Pluvo: Decentralization Through Distribution

Pluvo: Decentralization Through Distribution

Shepard Moore-Berg

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

March 2019

Abstract

This paper introduces a new method for cryptocurrency distribution, Pluvo. Unlike traditional cryptocurrency networks, which typically only charge fees to currency spenders, Pluvo charges fees to currency holders. The proceeds of the fees are shared equally by all unique human members of the network. The result encourages use, discourages hoarding, heals ownership imbalances over time, and maximizes ownership decentralization. Pluvo serves as a template for all cryptocurrencies in which the health of the ecosystem relies on the ability of all members to maintain ownership of the currency and in which the utility of the network suffers in the face of extreme ownership concentration. It also serves as a model distribution mechanism for monetary authorities experimenting with new forms of digital money.

Acknowledgments

Thank you to my thesis director, Scott Bradner, and to my wife, Dr. Samantha Moore-Berg. Your help was invaluable, from inception to completion of this paper.

Table of Contents

List of Figures

Chapter I.

Introduction

Satoshi Nakamoto introduced the concept of a cryptocurrency with Bitcoin (Nakamoto, 2008). The Bitcoin network lays out a way for users to transfer electronic tokens directly between themselves without the help of a central intermediary. The intended purpose was for the Bitcoin network to function as a payment network. As Nakamoto (2008) puts it in the Bitcoin whitepaper, “A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution” (p. 1).

The fundamental innovation behind Bitcoin is that an entire currency system can be represented by a list of token transfers[1]. Rather than having physical units of currency mediate face-to-face transfers, Bitcoin allows for transfers to take place at a distance over a network by maintaining a list of every transfer enacted on the network. By following the path of a token on the list from its genesis through its most recent transfer, anyone can see all previous owners of the token, as well as the current owner of the token (the recipient of the most recent transfer). Because Bitcoin uses several elements from the field of cryptography to create the currency system, it is called a *cryptocurrency*.[2]

[1] This Bitcoin whitepaper (Nakamoto, 2008) uses the term “transaction” to refer to a transfer of bitcoins from one party to another. For clarity, this paper uses the term “transfer” to refer to a one-way movement of tokens (e.g., from party A to party B), whereas a “transaction” refers to a two-way exchange.

[2] *Cryptocurrencies*, which are elsewhere called “crypto assets” or simply “crypto,” here refer to any network comprising a distributed digital ledger and an accompanying consensus mechanism for updating that ledger. For the purposes of this paper, the terms *token* and *coin* are used interchangeably to refer to the native unit of a cryptocurrency.

Decentralization underpins Bitcoin and other cryptocurrencies. In Bitcoin, no central party controls the list of token transfers or the list of participants on the network (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016). In fact, the concept of decentralization extends to many facets of the cryptocurrency ecosystem, in addition to decentralization of control of the list of transfers. There can be decentralization in the software implementations of the network protocols, decentralization in developer communities, decentralization of currency exchanges, and decentralization of token ownership (Srinivasan, 2017). Because centralization/decentralization is not binary—rather, it exists as a spectrum (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016)—the cryptocurrency community has worked to increase the degree of decentralization in their networks along many of these dimensions. For example, projects such as Hashgraph (Baird, 2016), Stellar Consensus Protocol (Mazières, 2015), Algorand (Micali, 2016), and Casper (Buterin & Griffith, Casper the Friendly Finality Gadget, 2017) propose new consensus mechanisms that decentralize control of the list of transfers. Projects such as 0x (Warren & Bandeali, 2017) and OmiseGO (Poon, 2017) aim to remove the centralization inherent in most currency exchanges.

Relative to the amount of research done on consensus mechanisms and exchange protocols (examples mentioned above), comparatively little work has gone into increasing ownership decentralization—that is, eliminating centralized control of cryptocurrency tokens. There are a few projects whose creators have attempted to broaden distribution of their tokens. Some projects—for example, Stellar (Stellar, 2018)—performed initial distributions via airdrops, which are distribution mechanisms in which tokens are delivered for free to new users (AirdropAlert B.V., 2018). Usually there is an imposed condition for users to receive tokens from the airdrop; for example, users may have to post about the token project to social media or sign up for a Telegram channel. Other token sales, like Civic's, attempted to minimize ownership centralization by requiring buyer identification (Lingham & Smith, 2017) and then limiting the number of tokens any individual buyer could buy. In all cases, these attempts aim to increase the decentralization of token ownership.

Ownership decentralization depends on the distribution of tokens in the network. One can distinguish between two types of distribution:

- *Initial distribution* refers to the distribution of new tokens on the network. The goal of maximizing decentralization of the initial distribution is to get the tokens in as many hands as possible.
- *Forward distribution* refers to ongoing token ownership over time. Maximizing decentralization of the forward distribution means keeping the tokens in as many hands as possible.

How ownership distribution begins and evolves depends on the rules of the environment in which the currency operates. Some environments are more conducive to maximizing ownership decentralization than others. To explore ownership distribution in cryptocurrencies, consider the case of Bitcoin.

Case Study: Bitcoin Ownership Distribution

Bitcoin has limited decentralization of both its initial and forward distributions.

The lack of decentralization of its initial distribution is a direct result of how new bitcoins are created. In order to avoid network control by a central entity (e.g., a list-keeper who has sole control over all token transfers on the ledger), the Bitcoin protocol allows anyone to keep a local copy of the list (Nakamoto, 2008). However, because the list of transfers is equivalent to the state of token ownership on the network, it is crucial that all local copies remain identical.

To keep all local lists identical to one another, Nakamoto devised a system of distributed consensus so that the ledger of transfers is redundantly stored on servers all over the world (Nakamoto, 2008). Each list-keeper—called a "miner" in Bitcoin parlance—listens for recent token transfers on the network, validates those transfers, and then competes in a race with all other miners for the right to add their list of recently-validated transfers to the running list of all transfers ever recorded. To incentivize miners to maintain the list and validate all transfers, the Bitcoin protocol rewards with brand-new bitcoins the miner who wins the race. This race occurs continuously and is won on average every 10 minutes, meaning that new transfers are added to the running list every 10 minutes, and new bitcoins are generated every 10 minutes (Nakamoto, 2008).

From a currency-distribution perspective, the result of this process is that new coins in the Bitcoin network accrue only to the miners who maintain the network. Miners represent a small fraction of total users in the Bitcoin network (Yeow, 2018), but they represent the recipients of 100% of all new coins. Thus, the initial distribution is quite limited.

(Readers with Bitcoin experience may wonder what the problem is with miners receiving 100% of new bitcoins. After all, with Nakamoto's consensus mechanism, miners must be incentivized to run the network and expend the energy required to participate in consensus. But the question here is not about network incentives or consensus mechanisms. Rather, the question is about token distribution, ownership decentralization, and what they mean for the integrity of a currency system. In a system in which only a few participants receive 100% of new supply, the initial distribution will be limited.)

Decentralization of the forward distribution is inhibited by Bitcoin's low level of circulation. Owners tend to hoard, rather than spend, their coins (Baur, Hong, & Lee, 2016). There are a number of contributing factors here. One is the ethos of HODL. (HODL was initially a misspelling of "hold" on a Bitcoin message board, but since then it has been given the backronym "hold on for dear life") (Literally Media Ltd., 2018). HODL is a mindset which Bitcoin enthusiasts employ to ride through the wild swings in Bitcoin's price. HODLers are hoarders, though, in the sense that HODLing entails never spending or selling bitcoins. Another factor contributing to bitcoin hoarding is the lack of required spending, for example, on taxes. When a government requires payment of taxes in the currency issued by that government, it guarantees a use value for the currency. Bitcoin has no such requirement.

Regardless of motive, the fact that Bitcoin holders tend not to spend their bitcoins can be seen directly on the blockchain. As of September 9, 2018, 47.42% of bitcoins had not been spent in over 1 year, representing over $50bn in dormant value (Unchained, 2018). The centralized initial and forward distributions have resulted in an ownership distribution where, as of September 25, 2018, the richest 0.66% of addresses owned 87% of coins[3] (bitinfocharts.com, 2018). The distribution of bitcoins follows a power law,

[3] It is not clear how many humans are represented by these 0.66% of addresses. Individuals can control multiple addresses, and addresses can be jointly owned by many people.

with a handful of holders owning the vast majority of the currency (Kondor, Pósfai, Csabai, & Vattay, 2014).

Although this case study only looks at Bitcoin, many other cryptocurrencies have a similarly centralized distribution. For example, as of October 4, 2018, Bitcoin Cash has 92% of coins held by 0.72% of addresses, Litecoin has 85% of coins held by 1.37% of addresses, and Dash has 89% of coins held by 1.12% of addresses (bitinfocharts.com, 2018).

A New Distribution Mechanism

The Pluvo distribution mechanism is constructed as follows:

1. At the onset of the network, set three system parameters: an initial maximum supply, *m*; an evaporation percent, *r*; and an evaporation interval period, *t*. The initial maximum supply *m* is held in an unspendable reserve.
2. Register unique human participants to the network. Registration and unregistration occur continuously.
3. Periodically (every *t*), *r* percent of tokens evaporate from all accounts (including from the initial reserve) and are distributed equally to all registered participants.

The evaporation component mentioned above is equivalent to a demurrage fee, which is a fee for a delay, in this case, a delaying in spending the token. Chapter II discusses demurrage fees further, providing context on its use in currency systems and elsewhere.

The Pluvo distribution mechanism takes its inspiration from the water cycle, which also has components of evaporation and rain (U.S. Geological Survey, 2017).

Large bodies of water (e.g., the oceans) see greater quantities of evaporation than of rain. Dry land, on the other hand, sees more rain that it sees evaporation. The process of evaporation and rain keeps water cycling from the large bodies of water where it collects to the dry land where it sustains life.

