ACL: Cryptocurrency Infrastructure for Global Scale and Stable Value

ABSTRACT: We present a cryptocurrency design pattern scalable and efficient enough to operate on global scale without specialized hardware and without consensus. We combine the principles of double-entry accounting with cryptographic signing to instantiate a new form of tokenless cryptocurrency. We provide optimizations to enable the tracking and billing for the trillions of network interactions per second required to run a global-scale, decentralized hosting platform. We propose a blend of automation (akin to smart contracts) with engineered visibility to enable better decisions by human agents. Finally, we explore security considerations relative to blockchain based currencies.

This paper focuses on ACL's currency and crypto-accounting system and is published in tandem with another paper detailing ACL's approach to practical scaling of distributed applications to serve mainstream Internet users. These solutions are intertwined. The distributed computation powering ACL applications is the asset backing its currency. And the operation of that hosting infrastructure relies on the currency engine described herein to manage its internal economy.

To increase accessibility to a wider audience, each formalized mathematical section is preceded by an explanation in more plain language.

Introduction: Clearing Blockchain Hurdles

ACL is a decentralized hosting platform built on an architecture called ACLchain. ACL fulfills on the promise of blockchain by leveraging the scalable architecture of ACLchain to accomplish two primary feats:

1. Enable fully-functional decentralized applications (far more sophisticated than smart contracts) to serve mainstream Internet users, and
2. Provide the massively scalable crypto-accounting infrastructure required to host and manage applications at such enormous volumes of usage.

In recent years, blockchain, and things that run on blockchain, have risen in popularity and entered into more mainstream awareness. This has generated excitement around new possibilities for decentralized approaches to currencies, web applications, data ownership, and collective governance. Some early implementations have been demonstrably useful, yet large scale applications have been hampered by current architectures. The focus on the specific architecture of blockchain has yielded a kind of tunnel-vision, even though its massive inefficiencies and scalability hurdles are well known.

Blockchain currently consumes more than 0.1% of the world's electricity to power less than 0.0001% of the world's finances in the form of crypto-tokens with wildly fluctuating, speculative value, and no inherent redeemability. Some devotees claim the value of a cryptocurrency is based on how much energy it wastes to use it. Rather than requiring specialized processors for massive computation for mining, we present an alternative approach to reach large scales even with low-powered computing devices.

This paper has three major sections. After a brief description of the underlying platform ACL is built on, the first section details the design constraints and considerations that drive the architectural aspects of the cryptocurrency. The second section outlines the details and dynamics by which currency operates. The third section identifies security considerations and how they are addressed, security issues that remain in question, and suggestions for what can be done to minimize risks.

ACLchain: The Underlying Technology Framework

In order to cover details of the currency design, such as countersigned transactions, pre-authorization tokens, and supply dynamics, we need to establish some terminology and basic understanding of the ACLchain architectural approach. ACLchain is a crypto-operating system. While it does not follow the pattern of blockchain or other decentralized consensus systems, it provides a wholly innovative means of ensuring data integrity for peer-to-peer applications without using consensus.

In this paper, the absence of references to Proof-of-Work, Proof-of-Stake, or leader selection algorithms are not an oversight, nor due to unfamiliarity with established approaches to distributed computing or blockchain architectures. The approach used in ACL does not waste computing power on those proofs, nor on any type of global ledger consensus at all. ACLchain ensures data integrity for distributed applications through careful provenance of data published from each agent's local, immutable chain. Public entries to local chains are then shared to a content-addressable, distributed hash table (DHT). Unpredictably selected peers validate cryptographic signatures, enforce data schemas, and application logic.

Using this pattern of local immutable chains with headers notarized by an eventually consistent DHT, ACLchain introduces an agent-centric pattern for data integrity, rather than a data-centric, absolutist frame (which is what creates the need for consensus). Once a developer learns to do this kind of inversion in their thinking (from data-centric to agent-centric), then building a distributed application on ACLchain becomes much easier, and many orders of magnitude more computationally efficient than applications built for scale on blockchain.

Please consult the ACLchain white paper for a more technical formalized information about its architecture. At the time of publication, ACLchain is already in Alpha release with a number of important early applications (such as a distributed Twitter-clone, and distributed public key infrastructure), which have been designed to demonstrate its scalability and ease of use.

Design Constraints: ACL Currency Requirements

ACL is intended to be scalable, secure, and stable enough to be the foundation for a new crypto-economy. To accomplish this ACL must provide high-speed crypto-accounting, massive transaction volumes with high enough efficiency to minimize overhead, and a cryptocurrency which is stable enough for normal people to rely on for their livelihoods. It must also be structured with proper incentives for participation and provide adequate resources for maintenance of its infrastructure.

Crypto-accounting with Low Overhead

ACL mobilizes computing resources which can be fairly easily measured in terms of bandwidth, storage, and CPU time or cycles. As network computing power and access to high speed connectivity expands, costs and prices will be driven downward. The fee infrastructure involved with traditional financial transactions does not map well onto this problem space, owing to the very small, but very frequent transactions it services. The toll of having to pay a bank a 3% charge plus a $0.25 transaction fee on a $0.00025 transaction obviously makes such a system unfeasible. People can't pay over a thousand times the cost of the computing power in transaction fees.

