. This week and for the next couple of posts, though, I'm shifting my focus slightly towards the business side of blockchain technology. I'll attempt to explore the real-world problems that blockchains can solve and the ways that different projects have positioned themselves to suit their target markets.

These subjects are a bit outside my comfort zone, so I'll thank you in advance for your patience with me in case I say something ignorant or naive. And as always, I greatly appreciate constructive criticism. :)

This disclaimer is especially important this week, because this week I studied Waves.

As a newcomer to Nxt, I've read just enough about its history to know that the founder of Waves, Sasha Ivanov (a.k.a. Coinomat on nxtforum.org), had been an active member of the Nxt community until the turbulent period of early 2016, at which time he left to found Waves. I won't attempt to rehash the debate over Ardor and the future of Nxt, which I understand ended with many asset issuers like Sasha leaving the community, but if you're interested I'd highly recommend apenzl's summary in SNAPSHOT and the references therein.

Instead, for this post I'll mostly ignore the histories of Nxt and Waves, and will approach both projects with an open mind and a view towards the future. I do think there would probably be some value in a proper historical analysis, but I simply am not qualified to offer one.

With that out of the way, let's talk about Waves.

Waves

At first glance, Waves looks a lot like a stripped-down version of Nxt. It is primarily a decentralized exchange (DEX), inspired by and conceptually similar to the Nxt Asset Exchange. Like Nxt, it uses a proof-of-stake consensus algorithm and allows users to lease their balances to other accounts in order to forge in pools. It recently added a way to associate a human-readable alias to an account number, partially replicating the functionality of Nxt's Alias System. Even a couple features still in development–namely, a voting system and a way to send encrypted messages–duplicate functionality that Nxt already offers.

At the same time, Waves is missing many of Nxt's most powerful features. For now, it doesn't support anything similar to Nxt's phased transactions or account control options, for example, though it is worth noting that both smart contracts and multisig transactions are on the agenda.

Additionally, the white paper suggests that crowdfunding will be one of the main uses of the Waves platform, but tokens on Waves lack the customizable properties that make Nxt's Monetary System currencies so useful for this application. For example, the Monetary System offers the ability to condition the transfer of funds on meeting a fundraising goal, a la Kickstarter, and also the option to restrict trading so as to prevent scalpers from creating a secondary market. Using this latter feature, called a "Controllable" currency in Nxt's terminology, it is even possible for issuers to dictate both a fixed asking price *and* a fixed bid for the currency, enabling them to offer buyers full or partial refunds for their tokens. Crowdfunding on Waves, in contrast, is limited to issuing a token essentially at the market rate.

These observations notwithstanding, in my opinion it would be a terrible mistake to dismiss Waves as just another Nxt copycat with fewer features. For one thing, Waves offers several key features that Nxt and other platforms do not have, which I'll describe next. Perhaps even more importantly, though, the Waves team has built a strong brand and has offered a clear and consistent vision since the platform's inception. The field is currently so crowded, and innovation so rapid, that the combination of a simple, clear message, a strong marketing effort, and a demonstrated ability to deliver on those promises might be even more important to the long-term success of a project than the richness or novelty of its underlying technology.

Unique Features

One interesting feature that distinguishes Waves from many other platforms is the design of its DEX. It is a hybrid approach that combines a centralized order-matching engine, called the Matcher, with decentralized settlement on the Waves blockchain.

When users place orders on Waves, the Waves client sends those orders to central Matcher nodes, which maintain the order books for all tradeable pairs. Each new order is either matched against existing orders or added to the order book for the pair in question, but either way the user who submitted the new order is notified immediately whether the order was filled. It is still necessary to wait for the next block(s) to be added to the blockchain to fully confirm the transaction, but in the meantime, the user knows with high confidence the result of the order.

This might not seem like a big improvement over a fully decentralized exchange, but from the handful of transactions I made on Waves, I must say I was quite impressed by the user experience. The ability to see real-time updates to the order book, and to know immediately whether my orders were filled, made a bigger difference than I had expected.

In principle, any full node can become a Matcher. The lite client currently only connects to Matchers at nodes.wavesnodes.com by default, though, so Matchers on the rest of the network probably do not see much volume. With new orders transmitted directly to these centralized nodes, and only broadcast to the whole network once they have been filled (I think), this design allows the order books to remain anonymous. I don't know for sure how important it is for open orders to be anonymous, but it certainly seems like a feature that traders might value highly.

Another distinguishing feature of Waves is the ability to trade any token against any other token without first converting to WAVES. Combined with the integrated gateways that issue tokens pegged to U.S. dollars, euros, and several cryptocurrencies, this feature enables Waves to function as a decentralized foreign exchange market. It also allows token issuers to conduct an initial offering directly in fiat-pegged tokens. With the full client, it is even possible to pay fees in tokens instead of WAVES.

Additionally, it is worth noting that there are several features in development or on the roadmap that also distinguish Waves from other platforms. One is a reputation system that will score accounts by their age, transaction history, and other factors. There are not many details yet, but the goal is to provide users with at least a rough indication of how trustworthy a given token issuer is. The white paper even goes so far as to suggest that the reputation system will serve as "some form of decentralized KYC/AML" (know your customer/anti-money laundering) system. While it's difficult to see how a decentralized reputation system could help issuers actually *comply* with KYC and AML laws, it's not unreasonable to suppose that it could serve some analogous purpose in a blockchain community.

Speaking of compliance issues, Waves has also announced a new project, Tokenomica, that will provide a "100% compliant legal framework for different types of token crowdsales, including private equity crowdsales." Unfortunately, that quote from the 2017 roadmap is just about the

full extent of information I've been able to find about Tokenomica. My impression is that the project is still in its early stages, but it shows that the team is taking regulatory compliance seriously.

For completeness, I should probably mention that the Waves team is also planning to incorporate smart contracts into Waves. The scripting language will not be Turing complete, and there will be no equivalent to Ethereum's concept of "gas," presumably because there will be no loops. Beyond these details, there isn't much other information available yet.

Finally, I must mention the approach that the Waves team has outlined for scaling. It consists primarily of two parts: a redesign of the forging process that breaks large blocks into "microblocks" to optimize bandwidth usage; and an optimization to how account balances are stored–or rather, *not* stored–that reduces memory requirements for full nodes.

The first of these two proposals, called Waves NG, is based on Bitcoin NG. In a nutshell, once a node has won the right to forge the next block, it immediately issues a key block, which is usually empty, and then broadcasts microblocks containing transactions every few seconds. The motivation for this design is that broadcasting one large block each block interval is a much less efficient way to use the network's bandwidth, and the corresponding spikes in network activity place an artificially low bound on the number of transactions that the network can handle. By spreading transactions out over a sequence of microblocks, it is possible to increase the average data rate over the network but *decrease* the peak data rate, lessening the constraints that bandwidth and latency impose on the maximum transaction rate.

The second component of the scaling plan is to implement the ideas described in this paper by Leonid Reyzin, Dmitry Meshkov, Alexander Chepurnoy, and Sasha Ivanov. I admit I haven't spent very much time with it, but the gist is that full nodes will not all be required to store every account's balance of every token in memory in order to validate transactions. Instead, they will store a compact digest of this information, and forgers that *do* store it in full–or some subset of it, if they choose to only forge transactions involving specific tokens–will generate cryptographic proofs that they have updated the account balances correctly. The forgers will then include the proofs and an updated digest in the header of each new block. Nodes that have chosen not to record the balances of all tokens involved in those transactions will still be able to validate them by using their current digest and the forger's proofs to compute an updated digest, which they can compare to the one the forger reported.

The authors argue that this approach can reduce the amount of memory required for a full node under realistic conditions by about a factor of four. Moreover, if this optimization is able to keep all required information in memory in cases where it would otherwise have to be stored on disk, the performance improvement could be far greater – about a factor of 20, the authors suggest.

Comparison with Ardor

Although a couple of the features described were not present in Nxt, there will be similar features available in Ardor.

Specifically, Ardor's parent-chain/child-chain architecture will allow users to trade all pairs of child chain coins, some of which could be pegged to fiat currencies and other cryptocurrencies. It will also be possible to price assets in any of the child chain coins, and to pay fees in the child chain coin when transacting on a given child chain. It will not be possible to trade *assets* against each other directly, but most of those trading pairs would probably have such low volume that it wouldn't really be worthwhile to add this feature anyway.

As for the improvements that the Waves team has made to their DEX by partially centralizing it, it should be possible to mimic this functionality pretty closely by building a centralized order matcher on top of Nxt/Ardor. Indeed, the InstantDEX project accomplished something similar in the past, using Nxt to settle transactions in a decentralized manner.

On the subject of scaling, the proposal to reduce in-memory storage requirements for full nodes is intriguing, but I wonder whether there might be a small trade-off with security. (If you've read the previous articles in this series, then you have probably figured out by now that I *always* suspect that performance improvements entail reductions in security.) In particular, if nodes are not required to store the current state of every account, and must use the proofs and digest in each new block's header to validate the transactions contained in it, then I assume that means that nodes will not be required, nor even will they be *able*, to validate unconfirmed transactions before broadcasting them to their peers. I don't know the consequences of allowing nodes to propagate potentially invalid transactions across the network, but the thought makes me a bit uneasy.

