THE FLARE NETWORK AND SPARK TOKEN

Flare Networks

Version 1.1

August 27, 2020

ABSTRACT

The Flare Network is a distributed network running the Avalanche consensus protocol adapted to Federated Byzantine Agreement and leveraging the Ethereum Virtual Machine. It can thus be leveraged as a scaling method for smart contract networks without relying on economic safety mechanisms. The absence of a link between network safety and the native token, the Spark, allows for greater flexibility as to how the native token can be used. The Spark Dependant Application model provides a blueprint for building applications on the Flare Network. This relies on three components: Spark used as collateral, Spark used to contribute to the Flare Time Series Oracle providing on-chain data estimates and Spark used as a participation token in governance schemes.

1 Introduction

The increasing interest in using smart contract platforms to facilitate decentralized applications has led to an effort to develop new consensus protocols with increased computation throughput. In particular, Proof of Stake protocols (POS) are a family of consensus protocols which leverage economic means to guard against attacks. More specifically, network participants lock their network's native token in a staking contract, which they may lose in case of malicious behavior. Thus, the network's safety is proportional to the value of stake committed. This in turn limits the use cases that it is possible or desirable to express on a POS based network. Furthermore, competing uses for the network's native token, such as Decentralized Finance (DeFi), can cannibalize the safety of the network [Chi20].

In contrast, Federated Byzantine Agreement (FBA) are a class of scalable consensus protocols that do *not* rely on economic mechanisms to secure the network [Maz15; SYB+14]. Flare introduced the first such iteration of a Turing complete FBA protocol in [RU19]. The Flare Network is a new Turing complete smart contract platform, based on the Avalanche protocol [Roc18] in a FBA setting [CC19], and integrating the Ethereum Virtual Machine (EVM). The resulting network is scalable, safe and decentralized. It is freed from the constraints and potential safety conflicts of utilizing economic means to solve the Sybil attack.

Flare's native token, the Spark, is required for spam control at the network level. Because the safety of the network is not contingent on the token itself, Spark can be used in ways which would be unsafe for other networks. First, Spark can be used as collateral within applications. Second, Spark as a contributor, but not the sole contributor, to an oracle providing on-chain time series data estimates, called the Flare Time Series Oracle (FTSO). Third, Spark as a governance methodology across all elements that rely on it. These three components form the building blocks of a collection of applications that can be built using Spark, termed Spark Dependent Applications (SDA's). Furthermore, as the network doesn't require staking or computation returns for safety, the FTSO becomes the inflation function for Spark itself.

One natural application for the SDA is to enable trustless token representations of non-Turing complete tokens. Indeed, approximately 75% of the value that has emerged in public blockchain is in crypto currencies on non-Turing complete blockchains[1]. Spark enables the value represented by these tokens to be used on a scalable smart contract platform in a trustless manner. Furthermore, once that value is represented trustlessly on Flare it can then potentially be propagated across other networks through interoperability networks such as Cosmos and/or PolkaDot. The first such representation

[1] As of June 2020 according to https://coinmarketcap.com/

is XRP as detailed in [Net20], which for the first time will allow XRP to be used trustlessly with Turing complete smart contracts.

This document proceeds as follows. Section 2 introduces the Flare network and its consensus mechanism. Section 3 outlines the specification of Spark as a token that works like a master key for a potential series of other tokens and applications. In section 4, the Flare time series oracle is presented. Next, in section 5, the governance of the network is discussed, whereby Spark token holders can vote to modify network parameters. Finally, section 6 presents the Flare Foundation which is structured with the aims of advancing the network and helping implement certain governance decisions whilst being transparent, neutral and naturally reducing in prominence. Crucially, it can be dissolved by Spark token holders if it does not perform to expectations.

2 The Flare Network

In this section, an overview of the Flare Network is first presented in 2.1, which includes a discussion on the network consensus, as well as its safety, complexity and liveness considerations. In subsection 2.2, the process of achieving consensus on the state of any deterministic system external to the network for use in downstream applications, such as the XRP Ledger state for use in FXRP on Flare, is presented.

2.1 Overview

The *Flare Network* is a distributed network whereby nodes run the Avalanche consensus protocol [Roc18] with a key adaptation to a Federated Byzantine Agreement (FBA) consensus topology [CC19]. The Flare Network leverages the Ethereum Virtual Machine (EVM), enabling the network to run Turing complete smart contracts. The combined usage of the FBA consensus topology with a Turing complete smart contract layer makes Flare a scalable public smart contract network that does not require a native token for safety. This is a useful property as the cost to attack network safety in networks that do leverage a token for safety is related to the speculative value of the token, therefore facilitating an incentive in these networks to create a double-spend attack whereby the cost of an attack is lower than the reward from the successful double-spend [Lok+19].

Definition 2.1. The *Flare Network* is a distributed network whereby nodes run the Avalanche consensus protocol [Roc18] adapted to the FBA [CC19] consensus topology and leveraging the EVM.

Instead of being used for network safety, Flare's native token, the *Spark* introduced in section 3, is used to trustlessly issue tokens from non-Turing complete networks, such as XRP, onto Flare so that they can be leveraged in Turing complete smart contracts. Crucially, this trustless issuance on Flare does not require the cooperation of the non-Turing complete networks, meaning that the other networks do not have to make any changes to their protocol to enable the issuance on Flare. The Flare Network can then cooperate with interoperability protocols that do require bilateral protocol cooperation for cross-network asset issuance, such as Cosmos [ICS20], meaning that Flare can act as a unified frictionless pipeline of non-Turing complete assets into these networks.

Governance over the network and a wide set of parameters is achieved through Spark token ownership, as discussed in section 5. This allows Flare and its core applications to be highly dynamic, upgradable and increasing in utility.

2.1.1 Consensus

The Flare Network is organised in an FBA topology for its network-level consensus. FBA is unique as a consensus topology in that it achieves safety without relying on economic incentives that can interfere with high-value and high-risk use-cases [Maz15].

However, a criticism of pure FBA is that it leads to fragile structures of constituent nodes, permitting topology scenarios where a single node failure can cause a network-wide failure [Mac18]. For this reason, a specific setting of FBA called a Unique Node List (UNL) topology [CM18] is prioritised that emphasises clarity and ease-of-use while maintaining the open-membership property of FBA. The basis of the UNL topology is symmetry in network structure by requiring that the set of nodes that each node relies on for consensus intersects with other node's sets by a governance-defined threshold, called the *UNL intersection set*, which prevents the single node failure mode of the pure FBA topology. In the following, the total number of nodes in a UNL is defined as n. Note that the value of n can technically differ between different node operators as it is a privately set variable that cannot be forcibly set by others. However, for simplicity of safety analysis, it is recommended that nodes agree on a value for n through a non-automated Spark token holder governance vote, see section 5.3.3.

Definition 2.2. A node operator's *unique node list (UNL)* is that node operator's private definition of nodes that they voluntarily choose to rely on for consensus.

Definition 2.3. A *UNL intersection set* is a minimal UNL subset that overlaps a node operator's UNL such that any two node operators that wish to have their nodes be consistent in network state must share at least a common UNL intersection set. The UNL intersection set size is a governance-defined parameter and is defined as $I\%$ of n, where n is the number of nodes in a node operator's unique node list.

The underlying consensus protocol leveraged by the Flare Network is the Avalanche Consensus Protocol [Roc18]. Avalanche is a flexible protocol in terms of the different Sybil-resistance approaches that can be used with it, including the FBA (and therefore UNL) topology [CC19]. The Avalanche parameter k is the size of the uniform random sample of unique nodes from the network that are queried during consensus. In [CC19], Chitra et. al show that if any two samples of the nodes during consensus in an Avalanche network are guaranteed to intersect by at least one node, then a property necessary for safety in FBA networks called the quorum intersection property (QIP) is attained.

Definition 2.4. An FBA network has the *quorum intersection property (QIP)* if there do not exist disjoint quorums of the FBA network.

In the UNL topology setting of Avalanche for the Flare Network deployment, k is tuned such that any sample of the network contains more than 50% of a UNL intersection set. In other words, k is set to

$$k > n(1 - I/2), \tag{1}$$

for $0 < I \leq 1$, and where n is the UNL size. This guarantees that any two samples of the network during consensus by nodes with UNLs that overlap by a UNL intersection set will contain intersection by at least one node, and therefore attain the quorum intersection property (QIP).

Example 2.1. Suppose that the UNL intersection set size is tuned to 90% i.e. $I = 0.9$. The value of k is then set to strictly greater than $n(1 - I/2) = 0.55n$ i.e. to 55% of the UNL size. This will guarantee that a sample of the network will contain more than 50% of a UNL intersection set.

2.1.2 Network safety

The safety consideration, see definition 2.5, of the Flare Network is that the quorum size should be at least greater than or equal to $k > n(1 - I/2),\ 0 < I \leq 1$, in order to ensure that any two quorums intersect by at least one node. Indeed, the toy example 2.2, illustrates a risk in pure FBA where a quorum intersection of one node is a permitted network topology. This example shows why the Flare Network prioritises the use of the UNL topology over the pure FBA topology, as the following permitted setting in pure FBA would enable a single node failure to impact the entire network's safety.