The computational overhead of blockchain poses the same problem. When every node must agree about every state change on a global ledger, it takes drastically more computing to account for that bit of computation than the original computation provided. The computational cost of consensus management, whether through the hash-churning of proof-of-work or the betting markets of proof-of-stake, requires an unnecessary amount of extra state processing that carries inherent scaling limitations. One should not incur millions of times the computing power in accounting overhead to provide a small amount of hosting.

In summary, the first constraint which guides the design of ACL's currency system is that it must cost less in computing cycles and in fees than the original computing and funds being counted.

Massive Transaction Volume

For ACL to service mainstream Internet markets, it must be able to host large peer-to-peer versions of applications similar to Slack, Twitter, or even Facebook. Accounting capacities for the computation obviously must scale with the application's usage. As such, ACL must exceed the capacities of current blockchain throughput of a few transactions per second by at least a million-fold. With widespread enough usage, we expect to surpass financial exchange backbones like the Visa network, which has a maximum estimated capacity of 56,000 transactions per second.

Proof-of-Service

For ACL to serve the trillions of small computational interactions that it will need to handle, we've designed its currency system to support millions of transactions per second and account for service provision in batches. The batching is facilitated by signed service logs, which record signed requests and their associated signed responses, along with the underlying computational metrics to deliver them. When the log on a device accumulates enough micro-charges to cross the billing threshold (in terms of time elapsed or charges accumulated), it generates a Proof-of-Service invoice

for the services performed.

The approach of delayed invoicing parcels the accumulation of computational micro units into billable chunks for which banking fees would still be impractical, since they may still be less than a dollar. The Proof-of-Service invoice is sent asynchronously by attaching a link to the Payor's ID in the shared DHT. If the Payor fails to pay the invoices promptly, the host stops serving that application. This invoice acts as both a request for payment and a pre-authorization token, which accelerates payment by permitting it to be automatically signed to the recipient's chain without needing human approval for the transaction.

Infrastructure and Fees

ACLchain is a fully peer-to-peer architecture that enables applications to run with no centralization at all. However, to bridge between a fully peered world and the web site functionality people are familiar with, ACL builds the bridge to the "centralized web." Full details are in the Hosting Paper, but one example involves enabling people to type a domain name into their web browser and provide Domain Name Services (DNS) to resolve to, in order to connect them with a cluster of hosts for the application they are trying to reach. We are used to thinking of the web as decentralized, but once someone has experienced fully decentralized spaces with no global address or name authorities, no root servers, only content addressability via cryptographic hashes, then the aspects of centralization in the existing pattern of web and Internet become apparent.

So the ACL organization not only provides the necessary operate the infrastructure to interface with centralized web, but maintains that infrastructure and provides security updates for the ACL software and DNS services. The funds for doing this are built into tiny transaction fees on each transaction. Since ongoing funding for open source infrastructure projects has often been challenging, this is one strategy to alleviate that issue. Half of ACL transaction fees go toward ACLchain.

The fees are not only for current maintenance and services but to fund future features and infrastructure expansion. We will provide tools for crowd-direction of a significant portion of these funds, as part of our commitment to transition ACL to a purely digital entity managed by its participants. At the time of launch, the percentages and timelines will be established so that the structure of this management can be coded into the governance of ACL itself.

Accordingly, every currency transaction earmarks a small promise for payment of the transaction fee. This is akin to the Proof-of-Service process but involves recording a promise of payment rather than a request for payment. Whenever transaction fee promises accumulate to a threshold level (initially 1 credit), the next spending transaction must be a payment to settle those promises. No central authority needs to police payment when it's built into everybody's validation rules. Keep in mind, if someone alters their validation rules, they essentially exile themselves from everyone else's shared agreements, since future transactions will be rejected by all non-corrupt nodes.

Value Stability as Foundation for Mainstream Participation

While speculators may enjoy the volatility of cryptocurrency markets, most individuals and businesses are wary of relying on a form of payment with large and drastic fluctuations in value. For a majority to engage with ACL and its underlying cryptocurrency, the market value of its units should be optimized for a steady and stable trajectory, preferably growing in value over time. While this may sound to many like an impossible or naive design parameter, it is incorporated into the design of ACL's cryptocurrency.

A New Breed of Cryptocurrency

ACL offers a whole new breed of cryptocurrency. Unlike its predecessors, it is not a crypto-token or cryptocoin, but a mutual credit accounting system where every transaction is countersigned on the local chains of both counterparties. This allows us to design the crypto-credits to forge new patterns of social and market behaviors that have not previously been possible.

No tokens. No coins.

Blockchain-based currencies are token-based. A cryptocoin is actually a cryptographic token with an associated unit value and a private key, which is used to spend that coin. A blockchain ledger provides the definitive list of all coins, by recording the creation of each coin and each following transaction. In blockchain, each transaction actually destroys prior coins when they are spent and creates new coins. The recipient of the transaction controls the private key of the coins paid to them, while the sender maintains the control of keys for any coin created as "change" left over from the original coins sent. Blockchain ledgers require consensus strategies because everyone must agree about what coins exist and what coins have been destroyed.

