Overview of Polkadot Design

Jeff Brudges[1], Alfonso Cevallos[1], Peter Czaban[1]
Rob Habermeier[2], Syed Hosseini[1], Handan Kılınç Alper[1]
Ximin Luo[1], Fatemeh Shirazi[1], Alistair Stewart[1]
Gavin Wood[1,2]

[1] Web3 Foundation,
[2] Parity Technologies

January 9, 2020

Abstract

In this paper we describe the design components of the heterogenous multi-chain Polkadot. We review the properties that Polkadot aims to achieve and go over all the components that are designed to achieve those properties.

Contents

1 Introduction

Until recently, most of the applications on the web were controlled in a centralised fashion.(I'd start less abruptly plus Web is still centralized, "until recently" suggests otherwise, I'd suggest something like this: Internet was originally designed to be a collection of interconnected decentralized entities as it is evident from the design of its early protocols such as TCP/IP and SMTP. However, its commercialisation has led to the centralisation of most of the applications on the web). We refer not to the centralisation of physical infrastructure, which is often economically efficient, but rather to the logical centralisation of the ability to deploy, take down, alter, and change the rules of the application. Two good(Prominent) examples are Google and Facebook: while they do have servers all around the world, they are each ultimately controlled by a single legal entity.

Giving a central entity control over a system comes(poses) several risks for users and(as well as) application developers. T(For example t)he central entity can stop the service at any moment, can sell users' data depending on the jurisdictions they provide service to and manipulate how the service is working without user consent. With the need for security and the urge for more freedom and fairness, the era of decentralised web applications, where no single entity can control the system, is emerging. This decentralised web is comprised of many types of applications such as games, storage, decentralised exchanges, auctions, financial systems, etc.(we may add some references for each applications.)

One of the fundamental challenges of the(developing a) decentralised web(application) is keeping state(keeping state is too technical to jump at the intro, maybe a better alternative is: storing and communicating the current state of the application) state. T(That is because t)here is no central entity who follows the current state or can decide what the current valid state is if there is any doubt. Blockchains are one of the(-remove-) technologies who(which are proposed to) address this problem.

In order to make the decentralised web usable for end-users, these separate blockchains need to interact, otherwise, each will become isolated and not adopted by as many users, and the sum(replace: overall?) functionality of the system will be insufficient to compete with the centralised web. However, to interact with different chains, we need(one needs) to build an interoperability mechanisms, which introduces challenges. Many of these challenges arise from the ways that different blockchains have different technical characteristics.(Therefore, interoperability mechanisms are urgent for blockchain systems but such mechanisms bring along new challenges. Many of these challenges arise because of different technical infrastructures and functionalities that blockchains have.(I would argue that neither technical nor functional differences are not the main source of the problem but the lack of trust is the culprit as it actually argued in the following sentences, and the fact that why Cosmos is not very useful system to solve the interoperability problem. I'd suggest this: Many of these challenges which are missing in the centralized model arise because of the fundamental differences in the trust model between

the two paradigms. The simple authentication mechanisms such as OAuth [?] can not easily be adapted to the decentralized web where there is no central entity guarding the secret key. Furthermore, it is not straight forward for decentralized entities to establish trust between each other when they use different mechanism to ensure security of their respective systems)). For example, Bitcoin and Ethereum(refs) are proof-of-work (PoW) blockchains where preventing the control of the whole system by one entity(replace: decentralization?) is carried out by introducing challenges(maybe puzzle or something else is better than 'challenges' not to confuse with the challenges that we talked one sentence before) that need a large amount of processing(maybe computing?) power to solve. Their security relies on whether this amount of processing power exceeds the amount of processing power any single entity would possibly have. An alternative to PoW are(is a) proof-of-stake (PoS) systems(system), where the entities who(that) control the system have to lock a large amount of funds and would be punished if they misbehave in any way. The security relies on the fact that the total amount of locked funds exceeds the budget any adversary is able to invest in the attack. These technical differences present difficulties for one blockchain to be able to trust another.

(this paragraph looks like repetition of the previous one. Maybe it is better to combine them.)(it can be integrated into the previous paragraph). Currently, there have been hundreds of chains implemented in the wild for various functions and non-compatible underlying technology. Most one(-remove-) of these systems have different properties they aim to achieve and (a) certain security threshold that they set up. Hence, there is a need for a system that can connect these heterogeneous chains(enables these heterogeneous chains to communicate and interact based on a common security framework). Moreover, having multiple security setups causes a split in the security these systems can provide. Gathering all this security power and have a shared security system increases security substantially.(This part should clearly goes behind the hence part)

(we need to improve this paragraph)I(As a solution to the above mentioned interoperability problem), in this paper, we introduce Polkadot, a multi-chain system with shared security guarantees. (There is an abrupt jump here into technicality)Polkadot consists of the main chain(To provide such a shared security framework, Polkadot utilizes a central chain) called the *Relay Chain* that guarantees the security and(which communicates with) multiple heterogeneous parallel(independent) chains called *Parachains*(portmanteau of parallel chains)(.) The security goal of Polkadot is to be Byzantine fault-tolerant(This BFT comment is out of its place). Polkadot enables the parachains to communicate together and have shared security.(So we do not need to repeat this)

Furthermore, scalability is another challenge for blockchains to make them comparable with centralised services in functionality.(it may be better to talk about it when we mention challenges of interoperability)(I agree with Handan, specially that scalability is partially related to interoperability, for example, the reason that Ethereum tried to be generic computer is because it wanted to

have all Dapps lives inside so they can communicate efficiently which resulted in scalability problem. But we also can talk much more about scalability) Polkadot's hierarchical structure addresses this challenge.

2 Properties

Polkadot provides the following properties.

2.1 Utility

The state machine of each parachain in Polkadot provides a utility to the system participants. It is ensured by state machines that interprets the willingness of a participant to pay for a transaction to be included. Polkadot governance mechanism enables participants to decide what state machines should be included based on needs of participants.

2.2 Validity

Validity of a new state of a parachain is defined by its own state transition function that defines how a parachain can move from a state to another state. Validators who are responsible to (in)validate and produce blocks in the relay chain know these functions so that they can check whether a given state of a parachain is valid or not. In Polkadot, we have three levels of validity checks for each state of each parachain. The first-level check of a parachain state is executed by validators who are responsible for this parachain. These validators are called parachain validators and they shift from one parachain to another parachain periodically. The second level of check is executed by staked parties called fishermen which report to validators if they see any invalid state to receive some reward. And the last level of check is executed by randomly chosen validators after the block including the state is produced. These checks guarantee that it is almost impossible to have a finalized invalid state in the relay chain.

2.3 Finality

It is necessary to provide finality on a state of a parachain to provide a reliable communication in Polkadot so that parachains act by relying on the fact that the data provided by Polkadot related to other parachains will never change. Polkadot provides the finality property via relay chain which is based on a heterogeneous consensus mechanism: provable consensus and probable consensus. The relay chain provides provable consensus with GRANDPA (GHOST-based Recursive ANcestor Deriving Prefix Agreement) finality gadget. Validators are supposed to vote for a chain in GRANDPA which has valid blocks (e.g., having valid states of parachains). The probable consensus is based on the block production mechanism of the relay chain that is called BABE (Blind Assignment for Blockchain Extension). BABE is a proof-of-stake based block production mechanism that privately and evenly assigns validators to produce blocks.

2.4 Decentralization

In some of the most popular blockchain projects, there are concerns of centralization of power. This centralization can be caused by several factors, such as a) a block-producer selection method where some minorities are over-represented or receive disproportionate power, or b) an incentive mechanism that concentrates wealth or encourages cartel formation. As we explain in the corresponding sections, our validator selection method (based on nominated proof-of-stake) as well as our incentive layer are particularly designed to fight centralization, and we provide new and precise decentralization guarantees.

2.5 Availability

The availability of data in parachains is very critical because only available data can be validated. In Polkadot, we provide availability via erasure codes of data which are distributed to validators. Thus, when a validator needs, he/she can contact with some subset of validators to construct it. We guarantee that any state with some unavailable data cannot be finalized in the relay chain.