Ardor's approach to scaling is for all nodes to validate all transactions, but for only the minimum possible amount of information to be permanently recorded on the Ardor blockchain. In particular, only those transactions that change the balances of ARDR, the forging token, need to

be stored on the blockchain in order for other nodes to trustlessly verify that each block was forged by an account that was eligible to do so. In contrast, the whole history of transactions involving only child chain coins and the assets and currencies traded on those child chains does *not* need to be stored on the blockchain, and hence can be pruned away, leaving only cryptographic hashes of that information behind. The result is that the blockchain stays much smaller and grows more slowly than would be the case if it stored all of this extra information.

Which approach is better depends on whether permanent storage of the blockchain or in-memory storage of current account balances presents a bigger problem as the two platforms grow. I don't know the answer to this question, but there are a couple of related points that are probably worth making. One is that the timescales of the two problems could be quite different: I could see an explosion of new assets on the Ardor platform placing an immediate strain on memory, whereas blockchain bloat would likely pose a severe long-term problem for Waves, especially if it reaches hundreds or thousands of transactions per second, which is the current goal. My other thought is that Ardor required an entirely new architecture to implement its scaling solution, whereas Waves's approach will not. It would no doubt be easier for Ardor to incorporate Waves's solution at some point in the future than for Waves to implement Ardor's solution.

Finally, perhaps the most interesting subject in this comparison is the issue of regulatory compliance. Waves has positioned itself as a platform for creating and issuing tokens, with a special focus on crowdfunding. To that end, the Waves team has indicated that they are taking a serious look at the regulatory complications that go along with crowdfunding–which might involve selling securities, for example–in order to help users comply with the law. While the suggestion that a decentralized reputation system might eventually replace traditional KYC/AML requirements strains credulity, it could at least help suppress scams and reduce the opportunities for bad actors to take advantage of others. In that sense, it might accomplish some of the same goals that regulators aim to achieve.

Ardor, for its part, will offer a couple of enhancements over Nxt that will be quite valuable for regulatory compliance. One is the ability to issue assets that can only be traded with a certain type of phased transaction, and the other is the addition of a new phased transaction type, which allows an account to approve a transaction only if the account has a certain specific property. Combining these two features, a user can issue an asset which can only be purchased by accounts that have a property that, for example, a KYC/AML-compliant identity provider has added to designate that it has verified the owner's identity.

If your asset represents shares of a company, or a mutual fund, or some other type of security, this feature would enable you to prove to regulators that you know who is purchasing your tokens. Moreover, if you are a user interested in purchasing those types of tokens, recording a proof of your identity on the blockchain via your account's properties will hopefully allow you to spend less time trying to convince businesses that you are who you say you are and that you aren't laundering money.

In addition, it will be possible to create child chains that support only a subset of the features that the Ardor platform offers. This will allow child chain creators to disable certain features, such as coin shuffling, that might raise red flags with regulators in some jurisdictions.

Conclusion

What, then, do we make of Waves? There is definitely something to be said for choosing one problem and trying to solve it better than anybody else can do. Abandoning Nxt's "Swiss Army knife" approach and focusing instead on the single goal of building a great token-trading platform no doubt made it easier to pitch, develop, and market Waves. There is also a lot to be said for starting off well-funded, as Waves did with a $16M ICO.

At the same time, though, I'm not sure that an objective comparison of Waves and Ardor could conclude that Waves is as technologically mature as Ardor is. (For the record, I have tried to do a fair and objective comparison in this article, but I am not claiming that I succeeded. That's ultimately your call.) Nxt is already capable of almost all of what Waves can do, not to mention all of the things that Waves *cannot* do, and Ardor is adding new functionality, too.

Perhaps Ardor's biggest remaining challenge is to truly sell its vision the way that the Bitcoin community and the Ethereum Foundation have sold their visions, and this is where Waves has a sizable head start. Being capable of so many different things, but not purpose-built for anything in particular, Ardor faces a very difficult task here. The worst possible outcome would be for users and businesses to see it as "just another platform," or perhaps to fail to grasp the full range of what it can do, and to simply ignore it as a result.

As for Waves, I'm excited to see what the future holds. The improvements that it has made to the Nxt Asset Exchange, though modest in my opinion, have nonetheless distinguished it as a formidable DEX. If the Waves team can follow through on their roadmap, Waves will be a fierce competitor among exchanges–centralized and decentralized alike.

Stratis

Ardor vs The Competition Pt 5

This week I studied Stratis, a blockchain-as-a-service platform based on the Bitcoin protocol.

Stratis

The goal of the Stratis project is to enable businesses to create their own customizable blockchains, choosing from a set of prepackaged features. Additionally, the Stratis Group, which guides the development of Stratis, will offer consulting services to help businesses find ways to use blockchain technology effectively, and presumably will also help them configure and deploy custom blockchains on the Stratis platform.

Put this way, Stratis sounds an awful lot like Ardor. But in most of the details–to the extent that details about Stratis are available, anyway–the two platforms are quite different. More on those differences in a bit.

Currently, the Stratis platform comprises several parts:

- NBitcoin, a comprehensive Bitcoin implementation in C# inspired by Bitcoin Core;

- NStratis, a fork of NBitcoin that adds a proof-of-stake mining algorithm and an alternative proof-of-work algorithm;
- the Stratis Bitcoin Full Node, which can run on either the Bitcoin network or the Stratis network, and which serves as the basis for the rest of the platform;
- the Breeze Wallet, a simplified payment verification (SPV) wallet for both Bitcoin and Stratis that implements TumbleBit to make transactions private; and,
- the Stratis Identity module, which allows third parties to attest to the identity of the person controlling a Stratis account.

Note that most of these components are currently in alpha.

Particularly noteworthy in this list is the integration of TumbleBit into the Breeze Wallet. The TumbleBit paper is rather dense; if you're interested in the details, I recommend instead this excellent presentation by two of the authors. In a nutshell, TumbleBit uses one-way payment channels to transfer funds from a set of payers to an intermediary called the Tumbler, and from the Tumbler to a set of payees, without any of the parties having to trust one another. The key innovation over other payment channel implementations is that TumbleBit uses blind RSA signatures in a clever way to prevent the Tumbler from knowing which incoming transaction maps to a given outgoing transaction. If many accounts are transacting through the Tumbler, then it is impossible to trace the funds in an output account back to the input account that sent them. Not even the Tumbler can link the two accounts.

Stratis's Breeze Wallet provides TumbleBit functionality for both Bitcoin and Stratis, making it useful to a much larger audience than would be the case if it worked only on the Stratis network. Moreover, since the TumbleBit protocol uses off-blockchain payment channels, it is possible to make many payments through the Tumbler in approximately the same amount of time as it takes to make a single payment.

The Stratis Identity module is still at the proof-of-concept stage, but it is functional nevertheless. Users can log into their Microsoft, Google, or LinkedIn accounts using the Stratis Identity mobile app, and these services will notify Stratis of the successful login. A special account owned by Stratis then records an attestation to the successful login by hashing the corresponding personally identifiable information (e.g., name and email address) and storing it on the Stratis blockchain.

An attestation by Google that a person owns a particular Gmail account is perhaps not the most useful identity service, but it is easy to see how the same mechanism could be used to prove ownership of some piece of information that is much more difficult to verify. For example, a government agent might attest that somebody presented a valid photo ID, together with a name and address. If a user can provide the name and address that match the hash on the blockchain, that would probably convince a service provider that the user also owned the corroborating photo ID, since the government agent attested to all three pieces of information together.

TumbleBit integration in the Breeze Wallet and the Stratis Identity module are two examples of the kinds of features that Stratis intends to offer on their platform. I'm not completely sure I've grasped the overall architecture of Stratis, but from what I can understand, the idea is for the

Stratis blockchain to delegate the backend processing for each new feature, such as TumbleBit and Stratis Identity, to a dedicated set of masternodes. For example, the upcoming Breeze Node–not to be confused with the Breeze Wallet, which uses SPV instead of requiring a full node–will be a masternode that serves as a Tumbler. Similarly, there are plans to build masternodes that process Stratis Identity transactions, though I don't really know what that means and can't find any details.

Finally, it is worth mentioning that the Stratis team has planned several other features, most notably a way to deploy sidechains anchored to the Stratis chain. My understanding is that this will be the main mechanism that Stratis uses to provide customizable, private blockchains to clients.

Unfortunately, I haven't been able to find any details about how sidechains on Stratis will work. The Stratis white paper refers to Blockstream's sidechain paper, but that is the only hint I have found so far about Stratis's design. In particular, it is not so easy to securely and trustlessly transfer value between two blockchains without having at least some of the miners on each chain validate all transactions on both chains. The details, including how the sidechain protocol handles forks and reorginzations, are crucial in order to evaluate how secure the mechanism is.

Even supposing that transfers between the Stratis chain and sidechains are secure, there is also the matter of the security of the sidechains themselves. The Stratis white paper says in several places that the Stratis chain will somehow provide security for its sidechains, but it doesn't explain how that will work. Typically, sidechains are completely independent and must secure themselves.

Compared to Ardor

With Ardor, on the other hand, the parent chain *does* provide security for each child chain.

In fact, this is one of the most important differences between Ardor's parent-chain/child-chain architecture and typical sidechain implementations. Unfortunately, without more technical details from the Stratis team, it is impossible to do a proper comparison between their design and Ardor's approach.