Token distribution in Pluvo works similarly. Evaporation occurs proportionally from all accounts, meaning that no matter how many tokens an owner has, exactly *r* percent of them will evaporate every *t*. Rain, however, occurs equally for all registered participants. Because evaporation is proportional but rain is equal, users with above-average balances of tokens will see greater evaporation than rain, just like large bodies of water do in the water cycle. On the other hand, users with below-average balances of tokens will see greater rain than evaporation, just like dry land does in the water cycle. A registered user with an exactly average number of tokens will not be affected by the cycle of rain and evaporation, because the quantity of tokens that evaporate will be equal to the quantity of tokens collected from rain.

In Pluvo, the initial distribution is highly decentralized; all registered participants are on an equal footing, with no single group receiving more new tokens than any other registered participants. (As a reminder, new tokens come from the initial reserve of *m* tokens and are distributed by rain.) The forward distribution tends toward decentralization as well, precisely because users with above-average token balances will see their balances fall every *t*, while users with below-average token balances will see their balances rise every *t* (all else equal).

Outline

The rest of the paper proceeds as follows. Chapter II discusses prior work regarding currencies and crypto tokens that have built-in circulation or that aim for high levels of ownership decentralization. Chapter III describes the Pluvo protocol and the requirements for a system implementing the protocol. Chapter IV outlines the tools used in implementing the Pluvo protocol. Chapter V details the implementation of the protocol. Chapter VI discusses use cases for the Pluvo protocol, and Chapter VII concludes. Appendix A walks through the code that implements the protocol, and Appendix B walks through how to use the interface.

.

Chapter II.
Prior Work

This chapter discusses previous attempts to build currencies with the characteristics of Pluvo, namely demurrage and universality.

Demurrage Currencies

In its dictionary sense, *demurrage* is "a charge for detaining a ship, freight car, or truck" (Merriam-Webster, Inc., 2018). It is related to the word "demur," and it ultimately comes from the Latin root "morari" meaning "to delay." In the shipping sense, if a charterer detains a ship in port for longer than agreed upon with the owner, the charterer owes a fee to the owner of the ship. This fee for delay is called "demurrage."

Silvio Gesell introduced the idea of demurrage for currencies in his seminal work *The Natural Economic Order* (Gesell, 1916). In it, he describes the concept of "freigeld" or "free money[4]." His implementation relied on paper money that would lose 1/1000 of its value weekly, or around 5% annually. Demurrage in this context still represents a fee for delay, but here the delay is in spending the currency. The goal of implementing demurrage in a currency was to encourage use of the money. Anyone who delayed in spending their money would be charged a small but not insignificant fee.

In order for currency holders to retain the value of the currency, they had to affix stamps, similar to postage stamps, to the bills each week. Each stamp cost 1/1000 of the

[4] Here, the word "free" refers to the intent to make the currency interest-free.

face value of the bill. Those left holding the bill at the end of the week were responsible for affixing the stamp if they ever wanted to use the bill again. At the end of each year, bills that were filled with stamps could be returned to the currency office and exchanged for fresh bills.

The town of Wörgl, Austria, experimented with the idea in 1932, two years after Gesell’s death and in the throes of the Great Depression (Boyle, 2002). The new schillings circulated with unprecedented speed and were considered an unqualified success. The following excerpt from *The Week*, appearing May 17, 1933, near the end of the roughly one-year Wörgl experiment, explains how the new money worked in the impoverished town:

> On the first day when the new notes were used, eighteen hundred schilling worth was paid out. The recipients immediately hurried with them to the shops, and the shopkeepers and merchants hastened to use them for the payment of their tax arrears to the municipality. The municipality immediately used them to pay the bills. Within twenty-four hours of being issued, the greater part of this money had not only come back to the municipality in the form of tax payments, but had already been passed on its way again. During the first month, the money had made the complete circle no less that twenty times. (Boyle, 2002)

The experiment was soon copied in America, where economist Irving Fisher helped bring it to hundreds of cities and towns in America (Fisher, 1933). Both Austria’s and the USA’s experiments met similar fates. In Austria, the Austrian National Bank shut down the Wörgl experiment after just over a year (Unterguggenberger, 1934). In the USA, the Emergency Banking Act of 1933 put an end to similar experiments (Gatch, 2008).

In 2003, Christian Gelleri created a regional currency, Chiemgauer, to implement Gesell’s idea of a demurrage currency (Gelleri, 2009). Chiemgauer circulates solely in the town of Chiem, Germany, in southeast Bavaria. Although the currency began as a school project, Gelleri continued its administration even after it outgrew the school. The students gave Chiemgauer an 8% annual demurrage fee, charged quarterly; that is, a 1 CH banknote requires a 0.02 CH stamp to be affixed to it every 3 months for it to remain valid. Purchases of Chiemgauer are fixed at a rate of 1 EUR per CH, and a fee of 5% is charged for converting CH back to EUR. The person who purchased the CH initially can designate a local charity to receive 3% of the 5% fee; the issuing institution (of which there are 50) keeps the remaining 2% fee. Though Chiemgauer has continued to circulate since its introduction, its benefits have been confined to the town of Prien am Chiemsee, Germany.

In 2012, Freicoin attempted to implement demurrage in a cryptocurrency (Friedenbach & Timón, 2018). Freicoin is a fork of Bitcoin and shares its basic structures. It has a 5% annual demurrage fee that accrues to the miners, rather than to all members of the network as it does in Pluvo; this has the benefit of providing miners a constant stream of revenue at the expense of system-wide distribution.

Universal Currencies

Universal currencies aim to have both a wide initial and a wide forward distribution. In *The Relative Theory of Money* (Laborde, 2017), Stéphane Laborde discusses two dimensions upon which ownership should be decentralized:[5]

1. *Spatial*. A currency must not inordinately favor one group over another at any point in time.

[5] Writing in French, Laborde uses the phrases for "spatial symmetry" and "temporal symmetry" to describe the spatial and temporal dimensions of decentralization. He also describes universal currencies as "free money." Here, the word "free" does not mean "given without charge," as in "free pizza." Rather, it means "not subject to control by others," as in "free press.".

2. *Temporal*. The distribution of wealth among future generations must not be overly determined by the distribution of wealth of their ancestors.

To understand these dimensions of ownership decentralization in context, consider whether or not they are achieved in Bitcoin. In both of these regards, Bitcoin fails to serve as a universal currency. With regard to the spatial dimension, Bitcoin distributes all new coins to miners alone, rather than equally to all members of the network. With regard to the temporal dimension, Bitcoin's lack of circulation means wealth distribution is largely determined by the initial currency holders. Bitcoin enthusiasts who bought coins in 2009 or 2010 could buy them for less than $1 per bitcoin, and miners received 50 new bitcoins every 10 minutes. By 2013, half of all the bitcoins that will ever exist had been already placed into circulation. By 2016, the mining reward had fallen to 12.5 new bitcoins every 10 minutes, and by 2017 the price had risen to over $10,000 (blockchain.info, 2018). The way in which those bitcoins were distributed will have a large effect on the future distribution of bitcoins and the allocation of resources in the Bitcoin community.

Another example of ownership centralization can be seen with the US dollar, which has asymmetries similar to those of Bitcoin. When the government chooses to inject new money into the economy, it does so by increasing the reserves of commercial banks (The Economist, 2015). In theory, the banks would then lend out that money to borrowers, who would then spend the money. What this means is that all new money directly benefits the banks and borrowers—spatial asymmetry. Furthermore, the distribution of US dollars at any point in time strongly influences the future distribution of dollars. That is, the rich tend to get richer, which can be seen in the change in wealth distribution in the USA over time (Stone, Trisi, Sherman, & Horton, 2017).

To overcome these asymmetries—or at least provide a countervailing force—a universal currency must distribute all new coins according to the following principles:

1. To maintain spatial symmetry, all new coins must be distributed to members of the network equally. No single person or group should have an advantage in the creation of new money.
2. To maintain temporal symmetry, individuals' token balances should tend toward the average over time when the coins are not in use.

Note that these symmetries are not absolutes. A well-functioning economy requires a certain amount of temporal asymmetry. For example, if Alice sells Bob a baseball for $1 today, then Alice must be able to use that dollar to buy approximately one dollar's worth of goods in the future. If she could not do so, then she would not accept the dollar in exchange for the baseball in the first place. Perfect temporal symmetry would mean that everyone maintained an equal balance of currency every day; this would necessarily preclude or nullify any transfers of currency, and it would thereby render the currency useless.

However, large degrees of temporal asymmetry are likewise untenable. Consider a world in which Alice and Bob both work as farmers. One day Alice captures and enslaves Bob, and for the next 400 years, all of Alice's descendants enslave all of Bob's descendants. Alice and her family accumulate immense wealth on the backs of Bob and his family. Then one day Alice's family is forced to relinquish her control over Bob's family. Alice's family, despite being roughly equal in size to Bob's family, controls 99%

of the currency, and Bob’s family controls only 1%. Without a mechanism for evening out the distribution of wealth over time, historical asymmetries will persist.

Below are several examples of newly formed digital currencies that aim, usually explicitly, to be universal currencies. Some, like Duniter (Moreau, 2018), Circles (Circles, 2017), and Democracy Earth (Democracy Earth, 2018), hew closely to the principles of universal currencies. Other, like Manna (Mannabase, Inc., 2018), SwiftDemand (SwiftDemand, 2018), and Raha (Raha, 2018), depart from universality in one or more ways.

Duniter (Moreau, 2018) is a French currency aiming for universal acceptance. It implements ideas directly from the *Relative Theory of Money* (Laborde, 2017); namely, it aims for spatial and temporal symmetry. It achieves both via constant money supply growth, rather than via demurrage.

Duniter launched in March 2017 with a currency called Ğ1. To create the initial supply of currency, 10 Ğ1 are distributed daily to each member of the currency network. The distribution is called a Universal Dividend, and all members receive the same distribution. Although at first the 10 Ğ1 dividend represents rapid growth in the money supply, Duniter proposes to eventually cap the annual money supply growth rate at 10%.

