Availability and Validity

All existing sharded blockchain designs choose the validators assigned to each shard *before* the shards produce blocks, which gives adversaries advance knowledge of when they could approve invalid blocks, and perhaps manipulate shard assignments. In practice, real adversaries would behave somewhat adaptively to disrupt connectivity for some additional validators too, with disruption costs varying among validators. Adversaries wait until an entire shard consists of their corrupted validators plus some disruptable validators. We must therefore choose security paramaters to make these odds negligible, and their waiting time astronomical.

Instead, we make adversaries first commit to their blocks' correctness and availability before we select which validators check their block, so aversaries cannot simply wait until they control enough assigned validators. We achieve their commitment to availability using erasure coding ..

We require that approval validity checks complete before finality. We cannot however require that all validator checks conclude before finality, or even ask fishermen to begin checks before finality, so invalidity can be detected after finality.

In §1, we first recall our tools like verifiable random functions, erasure coding, and state witnesses, and then describe the sharding problem and introduce our security definitions.

We describe and instantiate our *availability and validity protocol* for efficient sharding as several phases across §**??**, §4, and §**??**. As a rough outline,

- a parachain phase in which collators produce candidate parachain blocks (§**??**) and parachain validators perform preliminary backing validity checks (§2),
- a relay chain submission phase that distributes candidate parachain blocks and produces relay chain blocks (§2 and §3.2),
- availability (§3) and unavailability (§**??**) subprotocols that enforce availability in GRANDPA and BABE respectively,
- assigning secondary checkers (4.2) who fuller approval validity checks (§4.1), and
- invokation of a Byzantine fault tolerant "finality gadget" that gives us finality (§4.2.8).

We depend upon slashing to provide economic security guarantees, so we discuss slashing both in §**??** and when relavant in the main protocol description.

We also discuss objection procedures for fishermen in §5.2 and collators in §5.3.

At any time, there are three candidate receipts associated to each parachain ρ, one accumulating backing votes, its predecessor accumulating availability votes, and its predecessor the current chain head that

1 Heterogeneous sharding

We shall describe a blockchain sharding scheme through which a single beacon or *relay chain* can validate numerous heterogeneous shard chains. We aim for true shared security, minimal latency, good liveness, and relative efficiency in terms of computation, network, and storage. We refer to our shard chains by the term *parachains* because shard alone connotes a non-heterogeneous approach. There exist other terms for heterogeneous approaches too, but they all connote bridged designs without shared security.

1.1 VRFs

A *verifiable random function (VRF)* is a cryptographic function that generates a pseudo-random output number from a given a secret key sk and an input string α called a seed, and produces a proof of correctness for the output. In a sence, VRFs resemble a public-key analog of a cryptographic hash function with a distinguished key, such as MACs that acts as PRFs, in that a keyed hash function requires the key be shared for verification.

Anyone with the input and the corresponding public key pk can verify the correctness of the VRF output, so VRFs should satisfy appropriate signature scheme security definitions, like EUF-CMA. We implement output as function applied to the signature and message that must satisfy *uniqueness* in that each public key and message admits only one output, *unpredictability* for anyone without the secret key, and *pseduo-randomness* even for the secret key holder.

We refer readers to [**?**, §1] for more formal definitions. Almost all proof-of-stake and sharded blockchain proposals make heavy usage of VRFs.

1.2 Chains

We expect that our sharded blockchain employs a Byzantine fault tolerant "finality gadget" for the relay chain that establishes agreement on finality for all shards or parachains too, like in the Polkadot and Ethereum 2.0 designs. We let $\mathcal{V}$ denote the full validator list for the relay chain, and set $n := |\mathcal{V}|$, which we term just validators since the relay chain remains clear from context. We expect here that validators act as block producers for the relay chain.

As part of our security model, we assume that $n \geq 3f + 1$ where f bounds the number of validators controlled by any adversary.