One comparison that we *can* do is between Stratis's TumbleBit feature and Ardor's Coin Shuffling feature. (Note that Coin Shuffling will not be available on the Ardor chain itself, but it will be available on Ignis, the first child chain, and other child chains can also choose to support it.) This feature is Nxt's implementation of the CoinShuffle algorithm, which allows a group of users to trustlessly agree to transfer a fixed quantity of coins from their (input) accounts to a set of output accounts, one per input, without any user being able to know which of the other users controls each of the other output accounts. The algorithm is not very complicated, and section 4.2 of the CoinShuffle paper gives a good overview of how it works.

I don't claim to be an expert on either algorithm, but the TumbleBit approach seems to me to have a couple of advantages over CoinShuffle. Because it uses off-blockchain payment channels, it is potentially capable of scaling to a high transaction rate in addition to adding a measure of privacy to payments, addressing two problems at once. Also, if the goal is to prevent an observer from noticing correlations between several payments–which might leak information about a business's customers or supply chain, for example–it would probably be more convenient to make the payments back-to-back from the same account via TumbleBit instead of having to first shuffle each payment to a new account.

On the subject of identity verification, I think the Stratis Identity module is an interesting proof of concept, but in my opinion Ardor provides a much richer set of tools for identity-related services. While a service like Stratis Identity can be built relatively easily on any blockchain, Ardor offers a couple of unique features that could extend such a service for some interesting applications.

On Ardor, identity validators will be able to attest to the identities of account owners using Account Properties. These are arbitrary bits of data that can be permanently associated with an account on the blockchain, rather like attestations in Stratis Identity. One novel feature that Ardor will add, though, is the ability to issue assets that can only be traded by accounts that have a specific property set.

In cases where government regulations require that asset issuers know who is purchasing their assets, this feature will allow issuers to restrict trading of their assets to accounts whose owners' identities have been verified by compliant identity providers. This level of control will hopefully help put blockchain-based securities on a firmer legal foundation, and will make it easier for asset issuers to comply with the law.

Even apart from regulatory compliance, asset issuers will probably find other uses for this feature. For example, a club or other private organization could express eligibility requirements for membership as a set of required account properties, issue an asset that only eligible accounts could obtain, and then use the asset to pay dividends to or conduct polls of members.

Some Thoughts on Marketing

Even having read this far, you might still be wondering what exactly the Stratis platform is and how it works. To be frank, I have found myself asking these questions too, even after many hours of reading about Stratis. At the risk of speaking perhaps a bit too close to the edge of my knowledge, I think it might be helpful to compare and contrast the marketing efforts of Jelurida and the Stratis Group in order to shed some light on why it is hard for me to answer these very basic questions.

Reading the Stratis website and the white paper (linked above), I got the distinct impression that, to be blunt, *those resources weren't really written for me*. The language they use reminds me of how the salespeople at my company talk, and I learned a while ago that engineers and salespeople tend not to understand each other very well.

I read that Stratis offers "simple and affordable end-to-end solutions" to "streamline and accelerate [my] blockchain project development"; that it is a "powerful and flexible blockchain development platform designed for the needs of real-world financial services businesses and other organizations that want to develop, test and deploy applications on the blockchain"; and that its "one-click process means that new chains can be launched with unprecedented speed, tailored for the needs of the organization"; but I still don't really understand what any of this means, much less *how* Stratis will accomplish these things.

This type of language conveys precisely zero information to me. Without technical details, I am completely, hopelessly lost. I know that there are plenty of people who are fluent in business-speak, though, and those people can probably read the Stratis white paper and come away with a decent, if very high-level, understanding of what the company plans to do. In contrast, it took me multiple passes through the white paper before I began to grasp the big picture, and I'm still not sure I have it right.

The Ardor white paper, on the other hand, contains substantial technical detail about how Ardor works and what distinguishes it from other blockchain platforms. It is obvious, both from its content and how that content is organized, that engineers played a significant role in writing it. Upon completing my first pass through it, I understood pretty well what problems Ardor solves and how it solves them.

The point I'm trying to make with this comparison is that business-minded people and technically-minded people often speak different languages, and the marketing materials that the Stratis Group and Jelurida have created seem to reflect this difference. Personally, I found it extremely frustrating to find so little technical substance in Stratis's resources, and this frustration has probably prevented me from really understanding Stratis.

Conclusion

Is my assessment of Stratis too harsh? Maybe. I do think that TumbleBit is an interesting piece of technology, and it seems smart for the Breeze Wallet to implement it for both Stratis and Bitcoin. Moreover, if we drop the white paper's contention that the Stratis chain will secure its sidechains, and instead assume that sidechains will be responsible for their own security, then I can use my imagination to fill in enough of the gaps to come up with at least a rough mental image of what Stratis will look like when it is complete.

This mental image, though, is basically a direct competitor to Lisk. Sure, Stratis is based on .NET and the Bitcoin protocol instead of JavaScript and Lisk's predefined transaction types, and the feature sets that the two teams intend to offer don't overlap perfectly, but essentially both projects aim to provide a central, public blockchain and a set of tools for easily creating sidechains on it. Both projects are in rather early stages of development, too, and for this reason it can be difficult to find technical details about them.

Ardor is quite different. Built on the Nxt codebase, it is already far more mature than Stratis, despite not having launched on its mainnet yet. Its parent-chain/child-chain architecture achieves the goal described in the Stratis white paper–a means for businesses to create customizable blockchains without having to worry about securing them–better than existing sidechain architectures. And the rich variety of features that Ardor already supports will take quite some time for Stratis to emulate.

Perhaps just as importantly, Jelurida and the Nxt community have done a great job of making technical information about Ardor and Nxt publicly available. This information lends credibility to the Ardor project and strengthens the community. In my opinion, it is what separates true marketing from hype.

Komodo / SuperNET

Ardor vs The Competition Pt 6

This week I studied Komodo, the blockchain platform that forms the basis of SuperNET.

SuperNET

Like Waves, SuperNET was founded by someone who was quite active in the Nxt community in the past. And as with my article about Waves, I won't attempt to rehash that history here.

Suffice it to say that James/jl777 was the developer behind SuperNET, the Multigateway, and several other projects on Nxt, including a number of assets on the Nxt Asset Exchange, but he left the Nxt community during the turbulent period of late 2015 and early 2016. Since then, he has created the Komodo platform, which now serves as the foundation of SuperNET.

The vision of SuperNET is to enable users to seamlessly transact with many different cryptocurrencies in order to enjoy the unique advantages of each coin. The experience is to be so seamless, in fact, that the user *might not even realize* that he or she is using multiple coins. For example, if I understand correctly, a SuperNET application might allow users to transact privately with Bitcoin by converting to and from a privacy coin like Komodo behind the scenes. From a user's perspective, it would be as if Bitcoin had "borrowed" Komodo's privacy feature.

SuperNET isn't itself a blockchain. Rather, it is a framework comprising several parts. The main ones are:

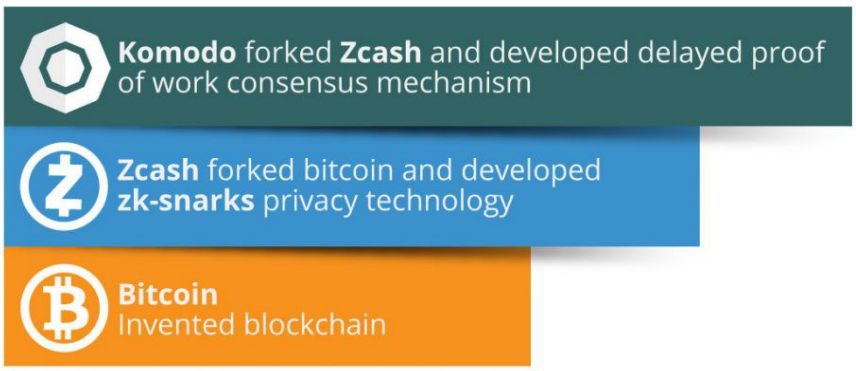
1. **Komodo**, a blockchain anchored to Bitcoin;
2. **assetchains and geckochains**, independent blockchains anchored to Komodo;
3. **the Agama wallet**, a multicoin wallet;
4. **BarterDEX**, a decentralized exchange (DEX) that will be integrated into the Agama wallet; and,
5. **Iguana**, the codebase that underlies the Agama wallet and part of Komodo.

Note that much of the literature about SuperNET refers to the Agama wallet as the "Iguana wallet," which was its previous name.

The "anchoring" process in items 1 and 2 is Komodo's delayed proof-of-work consensus algorithm, which I describe next. I'll return to BarterDEX later.

Delayed Proof of Work

Komodo is a fork of zCash, which is a blockchain that uses zero-knowledge proofs (via zk-SNARKs) to allow users to transact without publicly revealing their account numbers or the amounts that they exchange. Komodo has added several features to its branch of the zCash codebase, including the delayed proof-of-work (dPoW) consensus algorithm and a mechanism for creating additional blockchains that are periodically anchored to the Komodo chain.