In order to ensure that only unique individuals become members of the network, Duniter uses a Web of Trust: each new applicant must receive 5 certifications from existing members before gaining membership. The Web of Trust concept was first introduced by Phil Zimmerman in 1992 for the Pretty Good Privacy (PGP) email encryption scheme (Zimmerman, 1994). The scheme allows the network to grow gradually with members who recognize and trust each other. However, it does not defend against sybil attacks (that is, fake accounts, also known as “bots” or “clones”) should any subset of the members in the network privately collude to create fake accounts.

Circles (Circles, 2017) takes a different approach. Rather than issuing a single fungible currency, Circles allows members who join the network produce their own credits according to a predetermined supply schedule. All members get to mint their own money to themselves every day at a set rate. Other members get to decide whose credits to take and whose to deny—that is, coins minted by distinct users are not automatically fungible. To re-gain fungibility, members can join groups and pool their credits into group currencies; the credits of all members within a group are fungible. Because credits can only be minted according to a fixed schedule, all credits should be fungible (and therefore able to join into one fungible group), except in the cases where someone is suspected of having multiple accounts.

Circles explicitly allows the creation of multiple accounts; however, the lack of fungibility is what prevents sybil attacks from being meaningful. For example, if Alice accuses Bob of having two accounts, she can simply refuse to accept Bob’s credits. Additionally, groups can have validators to reduce the chances that their members have multiple accounts. Lastly, like Duniter, the money supply grows over time as new coins are minted by each user.

Democracy Earth (Democracy Earth, 2018) has a similar goal of giving an equal number of coins to all people on earth. Its mission is different—rather than serve as a currency system, Democracy Earth tokens are meant to serve as vote tokens in a global democracy. To prevent sybil attacks, Democracy Earth employs a video-based Proof of Identity. Users submit a three-minute video of themselves stating their names, reciting biometrics (birthday, height, weight, sex), showing their face, and sharing witnesses' certifications. A hash (cryptographic digest) of the video is stored on the Bitcoin blockchain (or, because Democracy Earth is still in its infancy, potentially via a Blockstack namespace). To validate the videos, Democracy Earth then uses a process called Attention Mining, originally pioneered by Luis von Ahn and others in the reCAPTCHA system (von Ahn, Maurer, McMillen, Abraham, & Blum, 2008). It relies on this crowd-sourced validation to weed out spoof and duplicate videos.

Manna (Mannabase, Inc., 2018) is cryptocurrency that aims to be universal through weekly distributions to all verified users. The supply of manna grows at a fixed rate of 3.5% per year, with all new manna being distributed to verified recipients. The initial distribution is not entirely universal—there is a temporary referral bonus program, meant to encourage growth of the nascent network, in which users who refer other users will receive additional weekly distributions. Additionally, because the Manna network operates on its own Bitcoin-like blockchain, it pays a 5 manna/block reward to miners.

SwiftDemand (SwiftDemand, 2018) is a digital currency that also aims to be universal by distributing 100 Swifts per day to each registered participant. It departs from universality in several important ways. First, like Manna, it uses a temporary referral bonus program, providing referrers with 500 new Swifts for each referral. Second, in order to ensure that users are still alive, users must claim their daily distributions. However, SwiftDemand only allows 7 days of Swifts to remain unclaimed. This means that if users do not claim their daily distributions for over a week, they will lose further distributions until they make a claim. Third, part of the income distribution is reserved for

Delegated Nodes, which create new blocks, and Identity Providers, which validate users' identities.

Raha (Raha, 2018) is a digital currency with an equal distribution method. Each user mints 10 Raha every week. It is close to universal, but departs from universality in a few ways. Like with SwiftDemand, Raha's users must mint Raha in a timely fashion; after 4 weeks, users can no longer accrue unminted Raha. To encourage use, users who are inactive for more than a year lose their Raha, with 80% of confiscated Raha distributed directly to members and 20% distributed to the Raha Parliament. Additionally, a referral bonus of 6 weeks of Raha is paid for each referral, up to 150 referrals; that is, someone who refers 150 people will have a seventeen-year head start in the distribution of Raha compared with people who do not make any referrals.

Project UBU (Project UBU, 2018) aims to create a universal currency with demurrage. All network members, called citizens, are entitled to 100 UBU per day. Additionally, there is a 1% daily demurrage fee on all balances. But it departs from universality in two important ways. Like Manna, SwiftDemand, and Raha, Project UBU implements a referral bonus program (here, 1,000 UBUs for the citizen making the referral and 500 UBUs for citizen being referred). Additionally, Project UBU allocates 11 UBU per citizen per day to the UBU Sphere, which is the organization that develops the system.

Frink (Frink, Inc., 2018), short for Friend Link, claims to have equal initial distribution. In practice, the initial distribution is given out in a referral bonus system, with only a portion of new members able to achieve the maximum initial distribution of

1,000 Frinks. Additionally, Frink Incorporated keeps 20% of all Frinks issued as part of its business model.

The BIG Foundation (BIG Foundation, 2017) claims to have an unconditional distribution to network members. Registered members receive 1 token per day. However, rather than being entirely distributed to network members equally, tokens are available for purchase directly from the BIG Foundation at a price of 0.01 ETH per BIG, with bonuses available for large purchases.

GoodDollar (Assia, 2018) claims to continuously mint coins for registered network participants in order to increase the forward distribution of the tokens. Although GoodDollar is still in draft phase and subject to change, it has a feature whereby new GoodDollars can be purchased in the primary market (i.e., directly from a smart contract) for a fixed price.

Enumivo (Enumivo, 2018) is a fork of the EOS blockchain that issues a weekly distribution of tokens. However, the initial distribution does not go equally to all members. Rather, some tokens are awarded for a verification mechanism in which community members vouch for the uniqueness of other community members. New members must receive sponsorship from an existing member and be voted in by the community.

UBIC (UBIC, 2018) is a currency network that aims to reward all members for participation in the network with distribution of currency. To become a member of the network, registrants must show a valid government-issued passport. Although the stated goal is to create a system that does not suffer from “asymmetric value distribution,” the system allocates the distribution of currency according to users’ countries of origin. For instance, the total annual currency distribution to members from the USA is 393,096,240 per year, whereas it is 81,730,800 to members from France.

Chapter III.
Requirements

As described in Chapter I, Pluvo is a cryptocurrency distribution protocol with two goals: decentralized initial distribution and decentralized forward distribution.

The Pluvo protocol describes a currency system with built-in circulation, which is a process for maintaining a decentralized initial and forward distribution. Circulation has two parts: evaporation and rain. In evaporation, a percentage r of every balance of tokens disappears, much like evaporation from a body of water. In rain, the evaporated tokens are distributed equally to all registered addresses. Rain is the source of this protocol's name; *pluvo* is the Esperanto word for *rain*.

Initial Maximum Supply

The initial maximum supply, m, of tokens in the system is established at the network's onset. This value is important only insofar as it creates reasonable whole number units for reckoning prices and balances. As long as the divisibility of the token is sufficient for it to be spread to a large number of currency holders, any number will do initially. As the network evolves, this value can be adjusted to alter the total supply of tokens in the system.

The initial maximum supply is not initially owned by any address. Rather, through the process of evaporation and rain, the initial maximum supply is distributed to registered addresses. It can be thought of as a reserve of tokens from which evaporation occurs but that does not receive rain.

Circulation

Periodically the following two-step process, called *circulation*, occurs:

1. *Evaporation*: All account balances, including the initial reserve, are reduced by the evaporation rate, *r*. For example, if the evaporation rate is 5% per year, then the evaporation would be ~0.43% if assessed monthly, or ~0.1% if assessed weekly.
2. *Rain*: All coins that evaporated in step 1 are distributed equally to all registered addresses.

A few notes about circulation:

- All addresses experience evaporation proportional to the number of tokens held by that address at the time of evaporation.
- Only registered addresses receive rain. All other addresses, which can be created arbitrarily and without registration, do not receive rain.
- Registration should map one-to-one with human participants in the network. That is, registration requires identity verification to ensure uniqueness, and it requires periodic reverification to keep registration lists up to date. This paper does not discuss the details of registration.
- For a given evaporation rate and maximum supply, the number of coins collected through evaporation and distributed through rain will always be the same from one period to the next. This occurs because, regardless of which addresses hold

the tokens, all addresses experience evaporation. Thus, the total amount of periodic rain is a function of the maximum supply *m* and the evaporation rate *r*. In particular: Total Periodic Rain = $m \times r$. This makes calculation of rain straightforward. If there are *n* registered participants, then each registered participant receives $\frac{mr}{n}$ tokens each period. This entails that, for a given maximum supply and evaporation rate, the rain per registered address is inversely proportional to the number of registered addresses. Note that this does not entail that the *value* of rain per registered address falls when the number of registered addresses rises, only that the *number* of tokens falls. Currency networks benefit from network effects (Dowd & Greenaway, 1993), so they become more valuable when they have more participants (all else equal).

- The total supply of currency in use (i.e., held by accounts, rather than in the initial reserve) approaches the maximum supply asymptotically over time. At a 5% annual evaporation rate, it will take about 10 years for the total supply in use to reach 40% of the maximum supply.

Registered Addresses

An identification system should have the following characteristics:

- *Universality.* Every person should have the ability to register an address for storing tokens.
- *Uniqueness.* Registered addresses should match one-to-one with real people. That is, no one person should be able to control multiple registered addresses, and no

single registered address should be controlled by multiple people (unless they have previously consented to sharing control of that address).

- *Security.* Control of registered addresses should be resistant to theft; that is, an identity should not be easily controllable by someone other than its owner.
- *Privacy.* The identification system should not impede users' ability to maintain privacy of their balances or transfer histories.

There are numerous attempts to bring identity to cryptocurrency ecosystems. The user "peacekeeper" maintains a comprehensive list of identity projects on GitHub (Sabadello, 2018). Identity implementations referenced there include Civic, uPort, Blockstack, ShoCard, Authenteq, and brightID.