An adversary who controls some $f' \leq f$ of the n (relay chain) validators knows the upcoming block producer for its own upcoming block production slots of course, but we assume Bernoulli trials with probability $\frac{f'(n-f')}{n^2}$ describe whether they know the honest block producer assigned to individual upcoming block production slots. In particular if $\frac{f}{n} < \frac{1}{3}$ then they know less than $\frac{5}{9}$ths of the upcoming honest block producers. We choose this assumption because it supports overall stronger block production mechanisms than proof-of-work or [**?**], although those would admit a better bound here.

As in other finality gadget designs, the relay chain and each parachain ρ has its own block production protocol run within its own infrastructure. We employ the simplifying assumption that the block producer list equals the validator list $\mathcal{V}$.

A parachain ρ refers to its block producers as *collators*, which could come from $\mathcal{V}$ or elsewhere. We let $\mathcal{V}_\rho \subset \mathcal{V}$ denote the sublist of validators assigned to parachain ρ, which we term the *parachain validators* of ρ. We also need a bound n_0 such that $|\mathcal{V}_\rho| \geq n_0$.

In fact, we always have an epoch parameter t associated to ρ, but we subsume t into ρ here for convenience. Aside from time, we similarly assume the relay chain provides some randomness r associated to each epoch t, which we similarly subsume into our ρ notation for convenience.

We prefer if subscripts always denote higher level properties, like identifying parachains, cryptographic keys, etc. We favor indexing bracket notation like $\mathcal{V}[i] = \mathcal{V}_\rho[j]$ when identifying individual list elements, as index notation simplifies definitions and avoids mixed role subscripts.

1.3 Blocks

We normally prevent blocks from referencing data in other blocks directly, e.g. UTXOs, because doing so incurs unacceptable storage overhead. Instead, all blockchains have some associated state for which blocks describe a state transition, normally stored as a Merkle tree so that the Merkle root acts as a commitment to the entire state. As a brutal convenience, we further subsume any current state commitments into our ρ notation. We let $\hat{\rho}$ denote the parachain state ρ but with full attached state required by collators, not just the state commitments.

Consider a block B. We distinguish a header B_{head} and body B_{body} of a block $B = (B_{\mathsf{head}}, B_{\mathsf{body}})$. We ask that B_{head} contain $H(B_{\mathsf{body}})$ and commitments to the associated state before and after running B, normally Merkle roots.

In principle, any block might incorporate additional associated data M also referenced by B_{head}. We shall not belabour the associated data M here, so anyone who requires it should read thoughtfully.

Definition 1. *We define a* state witness procedure *to consists of*

- *a randomized algorithm* $\mathsf{Prove}_{\hat{\rho}}$ *that executes a block B, and maybe M, and returns a witness proof π_B of any transitions to the associated state, as well as*
- *deterministic algorithms* Verify_ρ *and* $\mathsf{Verify}_{\hat{\rho}}$ *that take the blob $\bar{B} = (B, \pi_B, M)$ execute B applying the state transition to either the commitments in ρ or the full state in $\hat{\rho}$, and returns* true *and mutates ρ or $\hat{\rho}$, if and only if B executes correctly with the state transition given by the witness π_B.*

We disallow $\mathsf{Prove}_{\hat{\rho}}$ from modifying the associated state commitments in ρ and only permit $\mathsf{Verify}_{\hat{\rho}}$ to do so when returning true. We note that Verify_ρ and $\mathsf{Verify}_{\hat{\rho}}$ make B the more recent block of ρ, so they should normally fail if B already exists in ρ under most chain designs.

As a convenient Merkle tree notation, we let the notation $\overrightarrow{xz}$ denote the copath witness proving that z is a leaf of a Merkle tree with root x. We support internal nodes in this notation too, so $\overrightarrow{xz} = \overrightarrow{xy}\overrightarrow{yz}$ if y lies on the path from x to z. We write $B[\overrightarrow{xy}]$ to specify B as the underlying Merkle tree, when not clear from context, especially when B itself is a block. We shall always write just $\overrightarrow{xy}$ when x and y lie in the state tree associated to some blockchain.