Definition 2.5. A set of nodes in an FBA network exhibits *safety* if no two nodes externalize different values at the same time.

Example 2.2. Suppose that the UNL intersection set size is tuned to one node. The value of k is then set to 100% of the UNL size, as this would guarantee a sample of the network that always contains the one node in the UNL intersection set. Because this node alone intersects every possible quorum, it could cause a network-wide safety attack if it became Byzantine.

The following analysis demonstrates how the value of k can be tuned to avoid the one node intersection setting while still guaranteeing the preservation of the quorum intersection property. Suppose that greater than 66% of a UNL intersection set is included in any quorum. This means that any two possible quorums would intersect by at least 33% of the nodes in a UNL intersection set, instead of just by one node as in the previous example. Therefore, at least 33% of the nodes in a UNL intersection set would need to become Byzantine in order to cause a loss of safety between nodes that share a UNL intersection set, as this number would cause a loss of the quorum intersection property between at least two candidate quorums. 66% of a UNL intersection set can be defined mathematically as $(2n * I + 1)/3$. Then, any quorum must also account for the section outside of a UNL intersection set, defined as $n(1 - I)$, in order to ensure that any random sample of a node operator's UNL contains representation from at least 66% of a UNL intersection set. Combining the two sets together, the sample size k is set to $k = (2n * I + 1)/3 + n(1 - I) = n - (n * I - 1)/3 > n(1 - I/2),\ 0 < I \leq 1$, as the k-sized samples can be used to create *quorum slices* from the FBA topology framework [CC19], where a quorum slice is each node's private definition of what a quorum is [Maz15]. Note that for 100% UNL overlap, where $I = 1$, that the quorum size reverts to $(2n + 1)/3$, which is the quorum size in traditional Byzantine Agreement.

2.1.3 Network complexity

The communication complexity of the Avalanche consensus protocol is $O(kn \log n)$ [Roc18; CC19]. This means that it is necessary to bound the size of k while preserving the quorum intersection property. The following demonstrates an approach to reducing the communication complexity of Avalanche in the FBA setting, however, for the purposes

of simplicity in safety analysis of the Flare Network, we discuss why we recommend not following the approach. In [CC19], Chitra et. al propose a method that relies on Y independently drawn $k-1$ element subsets of the nodes in the network, where $Y \sim \text{Poisson}(\lambda)$. These subsets approximate the notion of *quorum slices* from the FBA consensus framework, and Chitra et. al prove that for $\log(n) < k \leq \frac{n}{2}$ and $\lambda < poly(k)$, that the quorum intersection property is guaranteed with high probability. However, the benefit of only needing to set one parameter is increased simplicity of node operation, which is a necessary trait for involving high numbers of node operators in the network in any case. Therefore, the method of only setting k to $k > n(1 - I/2),\ 0 < I \leq 1$ at the expense of higher communication complexity during consensus is prioritised.

2.1.4 Network liveness

The liveness, see definition 2.6, consideration of the Flare Network is that any set of nodes that blocks a node from being able to form a quorum during consensus can cause a loss of liveness for that node. In the UNL topology, nodes can freely change their UNL independently of other nodes in the network, and a node suffering from a loss of liveness can therefore change their UNL to substitute out failed nodes. This is not a feature of other networks that require global agreement on network membership.

Definition 2.6. A node in an FBA network exhibits *liveness* if it can externalize a value without the participation of any adversarial nodes.

2.2 State-Connectors: Consensus on the state of external systems

Consensus on the state of external systems, such as the XRP Ledger or a stock market, for use in applications on Flare such as FXRP [Net20], is a high-risk process. For example, the external system could suddenly issue a signal that a billion dollars worth of value has moved. As soon as possible then and without any human intervention, the Flare Network needs to be able to compute the finality of this external state such that it is available as a signal to the downstream applications on Flare, such as FXRP. In the case of FXRP, if the XRP Ledger state were recorded incorrectly to the Flare Network, then it would be possible to create a billion dollars worth of FXRP on the Flare Network that cannot be then converted back to XRP, undermining the value of the issued FXRP and any application that depends on it.

Thus, consensus on the state of external systems must be safely organised for immediate use in downstream applications on the Flare Network. This is achieved via *state-connectors*, defined in 2.7, which are responsible for safely encoding the state of external systems to the Flare Network.

Definition 2.7. The *state-connectors* are organised in a FBA topology in the UNL symmetric setting, just like the underlying Flare Network nodes. Each state-connector is run by a single Flare Network node, and the state-connector shares the same UNL definition as its Flare Network node. There are three phases of the state-connector system: 1) *sampling* the external-system state, 2) *registering* the state on the Flare Network and 3) *finalising* the state for use in smart contracts on the Flare Network.

The state-connectors are designed in order to fulfill the following criteria:

1. **Open-membership**. In the same way that anyone is permitted to participate in the consensus of the Flare Network, anyone is also permitted to run a state-connector.
2. **Consistency**. The recorded external state should be consistent between nodes on the Flare Network over time.
3. **Independent verification**. A state-connector should be able to independently verify an external state before it gets finalised to the Flare Network state. That is, if a state-connector's independent verification fails, then the state-connector does not permit the proposed external system state to be applied to its local Flare Network node.
4. **No reliance on economic incentives for safety**. Control over the state-connector system should not be based on who has the most money. Although state-connectors independently verify the external state such that they only permit the state that matches reality to be applied to the Flare Network, a state-connector with enough money could block the overall state-connector system from making progress on determining consistency between nodes on the Flare Network over time if control over this system were based on who has the most money. This has the same risk as basing the network consensus on who has the most money, that is that it would fundamentally limit the value that can be expressed in the system.

The sampling, registering and finalising phases of the state-connector system depend on the particular external system queried. In the following, these stages are concretely defined leveraging the XRP Ledger state example. The design can be straightforwardly generalised to other deterministic external systems of interest.

2.2.1 Sampling the external-system state

The primary engineering consideration in sampling the state of an external system is the rate at which the external system updates over time. For example, the XRP Ledger finalises rapidly at about once every 5 seconds. However, other systems such as Bitcoin have significantly longer periods of time in between state updates. In systems with faster state update times such as the XRP Ledger, it is useful to batch together *claims* about the state of the system into *claim periods* so that the state-connectors have enough time to both sample the external system and register its state to the Flare Network.

For example, a claim may represent a collateralization of XRP on the XRP Ledger or a derivative position in a stock market. Claims are external-system specific and are totally-ordered among other claims for each external system that is recorded to the Flare Network based on how they were originally ordered within the external system.

Definition 2.8. A *claim* is a unit of information about the state of an external system.

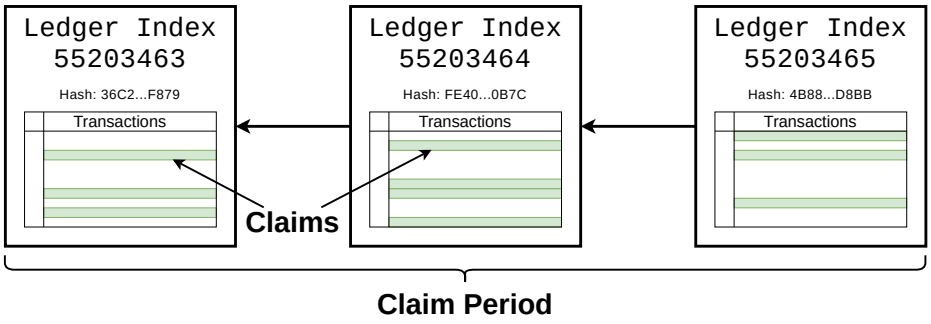

Figure 1: An example of *claims* within the XRP Ledger which are defined as a subset of the transactions within a ledger, for example all transactions relating to FXRP collateralisations, and where the *claim period* is set to three XRP Ledger closes.

The size of a claim-period set is flexible and should be related to the state update rate of its external-system. State-connectors must agree on the claim-period size for an external-system via governance, otherwise the state-connectors would lose consistency when coming to consensus on the existence and ordering of an external-system's claim-periods.

Definition 2.9. A *claim-period* is a totally-ordered set of claims.

Example 2.3. A claim-period in Bitcoin could represent a single Bitcoin block due to the longer length of time in between new blocks on Bitcoin.

Example 2.4. However, on the XRP Ledger, a claim period could be tuned to contain 10 XRP Ledger closes, as this gives a reasonable amount of time for state-connectors to query the XRP Ledger state, process the claims (e.g. filtering for spam claims), and then register the claims on the Flare Network to form a claim-period.

The state-connectors apply external-system state updates to the Flare Network by organising consensus on claim-periods. Claim-periods are totally-ordered among other claim-periods and are application-invariant. That is, the consensus system leveraged by the state-connectors in the smart contracts on Flare has no notion of the application details contained in claim payloads, the system instead is only concerned with organising consensus on the existence and ordering of claim-period sets.

2.2.2 Registering the external-system state on the Flare Network

Each state-connector periodically samples and then independently registers claims to the Flare Network that were previously unregistered by that state-connector in order to construct their own local definition of the current claim-period.