The uniqueness and security characteristics mean that the identification systems should take care to limit replication attacks and prevent attacks. A replication attack occurs when one person can register more than one address. A prevention attack occurs when a person is blocked from being able to successfully register an address or access a registered address. Identity theft—when an attacker steals another person's public address—is both a replication attack, because the thief can create multiple public addresses, and a prevention attack, because the person whose identity was stolen can no longer receive rain or spend their tokens. Any identity system compatible with Pluvo should prevent both types of attacks.

An identity system with the above characteristics of universality, uniqueness, security, and privacy is ideal for use with the Pluvo protocol. This paper does not specify a required implementation for an identity system.

Use

Registered participants interact with the network in two ways: they collect rain and make transfers. (Unregistered participants who receive tokens can also make transfers, but they cannot collect rain.) To collect rain, registered users initiate a collection using their wallet software; if any rainfall events have occurred since the user's last collection, then the user will collect tokens from those rainfalls. To make transfers (i.e., send tokens), users enter the transfer information (recipient and amount) into their wallet software, which broadcasts it to the network. Note that users do not have to perform any action to receive tokens from a transfer; recipients' balances update automatically.

Parameter Updates

Pluvo has three hard-coded constants: the maximum supply, *m*; evaporation rate, *r*; and the evaporation frequency, *t*. The implementation in Chapter V includes a way to manage the selection of system-wide constants.

Chapter IV.

Tools

The Pluvo protocol is implemented here as a distributed application on the Ethereum network (Ethereum Foundation, 2018). In particular, it is a smart contract (i.e., a program written for the Ethereum execution environment) written in Solidity v0.4.24 (Ethereum, 2018). It is based on OpenZeppelin standards (Zeppelin, 2018) and uses the Truffle framework (Consensys, 2018) for specifying tests and constructing the sample user interface. See Chapter V for details on the implementation.

Chapter V.

Implementation

The technical contribution of this thesis is the creation of a cryptocurrency token that allows for demurrage and the distribution of demurrage proceeds. It is implemented as a smart contract on the Ethereum network (Ethereum Foundation, 2018). Addresses are separated into two classes: registered and unregistered. Registered addresses should be provably unique, with a one-to-one correspondence between registered addresses and people. Users can hold keys for any number of unregistered addresses. All addresses participate in periodic demurrage (evaporation), but only the registered addresses receive regular distributions of the demurrage fees (rain).

Rain

As discussed in Chapter III, the amount of periodic rain per registered address is a function of the evaporation rate, maximum supply, and number of registered addresses. The straightforward way to distribute rain—loop through all registered addresses each period and automatically increase their balances—was avoided for four reasons. First, the number of registered addresses could grow so large that looping through them all would be too resource intensive. In particular, within the context of Ethereum, 5,000 gas is required to update a nonzero 256-bit integer in contract storage (Wood, 2018). (In the Ethereum virtual machine, every operation performed during the execution of a program costs a predefined amount of gas. This gas is payable in units of ETH, the native token on the Ethereum network (Blockgeeks, 2018).) Given current block size limits of around 8,000,000 gas (Etherscan, 2018), this would mean that a maximum of only 1,600 addresses could be updated in a single block (while also hogging the entire block and effectively shutting down the Ethereum network for all other use cases). Second, the data structure typically used to store address balances on Ethereum is a mapping, which does not permit looping (Ethereum, 2018). Third, Ethereum does not have a built-in system for automation. Calculating evaporation automatically each period requires a scheduler, but Ethereum does not have a native way to schedule events. Fourth, because Ethereum requires gas to be paid for every calculation, it is not clear who pays the gas to perform the calculations to distribute rain.

The solution is to use a pull method for rain disbursement. Each period, the amount of rain due to each registered address is stored in a list. When a user wants to collect tokens, the user calls the `collect()` function, which loops through the fixed rainfall totals for all the previous rainfalls since the user last collected rain, adds them up, and transfers the tokens to the user's registered address. (There is also a `collectRainfalls(n)` function, which only collets rain for n previous rainfalls; this function is implemented to allow a user to collect some rain when the total number of previous uncollected rainfalls is too large and expensive to be collected all at once.)

Because Ethereum does not have a native scheduler, the `rain()` function, which stores in the `rainfallPayouts` list the number of tokens due to each registered address in each period, is not run automatically each period. Rather, it is only called when `collect()` or `collectRainfalls()` are called or when the amount of rain per person would change (that is, when the evaporation rate changes, the rainfall period length changes, an address is registered, or an address is unregistered). When `rain()` is called, it appends the rain due to each registered address to the `rainfallPayouts` list once for each rainfall period elapsed since the last time that `rain()` was called.

This method—only calling `rain()` during a collection or when the amount would change—works because the `rainfallPayouts` list only needs to be updated when it is queried or when the rainfall per address amount would change. If the system goes into a period of dormancy, for instance, in which no addresses are registered or unregistered, nor is any rain collected, nor are any of the parameters updated, then there would be no need for the `rain()` function to run.

Note that there is a potential flaw with the implementation of rain as described in the above paragraph. Because the `rain()` function calculates the number of elapsed rainfalls since the last time the `rain()` function was called, and then it appends to the `rainfallPayouts` list once per elapsed rainfall period, there could be a case in which the number of elapsed rainfall periods is so large that the gas required to append to the `rainfallPayouts` list for each elapsed rainfall period would be prohibitively expensive. In that case, the system would effectively break, given that rain could never be registered. To overcome this, the public function `rainOnce()` can be called to append only one rainfall amount to the `rainfallPayouts` list. In the event that calling `rain()` would cost more gas than the available block size, for instance, then `rainOnce()` can be called several times (over several blocks) to bring the rainfall payouts up to date.

Evaporation

Each address's balance is reduced by Evaporation Rate * Address Balance each period. The straightforward way to perform evaporation—loop through all addresses each period and automatically change their balances to pay the demurrage—was avoided for the same reasons that it was avoided to have rain automatically increase the balance of all registered addresses.

The solution: each address's balance is reduced by the evaporation rate only when that address is a sender or recipient of a transfer. That is, if an address is the sender or the recipient of a transfer or a `transferFrom()` function, or is the recipient of a `collect()` or `collectRainfalls()` function, the `evaporate()` function is called on that address prior to the transfer. Importantly, the `evaporate()` function reduces the balance of that address according to the number of elapsed evaporation periods. For example, if periodic evaporation rate is 50% and two evaporation periods have elapsed, then the address's balance is reduced by 75% (which is two reductions of 50%) when that address is member to a transfer.

The `evaporate()` function uses the `calculateEvaporation()` function to calculate the number of tokens by which the address should be reduced. This presents its own difficulties, because Ethereum does not have native support for floating point numbers (Ethereum, 2018). For an address with balance b, evaporation rate r, and elapsed evaporation periods p, the number of tokens to evaporate is $b(1 - (1 - r)^p)$. Because the evaporation rate is between 0 and 1 inclusive, this calculation would ordinarily require floating point numbers to calculate.

Given the lack of floating point number support, the evaporation rate is stored as two integers, `evaporationNumerator` and `evaporationDenominator`, where the denominator must be nonzero and no less than the numerator. Then, to calculate the amount of evaporation by which to reduce an address's balance, the `calculateEvaporation()` function makes a call to the `fractionalExponentiation()` function, which approximates the evaporation amount using only unsigned integers. This function uses the binomial expansion of the term $(1 - r)^p$ to approximate the result. The binomial theorem states:

$$(x + a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k a^{n-k}$$

Into the equation above, substitute x as $-r$ and a as 1. Because all powers of 1 are 1, the a^{n-k} terms disappear. Given that r is between 0 and 1, inclusive, the x^kterms quickly approach 0 as k increases. Thus, only the first few terms of this equation are necessary to closely approximate the true answer. The number of terms of the equation to calculate is a tradeoff between precision and gas cost. The higher the precision, the more expensive it is to calculate the exponentiation. Given that this equation converges quickly, especially for low values for *r*, a value of 8 is sufficiently precise without being too computationally expensive.

Although evaporation is only removed from an address's balance when that address is party to a transfer or rain collection, the value of pending evaporation can be calculated at any time. The `balanceOf()` function makes use of this to accurately display the address's balance, taking into consideration any pending evaporation that will be paid at the next transfer. Thus, although the internal representation of an address's balance only updates for evaporation at the time of a transfer or rain collection, balance inquiries using the `balanceOf()` function will always return the number of spendable tokens.

It is worth noting that, although transfers and rain collection cause evaporation to settle on the blockchain, it is really the passage of time that causes the evaporation to occur. An example will make this clear. Say Alice receives a rainfall distribution of 100 tokens, and the evaporation rate is set to a value of 5% per year. After one year, Alice's number of spendable tokens, as determined by making a `balanceOf()` inquiry, is 95 tokens. However, technically, her address still owns 100 tokens. Under the hood, the balances mapping, which associates Ethereum addresses with token balances, does not change with the passage of time, so Alice's balance will continue to be 100 until she is party to a transfer or rain collection. However, those 100 tokens are not spendable, because the moment she initiates a transfer, the pending evaporation of 5 tokens will be debited from her balance. Therefore, the `evaporate()` function really only causes the evaporation to reflect appropriately in the balances mapping. If, at this point, Alice makes a transfer of 0 tokens to Bob, her balance in the balances mapping will fall from 100 to 95, but her balance as determined by a `balanceOf()` inquiry will still be 95.

In most ERC-20 tokens, the balances mapping is an association between addresses and integers, where the integer represents the number of tokens allocated to that address. In the Pluvo contract, the balances mapping is an association between addresses and `Balance` structs, where a `Balance` struct is a pair of two integers. The first integer represents the number of tokens allocated to that address, and the second integer represents the last time that the address underwent evaporation. By keeping track of the last time that an address underwent evaporation, the `calculateEvaporation()` function can determine the accurate number of evaporation periods elapsed since the last time that the address had some of its tokens evaporate.