We shall not restrict this proof π_B here, but often π_B consists of the union of all copath witnesses $\overrightarrow{rx}$ in the associated state tree that arise from read and write operations when executing B. If B changes the state tree's Merkle root from r_{in} to r_{out} then

$$\pi_B = \bigcup \{ \overrightarrow{r_{\mathsf{in}}x} \mid B \text{ reads } x \} \cup \{ \overrightarrow{r_{\mathsf{out}}x} \mid B \text{ writes } x \} .$$

There are however other instantiations for π_B, like some zkSNARKs that takes the read or written leaves x as public inputs. We define the *blob* $\bar{B} = (B, \pi_B, M)$ for our block B and perhaps its additional associated data M, although some plausible instantiations for π_B with zkSNARKs allow using only B_{head} in $\bar{B}$.

We observe this definition satisfies:

- *witness correctness* in that $\mathsf{Verify}_\rho(B, \mathsf{Prove}_\rho(B, M), M) = \mathsf{true}$ holds with good probability for all B, M for ρ, and
- *witness security* in that any probabilistic polynomial-time adversary has only negligible odds of constructing a (B, π, M) that satisfies $\mathsf{Verify}_\rho(B, \pi, M) = \mathsf{true}$ and $\pi \neq \mathsf{Prove}_\rho(B, M)$.

1.4 Availability

A *rigid/cryptographic k-of-n* erasure code consists of two algorithms:

- $\mathsf{Encode}_{k,n}$ creates n encoded strings from one source string B, while
- $\mathsf{Decode}_{k,n}$ recreates the original source string B from *any* k of the n encoded strings.

We call the encoded strings *pieces*, although the literature sometimes calls them symbols. We caution that many modern erasure codes are not rigid/cryptographic, especially rateless codes, because there exist encoded string sets S with $|S| > k$ but for which $\mathsf{Decode}_{k,n}$ fails. Reed-Solomon codes satisfy this rigid/cryptographic property.

There exist *availability sharding protocols* that preserve some string B by distributing its encoded string pieces among distinct locations, like our validators $\mathcal{V}$. In this context, we say B is *available from* $\mathcal{V}$ or *unavailable from* $\mathcal{V}$ if at least k pieces or less than k can be obtained by an honest connected party, respectively. We write only *available* or *unavailable* whenever the locations $\mathcal{V}$ seems clear from context.

1.5 Security

We discuss a "crypto-economic" protocol that hinges upon punishing nodes for incorrect behavior. All such protocols entail a conviction game in which an adversary engages in some specified missbehavior and the correctly behaving nodes must obtain evidence of incorrect behavior by enough incorrectly behaving nodes, and this evidence be actionable in that correctly behaving nodes have effective defenses against false accusations. In this article, we only discuss subprotocols that produce direct proofs of incorrect behavior, but our overall protocols depend upon interactive evidentiary standards, like in the ... phase of GRANDPA [**?**].

Assume again that at most $f' \leq f$ out of the n (relay chain) validators deviate from the correct protocol.

There are sharding protocols like ByzCoin [**?**] and .. in which validity of a block on a shard ρ is determined solely by the n_0 validators assigned to ρ, normally by reaching some vote threshold $\mathtt{min_backers} \in [n_0/2, n_0)$ for each block.

At first blush, one models these protocols by saying an adversary wins the *naively sharded validity game* if they first learn the role assignments for all nodes $\rho \mapsto \mathcal{V}_\rho$, only then proposes an invalid shard block B for some shard ρ, and convince the relay chain to finalize B, all

while fewer than $\texttt{min}_\texttt{b}\texttt{ackers}$ of the incorrectly behaving nodes in $\mathcal{V}_\rho$ provide proofs of their incorrect behavior to the correctly behaving nodes.

An adversary wins some random instance of a naively sharded validity game with odds $\epsilon'_{n_0,\texttt{min}_\texttt{b}\texttt{ackers}}$ of order roughly $\binom{f'}{n_0}/\binom{n}{n_0}$ In these, security depends upon $\epsilon'_{n_0,\texttt{min}_\texttt{b}\texttt{ackers}}$ being "negligible" so that adversaries cannot simply wait until they win the shard assignment lottery before launching an attack. For this, one requires that n_0 be a significant fraction of n, and that n itself be large enough.

We mildly distrust requirements for a large network size n for two reasons: Almost all networks start small at least in the sense of truly independent nodes, and the finality gadget and its gossip assumption limit n.