The dPoW white paper argues that the dPoW mechanism allows any blockchain to secure itself using Bitcoin's hashpower by periodically notarizing itself to Bitcoin. In a nutshell, consensus on the weaker blockchain occurs in two stages: an initial consensus by normal means (e.g., PoW or PoS), and a second layer of consensus established periodically by a set of notary nodes, elected by stakeholders, that record a hash of the weaker chain's most recent block on the Bitcoin

blockchain. All nodes on the network agree that, in the event of a fork, they will not reorganize the blockchain past the last time it was notarized on Bitcoin.

In this way, the author argues, the weaker blockchain inherits some of the security of Bitcoin. Even an attacker with a large majority of the network's hashpower won't be able to modify the blockchain back past the most recently notarized block. Accordingly, somebody who waits for a transaction on the weaker chain to be notarized on Bitcoin can be confident that it won't be reversed.

The white paper also proposes a mechanism to allow the network to fall back to the initial consensus mechanism in the event that the notary nodes become unavailable. The idea is that all nodes on the network are eligible to mine, but the notary nodes are assigned a lower difficulty level than normal nodes. As a result, notary nodes will normally win most or all blocks, but if an attacker were to somehow take them offline–by a DDoS attack, for example–normal nodes would be able to continue mining blocks and the blockchain would continue uninterrupted, except without the added security of Bitcoin. In this way, the dPoW chain is somewhat less centralized than it appears at first blush.

This line of reasoning does beg the question of exactly what is gained by the notarization mechanism, though. In particular, if an attacker can gain control of the notary nodes, he can prevent them from signing the Bitcoin transactions that notarize the weaker chain's blocks, forcing the weaker blockchain to rely only on its initial consensus. So it appears that the extra security provided by the notarization process depends implicitly on an honest majority of notary nodes.

[EDIT: After talking with jl777, I learned that Komodo allows a minority of notaries, 13 out of 64, to sign each notarizing transaction. This simultaneously reduces the Bitcoin fees that must be paid and makes the proposed attack harder, since an attacker would have to control a supermajority of notaries to defeat the notarization mechanism. My original statements were based off of what he wrote in the dPoW white paper, which suggests that 33 of the 64 notaries must sign the notarizing transactions.]

This is basically the security model of delegated proof-of-stake (DPOS) blockchains like BitShares. In both dPoW and DPOS, users vote by stake for a set of "special" accounts that the rest of the network depends upon for its security. Both systems suffer the same weaknesses, too: a burden on users to keep up with the "politics" of the system to know which accounts are trustworthy enough to vote for, and the corresponding voter apathy that this burden produces.

All things considered, I'm not sure I see a strong case for dPoW over and above other alternatives. If the weaker chain's initial consensus mechanism is strong enough to secure it, given its current economic value, then paying Bitcoin fees to notarize it seems like a waste of money. If the initial consensus is *not* sufficient, on the other hand, then it seems that the security of the chain rests entirely on the election of honest notaries. But in that case, why not use DPOS and take advantage of the increased transaction throughput that DPOS chains have achieved?

Setting these considerations aside, though, it is worth noting that the Komodo platform uses nested dPoW chains to help achieve SuperNET's vision of interconnecting a variety of different blockchains. Komodo's additional chains are called "assetchains" and "geckochains". These chains notarize themselves to Komodo, which in turn notarizes itself to Bitcoin. Again, the claim is that all chains involved inherit the level of security of Bitcoin, but as described above, a lot depends on each chain's notary nodes.

Unlike assets on Nxt and Ardor, or even child chains on Ardor, Komodo's assetchains are fully independent blockchains. Their only connection to the Komodo chain is the dPoW notarization mechanism. In this way, they are perhaps closer to the sidechains that Lisk and Stratis envision than they are to Ardor's tightly-coupled child chains.

Geckochains are like assetchains but with support for smart contracts. I haven't found many details about geckochains, and they don't appear to be available yet, but the Komodo client does currently support assetchains via a command-line interface.

BarterDEX

SuperNET's decentralized exchange, called BarterDEX, allows users to atomically trade coins across supported blockchains in a trustless way. The team has not yet integrated it into the Agama wallet's user interface, but they're working on it now, and in the meantime BarterDEX can be used on its own.

BarterDEX consists of three main components: a designated set of nodes for matching orders; a set of "liquidity provider" nodes to act as market makers; and a protocol for users to exchange coins from two different blockchains with each other as a single, atomic operation.

The order-matching nodes serve the same role as they do in Waves: they partially centralize the task of matching buy and sell orders in order to provide a more responsive user experience. This way, traders don't have to wait for the next blocks on the blockchains in question to know whether their orders have been filled or to cancel an order.

Liquidity provider (LP) nodes maintain balances of at least two supported coins and automatically trade them at a user-defined profit margin relative to a centralized exchange. For example, it is possible to set up an LP node that trades BTC and KMD on BarterDEX and also on Bittrex. Operators of LP nodes assume the risk associated with holding funds on a centralized exchange, and in return they profit from arbitrage opportunities between the two markets. Other BarterDEX users, for their part, get more liquidity and tighter bid-ask spreads than they would see otherwise, without having to store their coins on centralized exchanges.

After a user's order is matched, likely to an order submitted by an LP node, BarterDEX uses an atomic cross-chain swap protocol to settle the trade on the two blockchains involved. Presumably the details vary somewhat depending on the trading pair, but conceptually the process is similar in each case. One blockchain is assumed to be compatible with Bitcoin, or at

least to support the equivalent of Bitcoin's hashed timelocked contracts (HTLCs). The other blockchain must support 2-of-2 multisig transactions.

Suppose Bob is trading his funds on the Bitcoin-compatible chain for Alice's coins on the other chain. Alice and Bob each create a public key/private key pair and exchange public keys and hashes of the private keys. Alice sends Bob a 2-of-2 multisig transaction that he can spend once he knows both private keys, and Bob sends Alice a hashed timelocked transaction that Alice can spend by revealing her private key. Once she does, Bob uses it to unlock her multisig transaction and the trade is complete.

The protocol adds a bit of complexity to protect each party in the case that the other exits the process early. If Alice walks away without spending the transaction that Bob sent, Bob can recover his funds after the timelock on that transaction expires by using his own private key. Conversely, in order to protect Alice from the same risk, the protocol requires Bob to submit an initial "deposit" in the form of a hashed timelocked transaction. If he walks away before paying Alice, she can wait for the timelock on this deposit to expire and claim it for herself.

This is admittedly only a high-level overview of the atomic swap protocol, but hopefully it gives you an idea of how it works. The most important part is that there is no centralized exchange to facilitate the trade: Alice and Bob have exchanged coins on different blockchains without having to trust each other or some intermediary. You can find more details in the BarterDEX white paper.

Compared to Ardor

What do we make of Komodo and SuperNET, then? This question largely hinges on whether Komodo's delayed proof-of-work algorithm offers a substantial degree of additional security to Komodo and its assetchains. In my view, it does not: it offers roughly the same degree of security

as the delegated proof-of-stake algorithm, even if the notary blockchain is assumed to be perfectly immutable.

In this light, Komodo's assetchains look a lot like the user-deployable sidechains that Lisk and Stratis aim to offer. In all three projects, and in contrast to Ardor's child chains, each assetchain or sidechain is responsible for its own security. Komodo seems to have a head start on both Lisk and Stratis in terms of functionality, though, as users can already deploy their own assetchains and conduct atomic swaps on some pairs.

Note that Ardor's child chains store hashes of their blocks on the Ardor chain, rather like Komodo stores hashes of its blocks on Bitcoin, but there is a crucial difference: Ardor's forging nodes validate all child chain transactions. Each child chain effectively inherits all of the forging power of the Ardor chain, rendering it just as secure as Ardor and obviating the need for separate miners or forgers.

With regard to cross-chain atomic swaps, Ardor and Komodo are perhaps a bit more comparable. Ardor natively supports transactions among child chains and also between each child chain and the parent chain. Moreover, it supports a phased transaction type that is equivalent to 2-of-2 multisig, enabling the same kinds of atomic swaps with Bitcoin-compatible blockchains that BarterDEX uses. Ardor even adds the ability to combine multiple phasing conditions with Boolean AND, OR, and NOT operators, potentially allowing users to create the equivalent of a hashed timelocked transaction. Using BarterDEX's approach, this feature could enable atomic cross-chain swaps to any blockchain that supports 2-of-2 multisig.

Conclusion

SuperNET's vision of independent but interconnected blockchains is quite compelling, and between the Komodo platform, the Agama wallet, and the BarterDEX exchange, SuperNET has made real progress towards realizing that vision. While I am skeptical that the delayed proof-of-work algorithm provides substantial additional security to Komodo and its assetchains, the ability to quickly deploy an assetchain at least puts Komodo ahead of Lisk and Stratis in the race to build a functioning sidechain platform. Also, I see a lot of value in the ability to easily conduct cross-chain atomic swaps using BarterDEX.

Even so, I have to wonder whether there exists at the heart of SuperNET a fundamental tension between two of its goals. On the one hand, it aims to integrate the best features of many disparate blockchains, providing users and developers a seamless way to enjoy the unique advantages that each chain offers. On the other hand, it has offered Komodo as a single platform to solve most problems, supporting as it does private transactions, user-provisioned sidechains, and, in the future, smart contracts. Success at either of these goals seems to undermine efforts to achieve the other.