Just as evaporation occurs for balances held by addresses, evaporation also occurs for unclaimed rain. Apart from the fact that rain must be claimed before it is spent, it makes no difference, from an evaporation standpoint, whether rain is claimed or not. As an example, say that the current rainfall is 100 tokens per person. A rainfall just occurred, and Alice is able to make a claim of 100 tokens, but she chooses not to make the claim yet. After another circulation period passes, during which the evaporation rate was 5% per period, she then makes her claim. The number of tokens collected by Alice will be 195, which is composed of the 100 tokens from the most recent rainfall, plus the 95 tokens from the first rainfall, which evaporated by 5% since it rained.

Evaporation also occurs for undistributed supply. This is how the money supply is initially distributed to network participants. One way to think about the initial maximum supply is to pretend that the entire initial supply is held in reserve by an Ethereum address from which tokens cannot be spent. Over time, this address experiences evaporation (but not rain) so that its tokens are distributed to all registered participants over time.[6]

ERC-20 Conformance

The Pluvo contract conforms to the ERC-20 interface (The Ethereum Wiki, 2018), meaning that it implements the functions and events required by the interface. By

[6] Although the Pluvo contract does not implement undistributed supply as described in this paragraph, an alpha version of the Pluvo contract did. The initial supply was held in an address called the Ocean. The Ocean's sole purpose was to be the site of the initial supply; its tokens could not be spent, and its address could not be registered to receive rain. However, its existence is was superfluous. Given that evaporation occurs equally for all tokens everywhere—whether held by user addresses, sitting in unclaimed rain, or held by the Ocean—then the total amount of rain is merely a function of the evaporation rate and the max supply, where the max supply is the sum of tokens held by user addresses, sitting in unclaimed rain, or held by the Ocean. Because the max supply is static (or, more precisely, alterable only by the parameter setter, as described in the section below), there was no need to have a separate address represent the Ocean.

conforming to the ERC-20 interface, tokens governed by the Pluvo contract will be able to interact with tools, wallet software, and exchanges that are built to handle ERC-20 tokens.

The six ERC-20 functions implemented are:

- `totalSupply()`: returns the total number of tokens that have been collected from rain but not yet evaporated. Note that, in general, the value returned by this function will be greater than the spendable supply, because it does not take into consideration any pending evaporation. For the same reason, this function may in fact return a value that is greater than the maximum supply.
- `balanceOf(address)`: returns the spendable balance of an address. This function takes into consideration pending evaporation, so it will decrease over time if there are no transfers and no rain collections.
- `allowance(tokenOwner, spender)`: returns the number of tokens that the spender is allowed to spend of from the `tokenOwner`'s address. This function does not make any adjustments for evaporation.
- `transfer(to, amount)`: transfers a given number of tokens to the specified address from the message sender's account. This function emits the `Transfer` event and returns `true`/`false` for successful/unsuccessful transfers. By initiating a transfer, the sender also causes evaporation to occur in the sender's account and the recipient's account. The transfer will fail if the amount specified exceeds the sender's balance after evaporation.

- `approve(spender, amount)`: authorizes the spender to spend the given number of tokens from the message sender's account. This value does not change due to evaporation. The approve function emits the `Approve` event.
- `transferFrom(from, to, tokens)`: transfers a given number of tokens from the `from` address to the `to` address. The message sender must be authorized (via a previous `approve(spender, amount)` function call) to spend the given number of tokens from the `from` address. Just like the `transfer(to, amount)` function, this function emits the `Transfer` event, returns `true/false` for successful/unsuccessful transfers, causes evaporation to occur in the from account and to account, and will fail if the amount specified exceeds the sender's balance after evaporation.

Registration

There are two special addresses that are established at the construction of the contract. One of those addresses is the `registrar`. The `registrar` is responsible for registering and unregistering addresses. The `registrar` registers an address by calling the `registerAddress()` function and passing in the desired address as a parameter. When an address is registered, its value in the `rainees` mapping is updated to the current rainfall index, where the current rainfall index is the number of elapsed rainfall periods since the construction of the Pluvo contract. A value in the `rainees` mapping of greater than 0 means that the address is registered and allowed to collect all rainfalls that occur after the current rainfall index. The registrar can also remove a registered address by calling the `unregisterAddress()` function, which sets that address's value in the `rainees` mapping back to 0.

Parameter Updates

The other special address is the `parameterSetter`. The `parameterSetter` can update the evaporation rate, the rainfall period (which is the number of seconds between rainfalls), the maximum supply, and the precision used in the `fractionalExponentiation()` function, as described above.

Use

Pluvo can be used like any other ERC-20 token: any ERC-20-compatible Ethereum wallet can send Pluvo tokens or check balances. Additionally, users can register one address with the registrar. Doing so entitles that address to collect tokens from periodic rainfalls. The user can initiate a collection of rain by calling the `collect()` or `collectOnce()` functions.

Chapter VI.

Discussion

The protocol introduced here serves as a template distribution mechanism for any monetary authority considering introducing a digital currency. It also serves a template for cryptocurrency developers who want to ensure their tokens continue to circulate.

Pluvo as a Template for Central Bank Currency

A monetary authority that implements a system like Pluvo would have two important new tools in its toolkit. One is the evaporation rate, which can be used to directly accelerate or decelerate the velocity of money. Another is the ability to directly change the money supply by adjusting the maximum supply parameter. Changing the maximum supply changes the size of rain payouts without affecting the evaporation rate.

From a policymaker's perspective, evaporation and rain create an effective interest rate as a function of token balances. The effective interest rate is given by the function $f_{m,n,r}(b) = r\left(\frac{m}{nb} - 1\right)$, where the function *f* defines the effective interest rate from evaporation and rain for a specific user of the network with token balance *b*. The function is parametrized by the maximum supply, *m*; the evaporation rate, *r*; and the number of registered users, *n*, on the network.

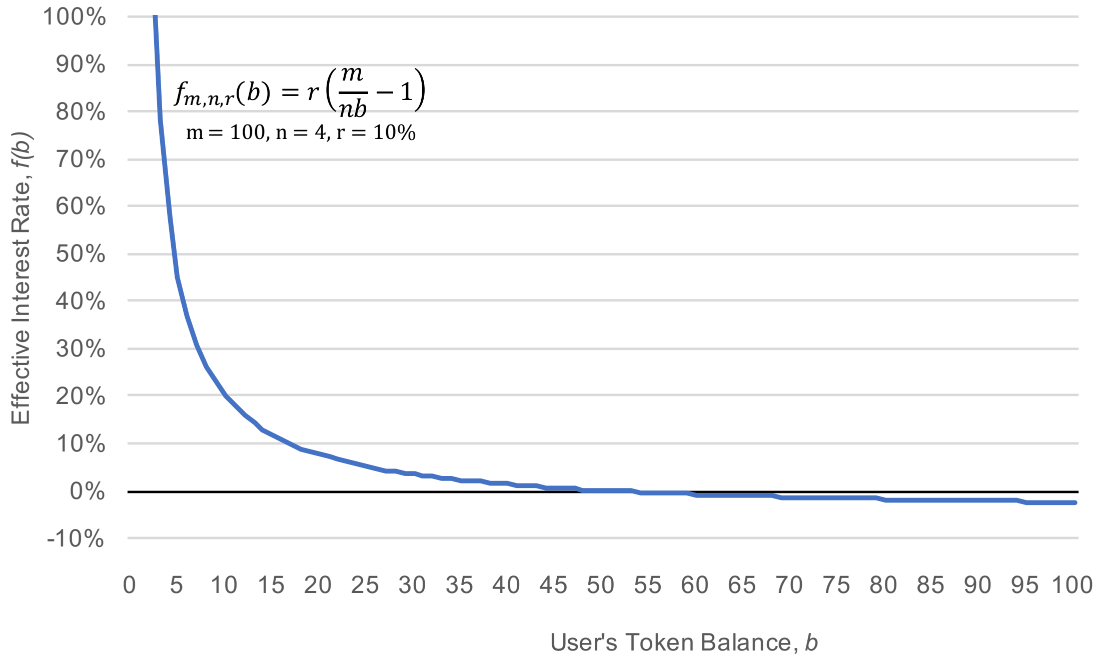

Figure 1. Effective Interest Rate by User's Token Balance.

The effective interest rate for a user is a function of that user's token balance, parametrized by the maximum supply, number of users, and evaporation rate, given by the equation $f_{m,n,r}(b) = r\left(\frac{m}{nb} - 1\right)$*. The effective interest rate is 0% at the average balance (here, 25). Parameters: max supply m = 100, number of users n = 4, evaporation rate r = 10%.*

For registered users with token balances less than the average balance, *f(b)* will be positive, meaning that proceeds from rain will exceed evaporation. For registered users with an average token balance such that $nb = m$, rain and evaporation are equal, and the effective interest rate is 0. Registered users with above-average token balances will see a negative effective interest rate.

The equation $\mathrm{f}_{\mathrm{m,n,r}}(\mathrm{b}) = \mathrm{r}\left(\frac{\mathrm{m}}{\mathrm{nb}} - 1\right)$ also shows how policy makers can alter users' effective interest rates by controlling the maximum supply *m* and evaporation rate *r*. An increase in *m* serves to increase the effective interest rate for all registered users; a larger *m* entails larger rain payouts without increased evaporation. By increasing the maximum supply, policymakers would be able to increase the money supply and have it be automatically distributed, via rain, to all participants in the currency network.

An increase in the evaporation rate, *r*, increases evaporation and rain, but how it changes a user's effective interest rate depends on the value of $\left(\frac{\mathrm{m}}{\mathrm{nb}} - 1\right)$. For users with below-average balances (i.e., those for whom $\left(\frac{\mathrm{m}}{\mathrm{nb}} - 1\right)$ is positive), an increase in *r* results in a higher effective interest rate. For users with above-average balances (i.e., those for whom $\left(\frac{\mathrm{m}}{\mathrm{nb}} - 1\right)$ is negative), an increase in *r* results in a more-negative effective interest rate. Changing *r* does not change the average balance, thus it does not change which users see positive or negative interest rates. An increase in *r* does, however, increase the absolute value of *f(b)*, meaning it increases the rate at which token balances approach the average.