Claims do not automatically reach finality such that a non-reversible action occurs on the Flare Network given only a registration of a claim by a state-connector. A state-connector requires that a sufficient number of other state-connectors which it has chosen to rely on for consensus within its node operator's UNL have also registered the claim and finalised it as part of a claim-period set. The finality of a claim-period set is defined concretely then in the following section.

2.2.3 Finalising the external-system state for use in smart contracts on the Flare Network

Once a state-connector registers all of the claims contained within a claim-period set, it issues a signal to the Flare Network that it is proposing the claim-period to be accepted for finality. When a quorum of state-connectors from

the perspective of a state-connector's UNL registers the same state for a claim-period N, then this state-connector is deemed to have accepted this state. A state-connector will also accept this state if a v-blocking set of nodes has accepted the state, where a v-blocking set of nodes is such that it intersects every possible quorum of the state-connector by at least one node.

Accepting this claim-period N state does not mean that the constituent claims are finalized yet such that they are available to smart contracts on the Flare Network however. The final stage is that a state-connector must have personally registered the same definition for the accepted claim-period N state, based on its own observation of the external system. If the state-connector's registration is present, then the claim-period N is automatically finalized from the perspective of that state-connector when the claim-period $N + 1$ is first accepted. If the state-connector's registration is not present, then the local state-connector was not able to personally observe the claim-period N state and it therefore does not reach finality from the perspective of this state-connector, causing the state-connector's node to come to a safe halt[2].

Engineering considerations regarding the encoding of bespoke definitions of the unique node list within the smart contract layer for use in computing the consensus of the state-connectors are discussed in appendix A.

Therefore, the state of external systems, such as the XRP Ledger, will be available on the Flare Network for use in smart contracts in a way that provides: open-membership, consistency, independent verification and no reliance on economic incentives for safety.

3 The Spark token

The Spark token is the intrinsic currency of the Flare Network. Its technical use to curtail network spam is introduced in 3.1. Next, the applications of Spark are presented in section 3.2. The creation, distribution and apportionment methodologies for Spark are then specified in 3.3.

Definition 3.1. The *Spark* is the native token of Flare Network.

3.1 Spark technical use

Spark's technical purpose is to impose a cost upon transactions so as to disincentivize superfluous transactions i.e. network spamming. This is achieved via *transaction fees*.

The Flare Network uses the Ethereum Virtual Machine (EVM) [Woo+]. The EVM defines a transaction's computational complexity in terms of units of Gas. To avoid extremely lengthy or interminable transactions, Flare imposes a *complexity limit*, defined in Gas units. A *Gas conversion rate* between Gas and Spark tokens is set by the network. Thus, a *transaction cost* is defined as the complexity limit multiplied by the conversion between Gas and Spark. The transaction cost is burned rather than accruing to some set of participants. Both the complexity limit and Gas to Spark conversion rate are governance parameters, see section 5 for further information on governance.

Definition 3.2. The *complexity limit* is the maximum amount of Gas units that a single transaction can expend.

Definition 3.3. The *Gas conversion rate* is the cost of a Gas unit in Spark tokens.

Definition 3.4. There is a single *transaction cost* for any transaction of complexity up to the complexity limit. In Spark tokens it is given by $T = c \cdot \gamma$, where c is the complexity limit and γ is the Gas conversion rate.

3.2 Spark Dependent Applications

Spark is used as the arbiter of network governance and a central component of an interlocking set of utilities that collectively form a novel way of structuring decentralized applications, termed *Spark Dependent Applications* (SDA).

Definition 3.5. A *Spark Dependent Application* (SDA) is an application which can be built utilising any combination of the following three components: a Spark mediated time series oracle (see section 4), Spark used as collateral and Spark mediated governance (see section 5).

Each of these components is set in brief below.

The SDA structure is intended to provide utility by allowing applications that deploy on Flare to optionally benefit from:

[2]There are two ways a node can recover from this state: 1) Issue a signature that ratifies the proposed claim-period N state, or 2) If the node does not agree to the network-wide decision on the claim-period N state from the perspective of its UNL, then it can bootstrap to a different UNL that matches reality. During bootstrapping, signatures on ledger periods can be batched into one step in order to sync to the latest state immediately.

1. Access to assets and ecosystems that were not previously available through trustless representations of non-Turing complete assets, as exemplified by FXRP [Net20].
2. The value of Spark as collateral or a usage medium.
3. The use of the Flare Time Series Oracle (FTSO), as described in section 4.
4. The depth of interest and expertise that could be brought to bear over governance questions.
5. The increased likelihood of attention for a new application from the Spark owning community against the attention that application would have through bootstrapping from zero.

Collectively these can be termed the "network effects"[3] of the SDA as the benefits increase with usage.

3.2.1 SDA components

The first component of the SDA is the usage of Spark as collateral for any application that requires collateralization. Unlike the native tokens of smart contract platforms that use some form of Proof of Stake that incorporates slashing [Ben], the use of the Spark token as collateral does not compete with network safety itself because Flare network safety is not reliant on the token. Any application on the network can use the Spark token as collateral. The first such application uses the Spark token to create a trustless 1:1 representation of XRP, FXRP [Net20].

The second component of the SDA is the Flare Time Series Oracle (see section 4), which provides a periodic on-chain estimate of the current value of any number of off-chain time series. The contributors to a specific time series consists at a minimum of Spark token holders but will under certain circumstances include the token holders of an application that relies on that time series, called *F-asset* holders.

Definition 3.6. An *F-asset* is a token issued by an SDA that is permitted by Spark governance(see section 5) to participate in the Flare Time Series Oracle over one or several data estimates.

An F-asset could take any form given consent via governance from the Spark token holders. Two such examples are presented.

Example 3.1. *One-to-one representation* FXRP is a trustless representation of XRP on the Flare Network [Net20]. The FXRP application requires a price feed (data estimate) of the XRP/Spark price. FXRP is the F-asset relating to the XRP/Spark price feed.

Example 3.2. *One-to-many representation* An example application enables the automated accrual and realization of profits and losses, denominated in the Spark token, against peer to peer bets over a variety of different time series. The time series are estimated by the Flare Time Series Oracle. The application has a governance token which determines certain parameters internal to the application, such as collateralization, but is also the applications F-asset and contributes to the Flare Time Series Oracle (FTSO) for the estimation of certain series used by the application. *The design of applications that use a one-to-many F-asset must be carefully considered so as to avoid providing incentive to attack the FTSO. This is a key consideration of the Spark governance set when deciding to allow an applications proposed F-asset to participate in the FTSO.*

The Flare Time Series Oracle (FTSO) has a reward function which generates new Spark tokens. The new Spark tokens are used to reward Spark holding contributors. The determination by which contributors are rewarded is based on a process designed to economically incentivize honesty and is detailed in section 4. The total amount of the reward is the inflation rate of the Spark token and is determined by governance. F-asset holders who contribute data are not directly compensated by the FTSO. They may be incentivized to contribute because doing so provides greater surety over the validity of the data and/or contribution rewards specified within the application itself.

The FTSO will initially provide data estimates for XRP/Spark, USD/Spark, BTC/Spark and XLM/Spark. Of those, the token holders of the initial dependent application, FXRP [Net20], will contribute to the XRP/Spark price in addition to the Spark holders. The FTSO is extendable to include additional time series, as detailed in section 5. All data produced by the FTSO is freely usable by any application in the system. Thus any application can reformulate and augment the data in anyway the designers see fit.

The third component of the SDA is the use of Spark for governance purposes. Spark is used in the governance of the network and parameters around the Spark token. It can *optionally* be used by applications to involve a wider set of participants in critical decision making. There are no set rules as to how Spark is used in the governance of an SDA or for how long. FXRP is an example of an application that derives its governance from the Spark token which sets the collateral ratio, fees and various other parameters. A full presentation of the governance system is detailed in section 5.

[3] https://en.wikipedia.org/wiki/Network_effect

3.2.2 Mitigating a potential trade off

The three elements of the SDA structure deliver optimal utility only if they work together. Indeed, it is necessary that there be no competition between using Spark as collateral and those same Spark tokens being used to contribute to the FTSO or governance systems. An impediment to using Spark for all three elements simultaneously is generated by the use of Spark as collateral. This is because it is locked in a smart contract so as to be available to compensate one or many counterparties upon the determination of the application.

When Spark is locked in an application as collateral, the provider of that collateral still potentially has a claim on all or part of it, which is enforced algorithmically by the smart contract. In most applications, that claim will be the original collateral amount plus any additions less any losses and/or expenses. Usually the claim would correspond to the amount that the collateral provider could extract from the system if they were to unwind their participation in the application. This claim will likely be variable over time and is termed the *Spark claim*.

Definition 3.7. The *Spark claim* is the amount of Spark tokens that are realizable from an application by an address if all participation in the application was unwound.