Ardor, for its part, also has a compelling vision, and one that is perhaps a bit more coherent: to support a multitude of businesses and projects on its child chains, making available to each a set of prepackaged features, allowing each to interact with the others, and requiring none to

provide for its own security or to store forever the histories of the others. Ardor already offers most of the technology required to realize this vision; what remains is for businesses, developers, and users to put that technology to good use.

Ethereum (smart contracts)

Ardor vs The Competition Pt 7

This week I studied Ethereum, which probably needs no introduction.

For several of the projects I've surveyed throughout this series, it has been rather difficult to find detailed, technical information. Ethereum has exactly the opposite problem: there is *so much* information available that it is difficult to distill it into a reasonable-length article without oversimplifying important ideas.

For this reason, I have chosen only two aspects of Ethereum to compare to Ardor. This installment compares its smart contracts to Ardor's smart transactions, and the next article will compare the approaches that the two platforms take to managing blockchain bloat. There are many more topics I would have liked to cover–its plans to move to Proof-of-Stake (Casper), its state-channel strategies (Raiden and Plasma), its partnerships with large companies through the Enterprise Ethereum Alliance, and a sampling of the projects running on it, for example – but discussing even a couple of these topics in satisfactory depth is a daunting enough task. Besides, the two topics I chose offer the most interesting comparisons between the two platforms, in my opinion (but see the Ardor vs. Plasma post, linked above, for some thoughts on Plasma).

Without further ado, let's talk about smart contracts.

Smart Contracts and “Rich Statefulness”

Ethereum’s design combines elements of Bitcoin and Nxt, and adds several novel features. Like Bitcoin, Ethereum uses a low-level scripting language to encode transactions, and it stores the contents of each block in Merkle trees whose root hashes are recorded in the block headers (more on this in the next article). And like Nxt, it tracks the current state of account balances and other account-specific data *directly* instead of using Bitcoin’s unspent transaction output (UTXO) model.

The most important innovations that Ethereum adds to this mixture are twofold: the ability to store scripts (contracts) in so-called “contract accounts,” which transact autonomously instead of being controlled by a user; and the ability to persist data in an account from one transaction to the next. Ethereum’s scripting language is also somewhat more powerful than Bitcoin’s language, allowing contracts to include loops and to invoke other contracts.

Combining these ideas, it is possible to create stateful “smart contracts,” which are bits of code and data that live in contract accounts and act as autonomous agents, listening for input from users and other contracts and transacting with them according to the rules defined in their contract code. The “stateful” modifier in the previous sentence is crucial: because a smart contract can have its own internal state, it is possible for one transaction to affect how subsequent transactions are processed. This is a significant departure from Bitcoin’s model, where transaction scripts only execute a single time and where the notion of the “state” available to a script is essentially limited to whether a given output is spent or unspent.

(You might have noticed that I haven’t said anything about Turing completeness. Depending on how pedantic you’re feeling, you could argue either side of the question of whether Ethereum’s scripting language is actually Turing complete. As the speaker in this excellent video explains, though, Turing completeness is a bit of a red herring anyway. Much more important is the fact that smart contracts are stateful and can transact with one another and with users in interesting ways.)

The potential applications of smart contracts extend far beyond setting conditions on the transfer of money from one account to another. Even the original white paper (which is a great read, by the way) proposed a handful of non-financial uses, including file storage, voting, distributed computing, governance of decentralized organizations, and decentralized marketplaces. Since then, developers have found plenty of other applications, too, such as decentralized messaging. And of course, the most common application of Ethereum so far, seemingly by an overwhelming margin, has been to conduct token sales for various projects.

Ardor’s “Smart Transactions”

If that list of applications sounds familiar, it might be because all but one of them have already been implemented in Nxt and Ardor as prepackaged “smart transactions.” Pioneered by Ardor’s predecessor, Nxt, smart transactions are bits of “blockchain 2.0” functionality that the Nxt and

Ardor developers have made available as part of the protocol itself. They allow developers to create blockchain applications without having to write and test their own smart contracts.

In order to enable ordinary users (i.e., non-developers) to take advantage of this functionality, too, the official Nxt and Ardor wallets include a handful of features built from smart transactions. These include:

- the Asset Exchange, where users can issue assets, trade them, and pay dividends to asset holders;
- the Monetary System, where users can issue currencies and conduct several different types of crowdfunding campaigns;
- a messaging system, which allows users to send each other plain-text or encrypted messages;
- a voting system, which allows users to conduct polls by account, account balance, asset balance, or currency balance;
- an integrated coin shuffler, which can afford users a degree of privacy by obscuring their transaction histories;
- a decentralized data store, which can record the hash of a file permanently on the blockchain and, optionally, record the file itself permanently in special archival nodes;
- a decentralized marketplace, where users can buy and sell goods and services peer-to-peer;
- a new Coin Exchange (Ardor only), where users can trade child-chain coins directly for one another; and,
- a number of advanced features, such as phased transactions, which allow users to set constraints on when and how other transactions are executed, and account properites, which can be used to associate arbitrary data with an account.

These are not the *only* applications that can be built from smart transactions, of course, but they do illustrate the breadth of what can be achieved with them. All of these features, plus a few more, will be available on Ignis, Ardor's first child chain. Creators of other child chains will have the option to implement as many of these features as needed to suit their projects.

I've heard several analogies to describe smart transactions, but my favorite is that they are like Legos, while smart contracts are like clay: the former don't provide the same degree of control over the finer details, but they are quicker and easier to use than the latter, and can still be combined to form some quite impressive final products.

The analogy isn't perfect, of course. A strong argument for smart contracts is that it is possible for potentially *all* of the business logic of a decentralized application (Dapp) to be recorded permanently and immutably on the blockchain, for example, whereas a Dapp built from a combination of smart transactions likely includes some external code. In the latter case, using the Dapp might require some degree of trust in the developer not to change the rules in later versions of it.

Viewed from another angle, though, this comparison hints at arguably the biggest drawback of smart contracts: the ease with which they allow programmers to make multimillion-dollar mistakes that cannot be corrected.

Security Considerations

Just about all software that is even modestly complex contains flaws, and too often these flaws make the software vulnerable to exploitation by an attacker. Smart contract developers face a particularly difficult task because the code they write is immutable, and as a result its vulnerabilities are permanent.

Unfortunately, catastrophic failures of buggy smart contracts have not been rare. The attack that froze $150 M worth of ether stored in multisig Parity wallets and the $30 M hack of that same wallet several months prior are the most recent examples to grab headlines, but they are not the first and almost certainly not the last. For an overview of some common vulnerabilities and analysis of several real attacks, including the infamous DAO hack, I strongly recommend this excellent paper by three researchers from the University of Cagliari.

It is worth noting that the Ethereum protocol and the Ethereum Virtual Machine (EVM) were not responsible for any of these attacks. Ethereum's supporters sometimes point this out, arguing that Ethereum itself is quite secure, and all that is needed is for developers to write better smart contracts. In a literal sense they're right, of course: in all cases, Ethereum did what it was supposed to do, and ultimately the blame lies with smart contract developers.

But personally, I wonder whether this assessment is too quick to absolve Ethereum, and whether the problem might run a bit deeper than just a few buggy smart contracts. For now, anyway, it seems to me that Ethereum's fundamental predicament is that it gives programmers tremendous power, but insufficient tools to use that power safely.

Developers' ambitions almost always exceed the level of complexity that they can achieve while keeping their code *perfectly* bug-free, and there will therefore be a constant temptation to make functionality a higher priority than security (this is nearly universal in software development, by the way). Immortalizing the buggy code that they produce by storing it in the blockchain, Ethereum brutally and mercilessly holds them to account for their sins.

Thankfully, there are certainly ways to mitigate the risk of writing vulnerable smart contracts. For example, it is possible to design a smart contract that *can* be updated by having it delegate its responsibilities to a second contract, commonly called a "library contract," at an address that can be changed to point to a different library contract later.

This approach allows developers to patch vulnerabilities, but as a consequence, it introduces the thorny question of who is allowed to switch to a new library contract. If it is a single third-party account, then the design reintroduces some degree of trust between that account and users. On the other hand, if the developers take another approach, such as allowing a majority of users to vote in order to approve each new library contract, then there are potentially further problems to solve, such as writing a secure voting mechanism, making sure that users are sufficiently informed and engaged to vote, and preventing an attacker from doing significant damage in the time it takes to organize a vote.

Another very promising approach towards securing smart contracts is to use techniques of formal verification borrowed from mathematics. I do not know much about formal methods, so please take what I write here with a grain of salt, but I do know that it is easiest (or indeed, feasible at all) with simple programs whose proper functioning can be expressed as a set of short, simple rules. In such cases, it can be possible to prove with certainty that the program contains no bugs. Even straightforward techniques like looping and recursion can complicate the analysis significantly, though, so it is best if the program under test is as simple as possible.

Why am I droning on and on about all this? Putting these thoughts together, it would seem that the best way to write smart contracts might involve: 1) keeping them as short and as simple as possible; 2) delegating the core business logic to library contracts that can be updated if necessary; and 3) reusing libraries that have been thoroughly vetted, so as to keep the amount of new code to a minimum. If the second of these points requires that users trust the contract's author to some degree, as is often the case, then contracts designed according to these three guidelines start to look a lot like Ardor's smart transactions: bits of stable, thoroughly tested code that expose the most commonly needed functionality, which developers can assemble into more complex programs.