Because users' rain proceeds do not depend on their token balances, users can always increase their effective interest rates by decreasing their token balances; a smaller token balance entails less evaporation without a change in the amount of rain receivable. Thus, the incentive to spend tokens (i.e., the disincentive to hoard) can be thought of either as coming from users' desire to decrease the cost of evaporation or to increase the effective interest rate.

Pluvo as Money

This paper implemented the Pluvo protocol as a cryptocurrency. However, the Pluvo protocol applies broadly to cases where the token ought to have no value in itself—that is, where it should only be valuable as a means of access to or control of something else—and where whatever the token accesses should be controlled by a large number of people.

As Gesell (1916) writes in *The Natural Economic Order*, the economy is exactly such a case. Money is the token that provides access to and control of the economy, and money is only useful insofar as it serves as a medium of exchange. “Money is an instrument of exchange and nothing else,” he writes. “The criterion of good money, of an efficient instrument of exchange, is that it shall secure the exchange of goods…, that it shall accelerate exchange…[and] that it shall cheapen exchange” (p. 266). Money has value because it controls the resources of the economy, and the economy is, by its very nature, is accessed by a large number of people. Money is thus a natural use case for the Pluvo distribution mechanism. Tokens subject to evaporation are perishable like the goods and services they buy. Just as sellers are induced to quickly sell their wares—which may decay, rot, perish, or require maintenance and storage costs—so too are buyers induced to quickly spend their decaying currency (Gesell, 1916). And because of the rain mechanism, a large number of people will always have access to some portion of the currency, and they will therefore always have access to some portion of the economy.

No implementation of this protocol will obtain positive value unless it is adopted by people for use as a medium of exchange. Once it is thus adopted, its value will depend on the goods and services it can buy. For an already-existing currency, the value of a currency appreciates when prices denominated in that currency decrease. For a new currency, the same logic applies. At first, prices are effectively infinite, given that when sellers will not accept a currency, no amount of that currency can buy goods or services.

Any finite price, then, represents deflation (from infinity) and a resulting increase in the value of the currency.

Prolonged deflation, though, does not encourage use of a currency. As prices fall, users benefit more by holding the currency—waiting for goods and services to get even cheaper—than by spending it. (This assumes that purchases can be postponed.) With Pluvo, however, there is a countervailing effect: the evaporation rate. If prices denominated in Pluvo are indeed falling, then holders of the currency must decide whether the predicted deflation rate will exceed the known evaporation rate. If the evaporation rate exceeds the predicted deflation rate, then the incentive to spend will outweigh the incentive to hoard. As users spend their tokens, the resulting increase in demand of goods and services will reduce or eliminate the deflation.

Pluvo as a Template for Cryptocurrency Distribution

For cryptocurrency developers who want to avoid speculation, the evaporation rate is an important tool. Other cryptocurrencies that lack built-in circulation (or demurrage) encourage a "land grab" mentality: early adopters can purchase coins when they are cheap and squat on them indefinitely, hoping to sell the coins at a later date when the network is more mature. The irony, though, is that by hoarding coins and not using them for their intended purpose, these early adopters actually inhibit the utilization of the network they hope to see succeed. With Pluvo, the evaporation rate impels users to spend their tokens, thus contributing to the utility of the network. But rather than speculators profiting from the network growth, all registered users will benefit over time, because all registered users share rain equally.

Because it has a built-in decay, Pluvo is far from an ideal speculative investment. That does not mean that it will be immune to speculation. Speculators can speculate on any asset, even perishables like tulip bulbs. If the evaporation rate is too low and the potential future value of the currency is high enough, then speculators will inevitably arrive. However, a bit of early speculation is not unwelcome. A new currency necessarily starts with no value, so there has to be *someone* willing to be the first person to exchange something of value for Pluvo. This person is speculating that Pluvo will have value in the future.

Over time, evaporation renders speculation increasingly less valuable, making Pluvo an unwise long-term buy-and-hold investment. At a 5% annual evaporation rate, nearly 80% of an unregistered speculator's holdings will have evaporated in 30 years. Higher evaporation rates make speculation even less attractive.

In addition to reducing speculation, the evaporation rate may also help reduce exchange-rate volatility. Many cryptocurrencies, most notably Bitcoin, have seen significant volatility in exchange rates against the US Dollar. For example, in 2017, the Bitcoin-USD daily exchange-rate volatility was 4.87% (Blockchain Luxembourg S.A., 2017). By comparison, the Euro-USD daily exchange-rate volatility in 2017 was 0.46% (Board of Governors of the Federal Reserve System (US), 2017), less than one-tenth that of Bitcoin.

The volatility in Bitcoin's exchange rate with the US Dollar—especially in 2017—stems in large part from Bitcoin's introduction to new communities. After people first hear about Bitcoin, the only way for them to participate in the Bitcoin network is for them to buy bitcoins from people who already own them, or to mine them. Mining requires technical expertise and a large up-front capital expense; the far simpler route is to purchase bitcoins on an exchange. Large fluctuations in demand, without any changes in the supply schedule, lead to highly volatile exchange rates. In contrast, Pluvo allows new members to directly own tokens shortly after registering to be recipients of rain. Thus, new members need not purchase Pluvo tokens from existing members simply to participate.

Chapter VII.
Conclusion

For cryptocurrency developers who want to foster a large community of token holders, who want a token that is resistant to hoarding and speculation, and who want a token with lower exchange-rate volatility, the Pluvo protocol serves as a template. For monetary authorities who want direct control over the money supply and the velocity of money, who want the capacity to set negative interest rates, and who want to be able to distribute new money quickly and easily, the Pluvo protocol serves as a template.

Because the Pluvo mechanism distributes all new coins evenly to all registered addresses, it represents a fundamentally different distribution mechanism from that of the proof-of-work consensus mechanism discussed in the introduction. However, Pluvo is not incompatible with mechanisms that distribute new coins as a reward for certain network actions. The Pluvo mechanism can be modified to introduce evaporation and rain after tokens are initially distributed by some other means.

Pluvo deserves a robust governance system to ensure that the parameters represent the collective interest of its current and future users. Because the evaporation rate factors into the calculations for both evaporation and rain, it has two purposes: it controls both the disincentive to hoard and the level of decentralization of ownership distribution. When the evaporation rate is decreased, current owners benefit at the expense of future owners, holders of large token balances benefit at the expense of holders of small token balances, the velocity of money slows, and decentralization of ownership distribution decreases. The reverse holds for increases in the evaporation rate. Because money is a tool to serve the people in an economy over time, any governance system that controls the parameters of Pluvo should ensure that the parameters best meet the needs of those people over time.

Appendix A.

Code Walkthrough

Below is a walkthrough of the Solidity file that defines the Pluvo smart contract. The code is available at https://github.com/theshepster/pluvo.

ERC-20 Public Functions

The signatures for the six ERC-20 public functions are defined in the ERC-20 standard (The Ethereum Wiki, 2018).

```
function totalSupply() public view returns (uint256)
```

Returns the total supply in circulation. That is, `totalSupply()` will return the total number of tokens that have entered circulation via collection of rain, less all tokens that have evaporated. It will always be equal to the sum of all tokens held in the balances mapping. However, because pending evaporation only evaporates from an account when it receives or transfers tokens, `totalSupply()` will typically return a value that is greater than the sum of `balanceOf()` for each account with Pluvo. See `balanceOf()` below for more details.

```
function balanceOf(address _owner) public view returns (uint256
balance)
```

Returns the balance of tokens owned by `_owner`, less any pending evaporation. Evaporation only occurs from an Ethereum account when the account is the sender or recipient in a transfer; until then, the evaporation is pending. The `balanceOf()` function accounts for pending evaporation, which is unspendable, when showing the balance of the given account.

```
function transfer(address _to, uint256 _value) public returns (bool
success)
```

Transfers `_value` tokens from the account of the message sender to the `_to` account. This function causes any pending evaporation to evaporate from the sender's and recipient's accounts.

```
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success)
```

Transfers `_value` tokens from the `_from` account to the `_to` account. Checks to ensure that the message sender is allowed to send `_value` tokens from the `_from` account (see `approve()` and `allowance()` below). This function causes any pending evaporation to evaporate from the `_from` and `_to` accounts.

```
function approve(address _spender, uint256 _value) public returns (bool
success)
```

The message sender approves `_spender` to transfer up to `_value` tokens from the message sender's account.

```
function allowance(address _owner, address _spender) public view
returns (uint256 remaining)
```

Returns how many tokens `_spender` is allowed to spend from `_owner`'s account.

ERC-20 Public Variables with Getters

The following two arrays are public and therefore have associated getter functions:

```
mapping (address => Balance) balances
mapping (address => mapping (address => uint256)) allowed
```

For reference, the `Balance` struct is defined as:

```
struct Balance {
    uint256 amount;
    uint256 lastEvaporationTime;
}
```

The optional ERC-20 functions `name()`, `symbol()`, and `decimals()` are also implemented, returning "Pluvo", "PLV", and 18 respectively.