When no coins exist, no consensus is required. Instead of a token-centric ontology that requires computational overhead invested in establishing consensus, ACL uses an agent-centric ontology. Basically, this provides an upgrade to traditional double-entry accounting by using cryptographic signatures committed to immutable chains as accounts. In this double-entry accounting method, instead of managing a ***global*** ledger of coins, each agent (or user, or account) manages its own ***local*** chain of transactions. This means that each person's balance is encoded on their own chain, so when two people transact, they only need to audit their counterparty's history to be sure they have the credits they're spending. They need neither permission nor consensus from anyone else. One party's balance goes up, and the other party's balance goes down, in equal measure.

Another result of having no coins is the fact that every transaction is perfectly counterbalanced -- including the initialization of the system. Rather than minting a plethora of coins for the organization in an ICO, all the money raised by presale of hosting credits is ***debited*** from ACL's account when the presold amounts are credited to the buyer's account. Every credit has an offsetting debit, and having received the funds for the purchases, ACL's account starts out by carrying this debt.

The transaction fees described earlier are the fundamental method for earning against that debt. This revenue structure ensures that incentives stay aligned with the need to support a robust infrastructure, and that ACL is incentivized to keep the fee from becoming so onerous that it makes competing systems too attractive. Moreover, the ACL account balance always shows the status of

its net value contribution, meaning that the organization is transparently held to account for all the funding it receives.

Countersigned Transactions

In ACL's design, all transactions are recorded as double-entry accounting records to the source chains of each agent participating in the transaction. As a general use example (not specifically paying for hosting), Alice agrees to sell a bike to Bob for some ACL credits. They communicate with each other via node-to-node messaging to build a transaction for a two-phase commit process.

Transaction Fields:

- Spender_ID: (hash of public key),
- Spender_Current_Header: (hash),
- Receiver_ID: (hash of public key),
- Receiver_Current_Header: (hash),
- Transaction_Amount: (double),
- Transaction _Fee_Promised (double),
- Transaction_Pre-auth: (hash -- optional) *[if non-empty; and if initiated by Spender, then it is a payment request token reference found in the receiver's source chain, else if not initiated by Spender, then it is a payment promise token found in the spenders chain], and*
- Initiated_by_Spender: (bit), [true = 1, false = 0]
- Transaction_Payload: (JSON string -- up to 512 characters - NULL terminated)

One party initiates the transaction, leaving blank the fields the counterparty must complete. Each agent exchanges the data required to validate the other's state. In the worst case, this involves each retrieving and auditing the other's chain to confirm that they are in a valid state to complete the transaction (for example, the spender has the credits they're spending). The person receiving funds indicates their approval by building a pre-flight header for their chain and sending it to the spender. When the sender responds with their header, they both commit it to their own source chains with the record of the counterparty's signature in the header they provided.

Note that BOTH parties actively participate in the transaction and must sign it to their chains. This is not the spending of a coin by a single keyholder, but a mutually agreed-upon transaction with the opportunity to validate the other party's state before transacting with them. Again, this underscores why no global consensus is needed. All nodes have the same validation rules, so if Alice can't validate that Bob can spend the credits he's wanting to spend, then her app will reject the transaction. It doesn't matter if others before her colluded with Bob. Every non-colluding actor will reject illegal operations. Bad actors, in this sense, can have no effect on legitimate users nor on the currency or its supply.

Dynamic Currency Supply

Since all valid transactions are double-entry accounting entries, ACL's internal crypto-accounting functions just like a balance sheet where every transaction keeps the sheet in balance. Every credit has an offsetting debit. Nobody ever gets to create something from nothing. There is no minting, mining, or burning of coins. This means the sum of all the positive balances is always equal to the sum of all the negative balances. When no coins exist, and a matching debit for every credit is always required, the trick to managing the currency supply is fundamentally different. The simple reason for this is that ***the net currency supply is always ZERO.***

A currency that is issued and operated this way is called a mutual credit currency. A currency where someone gets to create something from nothing is called a fiat currency, from the Latin word "fiat," meaning to "proclaim, declare, or speak into being."

In Bitcoin, being lucky enough to find a nonce for a block, via proof-of-work, gives a miner the authority to create new coins from nothing. The recent popularization of the term "fiat currency" to contrast bank-issued, national currencies from cryptocurrencies, fails to recognize that all token-based cryptocurrencies are also created by the same gesture of fiat. A token created by wasting energy and computing is not backed by anything. It is spoken into being by mining or staking.

Although the ***net supply*** of mutual credit currencies is always zero, the ***active supply*** can expand and contract in response to market demand. Issuance in a mutual credit supply can be configured to adjust with the actual market behaviors of its users. When a system issues coins by fiat (even if the fiat issuing authority is randomized by proof-of-work), it simply cannot be responsive to markets in this manner. In response to the reality of variable demand, the hard-coded supply dynamics of fiat issuance produce volatile market value. However, a mutual credit cryptocurrency can be designed for stability, instead of volatility.

The primary mechanism for managing value stability in a mutual credit currency is good credit limit algorithms that govern how much each account is allowed to go into the negative. First one must recognize when the supply is expanded or diminished via double-entry transactions. There is no change in active supply when an account with a positive balance transfers to another positive account. Same is true for a negative balance to negative balance. It is when someone spends their account into a negative balance while paying someone with a positive balance that the supply of credits in active use expands. And it is when a person with a positive balance spends to someone in

the negative that the supply of credits in active use contracts.