2.6 Messaging Reliability

Polkadot provides message reliability by regulating and ordering what messages have been sent and received by which parachain or parathread.

2.7 Reasonable Size and Bandwidth

In Polkadot, the block size is selected carefully with respect to scalability and security.

3 Preliminaries

(should we have this section before properties section? If prop section comes after this section we should remove some descriptions of entities in the properties section e.g., validator, fisherman)

3.1 Structural elements of Polkadot

Next, we review the structural elements and roles, shown in Figure 1, that are defined for the Polkadot protocols.

Parachains: Multiple heterogeneous blockchains run in parallel in Polkadot.

Relay Chain: This is a central chain in Polkadot. It keeps references to all canonical parachain blocks, and thus works as a single source of truth across parachains. It also keeps references of all cross-parachain messages.

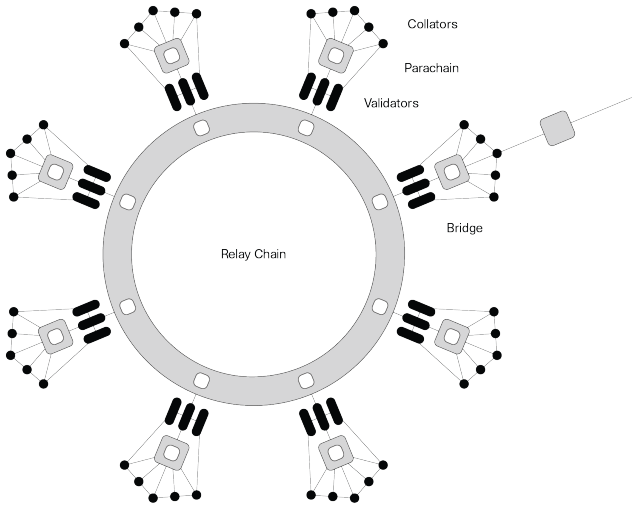

Figure 1: Polkadot network structur showing relay chain and parachain and including the roles collator, validator (Image credit: Ignasio Albero)

3.2 Roles

Validators: A fixed set of actors called validators are responsible to maintain the relay chain and keep its consensus, and by extension, maintain the consensus of all the parachains. In addition, at regular intervals validators are divided into random disjoint subsets that are each assigned to a particular parachain. Each of these validator subsets ensures the validity and availability of its parachain's blocks. Validators receive regular remunerations in DOTs, and are likewise heavily staked and prone to slashing in case of offenses.

Nominators: The role of validator is permissionless (like all roles in Polkadot), but the number of slots for validators is limited. Hence, validators must be elected by the community. In particular, any DOT holder may assume the role of a *nominator*, lock some stake, and publish a list of trusted validator candidates. Candidates with the most nominators' support get elected, and nominators share the economic rewards, or slashes, of their supported validators. Having nominators allows for an unlimited amount of actors and DOTs to participate in the security of Polkadot.

Collators: Every parachain has its own set of full nodes called collators. A collator produces new blocks and sends it to the parachain-assigned validators(why not just parachain validators?). It may only build on top of the parachain blocks that are referenced in the relay chain. Collators also process outgoing and incoming cross-chain messages.

Fishermen: Fishermen are staked actors that run background checks on the performance of validators. In particular, they raise reports in case that a validator-approved parachain block is unavailable or invalid. They are rewarded for correct reports, and slashed for bogus ones (which helps avoid report spamming).

3.3 Adversarial Model of Polkadot

Parties In Polkadot, honest parties follow the protocol while malicious ones can follow any arbitrary algorithm. We assume that less than one third of the validators are malicious. On the other hand, we do not have any limit on number of malicious fishermen.

Parachains:

Keys: We assume that malicious parties generate their keys with an arbitrary algorithm while honest ones always generate their keys securely.

Network and Communication: All validators have their own local clock and their clocks do not rely on any central clock. We assume that validators and collators are in a partially synchronous network. It means that a message sent by a validator or collator arrives at all parties in the network at most Δ units of time later where Δ is an unknown parameter. So, we assume an eventual delivery of a message in Polkadot. We also assume that collators and fishermen can connect to the relay chain network to submit their reports.

4 Overview and Components

Next, we summarise Polkadot functionality shortly for an overall picture and then continue to describe the individual components to show how to achieve that functionality.

Polkadot is a multi-chain system with shared security guarantees(we use this sentence in couple of places. I think we don't need to repeat it here. Also, in one of the places that we use this sentence, we need to explain shortly what the shared security means and implies). Polkadot consists of a main chain called the Relay Chain(is it "the relay chain" or "the Relay Chain". We should be consistent with this in whole paper.) and multiple parallel chains called Parachains. The relay chain is maintained by validators that are selected through the NPoS scheme 4.1 and is responsible for (add 'producing blocks of the Relay Chain 4.4.1 and') keeping the state of all the parachains 4.3. These validators need to vote on the consensus over all the parachains, see the consensus scheme 4.4.2 for more details. The security goal of Polkadot is to be Byzantine fault tolerant when the participants are rational (see 4.5 for more detail on incentives and economics). In addition, the validators are assigned to parachains by dividing the validators set into random disjoint subsets that are each assigned

to a particular parachain.(we repeat it one sentence later. We can remove it here.) For parachains, there are additional actors called collators and fishermen that are responsible for parachain block production 4.2.1 and reporting invalid parachain blocks respectively. The parachain validators assigned to each parachain validate each parachain block and are responsible to keep it available. For this purpose, they create erasure coded pieces from the parachain blocks and distribute them to all the remaining validators(do we need to give this detail here?), see 4.2.2. Moreover, another feature of Polkadot is enabling interchain messaging among parachains, see 4.2.3 for more details. Furthermore, Polkadot has a decentralised governance scheme 4.6 that can change any Polkadot design decisions and parameterisation.

4.1 Nominated proof-of-stake and validator selection

Polkadot will use Nominated Proof-of-Stake (NPoS), own version of proof-of-stake (PoS). Consensus protocols with deterministic finality, such as ours(maybe it is better to use 'in Polkadot' than saying 'ours'), require a set of registered validators of bounded size. Polkadot will maintain a number n_{val} of validators, in the order of hundreds or thousands. This number will be ultimately decided by governance, and is intended to grow along with the number of parachains; yet, it will be independent of the number of users in the network, to ensure scalability. However, NPoS allows for an unlimited number of DOT holders to participate as *nominators*, thus maintaining high levels of security by putting more value at stake. As such, NPoS is not only much more *efficient* than proof-of-work (PoW), but also considerably more *secure* than standard PoS. Furthermore, we introduce new guarantees on *decentralization* hitherto unmatched by any other PoS-based blockchain.

A new set of validators is selected at the beginning of every era (a period during roughly one day), to serve for that era, according to the nominators' preferences. More precisely, any DOT holder may choose to become a validator candidate or a nominator. Each candidate indicates the amount of stake he is willing to stake and his desired commission fee for operational costs. In turn, each nominator locks some stake and publishes a list (of any size) of the candidates that she trusts. Then, a public protocol (discussed below) takes these lists as input and selects the candidates with the most backing to serve as validators for the next era.

Nominators share the rewards, or eventual slashings, with the validators they selected, on a per-staked-DOT basis (see more details on the Economics section(add reference to this section)). Nominators are thus economically incentivised to act as watchdogs for the system, and they should base their preferences on parameters such as validators' stakes, commission fees, past performance, and security practices. Our scheme allows for the system to select validators with massive amounts of aggregate stake - much higher than any single party's DOT holdings - and thus helps turn the validator selection process into a meritocracy rather than a plutocracy. In fact, at any given moment we expect there to be a considerable fraction of all the DOT supply be staked in NPoS. This makes it very difficult for an adversarial entity to get validators elected (as it either needs a large amount of DOTs or high enough reputation to get the required nominators' backing) as well as very costly to attack the system (as it is liable to lose all of its stake, stake backing, and reputation).