If there were no way for the address to use the Spark claim amount to contribute to the FTSO (and potentially earn the FTSO reward) then the addresses owner faces an undesirable opportunity cost in deploying that Spark as collateral in the application. A system of *delegation* is thus introduced to resolve this. Delegation allows an address to bestow all or a fraction of the votes associated with its Spark tokens upon another address for the purposes of both FTSO and governance participation *without* moving or transferring those tokens. Each Spark token holds one vote that can be contributed to the FTSO and a separate vote that can be contributed to governance. These votes may be delegated to different parties. The opportunity cost described above is then solved when any application that creates a Spark claim automates delegation of the claim amount to an address specified by the collateral provider.

Definition 3.8. *Delegation* corresponds to an address on Flare assigning its own Spark token votes (or F-Asset votes) and any votes already assigned to it, to another address on Flare.

Token owners are free to delegate and un-delegate their votes at any time. Delegation can be structured such that one party can delegate to a second party and that second party can delegate to a third and so on. This leads naturally to the notion of an addresses *voting power*, which is used both by the FTSO and the governance system.

Definition 3.9. The number of non-delegated tokens held by an address plus any tokens delegated to that address, that are themselves not further delegated give the addresses *voting power*.

To summarise, applications that use Spark as collateral and allow delegation of Spark tokens do not impact the FTSO or governance functions and usage of Spark as collateral does not undermine network safety. Spark holders can simultaneously earn the FTSO return, return from any dependent tokens and continue to participate in governance.

Example 3.3. Flare XRP (FXRP) [Net20] is a representation of the XRP Ledger's (XRPL) intrinsic token, XRP. FXRP is backed by the Spark token and exchangeable 1:1 with XRP. The user does not need to trust a centralized party at any step throughout the creation, usage and redemption of FXRP, hence FXRP is trustless. The Spark token provides the collateral to underpin FXRP, hence it is only natural that the Spark token holders have a high level of governance over the FXRP system parameters. Spark token holders exert governance over the FXRP system through voting on system parameters that are encoded in smart contracts. An example of one such parameter is the FXRP collateral ratio. This is the value of Spark tokens that must be locked against issued FXRP, which at the outset is set to 2.5. It is then alterable via the proposal and governance process of Spark token holders and requires a super majority as defined in the governance section (see section 5) (at least 50% of all token holders participate and the proposal wins a 2/3 majority). The FXRP system requires a feed of the XRP/Spark price so that the value of collateral posted against issued FXRP remains above the collateral ratio. The price feed is secured by making it a responsibility of the two stakeholders in the FXRP system: the Spark and FXRP holders.

3.3 Spark creation and distribution

The Spark token will be distributed to the XRP ecosystem in what is intended to be a soft fork that generates considerable *utility* for the original chain (XRP), by enabling the use of XRP with Turing complete smart contracts, instead of trying to compete with it. A more apposite term for the process would be a *utility fork*.

3.3.1 Apportionment

A snapshot of the XRP Ledger will be taken at a specified date, detailed in section 3.3.2. At the inception of the Flare Network 100Bn tokens will be created. 45Bn tokens will be claimable for 6 months by XRP token holders excluding

known Ripple Labs accounts. The claim amount is calculated against XRP holdings at the time of the XRP Ledger snapshot via the *apportionment formula*, see definition 3.10. At the end of the claim period, any unclaimed tokens will be deleted. 25Bn tokens will go to Flare Networks Limited and 30Bn tokens will go to the Flare foundation. The Flare foundation, see section 6, is a non profit organisation intended to help develop the Flare Network and facilitate network governance.

Definition 3.10. XRP holders, bar Ripple labs, can claim Spark tokens according to the following *apportionment formula*:

$$x_{\text{Spark}} = x_{\text{XRP}}/(N - N_{\text{Ripple}}) * 45Bn, \tag{2}$$

where x_{Spark} is the amount of Spark claimable, x_{XRP} is the amount of XRP held at the snapshot date, N is the total amount of XRP in existence at the snapshot date, and N_{Ripple} is the total amount of XRP known to be held by Ripple Labs at the snapshot date.

3.3.2 XRP snapshot date

Because many XRP tokens are held on exchanges, the XRP Ledger snapshot will be taken at a point in the future when a sufficient number of exchanges have articulated to their clients whether they will claim the Spark token on their behalf. An announcement will be made on `https://flare.xyz` when the snapshot has taken place. The delay to taking the snapshot gives those XRP owners whose tokens are held in an exchange account but who wish to participate in the Spark distribution the means to do so should their exchange not provide such an option. Known exchange accounts who opt not to pass the Spark token on to their clients but still retain an XRP balance at the time of the snapshot will be removed from the token distribution. Any tokens that would have been claimable by those accounts will be reallocated pro-rata to valid claim holders.

4 Time series oracle

The *Flare Time Series Oracle* (FTSO) provides an estimate of information pertaining to the external world which is required on-chain. For the system to rely on one (or) more external information sources in order to provide this estimate would effectively introduce centralisation. By harnessing the decentralised nature of the network, estimates of a wide range of time series data can be computed via a *decentralised oracle*. This leverages the peer-to-peer nature of the network, whilst incentivising Spark users to contribute via the earning of rewards.

In the following, some background on decentralised oracles is first summarised in 4.1 and an overview of the FTSO is presented in 4.2. Next, in 4.3 the process of computing the estimate is detailed, followed by a description of the compensation scheme in 4.4 and of the pseudo-random number generator in 4.5. Finally, in 4.6, the FTSO's susceptibility to attacks by malicious parties is discussed.

4.1 Background

Decentralised oracles typically leverage the system participants in order to vote regarding the nature of a given proposition. For instance, the Schelling Coin protocol [But14], uses a commit and reveal scheme, in order to gather votes from network participants. This results in a distribution over the votes, of which the top and bottom 25% are then truncated. The median of the resulting distribution is taken as the price estimate, and the 50% of voters whose submissions were used are rewarded. Such schemes can be studied from a game theoretic perspective, whereby each participant has to make a choice in order to maximise their reward. For instance, the Schelling Coin scheme hinges on the idea that such a game has a Schelling point, i.e. a focal point [Sch80].

Over recent years, the need to query external information to the chain has sparked the development of various approaches to oracles, such as TruthCoin consensus [Szt15], Chainlink [EJN17], Astraea [Adl+18] or Terra [KDKP19]. The core feature of an oracle should be a decentralised data feed exploiting the distributed nature of the underlying system whilst incentivizing players to be honest.

4.2 FTSO overview

The Flare time series oracle is a decentralized application that aims to generate accurate estimates of off-chain time series data on the Flare Network. The oracle takes as inputs estimates from two sets of participants: Spark holders and relevant F-asset holders. At regular and prescribed time intervals (a governance parameter), participants from both sets may (but are not obligated to) submit to the system their current estimates.

The resulting set of votes forms a distribution over the data estimates, termed the *weighted estimate distribution*, see definition 4.5. In order to remove outliers, which likely consist of errors and attempts to be malicious, only the middle 50%, as determined by number of votes, is retained. This results in the updated *truncated estimate distribution*, see definition 4.6. The weighted mean of the truncated distribution is then used as the value estimate for the next A_{blocks} blocks, where A_{blocks} is a system parameter. Holders whose votes contributed to the final output are then compensated, as described in the compensation scheme in definition 4.12.

4.3 Computing the estimate

The time series oracle yields an *oracle estimate* of a given variable. These are computed by processing estimates from relevant token holders at regular time intervals, termed *voting rounds*. At the start of each voting round, relevant token holders (as later defined) submit their data estimate, termed their *vote*, from which the oracle computes the *oracle estimate*. The votes are submitted using a commit and reveal scheme [Gol07].

Definition 4.1. A *voting round* t is a time interval during which votes can be submitted. This is a system governance parameter.

Definition 4.2. The *vote* v_t^i at voting round t is the submitted estimate of the ith participating address.

Each oracle estimate is determined by two groups with competing interests: holders of Spark, and holders of the F-asset tokens issued in relation to the relevant data estimate. For simplicity of notation, in the following, a single data estimate is considered. The proposed procedure can then straightforwardly be extended to n different time series.

Example 4.1. The FXRP system [Net20] requires a robust estimate of the XRP/Spark price to ensure that sufficient collateral is held against issued FXRP. In this case, the F-asset is FXRP, and the time series oracle provides the on-chain XRP/Spark price, based on the estimates being submitted by both holders of Spark *and* FXRP. By virtue of holding either FXRP or Spark, both these groups have an implicit stake in the system i.e. an incentive to act honestly, as accurate pricing maintains the systems integrity and utility.

At each voting round t, let there be N_t^S Spark addresses and N_t^F F-asset addresses who wish to participate and submit a vote. These are termed the *partaking addresses*, defined in 4.3. Let Q_t^S be the total amount of Spark voting power on the N_t^S participating Spark addresses, and let Q_t^F be the total amount of F-asset voting power on the N_t^F F-asset participating addresses in voting round t. Thus, the total number of partaking addresses is given by $N_t = N_t^S + N_t^F$.

Definition 4.3. For a given voting round t, *partaking addresses* consist of:
1. Spark addresses who choose to participate in round t,
2. F-asset addresses who choose to participate in round t.