Since the supply is completely determined by credit limits the algorithms that determine the limits for each class of user are the only variables affecting the currency supply. When the extension of credit limits are well aligned with increase of value of the currency and its infrastructure, it is possible for the supply to grow without diminishing its value. Contrariwise, to support value stability, it is imperative to reduce the supply when market behavior indicates that demand for the currency is dwindling.

For the specific mechanics of this, see the section below on "Roles, Credit Limits, and Supply Algorithms."

Value Stable - Not Static

The most critical component to value stability of the ACL credits is the fact that they are backed by a vital modern asset: computing power. Estimated 2017 revenue for cloud hosting is $264 billion dollars, and is still growing annually. ACL's credits are not cryptographic tokens divorced from any specific value, they are integral to the operation of a large-scale computing infrastructure.

ACL credits are priced in computing units: processing time, bandwidth, and storage. They are available for purchase from ACL as well as the whole community of hosts. They are also redeemable across that whole community for computing power. Even though credits can also be used for general financial transactions, as the number of hosts grows, that mass of their computing power stabilizes valuation. Hosts set their own prices for their computing power which will tend toward stability when averaged across a large ecosystem of servers distributed across the planet.

Another feedback loop stabilizing the price is the fact that ACL hosting is feasible on commodity hardware. If the price of ACL credits rises significantly, people are incentivized to connect more computing power to the network. And since trades on exchanges are not likely to deviate wildly from the prices for which computing power is sold, this incentivization structure places a decentralized throttle on massive price pumps. This provides a substantial center of gravity for the price of ACL, tying it to the delivery of a real world asset with practical value.

Value-stable does not mean static -- there is not some kind of fixed price. The value of credits is still dependent on real world factors like the cost of electricity, computing hardware, and Internet connectivity. Variations of account balances moving closer and further from their credit limits enable small changes in the supply to strengthen immunity to wild fluctuations and pump-and-dump manipulations.

The ACL organization is a special class of user with an initial credit limit large enough to credit all the pre-sale purchases from their account. That limit is calculated from a valuation algorithm for the infrastructure services based on actual growth and demand (numbers of hosts, applications, users, etc.). Transaction fees should cover most of the costs of operation and maintenance, so this line of credit is to capitalize system improvements. Therefore, any significant expansions of supply would also be correlated to enhancing the value and capacity of the network.

Reserve Accounts

We expect the largest variations of supply to happen through Reserve Accounts. The supply of credits expands when a reserve account (such as the ACL organization) sells hosting credits for a correlated outside crypto or national currency. Reserve Accounts have a special algorithm for expanding their credit limit by committing a Proof-of-Reserve record to their chain to demonstrate receipt of a payment in the accounts' correlated outside currency for ACL credits. The outside funds are then held in reserve for network hosts who provide credit purchasers services. Any credits redeemed through a Reserve Account must have been earned for providing hosting, which can be confirmed by auditing hosts' Proof-of-Service entries and transaction histories.

While Reserve Accounts may temporarily expand the supply, they only do so in response to increased demand. As hosts redeem their hosting credits, the supply decreases again. The expansion of the supply raises liquidity for hosts giving them an opportunity to cash out increasing perceived value.

Roles, Credit limits, and Supply Algorithms

ACL credits are designed to power distributed applications operated by a network of hosts who provide computing power. Therefore, specific roles and responsibilities are defined within this ecosystem. Most of these roles can be stacked such that an agent (a node with private/public keys and the ability to interact with ACL and its hosted applications) may hold multiple roles. The exceptions to this rule are Reserve Accounts and the ACL organization, which acts as an Infrastructure Provider.

End-Users: ACL is designed to help users reach their applications, keep their data out of centralized services, and make it safe and easy to do crypto-transactions. Like all user roles, major rights and responsibilities involve following the rules encoded in the ACL app DNA, which may include payment of transaction fees when they accumulate to the payment threshold. End-Users cannot have a negative balance. They can only spend credits which are part of their positive balance.

Hosts: Hosts have the rights to copy and run applications on ACLchain, and copy and publish user data, but that does not grant them ownership of that data. If an App Provider removes an app, or an End-User removes their account, the host must purge that app and that user data from their system. Other than the underlying ACLchain engine for cryptographic services, and the ACL application for usage tracking and payment processing, no ACL host is required to host any particular application. Hosts can explicitly install particular apps, and also block specific applications. ACL applications are categorized when added to the framework.

Hosts can set their own priorities and filters by app categories, price brackets, and usage demands. If a host doesn't have much time or interest in customizing their applications, they can enable an app selection autopilot that will trigger installation of applications with more demand than available hosting power. Autopilot is a good way for the host to increase their revenue by bringing capacities

online as they are needed.

Hosts can also set their own prices. Some may opt to host certain applications for free -- consider supporting a P2P wikipedia, SETI, Genome Mapping, or other projects someone would want to share computing power with. Some people may choose higher thresholds than others, thinking they don't want to spend any bandwidth or electricity unless they are being paid above a certain amount.

Again, just like in application selection, hosts who don't want to pay much attention to pricing can configure an auto-pricing app. They can set basic priorities such as trying to serve the greatest demand, seeking the highest payers, or adjusting toward a nice middle of market zone to get some hosting volume without being flooded by it. Once a host has three months of hosting records they will receive a credit limit proportional to their hosting revenue. The credit algorithm includes anti-gaming mechanisms to deter faking transactions to inflate one's credit limit, so a host's credit will not be identical to their hosting revenue, but if they haven't been cheating it will be well correlated with it. They receive credit because we interpret a history of receiving hosting revenue as a good indicator of future capacity to earn against a negative balance.