Trade-offs between Security and Flexibility

I am not suggesting that Ardor's smart transactions can accomplish all of what Ethereum's smart contracts can securely accomplish, nor am I even arguing that combinations of smart transactions can always emulate smart contracts. What I *am* saying, though, is that I think there is a natural tension between the flexibility that a platform offers and the security of the code that developers inevitably write for it.

In this view, blockchain platforms can be located on a security-flexibility continuum. Near the "security" extreme is Bitcoin, whose scripting language is deliberately quite limited in order to prevent users from locking their coins with vulnerable scripts (though this is still possible, of course). Nxt and Ardor occupy a position somewhere toward the middle of the spectrum, limiting developers to a set of predefined transaction types but including an awful lot of functionality in those types.

Ethereum's smart contracts, on the other hand, occupy the entire spectrum. It is possible to write extremely simple, trivially secure scripts on Ethereum, and it is also possible to write more complicated scripts that contain very subtle vulnerabilities. Perhaps just as importantly, it is difficult for users to tell the difference between these cases–and unreasonable, in any event, to expect them to try. Using Ethereum safely necessarily means avoiding the "flexibility" end of the spectrum, even if it comes at the cost of introducing some extra trust between users and developers.

Finally, it is worth mentioning that Ardor offers a new feature, not previously available in Nxt, that helps it inch towards the "flexibility" end of the continuum: the ability to combine phasing conditions using Boolean AND, OR, and NOT operators to achieve primitive smart-contract-like behavior.

Briefly, phased transactions allow users to condition an underlying transaction on some event, such as approval by a certain number of specific accounts (m-of-n multisig), a vote by accounts holding a particular asset, the expiration of some amount of time (timelock), or the revelation of a secret (e.g., a hashlock). On Ardor, combinations of these phasing types can encode more complex conditions, such as, "transaction X is valid if a majority of ABC Corp.'s asset holders approve of it by date Y, unless it is vetoed by a supermajority of ABC Corp.'s board members."

It will no doubt be possible to combine phasing conditions in ways that allow for unexpected outcomes, possibly including theft or loss of funds. But the advantage over smart contracts in terms of security is still there, I would argue, since developers can focus on making sure the business logic of the transaction is sound, without having to worry about low-level bugs like race conditions. And of course, the drawback of offering less flexibility than a smart contract is still there, too.

Conclusion

With a protocol defined by a set of prepackaged smart transactions instead of a low-level scripting language, Ardor will probably never be able to offer developers as wide a range of possibilities as Ethereum does, at least in cases where everything must be done on-chain for minimal trust between parties. On the other hand, writing nontrivial contracts that follow security best practices might well require additional trust between users and developers anyway. And of course, Ethereum users ultimately have to trust the authors of smart contracts not to have made any mistakes and to have duly scrutinized and tested their code in order to make sure of it.

Naturally, you might say the same thing about any software, including Ardor's smart transactions, but there is a key difference: there is simply *so much more code* running on Ethereum. Nxt has been open-source since its inception, providing ample opportunity for peer review, and Ardor's code, which builds on the Nxt codebase, will be opened soon. Moreover, each new change to the protocol has been vetted thoroughly on a public testnet before being officially released. The same *ought* to be true of each and every smart contract, but with so much code being written, it seems like there are inevitably more opportunities for bugs to slip through into production.

In any event, I suspect that the degree to which most successful Dapps will rely on immutable *code* is still an open question. If access to an immutable *database* and a handful of simple operations on that data are sufficient for most applications, then Ardor's smart transactions seem to me to have an obvious advantage over smart contracts. If, in contrast, the notion that "code is law" turns out to be essential to the viability of most Dapps, with each Dapp requiring most of its unique code to be recorded on the blockchain in order to be truly trustless, then Ethereum's approach is probably superior.

I expect that there will be real-world applications that suit each platform. But I also wonder whether it will eventually become clear that one of the two approaches best handles a sizable majority of applications. Which approach will ultimately "win" is not at all clear to me, but I suspect that the deciding factor will be users' judgments of the degree of trust that each case requires. And since the entire appeal of blockchain technology is that it allows users to transact with minimal trust, I'd say that outcome would be quite appropriate.

Thanks for reading! If you enjoyed this article, be sure to read the next part of the series, which compares the ways that Ardor and Ethereum cope with blockchain bloat.

Ethereum (blockchain bloat)

Ardor vs The Competition Pt 8

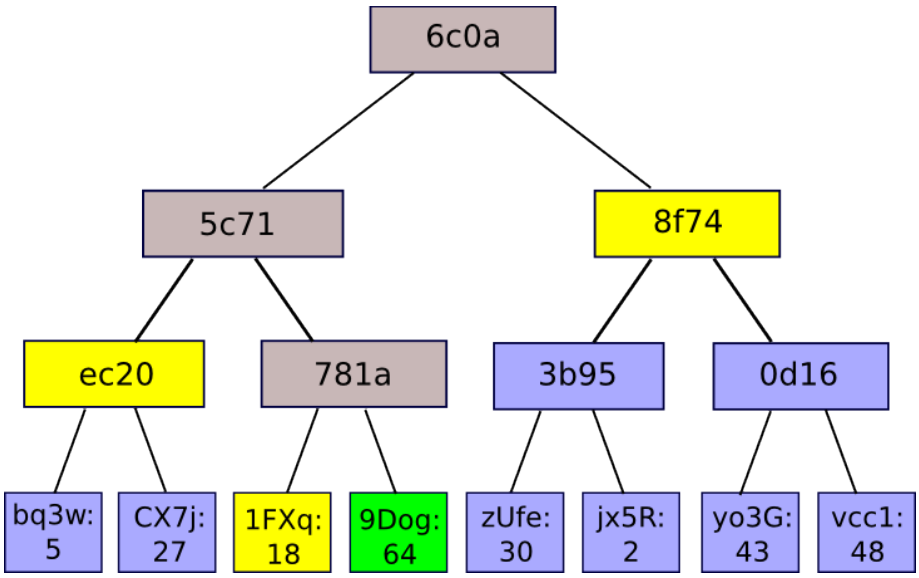
This article continues the previous installment's comparison between Ardor and Ethereum. This time, I explore how each platform approaches the problem of blockchain bloat. To my surprise, the two platforms are more similar in this regard than I had initially thought, though there are certainly significant differences, too.

Key to this comparison is an understanding of how the Ethereum blockchain is organized.

Ethereum's Structure

Like Nxt, Ethereum tracks the current state of all accounts with each new block. And like Bitcoin, Ethereum organizes the information in each block into a Merkle tree (actually, three of them) and stores its root hash in the block's header.

How exactly does this work? The diagrams from this article help to illustrate.



The leaf nodes of a Merkle tree (i.e., those at the bottom) represent all of the actual data stored in it. Each node above the leaves stores a cryptographic hash of its two children. (Note that I'm using "node" here to refer to items in the tree, not computers on the network. Each computer on the network stores the entire tree.)

This design has the property that if even a single leaf node changes by a single byte, the hash of its parent changes as well, along with the hash of its parent's parent, and so on all the way up to the topmost node, called the "Merkle root." In a sense, the Merkle root contains a digest of all of the information in the leaf nodes.

Simply grouping all of the leaf nodes together and hashing them all at once would produce a similar result, but the tree structure has a second nice property, which is that it is possible to prove that a single leaf is in the tree without seeing the entire tree. For example, in this diagram it is possible to prove that the green transaction has been included by supplying its sibling, in yellow, their parent, in gray, and the other siblings and parents along the path back to the root. Another user can compute the relevant hashes at each level in the tree, then compare the resulting Merkle root to the one stored in the blockchain. These "Merkle proofs" are the foundation of Bitcoin's simplified payment verification (SPV) clients, and also several of Ethereum's scaling proposals.

Ethereum uses three separate Merkle trees to record the data in each block: one for the block's transactions; a second for a set of "receipts" for those transactions, which represent each transaction's effects; and a third for recording the instantaneous state of all accounts, including their balances and associated data. Storing the entire state of the system with every block sounds tremendously wasteful, but since each block modifies only a very small subset of leaf nodes, most branches of the state tree do not change from block to block, and each new state tree can refer to entire branches of the previous one with minimal overhead. There are a few technical complications with this approach, and for that reason Ethereum actually uses a slightly different data structure called a Merkle-Patricia tree, but the concept is the same.

Ethereum's Fast-Sync Nodes

The most important fact in all of this is that the properties of cryptographic hash functions ensure that it is practically impossible to construct two different trees with the same root. As a result, the record of Merkle roots stored in Ethereum's block headers is sufficient to establish that the network *at the time* validated the corresponding transactions and state transitions.

In other words, even after a node has "forgotten" the contents of old blocks, as long as it keeps the (much smaller) block headers in storage, it can query a full node for a given block's contents and verify for itself that the full node has not tampered with any data. It does this simply by recomputing the relevant Merkle roots and comparing to the corresponding values in the block's header. (Note that here and for the remainder of the article, I've switched back to using "node" to refer to a peer on the network, not an item in a Merkle tree.)