Next, each account casts their vote i.e. submits their estimate. The number of tokens associated with the voting addresses at the start of the voting round are considered. Say the voting power of address i is x_i^S if it is a Spark address, and x_i^F F-asset if it is an F-asset address. If it is an F-asset address, the voting power is proportionally scaled by an *adjustment factor* $\frac{Q_t^S}{Q_t^F}$, resulting in the *adjusted voting power*, see definition 4.4.

Definition 4.4. The F-asset *adjusted voting power* is given by

$$\tilde{x}_i^F = \frac{Q_t^S}{Q_t^F} \cdot x_i^F, \tag{3}$$

where Q_t^S, Q_t^F are the total amount of tokens held on Spark and F-asset partaking accounts at the start of round t, and x_i^F is the voting power of the ith partaking F-asset address.

As the voting power of F-asset addresses are adjusted and scaled to the amount of assets held in voting Spark addresses, the total amount of voting power Q_t is given by $Q_t = 2Q_t^S$.

Then, the submitted votes weighted by voting power (or adjusted voting power depending on address type) result in a distribution over data estimates, termed the *weighted estimate distribution* $\mathcal{P}_t$. Each vote cast is weighted by the amount of tokens held as follows.

Definition 4.5. The *weighted estimate distribution* $\mathcal{P}_t$ for voting period t consists of a set of estimates $\{v_i\}_{i=1}^{N_t}$, and an associated set of weights $\{w_i\}_{i=1}^{N_t}$, where

$$w_i = \begin{cases} x_i^S/Q_t, & \text{if } i \text{ is a Spark address,} \\ \tilde{x}_i^F/Q_t & \text{if } i \text{ is an F-asset address.} \end{cases} \tag{4}$$

where $Q_t = 2Q_t^S$ and where furthermore $\sum_{i=1}^{N_t} w_i = 1$

Next, the distribution is truncated by computing $0.25Q_t$, and deleting the lowest and highest such votes i.e. 50% of the submitted votes are retained. This results in the truncated distribution $\mathcal{P}'_t$ over a new set of prices i.e. a same or smaller set. Indeed, if address i initially had x_i tokens, then all, some or none of these may be deleted after truncation. Thus, address i is now left with x'_i token votes. Thus, $x_i'^S$ and $\tilde{x}_i'^F$ correspond to respectively Spark and F-asset votes remaining after truncation.

Definition 4.6. The *truncated estimate distribution* $\mathcal{P}'_t$ for voting period t consists of a set of estimates $\{v_i\}_{i=1}^{N'_t}$, where $N'_t \leq N_t$, and an associated set of weights $\{w'_i\}_{i=1}^{N'_t}$, computed from where $x_i'^S$ and $\tilde{x}_i'^F$, which are the surviving votes of respectively Spark and F-asset account holders.

Finally, the output of the oracle can be computed by taking the weighted mean of the resulting distribution.

Definition 4.7. The *oracle estimate* is the computed estimate at voting round t based on partaking votes. It is computed by taking the weighted mean of the truncated estimate distribution $\mathcal{P}'_t$ and is denoted m_t.

The following two examples illustrate how the oracle estimates are computed. For simplicity sake, these consider the case where all partaking accounts are Spark accounts.

Example 4.2. Alice, Bob, Charlie and Eve hold respectively 10, 20, 30 and 20 Spark tokens, and decide to participate in round t of price voting. They are the only four participants who choose to vote. In terms of weighted holdings, this results in a total of 10+20+30+20=80 votes. Alice submits a vote for 3, Bob for 4, Charlie for 5 and Eve for 6, which results in a distribution over prices $\mathcal{P}_t$. Next, the top 25% and lower 25% of votes are discarded i.e. the top 20 and lower 20 votes are removed. The resulting distribution $\mathcal{P}'_t$ is now 10 votes for price 4 from Bob and 30 votes for price 5 from Charlie. The weighted mean of this distribution is $m = \frac{10}{40} \cdot 4 + \frac{30}{40} \cdot 5$ i.e. $m = 4.75$, which is the oracle output.

Example 4.3. Alice, Bob and Charlie hold respectively 100, 50 and 100 Spark tokens, and decide to participate in a round of price voting. They are the only three participants who choose to vote. In terms of weighted holdings, this results in a total of 100+50+100=250 votes. Alice submits a vote for 2, Bob for 3 and Charlie for 4, which results in a distribution over prices $\mathcal{P}_t$. Next, the top 25% and lower 25% of votes are discarded i.e. the top 62.5 and lower 62.5 votes are removed. The resulting distribution $\mathcal{P}'_t$ is now 37.5 votes for price 2 from Alice, 50 votes for price 3 from Bob and 37.5 votes for price 4 from Charlie i.e a total of 125 votes have survived truncation. The weighted mean of this distribution is $m = \frac{37.5}{125} \cdot 2 + \frac{50}{125} \cdot 3 + \frac{37.5}{125} \cdot 4$ i.e. $m = 3.0$, which is the oracle output.

More generally, if there are n estimates to compute, Spark addresses who choose to vote will no longer submit a single vote per round, but a list of n votes $\mathbf{v} = (v_1, \ldots, v_n)$. This will result in n estimate distributions, which will then each be truncated, resulting in n truncated distributions. The weighted means m_i for $i = 1, \ldots, n$ of the resulting truncated distributions are then computed, resulting in n oracle estimates $\mathbf{m} = (m_1, \ldots, m_n)$.

Definition 4.8. The *Flare time series oracle* (FTSO) provides data estimates as follows. In a given voting round t:
1. Partaking addresses submit their data estimate,
2. A weighted estimate distribution $\mathcal{P}_t$ is generated,
3. The top 25% and bottom 25% of this distribution is truncated, resulting in an updated distribution $\mathcal{P}'_t$,
4. The weighted mean m of the resulting truncated distribution is outputted, called the oracle estimate,
5. Holders whose estimates remain in the set are remunerated, as per compensation scheme 4.12,

4.4 Compensation

Spark holders that submitted an estimate that remained in the distribution after truncation are compensated. The precise amount of Spark tokens minted for reward is a network governance parameter, which is initially set at 10% of Spark tokens in circulation per annum, and is termed the *award rate*, see definition 4.9. The Spark token holders can subsequently vote in order to change this, as set out in section 5. Note that although F-asset holders can participate, they do not get compensated. Instead, F-asset holders are incentivized to vote to ensure the integrity of the data related to the F-asset.

Definition 4.9. The *award rate* r_A is the annual amount of Spark tokens minted as a percentage of Spark tokens in circulation at the beginning of the year to be awarded to successful votes i.e. votes surviving truncation. It is by default set to 10%.

The total minted tokens for compensation per year is then broken down over the number of voting rounds over the year, which is also a governance parameter. This determines the *award* a Spark holder can hope to earn for each voting round.

Definition 4.10. The *voting rounds per year* k is the number of voting rounds that occur in one calendar year.

Definition 4.11. The *award per period* A is the award available for a given voting round, and which is determined as follows: $A = r_A \cdot S^T_{tot}/k$, where S^T_{tot} is the total amount of Spark in circulation in year T and where k is the number of voting rounds per year.

Example 4.4. Say there are 1440 Spark in circulation in year 1 i.e. $S^1_{tot} = 1440$. At a default award per annum rate of 10%, 144Spark is minted for compensation. For $k = 12$, voting takes place on a monthly basis (note that in practice, it will be far more frequent). Then, every month 12Spark are available for compensation.

If n oracle estimates are computed at round t, then only one of these is rewarded. In order to determine which, a pseudo-random number between 1 and n inclusive is drawn, via the on-chain pseudo-random number generator, see section 4.5. Say the pseudo-random number drawn is k, with $1 \leq k \leq n$. Then, the addresses whose votes have survived truncation for the kth distribution and thus contributed to the computation of the kth oracle estimate are rewarded. Each address i whose vote contributed to the kth oracle estimate is rewarded a compensation amount a_i, computed as per definition 4.12. Thus, each voting round, a new oracle estimate will be rewarded, providing an incentive for participants to vote across *all* prices.

Definition 4.12. The *award* is the amount of Spark a Spark holder receives if their vote contributed to the oracle estimate i.e. survived truncation, and is computed as follows. Let $N_t^{'S}$ be the number of Spark addresses which survived truncation. The award for Spark address i is given by

$$a_i = A \cdot \frac{x_i^{'S}/x_i^S}{\sum_{j=1}^{N_t^{'S}} x_j^{'S}/x_j^S}, \tag{5}$$

for $i = 1, \ldots, N_t^{'S}$.

Example 4.5. Following example 4.4, say an amount 12 Spark is up for compensation. In example 4.2, Bob and Charlie have survived the truncation, and will thus be compensated for their participation. From Bob's votes, 50% contributed to the final distributed, and 100% of Charlie's contributed. Thus, Bob will obtain 1/3rd of the amount up for compensation and Charlie 2/3rds, i.e. 4 and 8 Spark respectively.