Pluvo Public Functions

The following functions are not ERC-20 functions. Rather, they are implemented specifically for Pluvo.

```
function currentRainfallIndex() public view returns (uint256)
```

Counts one more than the number of past rainfalls. This is because the `rainfallPayouts` array is seeded with a `(0, block.timestamp)` value in the constructor. (The `rainfallPayouts` array stores the amount that each registered address is allowed to collect for each periodic rainfall.) That is, this returns the index of the rainfall that is about to occur next. For example, a return value of 2 indicates that the next rainfall will be the second rainfall ever.

```
function lastRainTime() public view returns (uint256)
```

Returns the last time when rain happened. That is, it returns the timestamp of the last block when a `Rain` struct was added to the `rainfallPayouts` array.

```
function rainPerRainfallPerPerson() public view returns (uint256)
```

Returns the current amount of rain eligible for each registered address to collect in the next rainfall, if the next rainfall were to occur in the current block. Calculated by multiplying the maximum supply by the evaporation rate and dividing by the number of registered addresses.

```
function setEvaporationRate(uint256 _evaporationNumerator, uint256
_evaporationDenominator) public onlyBy(parameterSetter)
```

Sets the new evaporation rate to be `_evaporationNumerator / _evaporationDenominator`. Solidity does not support floating point math, so the numerator and denominator are both integers. Only the address designated as the `parameterSetter` is allowed to invoke this function.

```
function setRainfallPeriod(uint256 _secondsBetweenRainfalls) public
onlyBy(parameterSetter)
```

Sets the new number of seconds between rainfalls. Any pending rainfalls occur before the new period is established. Only the address designated as the `parameterSetter` is allowed to invoke this function.

```
function registerAddress(address _rainee) public onlyBy(registrar)
returns (bool success)
```

Registers a new address to receive rainfall payouts. Only the address designated as the registrar is allowed to invoke this function. The registrar, as designated by the `registrar` address, should implement separate functionality to allow users to request registration and to prevent the same person from registering multiple addresses. The Pluvo contract does not handle the implementation of the registrar's registration functionality.

```
function unregisterAddress(address _rainee) public onlyBy(registrar)
returns (bool success)
```

Removers a registered address from the list of rainfall payout recipients. Only the address designated as the registrar is allowed to invoke this function. The registrar should implement functionality to allow the person who registered the address (and only that person) to unregister.

```
function changeParameterSetter(address _parameterSetter) public
onlyBy(parameterSetter)
```

The `parameterSetter` can invoke this function to select a new `parameterSetter`.

```
function changeRegistrar(address _registrar) public onlyBy(registrar)
```

The `registrar` can invoke this function to select a new `registrar`.

```
function collectRainfalls(uint256 maxCollections) public returns
(uint256 fundsCollected)
```

Registered addresses can invoke this function to collect up to `maxCollections` of rain, where `maxCollections` represents the number of rainfalls, not the amount of rain, to collect. Note that, when rainfalls occur, the registered addresses only get the rain when they collect it by calling `collectRainfalls(maxCollections)` or `collect()`.

```
function collect() public returns (uint256 fundsCollected)
```

Registered addresses can invoke this function to collect all available rain.

```
function rain() public returns (uint256 rainfallsDue)
```

Causes one or more rainfalls to occur. The `rain()` function calculates the amount of time that has elapsed since the last rainfall, integer-divides that by the number of seconds between rainfalls, and pushes that many rainfall payouts (each one equal to the `rainPerRainfallPerPerson()`) to the `rainfallPayouts` array.

```
function calculateEvaporation(address _addr) public view returns
(uint256)
```

Returns the amount of pending evaporation from address `_addr`. Note that `_addr` does not have to be a registered address, because all addresses experience evaporation.

Pluvo Public Variables with Getters

The following variables are public and therefore have getter functions:

```
address parameterSetter
address registrar
mapping (address => uint256) rainees
Rain[] rainfallPayouts
uint256 evaporationRate
uint256 evaporationDenominator
uint256 secondsBetweenRainfalls
uint256 maxSupply
uint256 numberOfRainees
```

For reference, the `Rain` struct is defined as:

```
struct Rain {
    uint256 amount;
    uint256 rainTime;
}
```

Pluvo Events

Events are Solidity constructs that allow information about a contract to be viewable outside of the blockchain environment, for example, by JavaScript code (Ethereum, 2018). In particular, the return value from Solidity functions that modify the blockchain cannot be seen from JavaScript, so events are used to pass important information outside of the blockchain for use in the application.

```
event Collection(address indexed recipient, uint256 amount)
```

Event emitted by the `collect()` function. Exposes the collector (`recipient`) and amount collected (`amount`).

Gas Usage

Ethereum requires that all functions that modify the blockchain pay gas (Ethereum, 2017). This means that inquiry functions, such as `balanceOf()` and `totalSupply()` can be called without the user having to pay gas. On the other hand, functions that modify storage on the blockchain, such as `transfer()` and `collect()`, require the user to pay for the computations. Unless Ethereum implements economic abstraction (Buterin, On Abstraction, 2015), which would allow gas to be paid in something other than Ether, the gas owed for each of these functions is payable only in Ether. Thus, should Pluvo launch on the Ethereum mainnet, users will be required to pay a small amount of Ether in order to collect rain or transfer tokens.

Appendix B.

User Interface Guides

The below guide covers the interface available at https://github.com/theshepster/pluvo.

Setting Up a Test Environment

First, clone the git repository and install the dependencies by executing the following code:

```
git clone https://github.com/theshepster/pluvo.git
cd pluvo
npm install
```

Additionally, install Truffle and Ganache-CLI globally by executing:

```
npm i -g truffle ganache-cli
```

The `truffle` command will be what you use for compiling the Solidity smart contracts, migrating them to the blockchain, and running the tests. The ganache-cli tool is a custom development Ethereum blockchain and will be used as the private test network.

The last prerequisite is to have MetaMask installed. Get it by either downloading the Chrome extension or downloading the Brave browser.

Once you have finished the above npm commands and installed MetaMask, get the development blockchain ready. Open a new terminal window and type:

```
ganache-cli -d
```

This command initializes the Ganache Ethereum test blockchain, which runs in the background and connects to port 8545. The `-d` flag ensures that the 12-word recovery mnemonic will be the same one each time you run the `ganache-cli -d` command, so that when you reset the network, you will get the same private keys each time.

Open up MetaMask in your browser. Click in the upper-right corner and set your network to Localhost 8545. (If that is not an option, click Custom RPC and use 8545 as the port number.)

To connect MetaMask to your Ganache private keys, from the MetaMask login screen click "Import using account seed phrase." Copy the 12-word mnemonic from the command line (which appeared immediately after you entered the `ganache-cli -d` command) into the box in MetaMask, and choose a secure password.

Now that you are logged in, you should see your first account, called Account 1, have a balance of 100 Ether. If you do not, click in the upper-right-hand corner of MetaMask and double check that your network is pointing to Localhost 8545. **Never use the Ganache-CLI accounts with any other network in MetaMask. Localhost 8545 only!**

You are ready to deploy and test Pluvo. In a terminal window within the pluvo/ directory, enter the following commands in order:

```
truffle compile
truffle migrate
npm run start
```

The first command, `truffle compile`, compiles the Solidity contracts into JSON files that can be deployed to the blockchain. The compiled contracts live in the build/contracts directory. If you make any changes to the Pluvo.sol file, `run truffle compile` to prepare the updated contract for deployment.

The second command, `truffle migrate`, deploys the compiled contracts to the custom ganache-cli blockchain that is running in the other terminal window. Truffle knows to deploy to the ganache-cli blockchain because in the truffle.js file, the development network is set to host 127.0.0.1 port 8545, which is where the ganache-cli blockchain is connected.

Before deploying with `truffle migrate`, feel free to edit the Pluvo parameters in migrations/2_deploy_contracts.js. The four parameters to edit are:

```
const maxSupply;
const evaporationNumerator;
const evaporationDenominator;
const secondsBetweenRainfalls;
```

The last command, `npm run start`, runs the scripts/start.js script. It compiles the user interface and serves it up to port 3000. It uses hot-reloading, so any changes you make to the user interface will load automatically without requiring recompilation.

After several seconds, a browser window should open pointing to localhost:3000.

Understanding Accounts

The Ganache-CLI blockchain gives you 10 accounts. The first account will be the account that deploys the contracts to the blockchain, so you should see that, after deployment, the ETH balance of Account 1 is less than 100. This is because some ETH was spent to deploy the contracts. Additionally, in Pluvo, the parameterSetter and registrar addresses will be set to Account 1.

In order to play around with additional accounts, within MetaMask click on Add Accounts. Because the ganache-cli tool uses an HD wallet, the second account that MetaMask generates for you will automatically be the second account that ganache-cli created for you, meaning it will be pre-funded with 100 ETH.

Using Pluvo

Upon loading the application, the initial screen displays a few instructions on how to get started. In order for currency to first be distributed by the system, you must register an address to be a recipient of rain. Click the "Registration" link in the upper-right corner of the page. The number of accounts registered should display as 0, and the registrar's address should be the first address in your Ganache-CLI wallet. Copy the registrar's address (which should be the current address activated in MetaMask), paste it into the "Address to Register" field, and click Submit. This activates MetaMask, so you should see a pop-up from MetaMask that asks you to review and confirm the action. Although MetaMask may display a gas fee in USD, because this is a test environment there is no USD cost to gas. Click Confirm to submit the action to your test blockchain. The number of accounts registered should update to 1. (If it does not update automatically, it should update after refreshing the page.)

Now that Account 1 is registered to receive rain, click on Pluvo in the upper-left corner of the screen to return to the homepage. Then, click the Submit button underneath "Collect All Available Rain." After confirming the MetaMask pop-up and waiting a few seconds (and perhaps reloading the page), the value after "My Balance" should be positive.

To send tokens, fill out the two-field form in the Send Tokens section. To get an address to which to send coins, consult your ganache-cli blockchain accounts and choose an account from Accounts 2–10. Paste this address in the “To Address” field. Choose an amount equal to or less than the number of token you have, then click Submit. If you chose an amount greater than the number of tokens available for spending, MetaMask will warn you that there is an error, and you will not be able to confirm the transfer. If there are no errors, the transfer will go through after you click Confirm in the MetaMask pop-up.

If you would like to change any of the parameters, click on Parameters in the upper-right corner. If your active MetaMask account is Account 1 (which was automatically established as the parameter setter), you can update any of the available parameters by typing the parameter into its respective field and clicking Submit.

If you would like to change accounts, you may do so within MetaMask. Reload the page after switching accounts so that the application recognizes that the new account is the active account.

Troubleshooting

If you are not logged in to MetaMask when the application loads, you will see a screen that says, “We can't find any Ethereum accounts! Please check to make sure MetaMask in your browser is pointed at the correct network and your account is unlocked.” To fix this, log in to MetaMask and ensure that the network is pointing to the same port as the one where your local blockchain is running (port 8545 according to the directions above).