This approach is exactly how the Go Ethereum (geth) wallet's fast-sync option works. To perform a fast-sync, a new node first downloads and verifies all block headers, starting with the genesis block (actually, only every 100th block header must be verified; see the GitHub link for details). Since the headers contain the proof-of-work, this step is sufficient to show that the network came to consensus about the Merkle roots in each header at the time the block was mined.

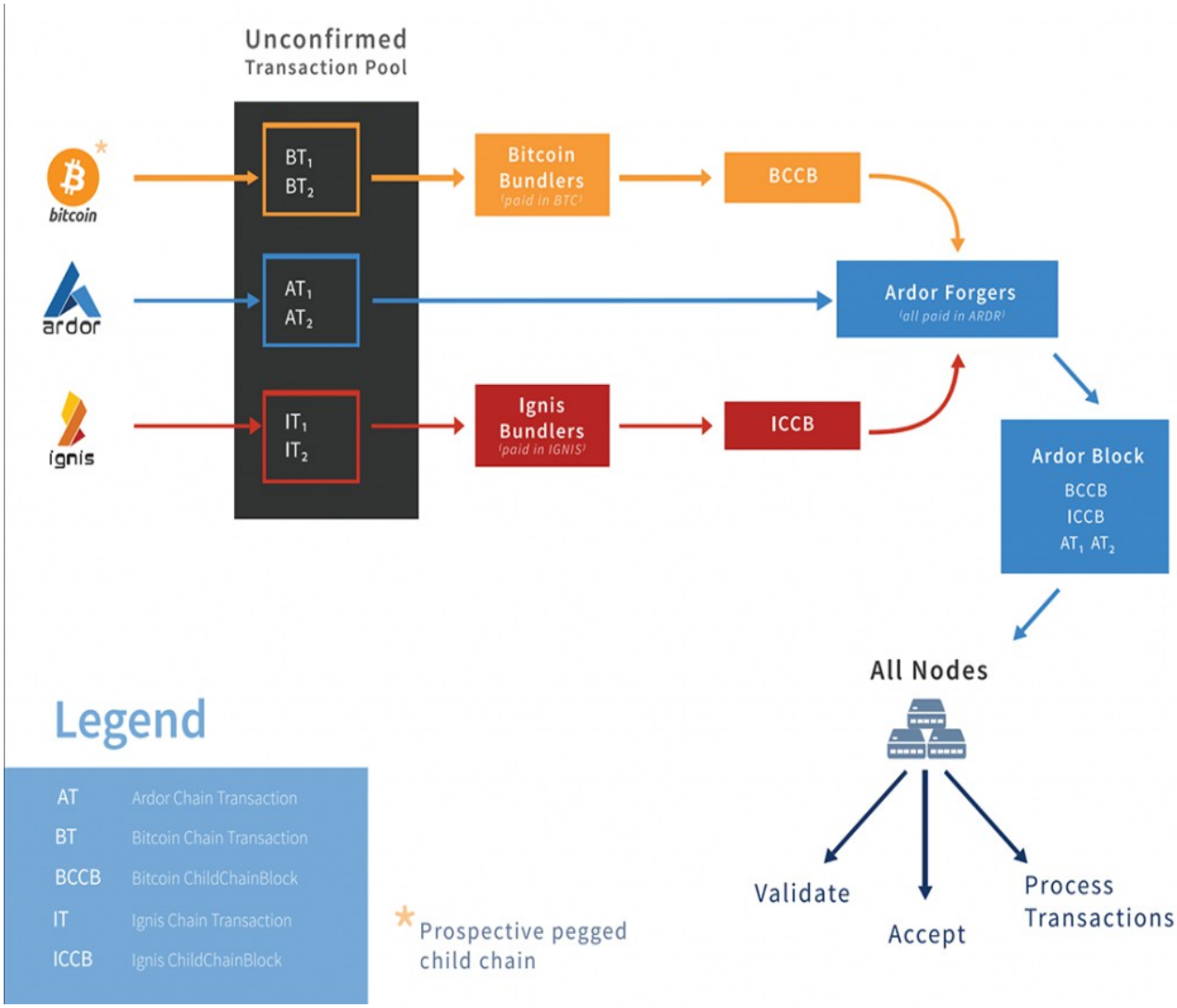
At some point in the recent past, say, 1024 blocks ago, the node gets a full version of the state tree from its peers and validates it against the Merkle root in the corresponding header. From that point forward, the node downloads full blocks from peers and replays all transactions until it has reached the most recent block, at which point it simply turns into an ordinary full node.

Although Go Ethereum does not currently support it, it is also possible for nodes to continuously prune the state tree as time progresses, keeping the amount of state data that must be stored to a minimum.

Child Chain Pruning on Ardor

If you have studied Ardor's parent-chain/child-chain architecture, this strategy hopefully sounds quite familiar. Ardor takes a very similar approach with regards to its child chains.

Briefly, the Ardor platform consists of a single proof-of-stake parent chain, also called Ardor, and a set of child chains. The parent chain supports only a few transaction types, basically, just those required for transferring ARDR around and for forging with it. The child chains, in turn, handle all of the actual business conducted on the platform using the smart transactions I described in the previous article in this series.



Only the parent chain's coin (ARDR) can be used to forge. Transactions involving only the child chains' coins do not affect the balances of the forging coin, so they are not essential to the security of the blockchain and do not need to be stored permanently. Special "bundler" nodes on each child chain collect these transactions, group them together, hash them, and report the hash to the network using a special transaction type called ChildChainBlock. They include the full transaction data along with each ChildChainBlock transaction, so forgers and other nodes can verify that the child-chain transactions are valid and do indeed produce the reported hash, but the transaction data itself is not stored in the blockchain, and after a specified time passes it can be pruned away. All that remains in the parent blockchain is the hash of this data.

Optionally, special archival nodes on each child chain can store the full history of that child chain's transactions. In cases where this history is needed, nodes can retrieve it, hash the original bundles of transactions, and verify that the hashes match the ones recorded on the blockchain.

Hopefully, the comparison to geth's fast-sync option is clear at this point: in both cases, nodes do not need to store the vast majority of transaction data to be able to verify that the network approved of those transactions at the time they were made. On Ethereum, it is sufficient to

verify the proof-of-work in the block headers and the accuracy of any given Merkle root to be able to trust the corresponding state tree. Ardor is slightly more complicated because it uses proof-of-stake for consensus, but storing the full record of ARDR transactions along with ChildChainBlock transactions ensures that nodes can verify, starting from the genesis block, that each block was forged by an eligible forger.

Comparing the Two Designs

At this point, I hope you agree with me that we can draw the following parallels between Ethereum and Ardor:

- An Ethereum full node is similar to an Ardor node that also stores the full history of every child chain.
- An Ethereum fast-sync node that continuously prunes the state tree is similar to an ordinary Ardor node, which stores the full parent chain but prunes away all child-chain data.
- Ardor offers the ability to run a node that stores the entire parent blockchain, plus the archived transaction data for a *single* child chain. This option currently has no equivalent on Ethereum.

These analogies are not perfect, of course. Specifically, it is worth noting that Ethereum's block headers are considerably smaller than full parent chain blocks on Ardor. I've also glossed over the mechanism that Ardor uses to track snapshots of the full state of the system and store hashes of those snapshots in the parent chain.

Still, I think this comparison is helpful. The third item in this list is especially interesting since it seems to be the biggest qualitative difference between the two designs. On Ardor, the ability to store each child chain's transaction history in a separate set of archival nodes allows for a type of vertical partitioning of the blockchain database. Since each child chain likely supports a

different business or project, partitioning the total set of all transactions along the lines defined by child chains seems like a natural choice. On Ethereum, perhaps the best analogy would be a design where a user could run a full node for a single project, like Golem, without having to simultaneously run full nodes for Augur and BAT and hundreds of other projects.

On that note, it strikes me that Ethereum's Merkle trees might naturally accommodate such a design, where a "Golem full node" would search the full blockchain for all transactions involving GNT, store Merkle proofs for those transactions and state transitions permanently, and discard the remaining data. I admit I haven't thought through the implications of this idea, though, so I won't say much more about it here.

In any event, neither this hypothetical strategy for Ethereum, nor Ardor's parent-chain/child-chain architecture, represents true sharding of the blockchain, since in both cases each node still must process all transactions from the whole network. These designs partition the storage, but not the bandwidth or computational power, required to run the blockchain. A proper scaling strategy must address all three bottlenecks.

Speaking of sharding…

Sharding

Ethereum's long-term vision for on-chain scaling is sharding, a way of partitioning both the storage of data *and* the processing of transactions. The goal is for most nodes on the network to have to process transactions from only a single shard, freeing them from the burden of validating and storing transactions that affect only other shards.

I won't even attempt to survey the Ethereum team's proposals here, as this article is already getting long, but if you're interested in this topic I strongly recommend their fantastic sharding FAQ on GitHub.

The reason I bring up sharding, though, is that Ardor's developers have suggested that they are exploring ways to push child-chain transaction processing to dedicated subnets of the Ardor network. They have not offered technical details yet, and I'll refrain from speculating here about how it might work, but to me, the idea certainly seems plausible.

