Overview of Polkadot Design

September 12, 2019

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

Say something in line with: blockchain are valuable technologies because...

Currently there have been hundreds chains implemented in the wild for various functions. (maybe a table with 10 famous but very different chains?) Each one has different characteristics and their own security set up.

There is a need for a system that can connect this heterogenous chains.

Moreover, having multiple security set ups causes a split in the security these systems can provide. Gathering all this security power and have a shared security system increases security substantially.

1.1 System Overview

- state machine - data structures

2 Background

Polkadot is a multi-chain system with shared security guarantees. Polkadot consists of a main chain called the *Relay Chain* and multiple parallel chains called *Parachains*. A fixed set of actors called *Validators* are responsible to maintain the relay chain. These validators need to vote on the consensus over all the parachains. The security goal of Polkadot is to be Byzantine fault tolerant. In addition, the validators are assigned to parachain by dividing the validators set into random disjoint subsets that are each assigned to a particular parachain. For the parachain, there are additional actors called collators and fishermen that are responsible for parachain block production and reporting invalid parachain blocks respectively.

3 Properties

3.1 Utility

3.2 Validity

3.3 Finality

3.4 Availability

3.5 Messaging Reliability

3.6 Reasonable Size and Bandwidth

How do we achieve them?

Briefly breaking them down into components.

4 Components

How do they work? What do they achieve (refer to properties)?

4.1 Validators (Network Maintainers)

Keys

4.1.1 NPoS mechanism for validator selection

The Polkadot network selects a new set of validators at the beginning of each new era. We denote by $n_{val}$the target number of validators to be selected, which is in the order of hundreds or thousands and is proportional to the number of parachains.

A validator node should satisfy certain requirements of speed, responsiveness and security. Any DOT holder who is up to the task can submit their candidacy to become a validator. If selected, they will have their stake locked and they will be economically compensated for their work, or slashed in the eventual case of a misconduct (see Section 4.5). Additionally, any DOT holder can participate in the security of the network as a *nominator*. Each nominator publishes a list (of any size) of the validator candidates that she trusts, together with an amount of DOTs that she is willing to lock. If one or more of her trusted candidates is elected during an era, she will share with them the validation rewards or slashings on a pro-rata basis.

Unlike the case for validators, there can be an unlimited number of nominators (passively) participating in any given era. This approach allows for a massive amount of DOTs to be staked, much larger than the DOTs owned solely by validators, which increases the security of the network. In fact, it is expected that a significant percentage of all available DOTs will be routinely locked in the NPoS mechanism.

The validator election problem. Formally, the validator election problem is a multi-winner election problem based on approval ballots, where nominators have a voting power proportional to their stake, and each nominator submits a list of validator candidates that she supports. A solution consists of a committee of k validators, together with a distribution of each nominator's stake among the elected validators that she backs. We consider two objectives, both of which have been recently introduced in the literature of computational social choice. The first one is achieving the property of *proportional justified representation* (PJR). The second objective, called *maximin support*, is maximizing the minimum amount of stake that backs any elected validator. The former objective aligns with the notions of decentralization among validators as well as user satisfaction in the platform, while the latter aligns with the security level of the consensus protocol.

A validator node should satisfy certain requirements of speed, responsiveness and security. And DOT holder who is up to the task can submit their candidacy to become a validator, and they will be compensated for it, or slashed in the eventual case of a misconduct; see Section 4.5.

This setup leads to a multi-winner election problem based on approval ballots, where nominators have voting power proportional to their stake, and each nominator submits a list of validator candidates that she supports. A solution

consists of a committee of k validators, together with a distribution of each nominator's stake among the elected validators that she backs. We consider two objectives, both of which have been recently introduced in the literature of computational social choice. The first one is achieving the property of *proportional justified representation* (PJR). The second objective, called *maximin support*, is maximizing the minimum amount of stake that backs any elected validator. The former objective aligns with the notions of decentralization among validators as well as user satisfaction in the platform, while the latter aligns with the security level of the consensus protocol.

We motivate the pursuit of these two objectives for the selection of validators in Polkadot as wells as any other decentralized protocol based on proof of stake with delegation. We also provide several constant-factor approximation algorithms for the maximin support objective, as well as a hardness result showing that these are, in a sense, best possible. Finally, we present a efficient algorithm which, when executed as a post-computation for any approximation algorithm for maximin support, ensures the PJR property while also preserving the approximation guarantee. Consequently, we provide the first efficient algorithms that simultaneously achieve PJR and constant-factor approximation guarantees for maximin support.