Example 4.6. Following example 4.4,say an amount 12 Spark is up for compensation. In example 4.3, Alice, Bob and Charlie have survived the truncation, and will thus be compensated for their participation. First, the proportion of surviving votes is computed. For both Alice and Charlie, this is 37.5/100=0.375 whereas for Bob this is 50/50=1. Computing the weights for each yields $0.375/(0.375 + 1 + 0.375) = 0.375/1.75$ for Alice and Charlie, and $1/1.75$ for Bob. This results in compensation of 2.57 Spark for Alice and 'Charlie, and 6.86Spark for Bob.

An important consideration is that sufficient number Spark addresses could wish to increase their chance of obtaining the reward by corrupting the system through the creation of dummy time series and related F-assets which are added to the FTSO. In turn, this could undermine the economic incentive for partaking addresses to provide honest estimates to the FTSO. In order to counteract this, when the number of F-assets on Flare is greater than one, a value threshold $Q^{\text{threshold}}$ for each F-asset is imposed. If $Q_t^F \geq Q^{\text{threshold}}$, then the compensation scheme proceeds as previously described. But, if $Q_t^F < Q^{\text{threshold}}$, F-assets retain the ability to contribute data to the time series that relate to them, however their votes are no longer included in the calculation of the award. More specifically, for each F-asset which is below threshold, the weighted estimate distribution P_t is re-computed with $\tilde{x}_i^F = 0$ for all partaking addresses i associated with the F-asset F. From this, the truncated distribution is re-computed and the award is computed for this truncated distribution.

4.5 Accessing pseudo-randomness on-chain

At each voting round, the compensation scheme requires access to a random number between 1 and n. Accessing randomness on-chain whilst not introducing an element of centralisation is an active area of blockchain research and development [EJN17; Gil+17]. In order to choose the data feed to reward, the Flare time series oracle relies on the *on-chain pseudo-random number generator*, see definition 4.13, and proceeds as follows.

At each voting round t, partaking accounts must submit a random number between 1 and n inclusive alongside their vote. This number is denoted y_t^i, where $i = 1, \ldots, N_t$. Next, these are added together resulting in a number Y_t. For n data feeds, this means $N_t^S + n \cdot \sum_{i=1}^n N_t^{F_i} \leq Y_t \leq n(N_t^S + n \cdot \sum_{i=1}^n N_t^{F_i})$, where the lower bound corresponds to the case where each account submits a 1, and the upper bound, n. Next, the hash function SHA-256 is applied to Y_t, the output of which is a 256-bit string, i.e. a number between 0 and 2^{256}. Next, this is taken modulo n, and a 1 is added to the result, in order to obtain an integer between 1 and n inclusive. This, is the output of the *on-chain pseudo-random number generator*, summarised in 4.13.

Definition 4.13. Each round t, the *the on-chain pseudo-random number generator* takes as input a random integer y_t^i from each partaking account i in voting round t and computes the following:

1. The sum of all the submitted random numbers $Y_t = \sum_{i=1}^{N_t} y_t^i$, for $i = 1, \ldots, N_t$,
2. The output is given by $N_t = 1 + \text{SHA}_{256}(Y_t) \mod n$.

The on-chain pseudo-random number generator relies on both Spark and F-asset holders to submit a random number. Note that it is not possible for the network to ascertain whether a given submitted value is indeed random or not. Instead, the system relies on the participants to be incentivised to submit a random number as requested. For Spark holders, the incentive is to obtain the reward and thus increase their Spark holdings. If they believe the outcome is indeed random, then this incentivizes them to submit what they believe are true prices for all n data estimates, as this maximises their chance of surviving distribution truncation. In contrast, for F-asset holders, the incentive is to ensure the FTSO functions correctly i.e. that the participants submit honest votes regarding their estimates.

4.6 Susceptibility to attacks

The FTSO builds on ideas introduced in the Schelling Coin [But14], which is susceptible to the $P + \varepsilon$ attack [But15]. The FTSO differs to the Schelling Coin in that it introduces implicit stake in the form of Spark and any contributing F-asset and where each series is formed by at least two sets of participants with differing incentives. In the following, a discussion regarding the FTSO's susceptibility to the $P + \varepsilon$ attack is presented. A quantitative analysis of all known attacks will be released separately together with any updates required to the FTSO structure.

The $P + \varepsilon$ [But15] attack can be briefly characterised as follows. In a binary outcome process, a participant votes 1 or 0 according to what they believe the majority will vote, in order to earn a reward P. Each participant has one vote and there are no restrictions on who the participants are. The attacker commits to paying out $P + \varepsilon$ to those who vote 1 when the majority vote 0, where ε is some marginal amount. This skews the economic benefit for all participants to vote 1, regardless of the truth. Hence the majority of participants vote 1 and the attacker does not need to pay out. Pricing mechanisms such as Schelling Coin are susceptible to the $P + \varepsilon$ attack as these hinge on how the majority of *participants* in the system will vote. In contrast, the FTSO weighs votes with token holdings. This means that the outcome of the vote is not dependent on the majority of participants, but instead on the majority of *stake*. This means that in order to corrupt the system, holders of a stake majority should vote a given way.

More specifically, the $P + \varepsilon$ hinges on the fact that each participant has a single vote i.e. the votes are effectively uniformly distributed across participants. Thus an attacker has to credibly demonstrate they have the capital to corrupt strictly more than 50% of participants. To translate this to Flare, each address would have 1 token giving it a voting power of 1 which cannot be delegated. A participant would also be limited to having a single address. So if there are 100 participants an attacker could only have a chance of success if they demonstrated sufficient capital to pay out $P + \varepsilon$ to 51 participants. However, on Flare each token equals a vote and tokens are not uniformly distributed across participants. Translated to Flare, the $P + \varepsilon$ attack would require an attacker to convince the holders of a majority of tokens to engage in the attack. This results in a situation for the attacker where the minimum economic commitment they must demonstrate for a successful attack must account for more than 50% of the tokens. However the minimum set of participants on Flare that own strictly more than 50% of the tokens could likely own some quantity reasonably greater than just over 50%. This minimum set of participants would not accept compensation on only a partial amount of their holdings so the entirety of the token holdings of that minimum set would need to be compensated. Thus the attacker would at least need to demonstrate a credible commitment an amount of capital that *may* be far greater than under the $P + \varepsilon$ analysis.

A second consideration it that the $P + \varepsilon$ requires the majority of participants to not be altruistic. For Flare, a scenario whereby a minority of participants could be altruistic but still own a majority of tokens could occur. This in turn would make such an attack impossible.

Third, given the unknown token ownership distribution at any point in the future, it cannot therefore be automatically assumed, as it is in $P + \varepsilon$, that an attacker will successfully attack some future round of voting, the effects of which backpropagate to the current round.

Finally, in order to vote a participant must own a Spark token or an F-asset. Hence every vote has a value which is represented by each token. The value of Spark is partially derived from the FTSO. The value of an F-asset is linked to both the oracle, through the time series in the application linked to the F-asset, and the value of Spark. Hence if the oracle loses utility through providing inaccurate estimates, the loss to both Spark token and F-asset holders will be considerable. So to replicate the incentive structure detailed in the $P + \varepsilon$ attack to Spark holders, the attacker would have to offer at a minimum the value of $P + \varepsilon$ before the attack plus the Spark loss value. Similarly to replicate the attack structure for an F-asset holder, the attacker would have to offer at a minimum the value of the F-asset loss.

Such a reward would only be credible if the attacker displayed an amount of capital committed to the attack that is the sum of the attack incentives for a sufficient amount of oracle participants, which for each time series must be more strictly than 50% of the scaled votes . Assuming economic rationality, this commitment could only occur if there was some economic mechanism by which the attacker could benefit in a way whereby for each unit value drop in Spark and the specified F-asset the attacker made more than one unit of value plus $P + \varepsilon$ *and* this mechanism is available to the

attacker at a scale to cover a proportion of holdings of participants that would be very large. Any counter-party or set of counter-parties offering such a mechanism to the attacker would themselves need to be economically irrational and hence the attack is highly improbable.

5 Governance

The Spark token is used as the primary governance mechanism over the Flare Network, the Spark token and the Flare Time Series Oracle (FTSO). These elements are detailed in what is set out below. Spark can also be optionally used for governance over dependant applications. This is determined by the application itself and is not relevant to this paper.

Certain updates from governance decisions can be automated, termed *automated processes*, whereas others, the *non-automated processes* can not. The latter typically require a more complex procedure, requiring for instance changes to the codebase. Automated processes that relate to a variable which can take a range of values are designed to increment slowly so as to avoid shocks to the system. An *emergency automated process* is defined for situations where there is general consensus amongst Spark owners that the variable under consideration needs to be altered rapidly.

Table 1 shows governance parameters on the Flare network. Decisions are categorized into whether they relate to the network and its technical parameters, the Spark token or the Flare Time Series Oracle.

Network	Spark	FTSO
Complexity limit	Spark to Gas unit conversion	FTSO reward rate
Consensus parameter I	Additional Spark distributions	F-asset value threshold
UNL parameter n		Oracle update frequency, T
Network code changes		Oracle voting period, t
		Oracle methodology update
		F-asset inclusion
		Change oracle series composition

Table 1: *Governance parameters on the Flare Network.* Decisions that can be automated are shown in black whereas decisions that require a code change, and are thus termed non-automated, are shown in blue.