If the application gets stuck on a screen that says “Loading dapp…”, it means that one of the JavaScript files has an error in it. In Chrome, open the Developer Console (Alt-Cmd-J on a Mac) and review the console output. If there were no errors, the “Loading dapp…” screen would not appear for more than a second or two, and the Chrome Developer Console would say “Injected web3 detected.” If there is an error, examine the error the Developer Console to see if you can glean any indication as to the root of the error.

Testing

To run the test suite, run one of the following two commands. If the ganache-cli blockchain is already running in the background, then at the terminal, from the pluvo/ directory, enter:

```
truffle test
```

This will compile the contracts, deploy them to the running ganache-cli blockchain, and then run all the tests in the test/ directory. If you only want to run a single test file, for example, TestPluvo.js, then type:

```
truffle test ./test/TestPluvo.js
```

Alternatively, if there is no ganache-cli blockchain already running, enter:

```
npm run test
```

This will launch a ganache-cli instance in the background, run truffle test, and then kill the ganache-cli instance.

References

AirdropAlert B.V. (2018). Retrieved from Airdrop Alert: https://airdropalert.com/

Assia, Y. (2018, September 17). *Whitepaper Draft.* Retrieved October 5, 2018, from GoodDollar: https://www.gooddollar.org/whitepaper-draft

Baird, L. (2016). *The Swirlds Hashgraph Consensus Algorithm: Fast, Fair, Byzantine Fault Tolerance.* Retrieved from http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

Baur, D. G., Hong, K., & Lee, A. D. (2016). *Virtual Currencies: Media of Exchange or Speculative Asset.* Retrieved from https://bravenewcoin.com/assets/Industry-Reports-2016/Bitcoin-Baur-et-al-2016-SWIFT-FINAL.pdf

BIG Foundation. (2017, October 12). *BIG Foundation Documentation.* Retrieved October 5, 2018, from Github: https://github.com/bigfoundation/Documentation/blob/master/BIGwhitepaperEN.md

bitinfocharts.com. (2018). *Bitcoin Rich List.* Retrieved September 25, 2018, from https://bitinfocharts.com/top-100-richest-bitcoin-addresses.html

Blockchain Luxembourg S.A. (2017, December 31). *Market Price (USD).* Retrieved October 9, 2018, from Blockchain: https://www.blockchain.com/charts/market-price?timespan=2years

blockchain.info. (2018). *Blockchain Charts*. Retrieved from https://blockchain.info/charts

Blockgeeks. (2018). *What is Ethereum Gas: Step-By-Step Guide*. Retrieved from Blockgeeks: https://blockgeeks.com/guides/ethereum-gas-step-by-step-guide/

Board of Governors of the Federal Reserve System (US). (2017, December 31). *U.S. / Euro Foreign Exchange Rate [DEXUSEU].* Retrieved October 9, 2018, from FRED, Federal Reserve Bank of St. Louis: https://fred.stlouisfed.org/series/DEXUSEU

Boyle, D. (2002). *The Money Changers: Currency Reform from Aristotle to E-cash* (1st Edition ed.). Routledge.

Buterin, V. (2015, July 5). *On Abstraction*. Retrieved from Ethereum Blog: https://blog.ethereum.org/2015/07/05/on-abstraction/

Buterin, V., & Griffith, V. (2017). *Casper the Friendly Finality Gadget.* Retrieved from https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf

Circles. (2017, December 11). *CirclesUBI/docs: Circles Money System Overview.* Retrieved April 5, 2018, from Github: https://github.com/CirclesUBI/docs/blob/master/Circles.md

Consensys. (2018). Retrieved from Truffle: https://truffleframework.com/

Democracy Earth. (2018, April 2). *DemocracyEarth/paper: The Social Smart Contract.* Retrieved April 7, 2018, from Github: https://github.com/DemocracyEarth/paper

Dowd, K., & Greenaway, D. (1993). Currency Competition, Network Externalities and Switching Costs: Towards an Alternative View of Optimum Currency Areas. *The Economic Journal, 103*(420), 1180-1189.

Enumivo. (2018). *Technical Whitepaper.* Retrieved October 5, 2018, from Enumivo: https://static1.squarespace.com/static/5a8049ac8dd041e820ad35fe/t/5b4964891ae6cf6f42e10247/1531536546110/whitepaper.pdf

Ethereum. (2017, December 20). *Contract Tutorial*. Retrieved from Github.com: https://github.com/ethereum/go-ethereum/wiki/Contract-Tutorial

Ethereum. (2018). Retrieved from Solidity: https://solidity.readthedocs.io/en/v0.4.24/

Ethereum Foundation. (2018). Retrieved from Ethereum: https://ethereum.org/

Etherscan. (2018). *Blocks*. Retrieved from Etherscan: https://etherscan.io/blocks

Fisher, I. (1933). *Stamp Scrip.* New York: Adelphi Company.

Friedenbach, M., & Timón, J. (2018). Retrieved April 5, 2018, from Freicoin: a peer-to-peer digital currency delivering freedom from usury: http://freico.in/

Frink, Inc. (2018). *Whitepaper.* Retrieved October 4, 2018, from Frink: https://frink.global/whitepaper

Gatch, L. (2008). Local Money in the United States During the Great Depression. *Essays in Economic & Business History, XXVI*, 47-61.

Gelleri, C. (2009). Chiemgauer Regiomoney: Theory and Practice of a Local Currency. *International Journal of Community Currency Research, 13*, 61-75.

Gesell, S. (1916). *The Natural Economic Order.* (P. P. M.A., Trans.) Bern.

Kondor, D., Pósfai, M., Csabai, I., & Vattay, G. (2014). Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network. *PLOS ONE, 9*, 1-10.

Laborde, S. (2017). *Relative Theory of Money v2.718.* Retrieved from http://en.trm.creationmonetaire.info/TheorieRelativedelaMonnaie.pdf

Lingham, V., & Smith, J. (2017). *Civic Token Sale*. Retrieved from Civic: https://tokensale.civic.com/

Literally Media Ltd. (2018). *HODL*. Retrieved from Know Your Meme: https://knowyourmeme.com/memes/hodl

Mannabase, Inc. (2018). Retrieved October 3, 2018, from Mannabase: https://mannabase.com/

Mazières, D. (2015). *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus.* Retrieved from https://www.stellar.org/papers/stellar-consensus-protocol.pdf

Merriam-Webster, Inc. (2018). *demurrage.* Retrieved October 28, 2018, from Merriam-Webster.com: https://www.merriam-webster.com/dictionary/demurrage

Micali, S. (2016). ALGORAND: The Efficient and Democratic Ledger. *CoRR, abs/1607.01341.*

Moreau, C. e. (2018). *Theoretical*. Retrieved April 5, 2018, from Duniter: https://duniter.org/

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved from http://bitcoin.org/bitcoin.pdf

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies* (Draft ed.). Princeton: Princeton University Press.

Poon, J. (2017). *OmiseGO Decentralized Exchange and Payments Platform.* Retrieved from https://cdn.omise.co/omg/whitepaper.pdf

Project UBU. (2018). *Project UBU Business Model and Economics Paper.* Retrieved October 4, 2018, from Project UBU: https://www.projectubu.com/pdf/ProjectUBU_EconomicsPaper.pdf

Raha. (2018). *What is Raha?* Retrieved October 4, 2018, from Raha: https://raha.app/what-is-raha/

Sabadello, M. (2018). *Blockchain and Identity*. Retrieved from Github: https://github.com/peacekeeper/blockchain-identity

Srinivasan, S. B. (2017). *Quantifying Decentralization*. Retrieved from Medium: https://news.earn.com/quantifying-decentralization-e39db233c28e

Stellar. (2018). *Stellar Invites*. Retrieved from (https://invite.stellar.org/).

Stone, C., Trisi, D., Sherman, A., & Horton, E. (2017). *A Guide to Statistics on Historical Trends in Income Inequality*. Retrieved from Center on Budget and Policy Priorities: https://www.cbpp.org/research/poverty-and-inequality/a-guide-to-statistics-on-historical-trends-in-income-inequality

SwiftDemand. (2018). *FAQ*. Retrieved October 3, 2018, from SwiftDemand: https://www.swiftdemand.com/faq

The Economist. (2015). *The Economist Explains: What is Quantitative Easing*. Retrieved from https://www.economist.com/blogs/economist-explains/2015/03/economist-explains-5

The Ethereum Wiki. (2018, June 16). *ERC20 Token Standard.* Retrieved from The Ethereum Wiki: https://theethereum.wiki/w/index.php/ERC20_Token_Standard

U.S. Geological Survey. (2017, November 15). *The Water Cycle*. Retrieved from USGS: https://water.usgs.gov/edu/watercycle.html

UBIC. (2018, August 17). *UBIC Whitepaper.* Retrieved October 5, 2018, from Github: https://github.com/UBIC-repo/Whitepaper/blob/master/README.md

Unchained. (2018, September 9). *Bitcoin UTXO Age Distribution*. Retrieved from Plotly: https://plot.ly/~unchained/37/bitcoin-utxo-age-distribution/#/

Unterguggenberger, M. (1934). The End Results of the Wörgl Experiment. *Annals of Collective Economy*.

von Ahn, L., Maurer, B., McMillen, C., Abraham, D., & Blum, M. (2008, September 12). reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science, 321*.

Warren, W., & Bandeali, A. (2017). *0x: An open protocol for decentralized exchange onthe Ethereum blockchain.* Retrieved from https://0xproject.com/pdfs/0x_white_paper.pdf

Wood, G. (2018). Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Byzantium Version*.

Yeow, A. (2018). *Bitnodes*. Retrieved January 12, 2018, from https://bitnodes.earn.com/

Zeppelin. (2018). Retrieved from OpenZeppelin: https://openzeppelin.org/

Zimmerman, P. (1994, October 11). *Volume I: Essential Topics.* Retrieved April 5, 2018, from PGP(tm) User's Guide: https://web.pa.msu.edu/reference/pgpdoc1.html