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, 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 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 4.2.3 for details such as the range needed) and 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 parachai 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.

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 guve 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. 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 blockhash 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 [**?**].

4.2.2 Validity and Availability

Once a block is created it is important that the parachain blob 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 block are available.

For a single chain, such as bitcoin, as long as 51% of hashpower is honest, not making block data available ensures that np honest miner builds on it so it will not be in the final chain. However parachain consensus for us is determined by relay chain consenus. 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 particpants, 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 [] 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 rely on nodes actin as *fishermen* to report invalidity in this case. 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 alrerady running full nodes, so all they need is stake in DOTs.

4.2.3 Inter Chain Messaging Protocol (ICMP)

ICMP is the protocol that parachains use to send messages. We would like guarantees that messages arrive quickly, that messages from one parachain to another arrive in order and that messages that arrive were indeed sent in the final history. 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 message was not sent, but then the message is recieved. 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 rootof outgoing messages, as well as a bitfield indicating which other parachains were sent messages from this parachain 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 actual messages.

However the messages themselves are passed, they should also be sent with the Merkle proof that allows nodes of the recieving parachain to authenticate that they were send by a particular parachain block whose header was in a particular relay chain block.

A parachain block contains a reference to the relay chain block, by block hash or block number, that it recieves all messages up to. This will likely be constrained to advance in some manner. To produce a parachain block on parachain P with builds on a particular relay chain block B, a collator would need to look at which parachain headers 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 occuring.

If any node is connected to both parachain networks, then it should forward messages from one parachain to the other, when the parachaim header is included ina relay chain block. The relay chain should at least act as a back up. The parachain validators are connected to the parachain network.

4.3 Relay Chain State Machine

Formally, Polkadot is a replicated sharded state machine where shards are the parachains and 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 govern by transactions grouped the relay chain blocks.

Transaction Inclusion

Block Building

4.4 Consensus

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 blocks production slots using the randomness generated with blocks. These assignments are completely private until the assigned parties produce their blocks. Therefore, we use "Blind Assignment" in the protocol name.

We may have slots without any assignments. We call these slot as empty slot. In order to fill the empty slots, we have secondary block production mechanism based on Aura. We note that these blocks do not contribute the security of BABE.

Next, we describe BABE together with its security properties. Then, we explain the secondary block production mechanism and discuss about its contribution on BABE.

BABE:

- 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. This protocol depends on the arrival time of blocks according to local clocks of validators. Thus, it is named as "Relative Time".

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 drift. In addition to this, a validator joins after the genesis block runs the relative time algorithm for synchronize with the other validators. Validators are synchronized if they have the same slot number in at least one moment in the original interval (Rephrase IT).

Assume that we have a validator who is a slot leader of the slot number sl and he wants to learn when sl starts according to his local clock. For this, he first collects n blocks. During the collection period, he stores the arrival time t_i of the block together with the slot number sl'_i in the block. Then, the validator computes some candidate start times of sl using the arrival times and the slot numbers i.e, given that $\delta_i = T(sl - sl'_i)$, $\mathcal{C}_T = \{t_i + \delta_i\}_{1 \leq i \leq n}$. The times in $\mathcal{C}_T$ is considered as candidates because of the following reason: We just for a moment imagine that all blocks are sent by honest and synchronized validators without any network delay. Given this fact, we say that sl should start $T(sl - sl'_i)$ seconds later. Therefore, all times are some 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 is in Figure 1.

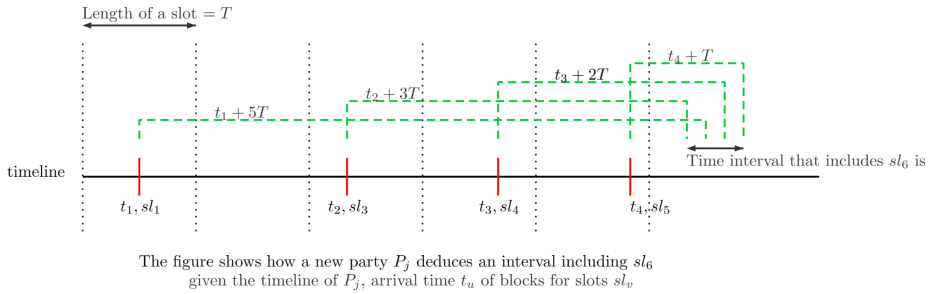

Figure 1: An example execution of the relative time protocol by a validator who wants to learn sl_6.