5.1 Decision categories

Governance decisions are made through voting using the Spark token. Each Spark token equals one vote. Because different decisions have different impacts on the network, three decision rules are specified: *simple majority* (definition 5.1), *super majority* (definition 5.2) and *super super majority* (definition 5.3). Each of these places specific requirements on the number of Spark tokens participating in the voting process, as well as the number of votes for a proposition to pass. The Spark tokens held by the Flare Foundation (section 6) may not be used to vote and are considered not to exist for the purpose of governance.

Definition 5.1. For a *simple majority decision*, a proposition is confirmed if strictly more than 50% of votes cast are in favor of it and there is a minimum turnout of 30% of total Spark tokens.

Definition 5.2. For a *super majority decision*, a proposition is confirmed if strictly more than 2/3rd of votes cast are in favor of it and there is a minimum turnout of 50% of total Spark tokens.

Definition 5.3. For a *super super majority decision*, a proposition is confirmed if strictly more than 80% of Spark tokens vote in favor of it and there is a minimum turnout of 70% of total Spark tokens.

Decisions fall into categories that are predefined prior to network instantiation. The category itself can be changed via the non-automated governance process.

Table 2 shows governance decision categories for various system parameters.

Simple Majority	Super Majority	Super Super Majority
Transaction cost parameters	Consensus parameter I & n	Additional Spark distributions
Oracle composition	F-asset inclusion	Oracle methodology update
	F-asset value threshold	Other network code changes
	FTSO reward rate	
	Oracle update frequency, T	
	Oracle voting period, t	

Table 2: *Decision Categories* Governance decisions fall into one of three categories, depending on the various impacts these can have on the network.

5.2 Direct and delegated voting

Flare will follow a *liquid democracy*[4] model whereby token holders are free to vote themselves or to delegate some or all of their tokens (votes) to another entity which votes on their behalf, as defined in 3.8. A voting entity that amasses delegated votes can be formed by any one or collection of people. One of those entities will be the Flare Foundation.

5.3 Voting considerations

The following sections cover governance voting over automated variables and more complex decisions which require code updates. Voting is based on an addresses voting power, as defined in 3.9. To avoid an attack on governance whereby tokens are shuttled from one address to another during the voting period, Flare restricts an address's voting power to its voting power at the beginning of any voting period.

Definition 5.4. The *governance voting period* T_G is the number of blocks over which voting takes place.

The voting power of an address for the purpose of governance is then the voting power of the address in the first block of T_G.

5.3.1 Automated Process

For variables which can be updated automatically, initial parameters will be set at the instantiation of the network. Variables that can be automated are either binary or can take values that are a finite subset of the real numbers.

Definition 5.5. An *automated variable* is either binary or takes on one of a finite set of values. It is set through network consensus.

For non binary automated variables, reaching agreement on both direction and magnitude of a change could be complex. Therefore, for such a variable, a preset increment is set such that the governance decision is over whether to increment the variable in a particular direction, and not in the size of the step. This is initially set by the variable creator, but can itself be governed in an automated fashion.

Definition 5.6. A *variable increment* is a predefined incremental increase or decrease, relating to a specific automated variable.

For any automated variable, a minimum preset threshold of interest to increase or decrease the variable (or switch state if binary) must be reached to trigger a network wide governance vote. A *governance vote* is the process that decides on the change in a variable.

Definition 5.7. A *governance vote trigger* is a smart contract associated with each automated variable, and which runs as follows:
1. The smart contract accepts votes over a voting period T_G.
2. Each Spark address can place a vote, up to its voting power in the first block of T_G, for the variable to either change (binary) or decrease/increase (by the predefined increment).
3. If at any point during the voting period, the sum of the votes in a particular direction exceed the preset threshold, then a *network wide governance* vote takes place, as defined in 5.8.
4. If at the end of T_G an insufficient amount of votes are accumulated to lead to a governance vote, the governance vote trigger is reset and reinstated.

[4] https://en.wikipedia.org/wiki/Liquid_democracy

Definition 5.8. The *governance vote* is a network wide vote that takes place if a governance vote is triggered, and proceeds as follows:
1. The governance vote trigger system for that specific variable is suspended for the duration of the governance vote, as defined by the number of blocks T_G.
2. The vote trigger outcome (i.e. to change state or increase/decrease the variable value) becomes the proposition for the governance vote over the specified variable.
3. The votes for the proposition accumulated in the governance trigger are automatically set in favor of the proposition.
4. Additional votes are accumulated during T_G using each addresses voting power from the first block of T_G.
5. If at any point during the voting period the sum of the votes exceeds the minimum defined by the preset decision category, the variable is incremented by the predefined amount in the direction specified by the proposition.
6. If at the end of T_G an insufficient amount of votes are accumulated to lead to a change in the variable the governance vote trigger is reset and reinstated.

5.3.2 Emergency automated governance

The automated governance process is purposefully designed to iterate through values slowly so that the system and participants have time to adjust and arrange their affairs accordingly. In certain cases however it may be necessary to make larger adjustments quickly.

During each governance trigger voting period T_G, any address with voting power may enter a proposed numeric value that is at least $n + 2$ variable increments away from the current value, where n is an integer and governance parameter. This creates an additional governance trigger proposal that remains for the duration of the current voting period against which voting power can be expressed. The same rules as set out in 5.7 are applied. If sufficient votes are amassed, then a network wide governance vote is held over this value. Otherwise, the new governance trigger is deleted at the end of the current governance trigger vote period. Multiple additional governance triggers may be proposed provided they are at least n variable increments away from each other.

Definition 5.9. The *Emergency governance vote trigger* is a process which runs as follows:
For a specific variable that can be changed by automated governance and referring to that variables governance trigger:
1. If during a period T_G, an address with voting power sends a transaction to the governance trigger contract, containing a suggested numerical value for that variable *and* that numerical value is at least $n + 2$ variable increments away from the current value or n variable increments away from any other proposed value, then:
2. An additional voting proposition is created within the governance trigger contracts that sets the suggested numerical value as a new proposition which can accumulate votes.
3. As per the governance trigger process each Spark address can place a vote, up to its voting power in the first block of T_G, for the variable to either change (binary), decrease/increase (by the predefined increment) or for any additional voting proposition.
4. If at any point during the voting period, the sum of the votes for any proposition exceeds the preset threshold, then a *network wide governance* vote takes place for that proposition, as defined in definition 5.8.
5. If, at the end of T_G, an insufficient amount of votes are accumulated to lead to a governance vote, the governance vote trigger is reset, any additional propositions are deleted and the governance vote trigger restarted.

5.3.3 Non-automated process

Implementing governance decisions that require a codebase change follows a *non-automated process*, where definition 5.10 specifies the sequence of steps taken for such a change, from inception to implementation. The set of decisions that such a process encompasses is broad. For example, this would include decisions such as updating the set of dependent applications in order to add or remove an application. This, in turn, may require an addition or subtraction of a price feed at the FTSO or an issuance of tokens via a secondary utility fork.

Definition 5.10. A *non-automated process* proceeds as follows:
1. A *proposal submission*, whereby a proposal is submitted via a form hosted at `https://flare.foundation`.
2. The submitter will receive a unique code from the Flare website which must then be submitted to a smart contract on Flare.
3. The smart contract now sets a new variable (the identifier) against which voting power can be accumulated and a block by which to define a freeze of voting power.
4. Once the foundation observes that the transaction has occurred, the proposal is published on the website in order to be publicly accessible and viewed.
5. Additional votes for a proposal are submitted by transacting with the smart contract using the proposal identifier.
6. The voting period for the proposal is approximately 3 months after which the variable is terminated.
7. A *foundation analysis trigger*, the foundation will initiate analysis on any proposal which reaches 50% of all voting power.

8. A *foundation report* is an analysis on a proposal compiled by the Flare foundation, complete with findings and a recommendation. Foundation may reject the proposal and end the process for the following reasons: legality, the proposal if successful would exceed foundation resource management constraints, technical in-feasibility, network safety parameters.
9. A *first hurdle*, as per table 2 any proposal must accumulate voting power exceeding its decision category within its voting period.
10. If the proposal relates to updating the FTSO composition, F-Asset list and/or related time series, network parameter I or network parameter n, this will be the only vote and the changes will be directly implemented*.
11. A *build and test*, whereby if the proposal requires code changes, these will be implemented and tested on the Coston test network.
12. A *second report* is compiled by the foundation, whereby the foundation issues its recommendations for any proposal that requires a second vote.
13. An *implementation vote*. The foundation second report contains an identifier against which a second round of voting takes place. This round is always decided by a simple majority decision.

Certain non-automated decisions can be implemented by augmenting smart contracts or adopting new sets of smart contracts. Other decisions however require node operators to adopt a new version of the Flare Network code distribution. It must be noted that nodes cannot be forced to run a new code distribution. The purpose of the non-automated methodology is to sort for potentially beneficial updates, involve the technical expertise of the Flare foundation to advise or build the update where applicable, utilize Coston as a testbed for those potential updates and then through voting show a weight of Spark tokens behind those updates. This weight of Spark tokens is then likely to incentivize nodes to adopt the update.