Polkadot elects validators via a decentralized protocol with carefully selected, simple and publicly known rules, taking the nominators' lists of trusted candidates as input. Formally, the protocol solves a multi-winner election problem based on approval ballots, where nominators have voting power proportional to their stake, and where the goals are *decentralization* and *security*.

Decentralization. In the late 19th century, Swedish mathematician Edvard Phragmén proposed a method(refs) for electing members to his countrys parlia-

ment. He noticed that the election methods at the time tended to give all the seats to the most popular political party; in contrast, his new method ensured that the number of seats assigned to each party were proportional to the votes given to them, so it gave more representation to minorities.

In the literature of computational social choice, Phragmén's method has been recently revisited and shown to achieve a property called *proportional justified representation* (PJR). In the context of NPoS, this property turns out to be ideal to guarantee decentralization. It ensures that the set of selected validators represents as many nominator minorities as possible, proportional to their stake, and that no minority is under-represented. (These minorities may be defined by common political views, geographical location, etc.)

Our validator selection protocol will observe the PJR property. Formally, this means that if each nominator $n \in \mathcal{N}$ has stake $stake_n$ and backs a subset $\mathcal{C}_n \subseteq \mathcal{C}$ of candidates,[1] the protocol will select a set $\mathcal{V} \subseteq \mathcal{C}$ of n_{val} validators with the following property(it may be better to say in words why the following mathematical definition represents PJR property.):
if there is a group of nominators $\mathcal{N}' \subseteq \mathcal{N}$ and some $1 \leq t \leq n_{val}$ such that

$$|\cap_{n\in\mathcal{N}'} \mathcal{C}_n| \geq t \quad \text{and} \quad \frac{1}{t}\sum_{n\in\mathcal{N}'} stake_n \geq \frac{1}{n}\sum_{n\in\mathcal{N}} stake_n,$$

then $|\mathcal{V} \cap (\cup_{n\in\mathcal{N}'}\mathcal{C}_n)| \geq t$.

Security. If a nominator gets two or more of its trusted candidates elected as validators, the protocol must also decide how to split her stake and assign these fractions to them. In turn, these assignations define the total stake backing that each validator receives. Our objective is to make these validators' backings as high and as balanced as possible. In particular, we focus on maximizing the *minimum validator backing*. Intuitively, the minimum backing corresponds to a lower bound on the cost for an adversary to gain control over one validator, as well as a lower bound on the potential slashable amount for a misconduct.

Formally, if each nominator $n \in \mathcal{N}$ has $stake_n$ and backs a candidate subset $\mathcal{C}_n \subseteq \mathcal{C}$, the protocol must select not only a set $\mathcal{V} \subseteq \mathcal{C}$ of n_{val} validators with the PJR property, but also a distribution of each nominator's stake among the elected validators that she backs(verb of the sentence is missing), i.e. a function $f : \mathcal{N} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ so that

$$\sum_{v\in\mathcal{V}\cap\mathcal{C}_n} f(n,v) = stake_n \quad \text{for each nominator } n \in \mathcal{N},$$

and the objective is then

$$\max_{\text{solutions } (\mathcal{V},f)} \min_{v\in\mathcal{V}} backing_f(v), \quad \text{where } backing_f(v) := \sum_{n\in\mathcal{N}:\ v\in\mathcal{C}_n} f(n,v).$$

[1] For ease of presentation, we consider here that candidates have no stake. This can be achieved by representing a candidate's stake as an additional nominator that exclusively approves that candidate.

The problem defined by this objective is called *maximin support*(refs) in the literature, and is known to be NP-hard. We have developed for it several efficient algorithms(either we should write the algorithms here or give reference to these algorithms) which offer theoretical guarantees (constant-factor approximations), and which also scale well and perform well on our testnet.

4.2 Parachains

4.2.1 Block Production

In outline, a collator produces a parachain block, sends it to the parachain validators, who sign its header as valid, and the header with enough signatures is placed on the relay chain. At this point, the parachain block is as canonical as the relay chain block its header appeared in. If that is in the best chain according to BABE (See Section 4.4.1), so is the parachain block and when that is finalised, so is the parachain block.

Because the parachain validators switch parachains frequently, they are stateless clients of the parachain. Thus we distinguish between the parachain block B, which is enough for full nodes of the parachain such as collators to update the parachain state, and the *Proof of Validity(PoV)* block B_{PoV}, which a validator who does not have the parachain state can verify.

Any validator should be able to verify B_{PoV} given the relay chain state using the parachain's *state transition validation function* (STVF), the Wasm code for which is also on the relay chain. The STVF takes as an input the PoV block,

- the header of the last parachain block from this parachain and,
- the message roots of all parachain blocks that have sent messages to this parachain since the last parachain block (See Section 4.2.3 for details such as the range needed)

It should output the validity of the block and its outgoing messages. The PoV block should contain any outgoing messages and for most parachains, it will also contain the parachain block. If it does contain the parachain block, the parachain validators should gossip the block to the parachain network, as a back up to the collator itself doing so. If it does not, then the parachain will need its own scheme to make sure that the parachain block data is available (See Section 4.2.2).

The PoV block will frequently be the parachain block, its outgoing messages, its header and a light client proof, i.e. Merkle proofs that give all elements of the input and output state that are used or modified by the state transition from the input and output state roots. However, the inclusion of the light client proof is optional and other types of proof are possible. For example, a parachain may use succinct or zero knowledge proofs of validity that can give more scalability than light client proofs. Or a permissioned chain could take signatures from authorities in place of much of the proof.

To aid in censorship resistance, a parachain may want to use proof of work or proof of stake to selected collators(who are the selected collators). This can be implemented in the STVF and need not be a part of the Polkadot protocol. So for proof of work, the STVF would check that the hash of the block is sufficiently small. However, for speed, it would be useful to ensure that most relay chain blocks can include a parachain block. For PoW, this would necessitate it being probable that multiple collators are allowed to produce a block. As such we will

still need a tie-breaker for the parachain validators to coordinate on validating the same parachain block first. This may be the golden ticked scheme of [**?**].

(In general, when I read this section, I felt like some parts are not certain because we give options e.g. we could do this, parachain should do this etc. What if they don't do that? How do we enforce them? I think we should be more direct.)

4.2.2 Validity and Availability

Once a block is created it is important that the parachain blob consisting of the parachain block, the PoV block and set of outgoing messages from the parachain is available for a while. The naive solution for this would be broadcasting/gossip the parachain blobs to all relay chain nodes, which is not a feasible option because the are are many parachains and the PoV blocks may be big. We want to find an efficient solution to ensure PoV blocks from any recently created parachain blocks are available.

For a single chain, such as bitcoin, as long as 51% of hash power is honest, not making block data available ensures that an honest miner builds on it so it will not be in the final chain. However, parachain consensus for us is determined by relay chain consensus. A parachain block is canonical when it's header is in the relay chain. We have no guarantees that anyone other than the collator and parachain validators have seen the PoV block. If these collude then the rest of the parachain network need not have it and then most collators cannot build a new block and its invalidity may not be discovered. We would like the consensus participants, here the validators, to collectively guarantee the availability rather than relying on a few nodes.

To this end we designed an availability scheme that uses erasure coding [](ref) to distribute the parachain PoV to all validators. When any misbehaviour, unavailability particularly in relation to invalidity, is detected the PoV can be reconstructed from the distributed erasure coded pieces.

If a block is available, then full nodes of the parachain, and any light client that has the PoV block can check its validity. We have three-level of validity check in Polkadot. The first validity check of a blob is executed by the corresponding parachain validators. If they verify the blob then they sign and distribute the erasure codes of the blob to each validator. We rely on nodes acting as *fishermen* to report invalidity as second level of validity check. They would need to back any claim with their own stake in DOTs. We would assume that most collators would be fishermen, as they have a stake in continued validity of the chain and are already running full nodes, so all they need is stake in DOTs. The third level of validity check is executed by a few randomly and privately assigned validators just after a block containing the blob header is created in the relay chain. We determine the number of validator in the third check considering amount of fishermen reports and unavailability reports by collators. If any invalid blob is detected in the relay chain, the validators who signed for its validity are slashed.