The relative time protocol guarantees that the output of a validator can be at most Δ behind the slot number from the real slot number on assuming that at least half of the blocks during the collection period belong to honest and synchronized parties. The reason of this result based on the fact that half of n blocks belong to honest and synchronized validators with a very high probability. If the median is computed from an arrival time of block belonging to a honest and synchronized validator.

GRANDPA For the relay chain consensus, we want provably finality, which makes bridges to chains outside Polkadot easier, and we also want the ability to revert chains in the short term if parachain blocks are unavailable for invalid. So we combine a block production mechanism BABE with the ability to fork and that gives eventual consensus with a finality gadget GRANDPA, that can finalise blocks and provide proofs that they are final. BABE is similar to Ourobors Praos[citation needed] but with a fixed set of validators and improved resilience to clock drift. It works by dividing time into slots in each of which validators computing a Verifiable Random Function(VRF) which will decide whether they have the ability to produce a valid block in that slot. For fork-choice, we go with the longest chain that includes all blocks finalised by GRANDPA.

4.5 Economic Incentives and Slashing

4.6 Governance

Relay Chain Council Referenda

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 can't 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

In polkadot, we necessarily distinguish among different permissions and functionalities with different keys and key types, respectively. We roughly categories 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 durations, 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 may declare themselves to be *stash accounts* by registering a certificate on-chain that delegates all validator operation and nomination powers to some *controller account* and designates some proxy key for governance. 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 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 support or other external key management solution, while schnorrkel/sr25519 provides more blockchain-friendly functionality like HDKD and multi-signatures.

In particular, schnorrkel/sr25519 uses the Ristretto implementation [**?**] of Mike Hamburg's Decaf [**?**, §7], 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.

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.

We impose no prior restrictions on the cryptography employed by specific substrate modules or associated session keys types.

In BABE §**??**, validators use schnorrkel/sr25519 keys both for regular Schnorr signatures, as ell 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 sores a low enough VRF output $\texttt{VRF}(r_e||\texttt{slot}_\texttt{n}\texttt{umber})$, 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 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 §4.4.1, 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

Networking is an essential part of decentralized technologies. Polkadot needs networking for a number of its components as follows.

- distributing votes for the consensus protocol GRANDPA 4.4.1
- distributing relay chain blocks4.3, parachain blocks, and PoV blocks
- receiving parachain blocks that have been produced by parachains
- distributing and recovering of erasure coded pieces for the availability and validity scheme 4.2.2
- distributing attestations for the availability and validity scheme 4.2.2
- sending messages from a parachain to another parachain for the interchain messaging scheme 4.2.3

We use *gossiping* for all networking use cases mentioned above, except for distributing and recovery of erasure coded pieces for the availability and validity mechanism, where we use *direct routing* for scalability. For interchain messaging we use first gossiping with a fall back to direct routing if gossiping fails.

We will not discuss the networking details of parachain block production, since it is out of scope for Polkadot.

We refer to all the entites as nodes when describing networking.

4.8.1 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). For node discovery we use a similar networking scheme as many other blockchains do that is using the widely used distributed hast 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 [] Kademlia implementation with some changes for this purpose.

Moreover, we apply some restriction to the gossiping protocol to prevent bandwidth problems, e.g., denial-of-service (DoS-ed) as follows.

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.

To prevent Eclipsing attacks [] we allow for routing tables that are large enough to contain at least some honest nodes.

4.8.2 Direct Routing

The aim of direct routing is only to send messages to the intended receiver. This is saving lots of traffic and is useful when messages only have one receiver and do not need to reach more than one validator such as in the case of erasure coded pieces.

4.8.3 Interchain Messaging

If the sending parachain and the receiving parachain have common honest full nodes then the message will be gossiped from without issue. However, if parachain validators of the receiving parachain realize that the message has not arrived they request the message from the parachain validator of the sending parachain. Once, they receive it they gossip it in the receiving parachain.

4.8.4 Overview

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

* 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.

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

5 Summary

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.