App Providers: App Providers are responsible for the maintenance and security of apps they publish on ACL. They also agree to timely payment of Proof-of-Service invoices. Since unpaid invoices are visible to all, it is easy enough for a host to demonstrate a failure to pay. No central authority needs to intervene, and no smart-contract needs to enforce payment. Hosts can configure their app selection preferences to filter on age or accumulation of unpaid invoices. Hosts will simply stop serving App Providers if they fail to pay. The benefit of this form of accountability is that feedback loops are optimized for making smarter choices, while maintaining the power to choose. Maybe someone wants to support a start up project that can't pay yet. Maybe they're friends with the developers. Maybe they can afford to take the risk with slow payers and just charge them a little more. Why should a one-size-fits-all smart contract replace people's power to choose? When it's practical to have a smart contract as sophisticated as the ACL system, we can both automate optimal functioning and include human choice.

If an app developer sets up subscription, product, or service payments to be received in ACL, they will have access to a variant of the credit algorithm similar to Hosts. Many apps may take payments in traditional currencies, and would not qualify for credit from such activity, but they can use that cash to purchase hosting credits from ACL Reserve Accounts, which would then be held on reserve for redemption by hosts.

App Developers: If developers complete development bounties offered in ACL, they will also have access to a variant of the credit algorithm similar to Hosts. This will likely be a small portion of the ecosystem.

Selective Automation vs. Smart Contracts

There may be times that a transactional relationship is so simple that every important condition and contingency could be programmed. Imagine a world where everything just happens smoothly and on time -- even better than train schedules in Switzerland. Smart contract advocates argue for achieving this by removing all room for ambiguity or human error.

However, for the most part, the real world is much sloppier than this. Sometimes trains must stop because of an obstacle on the tracks, or a person stuck in a door. It can be tempting to think that the path to clear and corruption-proof systems is computerized automation. Yet many also recognize the irony that the first big launch of a "code is law" DAO was so faulty that the underlying platform it was built on had to do a reset just to stop it.

This goes far beyond the fact that one simply cannot count on bug-free code, to the reality that every circumstance and corner-case could never be accounted for. Attempting to do so yields high complexity, increasing the chances of bugs exponentially. If your smart contract says you don't have to deliver the thing you just built unless payment is received before block 2450928 is written, and your customer couldn't pay you because of a power outage, car accident, or sunspot, do you really want to eliminate your ability to accept the late payment?

ACL provides the informational feedback loops needed for good decision-making rather than replacing people's freedom to choose. Automating things in a ACL application is easy, but blending automation with better information for making decisions is much more resilient. ACL provides information to facilitate the following in its internal hosting market:

- Application Providers can select Hosts based on quality information in their performance records,
- Hosts can accept or reject Application Providers based on reputation for payment,
- All actors see trustworthiness data from "warrants" containing fraudulent actions signed by the actor's own key.

The most fundamental "terms and conditions" of the system are the code written into the ACL app DNA. However, ACL is also a hosted commons with expected standards of behavior. The full complement of social agreements simply cannot be encoded into smart contracts. Instead, we can facilitate better collective intelligence and healthy feedback loops that enhance people's ability to choose who they trust, while detecting unexpected or unpredictable cheating and fraud. All types of users are incentivized not to defraud others on the system with the understanding that when they are caught all privileges on ACL may be revoked. Keep in mind that EVERY communication and data element is signed by its author to their immutable chain, or it cannot propagate. If you are a bad actor, you have published a non-repudiable record of your actions – you've left your digital fingerprints all over the scene of the crime.

This enables the ACL ecosystem to have a high-functioning "immune system," because any node can create a "warrant" that flags fraudulent behaviors and provides the original signed records of the fraudster as proof. As new types of fraud are identified and able to be detected, it is easy to determine who has already done those things, then spread warrants as proof of fraud, so other nodes can opt to blacklist them.

Another agreement for all ACL users is to not generate wasteful, automated, or artificial traffic (to boost one's own or a conspirator's hosting fees). If a developer needs to test app scalability they must do that on ACLchain or on a distinct ACL testing network where test nodes (likely supplied by app providers and developers) agree to provide each other free computing space purely for the purposes of testing their apps at scale and for vulnerabilities.

For a complex ecosystem of relationships, it is far superior to set safe settings for default behaviors, and then enable people to choose to replace those settings with their own thresholds or make their own special exceptions. But because of the scale of the system (both micro-transactions and large volume), automating certain functions is crucial. Thus there are a small set of roles and processes for interactions. Below are elaborations of the types and operations of automatable processes we have thus far only explained briefly: pre-authorization tokens, Proof-of-Service invoices, and transaction fees (a special instance of the structure used for pre-authorization).

Pre-Authorized Tokens for Automated Transactions

Since every transaction requires mutual consent and participation by each party to commit it to their local chain, pre-authorization tokens are a mechanism to optimize for speed. One party can provide explicit permission in advance, reducing transaction time to milliseconds and eliminating the need to obtain approval from someone. This transforms a process that normally requires synchronous human interaction into one that can be asynchronous and automated. Or more precisely, it reduces the synchronous interacting with the pre-authorizer down to signing the transaction and providing the current top-hash of their chain.