The security of our availability and validity scheme based on the security

of GRANDPA finality gadget (See Section 4.4.2) and the good randomness generated in each BABE epoch (See Section 4.4.1). Please see [**?**] for more details about the availability and validity scheme.

4.2.3 Cross Chain Messaging Protocol (XCMP)

XCMP is the protocol that parachains use to send messages to each other. It aims to guarantee that messages arrive quickly, that messages from one parachain arrive to another in order, and that messages that arrive were indeed sent in the finalized history of the chain.

As a result of these properties, we will need to require that a parachain accepts all incoming messages. The way relay chain blocks include headers of parachain blocks gives a synchronous notion of time for parachain blocks, just by relay chain block numbers. Additionally it allows us to authenticate messages as being sent in the history given by the relay chain i.e. it is impossible that one parachain sends a message, then reorgs so that that a message was not sent, but has been received. This holds even though the system may not have reached finality over whether the message was sent, because any relay chain provides a consistent history. To this end, parachain headers contain a message root of outgoing messages, as well as a bitfield indicating which other parachains were sent messages in this block. The message root is the root of a Merkle tree of message roots for each parachain that messages are sent to, and these in turn are the root of a Merkle tree of the head of a hash chain that has hashes of the actual messages. This allows many messages from this source to one particular destination to be verified at once from the message root. However the messages themselves are passed, they should also be sent with the Merkle proof that allows nodes of the receiving parachain to authenticate that they were send by a particular parachain block whose header was in a particular relay chain block. Parachains act on incoming messages in order. It always acts on messages sent by parachain blocks whose header was in earlier relay chain blocks first. When several such parachains have a header in the relay chain block, they are acted on in some predetermined order, either sequentially in order of increasing parachain id, or some shuffled version of this. It acts on all messages sent by one parachain in one parachain block or none of them. A parachain header contains a reference to the relay chain block, by block hash or block number, that the parachain block receives all messages up to and to the parachain id. This will likely be constrained to advance in some manner. To produce a parachain block on parachain P which builds on a particular relay chain block B, a collator would need to look at which parachain headers were built between the relay chain block that the last parachain block of this chain built on, and for each of those that indicated that they sent messages to P, it needs the corresponding message data. Thus it can construct a PoV block so that the (STVF) can validate that all such messages were acted on. Since a parachain must accept all messages that are sent to it, we implement a method for parachains to make it illegal for another parachain to send it any messages that can be used in the case of spam occurring. If any node is connected to

both parachain networks, then it should forward messages and their proofs from one parachain to the other, when the parachaim header is included in a relay chain block. The relay chain should at least act as a back up. The receiving parachain validators are connected to the parachain network and if they do not receive messages on it, then they can ask for them from the parachain validators of the sending chain at the time the message was sent.

4.3 Relay Chain State Machine

Formally, Polkadot is a replicated sharded state machine where shards are the parachains and the Polkadot relay chain is part of the protocol ensuring global consensus among all the parachains. Therefore, the Polkadot relay chain protocol, can itself be considered as a replicated state machine on its own. In this sense, this section describes the relay chain protocol by specifying the state machine governing the relay chain. To that end, we describe the relay chain state and the detail of state transition governed by transactions grouped in the relay chain blocks.

State Polkadot relay chain state is represented similarly to the one of the Ethereum. In this sense, the state is represented through the use of an *associative array* data structure composed by a collection of $(key, value)$ pairs where each key is unique. There is no assumption on the format of the key or the value stored under it besides the fact that they both the key and the value need to be finite byte arrays.

A *Merkle radix-16 tree* keeps the Merkle hashes corresponding to the $(key, value)$ pairs stored in the relay chain state. They enable the identification of the current state using its root hash and provide efficient proof of inclusion of a specific pair.

To keep the state size in control, the relay chain state is solely used to facilitate the relay chain operations such as staking and identifying validators. The Merkle Radix tree is not supposed to store any information regarding the internal operation of the parachains.

State transition Like any transaction-based transition system, Polkadot state changes via an executing ordered set of instructions. These instructions, traditionally known as transactions, are referred as extrinsics in Polkadot Jargon. They cover any data provided from "outside" of the machine's state which can affect state transition. Polkadot relay chain is divided into two major components, namely the "Runtime" and the "Runtime environment". The execution logic of the state-transition function is mainly encapsulated in the Runtime while all other generic operations, commonly shared among modern blockchain-based replicated state machines, are embedded into the Runtime environment. In particular, the latter is in charge of network communication, block production and consensus engines.

Runtime functions are compiled into a Web assembly module and are stored as part of the state. The Runtime environment communicates the extrinsics with the Runtime and interacts with it to execute the state transition. In this way, the state transition logic itself can be upgraded as a part of the state transition.

Extrinsics Extrinsics are the input data supplied to the Polkadot Relay chain(relay or Relay?) state machine to transition to new states. Extrinsics

are needed to be stored into blocks of the relay chain in order to achieve consensus among the state machine replica. Extrinsics are divided into two broad categories namely Transactions and Inherents.

Transactions are signed and are gossiped around on the network between nodes. In contrast, Inherents are not signed and are not gossiped individually but rather only when they are included in a block. The Inherents in a block are assumed to be valid if a supermajority of validators assumes so. The Timestamp is an example of inherent extrinsics which must be included in each Polkadot Relay chain block.

Transactions on the relay chain are mainly concerned with the operation of the relay chain and Polkadot protocol as a whole, such as `set_code`, `transfer`, `bond`, `validate`, `nominate`, `vote`.

Relay chain block producers listen to all transaction network messages. Upon receiving a transaction message, the transaction(s) are validated by the Runtime. The valid transactions then are arranged in a queue based on their priority and dependency and are considered for inclusion in future blocks accordingly.

Block format A typical relay chain block consists of a header and a body. The body simply consists of a list of extrinsics.

The header contains the *hash of parent block*, *block number*, the *root of the state tree*, the *root of the Merkle tree* resulting from arranging the extrinsics in such a tree and the *digest*. The digest stores auxiliary information from the consensus engines which are required to validate the block and its origin as well as information helping light clients to validate the block without having access to the state storage.

Consensus Polkadot Consensus protocol, similar to other replicated state machines, is responsible to guarantee liveness and safety. Liveness is the property that ensures that the state machine continues to collate and execute transactions. Safety, on the other hand, is the canonicalization mechanism, or the means by which parties agree upon one of a number of possible, valid, histories and that ensures honest nodes do not agree on two conflicting states. This is also known as finality in the context of blockchains. These two properties ensure that valid transactions will eventually be included in the state transition history and finalized.

Unlike the consensus architecture of many former blockchains, which builds a scalability bottleneck by tying the liveness and safety logic together, Polkadot architecture decouples the safety from the state-transition mechanism, a hybrid consensus model that separates block production from finality on those blocks (See Section 4.4).

The block production layer of the consensus is to be fast and probabilistically safe. Block production randomly and secretly assigns the production of each block to a certain block producer to mitigate denial of service and eclipse attacks(Do we need to talk about it here? I think it makes more sense to say it in the beginning of 4.4.1). The detail of Polkadot block production consensus

sub-protocol is described in Section 4.4.1.

Simultaneously, the Polkadot execute a BFT-based consensus finality protocol which is tasked to observe (possibly) several incompatible (but likely valid) state transitions produced by the first layer and democratically finalize a canonical version as the valid history of the relay chain. Besides scalability, the choice of relay chain consensus protocol provides efficient absolute (in contrast to probabilistic) finality. That is, once a block is finalized, the canonical chain will always contain that block in the future. The finality subprotocol of Polkadot consensus is explained in Section 4.4.2

Block Building (I think we should integrate Block Building with BABE) In this section, we present a summary of various steps of relay chain operation. Validators of the relay-chain are nominated and get selected to produce and finalize a certain number of blocks known as an epoch. At the beginning of each epoch, each validator secretly knows the slot of the time which it is permitted and is supposed to produce a block.