We worry much more about requirements for a large number n_0 of validators assigned to each shard, and the large number $\texttt{min}_\texttt{b}\texttt{ackers}$ of validtors approving each shard block, because this limits the number n/n_0 of shards, and hence limits the utility of our scaling effort. Also liveness suffers if $\texttt{min}_\texttt{b}\texttt{ackers}$ is close to n_0, while security against network adversaries suffers otherwise.

We shall create an information asymmetry that turns these odds more in defenders favor: We require the adversary first declare their parachain block B and commit to its correctness with the stake of $\texttt{min}_\texttt{b}\texttt{ackers}$ backing validators. We then reveal the identity of the $\texttt{min}_\texttt{a}\texttt{pprovers}$ checking validators assigned to actually check B only after this commitment. We say an adversary wins the *parachain validity game* if they finalize an invalid block with fewer than $\texttt{min}_\texttt{b}\texttt{ackers}$ of the f' incorrectly behaving nodes providing proofs of their incorrect behavior to the correctly behaving nodes.

In this game, we tolerate larger $\epsilon'_{n_0,\texttt{min}_\texttt{a}\texttt{pprovers}}$ because attacks cost an amortised $\epsilon'_{n_0,\texttt{min}_\texttt{a}\texttt{pprovers}}\texttt{min}_\texttt{b}\texttt{ackers}$ validators' stake.

In advance, we assigned the set $\mathcal{V}_\rho \subset \mathcal{V}$ of at least n_0 to validate all blocks from a particular parachain ρ. In our game, any parachain block B from ρ requires preliminary backing commitments to correctness claim from at least $\texttt{min}_\texttt{b}\texttt{ackers}$ out of these n_0 validators. We accept $\texttt{min}_\texttt{b}\texttt{ackers} < n_0$ here since some unpredictable nodes might be offline or slow, so long as $\texttt{min}_\texttt{b}\texttt{ackers}$ allocates sufficient stake to the correctness commitments for B.

After this commitment, we must assign at least $\texttt{min}_\texttt{a}\texttt{pprovers}$ validators from $\mathcal{V} \setminus \mathcal{V}_\rho$ who perform approval validity checks to conclude the game. Now our security level dependes only upon this number $\texttt{min}_\texttt{a}\texttt{pprovers}$ of approval validity checks.

All nodes compute their own target number $\texttt{min}_\texttt{a}\texttt{pprovers}$ of later approval checkers dynamically based upon several report factors from parties who notice abnormalities in the protocol run, as well as $n_0 - \texttt{min}_\texttt{b}\texttt{ackers}$. If n_0 primary checks appears realistic then we might consider fewer than n_0 primary checks as slightly suspicious. In this case, we might simply target $\texttt{min}_\texttt{a}\texttt{pprovers}$ plus the number of missing backers from $\mathcal{V}_\rho$, meaning that approval checks can freely replace $n_0 - \texttt{min}_\texttt{b}\texttt{ackers}$ preliminary backing checks.

2 Backing

We now describe the subprotocols that provide the preliminary financial backing checks from parachain validators with which Polkadot begins processing parachain candidate blocks. After parachain candidates have enough backing, we shall next proceed to ensure their availability in §3 and only lastly perform the final approval validity checks in §4.

Our backing protocol has nodes play three roles:

- Collators for a parachain ρ prepare the candidate blocks's proof-of-validity block usable with stateless validation.
- Validators $\mathcal{V}_\rho$ assigned to back the parachain ρ update the block metadata and prepare the candidate receipt handled directly by the relay chain, attest to its validity, and submit it to the relay chain once it obtains sufficent backing.
- Some relay chain block producer selects among any competing candidates with sufficient backing.

Intuitively, there are two distinct protocols here, one between the collators and parachain backing validators, and one between parachain backing validators and relay chain block producers. We caution however that parachain validators must update metadata and prepare the candidate receipt before providing backing attestations.

2.1 Collators

We begin when some **collator** C of a parachain ρ possesses some candidate block B for ρ, which C should then propose for recognition by the relay chain. We let B' denotes the parent of B in ρ. As above, let r_{in} and r_{out} denote state root of ρ before and after executing B.