How Pre-Authorization Works: The pre-authorizing party, such as a Host in a Proof-of-Service invoice, commits an entry to their source chain with the data needed to complete the transaction. Depending on levels of transparency or privacy coded into the application, it could be a private entry with the pre-authorization token sent by private node-to-node messaging. And in other cases, it could be a public entry, shared to the DHT then linked to the recipient for them to detect, and made available for others to see.

Proof-of-Service Invoice

Proof-of-Service invoices are public because ACL uses them for more than just facilitating payments. The host must commit a public entry to their chain with: 1) ID of the Payor, 2) amount being invoiced, 3) ID of the hosted app, and 4) the start/end sequence IDs in the hosted app's service log. It automatically gets shared to the DHT, and a link to it is also published on the base hash of [ProviderID+AppID], tagged as an "Invoice."

Paying a Proof-of-Service Invoice: Each App Provider can periodically retrieve invoices linked from their own [ProviderID+AppID] hash and process them following these steps:

1. Process analysis queue: (Sort linked invoices by timestamp, starting with the oldest)
    a. Retrieve the referenced service logs in the invoice (and optionally save them). If the host does not respond / is not online, try the next invoice.
    b. Total the logged services and charges and compare to the total invoice charge.
    c. Perform default (or as customized and detailed as desired) statistical analysis for fraud detection. All statistical processing is on the App Provider's local machine, so they can make it as complex as they like without paying for distributed computation.
    d. If all looks good, then add to payment queue; else if an inaccurate amount, reject the invoice with a message to the sender requesting corrections; else if fraudulent, log the problem and generate a "warrant" for public notice of the fraud.
2. Process payment queue:
    a. Send a transaction to the host via node-to-node messaging with an empty Receiver_Current_Header. Upon timeout, move to the next invoice.
    b. The host automatically retrieves their pre-authorization entry, and if Payor and Amount match, then it's approved. The host fills in their current header hash, and "pre-flights" by generating their next chain header and sharing it but not committing it yet. (Or they reject it, if no matching pre-authorization token is found.)
    c. App Provider/Spender signs the transaction to their chain, and replies with the new header they have committed to their chain. The Spender can then use their signed and published transaction as proof of payment to clear the linked invoice from their hash.
    d. With the Spender's header, the Receiver can commit the pre-flight header to their own chain, and attach the sender's as an additional signature. If Receiver doesn't get the Spender's header, the Receiver is frozen in pre-flight mode where they cannot commit another chain entry until timeout. Receiver then checks Spender's headers on DHT. If found, they use header to finish commit, else drop pre-flight headers and unlock their local chain.

Non-Transactional Uses of Proof-of-Services Invoices:

- Public accountability for timeliness of payments from an App Provider, which enables hosts to determine whether they want to host certain apps. Haven't you ever wished you knew before doing work that your customer was going to take 6 months to pay you?
- Public visibility of hosts' performance levels.

- Delivery of access keys to application service logs. This lets the App Provider aggregate usage statistics for their application even though service provision may be scattered across thousands, or even millions of machines.
- The access keys to logs also enable the App Provider to perform statistical analysis and fraud detection asynchronously before authorizing a payment to the host.

- Host's log access can be public or private (with access keys encrypted using provider's public key)

Transaction Fees

The transaction fee is a special case of the preauthorization structure. To avoid processing an additional transaction and an additional pre-authorization entry for every transaction, the transaction fee is built into the standard transaction code and accounted for in standard state calculations of someone's balance. When calculating someone's balance from the history of their chain, funds received are added to their running balance, and funds spent are subtracted from that balance, along with a percentage also deducted and earmarked as promised for transaction fee.

If the earmarked fee reaches the payment threshold (initially 1 credit), then the next chain entry must be a Transaction_Fee_Promise for their accrued transaction fees. That fee promise (assuming it is equal to the accrued fees) resets the earmarked fees to zero, and the counter starts again. Like the Proof-of-Service, this will be a public entry (so it is visible during chain auditing), published to the DHT, and linked to ACL's infrastructure provider account.

Fees are collected in a similar process as described above for Proof-of-Service, but with the transaction initiated by the sender within a preset time period of the creation of the Promise. The Infrastructure Accounts are configured to automatically accept payments with correctly calculated Transaction_Fee_Promises without requiring human intervention. In a system optimized to perform micro transactions, that should allow fee accrual on each chain for many thousands of transactions before a transaction fee payment is due. If use for much larger transactions becomes common, then so will a pattern of alternating large payments with small transaction fees. Compliance with the payment time period is enforceable with mutual chain audits, so no elevated enforcement authority is required.

Breaking transaction fees into separate payments initiated by the sender allows for simple transaction structures limited to two counterparties. Yet with pre-authorization tokens, it is easy to write automated applications for splitting transactions among multiple parties -- a simple, lightweight, and easily composable approach to more complexity.