Meanwhile, Transactions(why capital T) ranging from validated parachain block hash, transfer, staking, nomination or slashing for protocol violation are submitted to the relay chain validators. The Validators(why capital V) examine the validity of the transactions and store them in their transaction pool. Once the time slot during which the validator is expected to produce the block has arrived, the validator estimates the block which most likely represents the state which is going to be finalized by the finality protocol and set it as the current state of the relay chain. Then it selects valid transactions with fulfilled prerequisite, executes them and updates the state accordingly. It executes and collates as much as the block capacity allows, includes the final stage of the chain, signs and publishes the block.

Upon receiving the new block, other validators examine that the producer's adherence to the protocol as well as the validity of included transactions and store the block in a tree which represents all possible candidates for a final state transition of the relay chain.

Simultaneously, the set of validators votes on various branches of the transition tree and prunes branches which conflict with the version agreed upon by the supermajority of the validators.

4.4 Consensus

In this section, we explain the hybrid consensus protocol of Polkadot which consists of BABE: a block production mechanism of the relay chain that provides probabilistic finality and GRANDPA which provides provable, deterministic finality and works independently from BABE. Informally, probabilistic finality implies that after certain time passed, a block in the relay chain will be finalized with very high probability (close to 1) and deterministic finality implies a finalized block stays final forever. Provable finality means that furthermore, we can prove to parties not actively involved in the consensus that a block is final.

We need provable finality to makes bridges to chains outside Polkadot easier and for that we need a Byzantine agreement tyoe of consensus. But the validity and availability scheme also may also require us to revert blocks, which would mean that getting Byzantine agreement on every block, as in Tendermint or Algorand, would not be suitable. However, this should happen rarely as a lot of stake will be slashed when we do this. As a result, we want a scheme that generates blocks and optimistically executes them, but may take some time to finalise them. The way XCMP works, mean that message passing speed is constrained by block time, but not by finality time so if we delay finality, but in the end do not revert, then message passing is fast. Even the speed at whoch we finalise blocks may be variable - if we do not recieve reports of invalidity and unavailability then we can finalise fast, but if we do then we may need to delay finality while we execute more involved checks.

As a result of these requirements, we have chosen to seperate the mechanisms for block production and finalising blocks as much as possible. In the next twos ections, we describe the protocols BABE and GRANDPA that do each of these.

4.4.1 Blind Assignment for Blockchain Extension (BABE)

In Polkadot, we produce relay chain blocks using our Blind Assignment for Blockchain Extension protocol, abbreviated BABE. BABE assigns validators randomly to block production slots using the randomness generated with blocks. These assignments are completely private until the assigned validators produce their blocks. Therefore, we use "Blind Assignment" in the protocol name. BABE is similar to Ouroboros Praos [**?**] with some significant differences in the chain selection rule and timing assumptions.

In BABE, we may have slots without any assignment which we call empty slot. In order to fill the empty slots, we have secondary block production mechanism based on Aura [**?**] that assigns validators to slots publicly. We note that these blocks do not contribute the security of BABE since the best chain selection algorithm works as if Aura blocks do not exist and the randomness generation do not consider randomness in the Aura blocks.

Here, we only describe BABE together with its security properties since Aura blocks is not the part of the security of BABE.

BABE: BABE [**?**] consists of *epochs* $(e_1, e_2, ...)$ and each epoch consists of a number of sequential block production slots $(e_i = \{sl_1^i, sl_2^i, \ldots, sl_t^i\})$ up to the bound R. Each validator knows in which slots that he is supposed to produce a block at the beginning of every epoch. When the time for their slot comes, they produce the block by proving that they are assigned this slot.

The blind assignment is based on the cryptographic primitive called verifiable random function (VRF) [**?**] (See Section 4.7.2). A validator in an epoch e_m where $m > 2$ does the following to learn if he is eligible to produce a block in slot sl_i^m: He retrieves the randomness r_{m-2} generated two epoch before (e_{m-2}). Then, he runs the VRF with his secret key and the input: randomness r_{m-2} and the slot number sl_i^m. Validators in e_1 and e_2 use the randomness defined in the genesis block when they run the VRF with their secret key for the slots belonging e_1 and e_2. If the output of the VRF is less than the threshold τ, then the validator is the slot leader meaning that he is eligible to produce a block for this slot. We select τ considering the network delay [**?**]. When a validator produces a block, he adds the output of the VRF and its proof to the block which shows that his VRF output is less than τ in order to convince other validators that he has a right to produce a block in the corresponding slot. The validators always generate their blocks on top of the best chain. The best chain selection rule in BABE says that ignore the Aura blocks and select the longest chain that includes the last finalized GRANDPA block. See Section 4.4.2 for the details how blocks are finalized in GRANDPA.

The randomness of an epoch e_m is generated by using the BABE blocks of the best chain that belongs to that epoch: Concatenate all VRF values in BABE blocks that belongs to e_m (let us assume the concatenation is ρ). Then, compute the randomness in epoch e_{m+1} as $r_m = H(m||\rho)$ where H is a hash function.

Validators run periodically the relative time algorithm described below to learn the at what time a slot starts according to their local clocks.

- Relative Time Protocol: The elected validators for a slot need to know when actually the right time to produce a block for the consistency and the security of the block production mechanism. For this purpose, they run the relative time protocol which let them to know when approximately a slot starts even if some clock drifts exist in their local clock. We note that in BABE, we do not rely on any centralized clock adjustment protocols such as Network Time Protocol [**?**]. Therefore, we design the relative time protocol. This protocol depends on the arrival time of blocks according to local clocks of validators. Thus, it is named as "Relative Time". Please see [**?**] for the formal security model of synchronization in blockchains and for further details about the relative time protocol.

In BABE, we assume that after the genesis block is released, elected validators of the first epoch store the arrival time of the genesis block with respect to their local clock. Then, they mark the start time of the first slot and increment the slot number every T seconds. After this point, they periodically run the

relative algorithm not to lose the synchronization with others because of their local clock drifts. In addition to this, a validator joins after the genesis block runs the relative time algorithm to be synchronized with the other validators.

In every sync-epochs (different than epochs in BABE), validators update their clock according to the result of the relative time protocol and use the new clock until the next sync-epoch. The first sync-epoch ε_1 starts just after the genesis block is released. The other sync-epochs ε_i start when the slot number of the last (probabilistically) finalized block is $\bar{sl}_\epsilon$ which is the smallest slot number such that $\bar{sl}_\varepsilon - \bar{sl}_{\varepsilon-1} \geq s_{cd}$ where $\bar{sl}_{\varepsilon-1}$ is the slot number of the last (probabilistically) finalized block in sync-epoch $\varepsilon - 1$. Here, s_{cd} is the parameter of the chain density (CD) property which will be defined according the chain growth. In more detail, each validator stores the arrival time t_j of blocks together with the slot number sl'_j in the block during a sync-epoch. At the end of a sync-epoch, the validator retrieves the arrival time of probabilistically finalized blocks generated during the sync-epoch and computes some candidate start times of the first slot sl of the next sync-epoch i.e, given that $a_j = T(sl - sl'_j)$, $\mathcal{C}_T = \{t_j + a_j\}$. The times in $\mathcal{C}_T$ are considered as candidates. In order to choose one candidate, the validator then sorts the list of candidates $\mathcal{C}_T$ and outputs the median of the sorted list as a start time of the sl. An example execution of the relative time protocol in the first sync-epoch is in Figure 2.

- Security Overview of BABE: Garay et al. [**?**] define the properties defined below in order to obtain a secure blockchain protocol. Informally, we can describe these properties as follows:

- *Common Prefix Property (CP):* It ensures that the blocks which are k-blocks before the last block of an honest validator's blockchain cannot be changed. We call all unchangeable blocks *finalized* blocks. BABE satisfies CP property thanks to the honest super majority since malicious validators are selected for a slot probabilistically much less than the honest validators. It means that malicious validators does not have enough source to construct another chain which does not include one of the finalized blocks.