5.4 SDA interaction with Spark governance

An SDA interacts with Spark governance in a number of defined ways.

An SDA or potential SDA that wishes to add a data stream to the FTSO will use the non-automated governance process to propose the addition which may or may not be successful. An SDA or potential SDA can use Spark token holders as their governance set in whole or in part for any duration without the consent of the collection of Spark token holders. The participation of Spark token holders is entirely optional.

6 Flare foundation

This section is intended as a brief overview of the Flare foundation. A complete constitution will be released separately.

The Flare foundation is to be a non-profit organization whose mandate is to participate in developing and improving the network. The foundation will undertake this in five ways: grants, investments, direct software development, education content and publicity & partnerships.

6.1 Guiding philosophy.

If operating correctly, the foundation should be a rallying point for the ecosystem to communicate, innovate and further develop. It should not have excessive power in the network, nor should it interrupt or distort any of the economic processes that operate on the network. To that end the foundation is bound by a set of rules, as formulated in definition 6.1.

Definition 6.1. The *foundation rules* consist of the following five rules:
1. *No voting*,
2. *No collateral*,
3. *No oracle participation*,
4. *The right to dissolve*,
5. *Reporting*.

No voting. The Flare foundation is intended to help enact the collective will of the network. Its role in the governance process is to give opinions and technical analysis over proposals and help implement accepted proposals. The foundation may not use its token holdings to vote at any stage in the governance process.

No collateral. Foundation tokens will not be used to collateralize dependent tokens. Indeed, the supply and demand for collateral in dependent tokens, such as the FXRP system, would be distorted were the foundation to deploy its collateral into these applications.

No oracle participation. It is desirable that over time the foundation reduces in importance relative to other participants. The large initial token holding, combined with sensible financial management, should suffice, provided the network gains traction, to make sure that the Foundation is capable of discharging its mandate for a sufficiently long period such that at the end of the period the foundation is no longer needed. Hence, the foundation will not participate in the FTSO. In effect the FTSO award rate then becomes a measure of the speed with which the foundation's percentage ownership decreases relative to the collective ownership of other participants. Note this does not mean that the foundations net worth will necessarily reduce in parallel as the Spark token value may change and/or investments using the Spark token could yield positive returns.

The right to dissolve. The network has the right to dissolve the foundation if desired. This is achieved by enshrining in the bylaws of the foundation legal structure a stipulation that it must be disbanded if the network votes to do so. It is highly likely that upon such a vote it would be impractical or impossible for the foundation to dissolve immediately, however the foundation will submit a plan within 60 days to wind down operations, liquidate assets and burn its remaining token holdings. Any non Spark assets will be used to purchase Spark tokens which will then be burned.

Reporting. Transparency is key. The foundation will issue a report every six months detailing the following: sales of Spark tokens, employee remuneration, expenses and other use of funds, investments and grants.

953fbdd4ac2d5a2f1e413cbd378be0f3135010d81b4b643c6020e96ca49fc0c9

References

[Adl+18] J. Adler et al. "Astraea: A decentralized blockchain oracle". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1145–1152.

[Ben] M. Bentov Gabizon. "Cryptocurrencies without Proof of Work". In: ().

[But14] V. Buterin. *SchellingCoin: A Minimal-Trust Universal Data Feed*. 2014. URL: https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/ (visited on 06/09/2020).

[But15] V. Buterin. *The P + epsilon Attack*. 2015. URL: https://blog.ethereum.org/2015/01/28/p-epsilon-attack/ (visited on 06/09/2020).

[CM18] B. Chase and E. MacBrough. "Analysis of the XRP Ledger Consensus Protocol". In: *CoRR* abs/1802.07242 (2018). arXiv: 1802.07242. URL: http://arxiv.org/abs/1802.07242.

[Chi20] T. Chitra. "Competitive equilibria between staking and on-chain lending". In: *arXiv preprint arXiv:2001.00919* (2020).

[CC19] T. Chitra and U. Chitra. "Committee selection is more similar than you think: Evidence from avalanche and stellar". In: *arXiv preprint arXiv:1904.09839* (2019).

[EJN17] S. Ellis, A. Juels, and S. Nazarov. "Chainlink a decentralized oracle network". In: *Retrieved March* 11 (2017), p. 2018.

[Gil+17] Y. Gilad et al. "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 51–68.

[Gol07] O. Goldreich. *Foundations of cryptography: volume 1, basic tools*. Cambridge university press, 2007.

[ICS20] ICS. "Inter-blockchain Communication Protocol (IBC)". In: (2020). URL: https://github.com/cosmos/ics/tree/master/ibc.

[KDKP19] E. Kereiakes, M. D. M. Do Kwon, and N. Platias. "Terra Money: Stability and Adoption". In: (2019).

[Lok+19] M. Lokhava et al. "Fast and Secure Global Payments with Stellar". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. SOSP '19. Huntsville, Ontario, Canada: Association for Computing Machinery, 2019, 80–96. ISBN: 9781450368735. DOI: 10.1145/3341301.3359636. URL: https://doi.org/10.1145/3341301.3359636.

[Mac18] E. MacBrough. "Cobalt: BFT Governance in Open Networks". In: *CoRR* abs/1802.07240 (2018). arXiv: 1802.07240. URL: http://arxiv.org/abs/1802.07240.

[Maz15] D. Mazieres. "The stellar consensus protocol: A federated model for internet-level consensus". In: *Stellar Development Foundation* 32 (2015).

[Net20] F. Networks. "FXRP". In: (2020).

[Roc18] T. Rocket. *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies*. 2018.

[RU19] S. Rowan and N. Usher. "Flare Consensus Protocol". In: (2019).

[Sch80] T. C. Schelling. *The strategy of conflict*. Harvard university press, 1980.

[SYB+14] D. Schwartz, N. Youngs, A. Britto, et al. "The ripple protocol consensus algorithm". In: *Ripple Labs Inc White Paper* 5.8 (2014).

[Szt15] P. Sztorc. "Truthcoin". In: *peer-to-peer oracle system and prediction marketplace.* (2015).

[Woo+] G. Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: ().

Appendix A: Encoding the UNL Within the Smart Contract Layer

When a UNL definition is accessed in the smart contract layer by the claim-period finality system for use in computing consensus on the state of an external system, the UNL leveraged should match each node operator's UNL that they use for network-level consensus. This is challenging however because the smart contract layer requires uniform execution and resulting state between nodes that are consistent, meaning that there is no obvious way to encode a unique definition of the UNL within the smart contract layer that controls the finality of the claim-period system across different nodes that are consistent.

In the remainder of this section, we present a novel solution to enable the encoding of unique UNL definitions within the smart contract layer for use in controlling contracts such as the claim-period finality system. We then discuss the engineering considerations in using the approach.

Indexing Into an Array of UNL Definitions Using Custom Values for `coinbase` Per Node

In leader-based networks such as Bitcoin, a single node operator controls the definition of each block. Per block, this node operator is selected to be in control based on for example solving a proof-of-work challenge or being selected by a pseudorandom algorithm according to how much stake they own in a proof-of-stake network. Within such networks, the leader node's beneficiary address used for rewarding them for being in control of a block is denoted using a field encoded into the block called the `coinbase` address. The `coinbase` field is disused within the Flare Network because it is a leaderless network and also because consensus safety does not rely on economic incentives. However, the Flare Network leverages a smart contract layer that still provides access to the `coinbase` field, as the smart contract layer is built to be flexible in accommodating many different consensus topologies including leader-based, leaderless, economically incentivised and non-incentivised. Therefore, because the `coinbase` field is invariant to the network-level consensus of the Flare Network, each node operator can set the `coinbase` field locally to their own address. This means that during smart contract execution on each local node, the `coinbase` field returns the executing node's address whenever it is referenced in smart contract execution.

For the purpose of computing the finality of a claim-period, the unique values of `coinbase` are used to index into an array of UNL definitions within the smart contract layer. At runtime execution then, the code for computing the finality of a claim-period will reach a line that calls for indexing into the UNL array using the `coinbase` address. This address will be different on each different node operator, and when run on their machine will index into that node operator's UNL definition.

Engineering Considerations for the Approach

There are two engineering considerations to consider however when leveraging unique values of `coinbase` per each node operator: 1) transaction costs should be constant-valued for contracts that leverage the unique `coinbase` values and 2) the unique values of `coinbase` should only be leveraged in governance-approved contracts such as the state-connector finality system. The reasoning for the first engineering consideration is that transaction gas costs do not only depend on the number of computational steps, but also on the data values involved in the computations; that is, unique values of `coinbase` would incur different gas costs during computation execution. The reasoning for the second engineering consideration is that unique values of `coinbase` could be erroneously leveraged in bespoke smart contracts to intentionally cause inconsistencies between node state. The state-connector finality system leverages the unique values of `coinbase` in a specific and safe way that prevents inconsistencies between node state while achieving the UNL consensus functionality required by the state-connector finality system.