If the devs can deliver on this idea, then the Ardor platform will look a *lot* like the "basic design of a sharded blockchain" described in the Ethereum team's document. That section of the paper describes a set of "collator" (bundler) nodes charged with collecting (bundling) transactions from a single shard (child chain), validating them, and recording their hash in a "collation header" (ChildChainBlock transaction) on the main (parent) blockchain. "Super-full nodes" (current parent-chain nodes) would process all transactions from all shards; "top-level nodes" (future parent-chain nodes) would process only the main chain blocks, but not the full contents of all collations; and "single-shard nodes" (future child-chain nodes) would process all transactions on the main chain and a single shard.

Almost all of the complications arise from cross-shard communication, and as a result, this design works best when the shards are largely independent. As I mentioned above, Ardor's child chains might naturally accomplish this kind of partitioning, with each chain supporting a separate project, where interactions between projects are allowed but still less common than transactions within a project.

Conclusion

At this early stage, these ideas are quite tentative, of course. But the possibilities are exciting nonetheless. Ardor's design already incorporates proof-of-stake consensus, a separate goal that the Ethereum team has set for itself, and a reasonable partitioning of the blockchain's data, which is an obvious requirement for any sharded solution. Notably absent in Ardor are Merkle proofs, or some other compact way for partitions to trustlessly communicate state information to one another, but it does seem like these features could be built into the platform via a hard fork. The snapshot hashes and child-chain block hashes that would become Merkle roots are already present in the protocol, after all.

But what can we say about the *current* state of the two projects? Perhaps the most interesting fact I learned in researching and writing this article is that Ethereum actually scales far better than I had originally thought. Go Ethereum's fast-sync option for full nodes affords some of the same advantages of Ardor's design, and if it eventually incorporates state-tree pruning the analogy will be even closer.

On the other hand, the main drawback of Ethereum's current design is that there must still be full nodes *somewhere* on the network, and those nodes must store all 300+ GB of the Ethereum blockchain. As it continues to grow, and the cost of running a full node grows along with it, one would expect the proportion of full nodes relative to fast-sync and light nodes to naturally decline. As a consequence, each full node will likely end up handling an increasing volume of

requests from other nodes, further increasing the cost (in terms of bandwidth and computational power) of running a full node.

Even without sharding, Ardor's design mitigates this potential problem by breaking Ethereum's monolithic full nodes into sets of archival nodes that each store the current state of only one child chain. It will be possible to store the histories of several child chains simultaneously, if desired, but few nodes, or potentially none at all, will be required to store the full history of the entire system.

Needless to say, scaling a blockchain is a hard problem. Out of the several projects that I have surveyed for this series, Ardor and Ethereum seem to me to offer the most compelling visions for on-chain scaling. And while I am hopeful that both will succeed, I must admit that, judging solely from the concrete progress that each project has already made towards achieving its vision, Ardor seems to me to have an ever-so-slight head start.

Closing Remarks

Ardor vs The Competition

This is the final installment of a series of articles that compares Ardor to other blockchain projects with similar features or goals. You can find the rest of the series here, and feel free to leave comments and critique:

- Ardor vs. Plasma
- Ardor vs. the Competition, Pt. 1: Lisk
- Ardor vs. the Competition, Pt. 2: NEM/Mijin/Catapult
- Ardor vs. the Competition, Pt. 3: IOTA
- Ardor vs. the Competition, Pt. 4: Waves
- Ardor vs. the Competition, Pt. 5: Stratis
- Ardor vs. the Competition, Pt. 6: Komodo/SuperNET
- Ardor vs. the Competition, Pt. 7: Ethereum (Smart Contracts)
- Ardor vs. the Competition, Pt. 8: Ethereum (Blockchain Bloat)

This series started with a brief, informal reddit post with my initial reactions to the Plasma paper. I didn't know at the time that it would launch me on a tour of half a dozen other cryptocurrency projects, ranging from sidechain platforms (Lisk, Stratis, arguably Komodo) to colored-coins platforms with unique features (NEM, Waves), to a project that eschews the blockchain altogether in favor of a completely different data structure (IOTA). Now that we have

come full-circle, with the last two articles focusing once again on Ethereum, I think we have reached a good place to conclude.

This series has covered a lot of ground, and I won't attempt to summarize everything here. Instead, I would like to share my thoughts on an overarching theme that emerged from my research on these projects.

Scaling Securely

As I've mentioned before, my primary interest throughout this series has been to survey various approaches to the difficult problem of scaling a blockchain. What I've learned is that there are many different strategies, but most involve a trade-off with security. I am certainly not the first one to make this observation, but I think it bears repeating here in the context of this series.

At one end of the spectrum, the most secure way to launch a new blockchain project is probably to issue a token on an existing blockchain that has already secured itself. This is the colored-coins approach that Nxt, NEM, Waves, and Ethereum use, for example. Transactions involving these tokens are recorded directly on the underlying blockchain and are therefore just as secure as any other transactions on it.

The obvious drawback of this approach is that it doesn't scale particularly well: every node on the network must process all transactions involving all tokens on the blockchain, even if the projects that those tokens represent have nothing to do with one another. Moreover, all of this transaction data is stored forever on the same blockchain, bloating it at a rate proportional to the *combined* transaction volume of all of the projects running on it.

So-called "vertical" scaling methods, which aim to allow each node to do the same amount of work faster, or store the same amount of data more efficiently, are the natural way to scale this strategy. NEM's Catapult project is a good example, as it focuses on optimizing the full client's code and the communication protocol used on the network. Waves NG, an optimization of the forging protocol, is another example.

This approach to scaling ultimately runs into limits, though. At some point, adding enough users and transactions will break these designs, and the only viable option is some form of "horizontal" scaling, where each node on the network processes and stores only a subset of all transactions.

One reasonable way to scale a blockchain platform horizontally is to push each project onto its own independent blockchain, which is the approach that sidechain platforms like Lisk and Stratis are taking. This approach occupies the other end of the security-scalability spectrum: it naturally partitions both the total computational work and storage required to run the platform and allows different nodes to handle each partition, but this scaling comes at the cost of decreased security. Specifically, with N projects running on a sidechain platform, the weakest sidechain is secured by at most 1/N of the total miners or forgers, and likely far fewer than that, especially in its infancy.

Ardor partially transcends the security-scalability spectrum, successfully partitioning the storage of child chain data without sacrificing security. The price of this benefit is that the entire network must still process each transaction. It will be interesting to see the details of Jelurida's plan to push child chain transaction processing onto dedicated subnets of the network, which would provide the missing computational and bandwidth scaling, but until then, we must refrain from speculating.

IOTA is a bit of a special case, as its design is fundamentally different from a blockchain in a couple of important ways. Without rehashing the whole mechanism of “eventual consensus” on the tangle, allow me to say that IOTA’s tangle (as it is implemented today) seems to me to be primarily a form of vertical scaling, with an element of horizontal scaling. Each node sees and stores every transaction, and although nodes can continuously prune the tangle over time, reducing the storage requirement, “permanodes” on the network must still store the entire history of the tangle in order to bootstrap new nodes trustlessly. On the other hand, nodes do not necessarily need to *validate* each transaction, as they can accept transactions that are sufficiently deep in the tangle as having been confirmed by other nodes on the network as long as they are referenced by all tips.

In other words, IOTA partitions the computational work required to validate transactions, but not the bandwidth required to relay them or the data that must be stored.

Eventually, IOTA plans to introduce “swarm” nodes to divide up the work of transaction validation and tangle storage. This will be a form of full horizontal partitioning, but I have not yet been able to find technical details, so in my opinion, it belongs in the same category as Ethereum’s Plasma and sharding proposals: a plausible-sounding idea that needs further development before it can be accepted as a real solution.

On that note, I’d like to make one final point about Ardor’s approach towards scaling: while it is not a panacea, at least at this early stage, it is important not to understate the value of an architecture that *exists and actually works*. Perhaps it goes without saying, but Ardor’s developers are not just hypothesizing about theoretical solutions to a difficult problem. They have proven that they can devise an ambitious but realistic design, implement it in a reasonable time frame, and in doing so make substantial, concrete progress towards a truly scalable blockchain. Not every team can make those claims, no matter how promising their initial ideas sound.

Final Thoughts

There is plenty more to be said about all of these projects, but this will have to suffice for now. I hope you’ve enjoyed reading these articles even half as much as I’ve enjoyed writing them. On a personal note, I would like to thank you for reading this far, and for sharing these articles with other blockchain enthusiasts. It has been immensely rewarding to see people offer their support, comments, critiques, and all manner of other reactions. I am humbled and deeply grateful that you took the time to engage with my work.

If I may leave you with a parting thought, it is this: after all is said and done, I see tremendous potential in several of these projects, but I am especially excited about Ardor. Its parent-chain/child-chain architecture simultaneously addresses two very important problems: how to cope with bloat, and how to offer a blockchain as a service to clients who do not have the resources or expertise to start their own blockchains. It is anybody's guess what economic value markets will ultimately assign to Ardor's solutions to these problems, but in my humble opinion, Ardor compares quite favorably to the competition on both points. I can't wait to see what the future holds.

Resources

Jelurida
https://jelurida.com

Whitepaper
https://www.jelurida.com/sites/default/files/JeluridaWhitepaper.pdf

NXTER.ORG
http://nxter.org

Ardor
https://ardorplatform.org

Nxtwiki
https://nxtwiki.org

NxtPlatform
https://nxtplatform.org

Source code
https://bitbucket.org/Jelurida/ardor/src