- *Chain Quality (CQ):* It ensures sufficient honest block contribution to any best chain owned by an honest party. We guarantee even in the worst case where a network delay is maximum that there will be at least one honest block in the best chain during an epoch so that the randomness cannot be biased.

- *Chain Growth (CG):* It guarantees a minimum growth between slots. Thanks to super majority of honest validators, malicious validators cannot prevent the growth of the best chain.

- *Chain Density (CD):* It ensures that in a sufficiently long portion of the best chain more than half of the blocks produced by honest validators. CQ and CD properties implies this property [**?**].

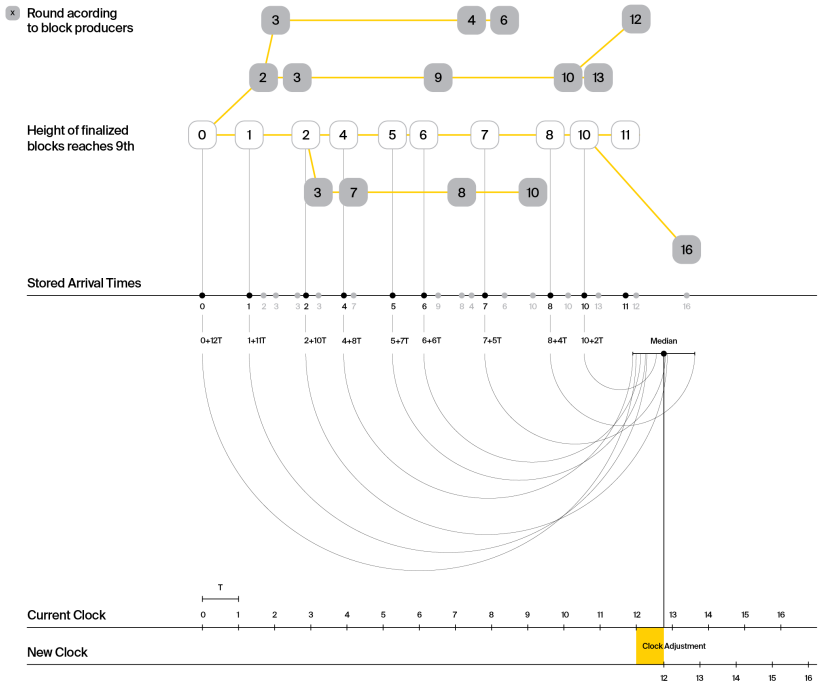

Figure 2: An example execution of the relative time protocol in the first epoch where $s_{cd} = 9$

Please see [**?**] for further details about BABE and its security analysis.

4.4.2 GRANDPA

As mentioned above, we want a finalisation mechanism that is flexible and seperated from block production, which is achieved by GRANDPA. The only modofication to BABE required for it to work with GRANDPA is to change the fork-choice rule: instead of building on the longest chain, a validitor producing a block should build on the longest chain including all blocks that it sees as finalised. GRANDPA can work with many different block production mechanisms and it should be possible to switch out BABE with another.

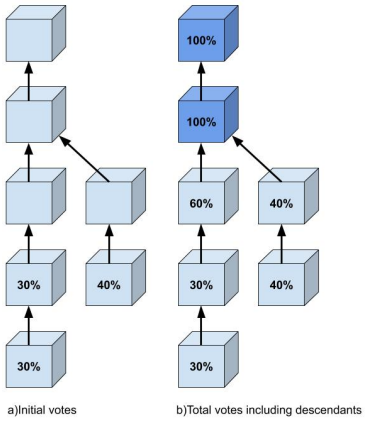

Figure 3: GRANDPA votes and how they are aggregated.

4.5 Economics and Incentive Layer

Polkadot will have a native token called DOT. Its various functions are described in this section.

4.5.1 NPoS rewards and inflation

In this section we consider NPoS rewards, i.e. payments to validators and nominators coming from the minting of new tokens, in normal circumstances(what is the difference btw normal and not normal circumstances). In particular we do not consider rewards from transaction fees, slashings, nor rewards to fishermen and other reporters of offences; these will be analyzed in further sections. NPoS rewards are the main driver of inflation in the system. This is because they are the only mechanism that mints new tokens, and because Treasury expenditure is designed to match the net token burning caused by the transaction fees and slashings mechanisms (see the section on Treasury(reference to the section)). Thus, it is natural to also study our inflation model in this section.

Recall that, according to the NPoS protocol, both validators and nominators stake DOTs. They get paid roughly proportional to their stake, but can be slashed up to 100% in case of a misconduct. We remark that, even though they are actively engaged for only one era at a time (where one era lasts approximately one day), they can be engaged for an unlimited number of consecutive eras. Furthermore, for added security, their stake remains locked for several weeks after their last active era, to keep them liable to slashing in case an offense is detected late.

Staking rate, interest rate, inflation rate. Let the "staking rate" be the total amount of DOTs currently staked by validators and nominators, divided by the current total DOT supply. Their average interest rate will be a function of the staking rate:[2] if the staking rate dips below a certain target value (be to selected by governance), the average interest rate will increase, thus incentivizing more participation in NPoS, and vice versa. For instance, a target value of 50% could be selected for the staking rate as a middle ground between security and liquidity. If the average interest rate is set to 20% p.a. at that staking rate value, then we can expect to maintain an inflation rate (due to NPoS rewards) of $50\% \times 20\% = 10\%$ on average. Hence, by controlling the staking rate we also control the inflation rate.

Rewards across validator slots. Once the total payout for the current era is computed as a function of the staking rate according to the paragraph above, we need to establish how it is distributed. Recall that the NPoS validator selection protocol (see Section 4.1) partitions all validators' and nominators' stake

[2] As we explain below, validators and nominators are not all paid exactly the same interest rate at a given time; in particular, validators are generally paid more than nominators. The average interest rate is simply the total payout divided by the total amount staked.

into the different validator slots(is it the same as BABE slots?) (where a nominator's stake can be fractionally split among several slots), so as to make the slots' stake backings as high and as uniformly distributed as possible, hence ensuring security and decentralization. A further incentive mechanism we set, to ensure decentralization over time, is *paying validator slots equally for equal work, NOT proportional to their backings*. As a consequence, nominators inside a validator slot with more backing will be paid less per staked DOT than nominators backing a less popular validator. Hence, they will be incentivized to change their preferences over time, converging to the ideal case where all slot backings are equal.

In particular, we devise a point system where validators accumulate points for payable actions performed, and at the end of each era validator slots are rewarded proportional to their points. This ensures that validators are always incentivized to stay efficient and highly responsive. Examples of payable actions are: producing a block in BABE, validating a parachain block, etc.

Rewards within a validator slot. As a nominator's stake can be split among several slots, that nominator's payout in an era corresponds to the sum of her payouts relative to each of these slots. Within a validator slot, the payment is as follows: First, the validator is paid his commission fee, which is an amount entirely up to him to decide and publicly announced before nominators select their preferences for the era. This fee is intended to cover the validator's operational costs. Then, the remainder is shared among all parties (both validator and nominators) proportional to their stake within the slot. In other words, the validator is paid twice: once as a non-stake validator with a fixed commission fee, and once as staked nominator. Notice that a higher commission fee means a higher total payout for the validator and lower payouts to his nominators, but since this fee is published in advance, nominators will generally prefer to back validators with lower fees. We thus let the market regulate itself. Validators who have built a strong reputation of reliability and performance may be able to charge a higher commission fee, which is fair.

Remark. As our decision to reward validator slots equally for equal actions independently of their stake is likely to raise eyebrows, we provide further arguments for it. First, notice that since validators are paid well and the number of slots is limited, candidates have an incentive to ensure a high enough backing to get elected, and we expect to have enough competition so that the minimum required backing is very high. Furthermore, even if the payouts are roughly the same across slots, within a slot each nominator and validator gets paid proportional to their stake, hence there is always an individual incentive to increase one's own stake. Finally, if a validator considers that his slot has more backing than necessary, he can either increase his commission fee (which has the effect of increasing his own reward at the risk of pushing some nominators away), or he can create a new candidate node and split his stake and nominators' support between all his nodes. On this last point, we welcome individuals and companies

with multiple validator nodes (and even aim to make their logistics simpler), so long as they disclose such correlations to nominators.