General Purpose Transfers and Transactions

ACL needs to be able to support general purpose transfers and transactions that are not just hosting transactions. Here are some basic examples of why:

- An App Provider has a few apps hosted on ACL. To maintain clear accounting records they maintain separate accounts for each app. App X is their main breadwinner which brings in lots of subscription revenue in ACL credits. They make periodic transfers from App X to the accounts of App Y and App Z to cover the hosting fees.
- A host has multiple devices with each one earning credits in that device's account. They would like to pool their credits into one account to make a large purchase.

- An App Provider needs to pay an App Developer for services.
- An App Developer needs to pay a subcontractor for development services.

The hosting transaction structure is already flexible enough to support general purpose transactions, and they are clearly needed for smooth operation of actors in the ACL hosting ecosystem, so there is no reason to artificially constrain people from making whatever payment transactions they need.

Security Considerations

Minimizing Attack Surfaces

For the security conscious, we will briefly review common cryptocoin vulnerabilities here.

Anonymity

There is a fundamental difference in motivation and purpose between the ACL currency and most existing cryptocoins. ACL credits do not constitute anonymous cryptographic cash. This is a value stable, mutually-accountable crypto-economy. People have a variety of reasons for anonymity (ideology, tax evasion, avoidance of repressive regimes, illegal activities, etc.). So, they can build other ACLchain-based currencies to provide anonymizers and transaction mixers. ACL is not natively optimized for anonymity.

This paper does not consider possible breaches of anonymity in ACL currency transactions as vulnerabilities, because anonymity is not a goal of our design. Continuity of accountability is included by design -- not a bug, but a feature. As such, the currency is not optimized for illegal, black market, or underground activities. ACL credits are optimized to build a consistently reliable, peered network of hosting providers.

An additional security measure to avoid Sybil accounts is that App Providers go through real world identity verification when they sign their agreements for hosting apps on ACL. Also, hosts who wish to cash out any of their ACL through Reserve Accounts must share identification information. To further reduce risks related to failures to pay, one could choose only to transact with identified people who have long-term stake in their identity on the system. ACLchain enables continuity of identity across application contexts with its DPKI app, which can interface with decentralized identity services of your choosing.

Consensus Attacks

There are a number of attacks on blockchain which target disrupting consensus. The usual thinking is that it takes a large number of nodes and massive amounts of computing power to prevent undue hijacking of consensus. However, since ACL's currency is not based on consensus of what coins exist, but on individual accountability for one's transaction history, nobody ever needs to trust a consensus lottery. You can always audit your counterparty's chain to validate their state and know that they have the credits they're spending. You need trust nobody but yourself and your installation

of the software. Therefore, attacks on consensus are not vulnerabilities for ACL. This includes Majority (or 51%) Attacks, most Sybil Attacks, Attacker with High Computing Power, High Energy Consumption (for Proof-of-Work), Selective Dropping of Transactions, etc.

Attacks on Absolutism

Blockchain is a strategy for managing consensus on a single authoritative reality about data, and manufacturing a single authoritative reality about time. On ACL there are no coins to double-spend, and no absolute time-sequence to hack with clock drift, only the local sequence and immutable history of each agent's chain. Thus, ACL is not vulnerable to attacks on a single authoritative data set nor attacks on a universal time sequence. This eliminates vulnerabilities to Double-Spend, Clock Drift, and most Segmentation and Scalability attacks.

The Finney Attack

An attempt to execute a fraudulent high value transaction with low confirmation is a special double-spending case called the Finney Attack. This attack merits more thorough coverage, because there is a potential ACL variant, and we must provide at least a basic defense.

Imagine Alice is not doing a micro-transaction of hosting credits, but is paying Bob 1 million in ACL credits in a general purpose transaction. First, Alice makes a backup of her local chain, then spends ♓ 1M with Bob. Alice immediately restores her old chain from the backup, and then tries to spend the same ♓ 1M again with Carol before her previous transaction can propagate throughout the DHT for Carol to detect it. This is the agent-centric variant of "double-spending" a coin -- rolling back one's source-chain to before a major transaction to try to spend from their previous balance.

First, since all mutual credit transactions are countersigned by both sender and recipient, and committed to the chains of each party, we are not depending on Alice's honesty for her transaction to propagate. As the recipient, Bob is highly incentivized to publish the transaction and make sure his receipt of ♓ 1M is recognized by the network. ACLchain has a few built in security features which are turned on in the configuration of the ACL app:

1) Source chain headers are published and placed as links on DHT entry of their author.
2) Sequence IDs of headers are marked and connected to their author's DHT hash.

Therefore, Alice's peers in her DHT neighborhood hold a copy of her source chain headers and their sequence IDs. When Bob publishes his copy of the transactions, along with Alice's signature and validation of it to her chain, it propagates to Alice's neighborhood peers in the DHT. Peers automatically detect header sequence collisions and flag them as invalid.

A basic defense against the Finney Attack can be added into any app by having the recipient of a transaction over some app specific threshold (e.g. ♓100) pause for a time correlated to the risk in accepting the transaction (e.g. log2(txAmt) * 10 seconds). So, for that ♓1M transaction, when Alice tries to trade with Carol, they are locked into a delay of a little over 3 minutes to allow any previous transactions time to propagate. Then Carol requests header confirmations from Alice's neighbors. If any peers report entries that were not included in the source chain Alice provided for audit, then

Carol has detected a Finney Attack. Since ACLchain peers have automatic detection of duplicate header sequence IDs, this kind of fraud is not worth risking for small transactions, because perpetrators will just end up blacklisted.