4.5.2 Relay-chain transaction fees and block limits

Limits on resource usage. Every relay-chain block can process a limited number of transactions, in order to ensure it can be processed efficiently even on less powerful nodes, and avoid delays in block production.

4.6 Governance

Relay Chain Council Referenda TODO

Parachain Allocation We use auctions to have a fair and transparent parachain allocation procedure. Since implementing seal-bid auctions are difficult and to avoid bid sniping we adopt an Candle auction [**?**] with a retroactively determined close as follows.

Once the auction has started within a fixed window bidders can post bids for the auction. Bids go into the block as transactions. Bidders are allowed to submit multiple bids. Bids that a bidder is submitting either should intersect with all winning bids by same bidder or be contiguous with winning bids by the same bidder. If an incoming bid is not changing the winners it is ignored.

For 4 lease periods we have 10 possible ranges. We store the winner for each one of the 10 ranges in a designated data structure. We need to make sure that a new bid does not have a gap with a winning bid on another interval from the same bidder. This means that once a bidder has won a bid for a given range, say for example, lease periods 1-2 then he cannot bid on lease period 4 unless someone overbids him for lease periods 1-2.

For any incoming bid the new winner is calculated by choosing the combination of bids where the average deposit for overall all 4 lease periods is most. Once a bid is added to the block, the amount of their bid gets reserved.

Once a fixed number of blocks have been produced for the auction a random numbers decides which one of the previous blocks was the closing block and we return the winners and their corresponding ranges for that closing block. The reserved funds of losers are going to be released once the ending time of the auction is determined and the final winners are decided.

For example, let us assume we have three bidders that want to submit bids for a parachain slot. Bidder B_1 submits the bid (1-4,75 DOT), bidder B_2 submits (3-4, 90 DOTs), and bidder B_3 submits (1-2, 30). In this example bidder B_1 wins because if bidder B_2 and bidder B_3 win each unit would only be locked for an average of 60 DOTs or something else equivalent to 240 DOT-intervals, while of bidder B_1 wins each unit is locked for 75 DOTs.

Treasury The system needs to continually raise funds, which we call the treasury. These funds are used to pay for developers that provide software updates, apply any changes decided by referenda, adjust parameters, and generally keep the system running smoothly.

Funds for treasury are raised in two ways:

1. by minting new tokens, leading to inflation, and 2. by channelling the tokens set for burning from transaction fees and slashing.

Notice that these methods to raise funds mimic the traditional ways that governements raise funds: by minting coins which leads to controlled inflation, and by collecting taxes and fines.

We could raise funds solely from minting new tokens, but we argue that it makes sense to redirect into treasure the tokens from transaction fees and

slashing that would otherwise be burned:

- By doing so we reduce the amount of actual stake burning, and this gives us better control over the inflation rate (notice that stake burning leads to deflation, and we cant control or predict the events that lead to burning).

- Following an event that produced heavy stake slashing, we might often have to reimburse the slashed stake, if there is evidence of no wrongdoing. Thus it makes sense to have the dots availabe in treasury, instead of burning and then minting.

- Suppose that there is a period in which there is an unusually high amount of stake burning, due to either misconducts or transaction fees. This fact is a symptom that there is something wrong with the system, that needs fixing. Hence, this will be precisely a period when we need to have more funds available in treasury to afford the development costs to fix the problem.

4.7 Cryptography

(can we change the title of the section e.g., Keys in Polkadot or Cryptographic Keys or something else? Cryptography is too generic name for this section)

In Polkadot, we necessarily distinguish among different permissions and functionalities with different keys and key types, respectively. We roughly categorize these into account keys with which users interact and session keys that nodes manage without operator intervention beyond the certification process.

4.7.1 Account keys

Account keys have an associated balance of which portions can be *locked* to play roles in staking, resource rental, and governance, including waiting out some unlocking period. We allow several locks of varying duration, both because these roles impose different restrictions, and for multiple unlocking periods running concurrently.

We encourage active participation in all these roles, but they all require occasional signatures from accounts. At the same time, account keys have better physical security when kept in inconvenient locations, like safety deposit boxes, which makes signing arduous. We avoid this friction for users as follows.

Accounts that lock funds for staking are called *stash accounts*. All stash accounts register a certificate on-chain that delegates all validator operation and nomination powers to some *controller account*, and also designates some *proxy key* for governance votes. In this state, the controller and proxy accounts can sign for the stash account in staking and governance functions respectively, but not transfer fund.

At present, we suport both ed25519 and schnorrkel/sr25519(ref or link) for account keys. These are both Schnorr-like signatures implemented using the Ed25519 curve, so both offer extremely similar security. We recommend ed25519 keys for users who require HSM(replace with: Hardware Security Module (HSM)) support or other external key management solution, while schnorrkel/sr25519 provides more blockchain-friendly functionality like HDKD(replace with: Hierarchical Deterministic Key Derivation (HDKD)) and multi-signatures(refs).

In particular, schnorrkel/sr25519 uses the Ristretto implementation [**?**] of Mike Hamburg's Decaf [**?**, §7](a strange symbol appears in pdf), which provide the 2-torsion free points of the Ed25519 curve as a prime order group. Avoiding the cofactor like this means Ristretto makes implementing more complex protocols significantly safer. We employ Blake2b for most conventional hashing in Polkadot, but schnorrkel/sr25519 itself uses STROBE128 [**?**], which is based on Keccak-f(1600) and provides a hashing interface well suited to signatures and NIZKs(replace with: Non-Interactive Zero-Knowledge Proofs (NIZK)).

4.7.2 Session keys

Session keys each fill roughly one particular role in consensus or security. As a rule, session keys gain authority only from a session certificate signed by some

controller key and that delegates appropriate stake.

At any time, the controller key can pause or revoke this session certificate and/or issue replacement with new session keys. All new session keys can be registered in advance, and some must be, so validators can cleanly transition to new hardware by issuing session certificates that only become valid after some future session. We suggest using pause for emergency maintenance and using revocation if a session key might be compromised.

We prefer if session keys remain tied to one physical machine because doing so minimizes the risk of accidental equivocation. We ask validator operators to issue session certificate using an RPC protocol, not (to) handle the session secret keys themselves.

Almost all currently running proof-of-stack(stake?) networks(refs) have a negligent public key infrastructure that encourages duplicating session secret keys across machines, which reduces security and leads to pointless slashing.

We impose no prior restrictions on the cryptography employed by specific Substrate modules(we have never mentioned Substrate before. We can add small info about Substrate module and give reference) or associated session keys types.

In BABE §4.4.1(We did not use § in the previous sections. We should remove §before all references and citations in this section to be consistent.), validators use schnorrkel/sr25519 keys both for regular Schnorr signatures, as well as for a verifiable random function (VRF) based on NSEC5 [**?**].

A VRF is the public-key analog of a pseudo-random function (PRF), aka cryptographic hash function with a distinguished key, such as many MACs. We award block productions slots when the block producer scores a low enough VRF output $\texttt{VRF}(r_e||\texttt{slot}_\texttt{n}\texttt{umber})$(it does not have the secret key as an input), so anyone with the VRF public keys can verify that blocks were produced in the correct slot, but only the block producers know their slots in advance via their VRF secret key.

As in [**?**], we provide a source of randomness r_e for the VRF inputs by hashing together all VRF outputs form the previous session, which requires that BABE keys be registered at least one full session(what is the relation between a session and epoch? if one session is one epoch then it should be at least two full session) before being used.

We reduce VRF output malleability by hashing the signer's public key along side the input, which dramatically improves security when used with HDKD. We also hash the VRF input and output together when providing output used elsewhere, which improves compossibility in security proofs. See the 2Hash-DH construction from Theorem 2 on page 32 in appendix C of [**?**].

In GRANDPA §4.4.2, validators shall vote using BLS signatures, which supports convenient signature aggregation and select ZCash's BLS12-381 curve for performance. There is a risk that BLS12-381 might drops significantly below 128 bits of security, due to number field sieve advancements. If and when this happens, we expect upgrading GRANDPA to another curve to be straightforward.

We treat libp2p's transport keys roughly like session keys too, but they include the transport keys for sentry nodes, not just for the validator itself. As such, the operator interacts slightly more with these.

We permit controller keys to revoke session key validity of course, but controllers could pause operation for shorter periods. We similarly permit controllers to register new session keys in advance, which enables a clean handover between validator machines.

4.8 Networking

In the above sections we talk abstractly about nodes sending data to another node or other set of nodes, without being too specific as to how this is achieved. We do this to simplify the model and to clearly delineate a separation of concerns between different layers. Of course, in a real-world decentralised system the networking part also has to be decentralised - it is no good if all communication passes through a few central servers, even if the protocol running on top of it is decentralised within itself. [3] In this section we outline and enumerate the specific sending primitives that we require in Polkadot, and sketch a high-level design on how we achieve these in a decentralised way, with the specifics to be refined as we move forward with a production system.

Polkadot needs networking for a number of its components as follows.

1. accepting transactions from users for (a) parachains and (b) the relay chain
2. collation of blocks within a parachain
3. validation of parachain blocks, via proofs and attestation, and making these efficiently available 4.2.2
4. distributing relay chain blocks 4.3 and votes 4.4.2
5. sending messages between different parachains 4.2.3
6. synchronising new state for (a) parachains and (b) the relay chain

1(a), 2 and 3(a) are strictly outside of the scope of Polkadot, being entirely chosen by each parachain for themselves, but schemes similar to the ones described below (to be used by the relay chain) may also be used by parachains if they decide they are suitable.

4.8.1 Message flows

Generally speaking, the communication data flow follows these patterns:

- messages within a parachain and within the relay chain are gossipped
- messages between a parachain and the relay chain are sent to some non-specific targets, e.g. a parachain validators to some collators, or a collator to some parachain validators. Gossip within the target set then propagates the messages to the entire target set.

[3] As a concrete example on why this is the case: in certain security models, including the traditional Byzantine fault-tolerant setting, nodes are modelled as capable of being malicious but no consideration is given to malicious edges. Under these models, if an edge is malicious in practice in the real-world, it is the corresponding node(s) that is treated as malicious in the model. If the underlying communications network is centralised, this effectively gives the central parties the ability to "corrupt" a large number (e.g. $> 1/3$) of nodes within this model, breaking its security, even if they don't actually have arbitrary execution rights on that many nodes.

- messages from users to submit transactions are sent to some non-specific targets, e.g. collators or validators
- as a special case, the erasure coded scheme for availability is sent directly from parachain validators to specific other validators. The scheme is design specifically to handle unreliability in the direct-sending primitive, which is generally harder to achieve reliability for than gossip or non-specific sending to a subset.

4.8.2 Authentication and discovery

In secure protocols in general, and likewise with Polkadot, entities refer to each other by their cryptographic public keys. There is no strong security association with weak references such as IP addresses since those are typically controlled not by the entities themselves but by their communications provider.

Nevertheless in order to communicate we need some sort of association between entities and their addresses. In Polkadot we use a similar scheme as many other blockchains do, that is using the widely used distributed hash table (DHT), Kademlia [**?**]. Kademlia is a DHT that uses XOR distance metric for finding a node and is often used for networks with high churn. We use Protocols Labs' libp2p libraries [](ref) Kademlia implementation with some changes for this purpose. To prevent Eclipse attacks [](ref) we allow for routing tables that are large enough to contain at least some honest nodes.

Currently, this "address book" service is also used as the primary discovery mechanism - nodes perform random lookups on the space of keys when they join the network, and connect to whichever set of addresses are returned. Likewise, nodes accept any incoming connections. This makes it easy to suport light clients and other unprivileged users, but also makes it easy to perform DoS attacks.

It is planned to develop a stronger authentication mechanism, where nodes authenticate themselves - their cryptographic identities and their roles as part of Polkadot - and verify other nodes both when (a) connections are mode, and when (b) looking up new nodes in the address book. Furthermore it should be possible to discover nodes that match a particular role, e.g. "collator for parachain C" and so on. Nevertheless it is also important to retain the ability to accept incoming connections from unauthenticated entities, e.g. users submitting transactions, or previously-privileged nodes that have been offline for a while that now want to synchronise, and this needs to be restricted on a resource basis, without starving the authenticated entities.

4.8.3 Gossiping

The main idea of gossiping is to broadcast every newly received message in the nodes local network (peers that the node is aware of). Moreover, we apply some restriction to the gossiping protocol to prevent bandwidth problems, e.g., denial-of-service (DoS-ed) as follows. Currently a naive flooding approach is

implemented, with various plans to make this more secure and efficient later on, which can be done independently of other components.

For GRANDPA we only allow two votes being received for each type of vote, round number, and voter. Any further votes will be ignored. For block production only valid block producers are allowed to produce one block per round and further blocks will be ignored.

We use *Sentry nodes* that are proxy servers who receive all the traffic that would go to a certain validator and forward the traffic as soon as the validator is able to receive it.

We also plan to implement some sort of set reconciliation protocol to reduce redundancy when many senders attempt to send the same object to the same recipient at once.

4.8.4 Specific and non-specific direct sending

In the naive initial version this simply involves looking up a particular key in the address book and connecting to that address.

For specific-target direct sending, the only higher-layer protocol in Polkadot that requires this primitive is the erasure coded availability protocol, which is designed to handle the case where some of the nodes are actually unreachable.

For non-specific target direct sending, we have a few issues to consider:

- How to store and lookup non-specific targets in the address book in a secure and balanced way, and what the representation of the query term(s) should be.
- Load-balancing actual transport-layer connection attempts over the whole target set, and ensuring availability.

As a special case, the source may already be a member of the target set, e.g. in an XCMP message the source parachain may share members with the target parachain. In this case, these members may bypass the above process entirely, and gossip between the two sets. However it may not be obvious to other members of the source set that the intersection member(s) are all online, and so they likely will want to attempt the process anyways.

4.8.5 Message types

Below we give an overview of where and how each type of message is sent. The column *Nets* refers to the networks where a type of message is traversing and the column *Mode* refers to the type of routing. The column *Static DHT Prefixes* refers to the DHT prefixes of the receivers if we use a one DHT for all and use prefixes to separate sub-networks.

We use gossiping mainly when the message type is small. For example, GRANDPA votes and attestation are very small. For bigger data structures we need to either use bloom filters or use direct routing.

Nets:

PC = Parachain Collator and parachain full nodes
PV= Parachain Validators
V = Validator and relay chain full nodes (-¿Validator Network ID on chain)
Mode:
D = Direct transfer
G = Gossip
B = Big / Bloomfiltered
R = Receving e.g., PC_R refers to the receiving parachain's collators and full nodes
S = Sending e.g., PC_S refers to the sending parachain's collators and full nodes

Message type	**Nets**	**Mode**	**Static DHT Prefixes**
Parachain TXs	PC	G	Depends on Parachain
PoV block	PC + PV	D	-
Parachain Block	PC + PV	G:PC, D:PV	P_0,...,P_n
Attestations	V	G	V
Relay chain TXs	V	G	V
Relay chain block	PC + V	G^B	General
Messages	PC_{R+S}	G^*	V
Erasure coded	V	G (D later)	V
GRANDPA Votes	V	G	V

* fallback-¿D:PV_R request PV_S and then uses G at PC_R to spread them, second fallback-¿D: PV_R recover messages from erasure codes obtained from V and use G at PC_R to spread them.

Acknowledgement

We would like to acknowledge Parity Technologies developers for their useful input and good discussions.

A Glossary and Background

- state machine - data structures

Name	Definition
Era	a period for which a new validator set is decided by NPoS
Epoch	a period for which randomness is generated by BABE
BABE Slot	a period for which a relay chain block is produced
GRANDPA Rounds	states of the GRANDPA algorithm which lead to a new relay chain block being fin
Shuffle	