Malicious Nodes / Rival Code

If an agent hacks their code, anomalous outputs that fail to validate with random DHT peers storing those outputs will be flagged as fraudulent and won't propagate. Bad transactions can't spread and will result in a blacklisting of the committing agent by peers. This is similar to committing bad blocks on a blockchain that won't validate, except a ACLchain node only succeeds in forking themselves into their own reality where nobody else acknowledges the validity of their chain. ACL operates in a strong Nash Equilibrium with all players incented to keep playing by the rules.

Spamming Transactions

A node cannot generate transactions with themselves, the counterparties in a transaction have to be distinct identities. Two parties (or one person controlling two accounts) could rapidly transact back and forth, paying transaction fees on each transfer. This pattern of behavior would mostly just bog down the two transacting nodes. Others would not be prevented from performing transactions, however, it would also create a flurry of gossip traffic in the neighborhoods of those peers. This could result in getting, at least, temporarily blacklisted if their peers detected the behavior as a Denial of Service attack.

Illegal Content

Since ACL credits are optimized for high volumes of micro-transactions, they do not carry much payload. So content (that it is illegal to be in possession of) cannot be stuffed into a transaction that you would be forced to hold. However, applications could certainly be built to hold such content, so hosts should take care in the selection of applications they choose to run. Note that the small payload size does not mean you can't accomplish powerful computational tasks or reference useful content, but that computation and content must be set up in an application and can only be triggered or referenced in a transaction.

Some of the Same Issues as Blockchain:

ACL has not introduced a new breakthrough in cryptography nor figured out how to prevent network congestion and human errors, so we are still subject to some of the same basic vulnerabilities as blockchain.

Breaking the Cryptography

SHA256 does not seem to be in any imminent danger of being cracked, but the crypto algorithms are configurable and can be replaced with new ones in later versions. We have not focused on Quantum-proof designs yet, and are happy to consider that for later releases as our community of cryptographers and developers grows.

DDOS

ACLchain has implemented some initial mechanisms to reduce gossip storms and to blacklist Denial of Service attackers, but many of these optimizations will need to improve over time. A Distributed Denial of Service attack is still a challenge to detect and block if there are enough attackers. A DDOS attack would not likely bring down entire applications or the ACL network as a whole, but if it targeted individual nodes on the network it could certainly disrupt their network services at least temporarily.

Human Error

People will still lose their keys, use weak passwords, get computer viruses, and sometimes they will abandon communities or even die, leaving inactive accounts behind. This is no different than things that happen with existing cryptocurrencies. ACLchain's core DPKI app (Distributed Public Key Infrastructure) can provide assistance in managing keys, managing revocation methods, and reclaiming control of applications when keys or devices have become compromised.

Conclusions

We have presented ACL as an approach to surpassing efficiency limitations of blockchain and other consensus-based distributed computing strategies. Specifically, ACL combines an agent-centric approach to cryptocurrency design that establishes a tokenless crypto-accounting engine, which eliminates the need to invest computing power or network traffic in consensus or synchronization of a global ledger of tokens.

We've illustrated how we leveraged principles from game theory and living systems feedback loops to establish an equilibrium for the value stability of currency units. We've transformed the computational efficiency from blockchains O(n) on number of transactions to O(log n). And we finally addressed common security issues for blockchain based currencies.

Optimizations

There are many optimizations that we concluded were outside the scope of presenting the basic operations of the ACL cryptocurrency engine. Some examples are:

- Chain data can be passed in large blocks from the counterparty.
- Headers can be retrieved from the DHT to ensure the chain matches what has been validated to the DHT.
- Agents can avoid auditing chain sections they've audited before by saving a bookmark of last audit point with state information.
- The ACL accounting application can be scheduled for quarterly updates to prevent unreasonable growth of transaction chains.

Criticisms

ACL is likely to draw criticism for diverging from expectations around consensus, anonymity, and a

perception of centralization for transaction fees paid to a single infrastructure provider. For these concerns, remember, ACL is just a bridge for fully distributed ACLchain apps to reach mainstream users, and its credits are just one of many currencies which can run on ACLchain has already provided a fully decentralized alternative and ACL relaxes some of the extreme decentralization characteristics to make it easier to access and more practical for widespread use.

Disclosure Regarding Long-Term Viability

Some advocates are convinced that Bitcoin will run forever. We hope that ACL does not. ACL is designed to put itself out of business by establishing widespread adoption of ACLchain. Once people are running ACLchain peers, they don't need anyone to host for them. It may take years or decades, but ACL is a transitional platform, and its semi-centralized features may be just enough motivation for people to complete the changeover

Questions Raised for Future Work

- Could ACL's currency engine be replicated for a new breed of cryptocurrency backed by distributed networks of providers of energy, food, housing, transportation, etc.?

- If governance of app changes and versioning is integrated into the app everyone is running, does this constitute a new DAO-like pattern for commons or platform cooperatives?

- Could a federation of interoperable asset-backed currencies leverage ACL's computing power and crypto-accounting engine to establish an accord solid enough for mainstream economic activity to migrate from petro-dollars to crypto-credits?

- If all modern currencies have been created by fiat, is all of economics just the economics of artificially-scarce fiat currencies? What do economics of sufficiency look like?