In practice, we want shared security so that parachains can communicate among themselves easily, among other reasons. As such, the candidate B should reference some *relay chain parent block* R^0_B that distinguishes any state ρ maintains on the relay chain, which contains an incoming message "watermark" and some commitments related to outgoing messages (see XCMP).

We let q denote the Merkle root of this state ρ maintains on the relay chain in R^0_B. Analogously with r, we shall let q_{in} and q_{out} denote the state ρ maintains on the relay chain before and after executing B below. We do not assume that C can compute q_{in} and q_{out} however, only that operations C performs transform them legally.

In principle, B could reference extra data M determined by R^0_B and supplied by the relay chain, but not actually included inside the relay chain state. We envision $M = \emptyset$ for the foreseeable future however.

First, C constructs the witness data π by evaluating the block with $\mathsf{Prove}_{\hat{\rho}}(B, M)$, so they can build the *candidate proof-of-validity blob* $\bar{B}_0 = (B, \pi)$, and also obtain the block metadata $(\rho.\mathsf{id}, H(B'), H(R^0_B), r_{\mathsf{in}}, r_{\mathsf{out}}, \ldots)$.

As Prove is a randomized algorithm, C must next reevaluate the block with $\mathsf{Verify}_{\hat{\rho}}(\bar{B}_0, M)$. We shall assume verification succeeds, but if this verification fails then C reports invalid parachain code for ρ, and discards B or possibly shuts down. Assuming no errors, C sends the candidate blob $\bar{B}_0$ to the corresponding parachain validators $\mathcal{V}_\rho$, along with any block metadata $(\rho.\mathsf{id}, H(B'), H(R^0_B), r_{\mathsf{in}}, r_{\mathsf{out}}, \ldots)$.

2.2 Backing checkers

We next describe the subprotocol by which the parachain validators $\mathcal{V}_\rho$ assigned to ρ produce the *preliminary backing validity checks* and *candidate backed transaction* with which they announce parachain candidates on the relay chain.

In essence, we shall construct a *candidate receipt* that describes B to

Assignment: Initially, our collator C sends the candidate's proof-of-validity blob $\bar{B}_0$ to some *parachain validator* $V_0 \in \mathcal{V}_\rho$, along with any metadata. We expect header data for $\bar{B}_0$ justifies C producing B, such as a VRF seal valid for this slot in ρ.

We caution the parachain backing validator sets $\mathcal{V}_\rho$ rotate. We determine the current assignment $\rho.\mathsf{id} \mapsto \mathcal{V}_\rho$ using the relay chain parent R^0_B of B, or perhaps merely its slot and r_e, so our collator C identifies the current $\mathcal{V}_\rho$. We permit slightly outdated R^0_B, but any R^0_B expires eventually due to rotation. In that case, V_0 should report failure to C and provide aid in selecting a newer R^0_B.

Candidate receipts: Any *parachain validator* $V \in \mathcal{V}_\rho$ who obtains $\bar{B}_0$, including our initial V_0, should first validate B by evaluating the block with $\mathsf{Verify}_\rho(\bar{B}_0, M)$, which checks the internal state of ρ witnessed by $\bar{B}_0$ of course.

At the same time, V observes the incoming and outgoing XCMP messages processed in B too. These invoke two relay state operations:

RW V computes the new relay chain state root q_{out} for ρ into which executing B transforms its old relay chain state root q_{in} of ρ. We cannot modify q except by executing parachain blocks on ρ, so this this RW transition remains valid in any $R > R^0_B$ until some parachain block acts.

RO V checks incoming messages' from outgoing message commitments in other parachains' relay chain states $\{q_{\rho'}\}$ in some $R^1_B > R^0_B$. These $\{q_{\rho'}\}$ would change in $R > R^1_B$, so operations involving them must remain valid indefinitely, although our watermark prevents replays.

We say these operations succeed if V successfully computes witnesses in R^1_B for them. We could regard this purely as a relay chain state transition that any validator checking $\bar{B}_0$ recomputes, except then (1) all relay chain validators check these operations under q and (2) any read operations occur under R^1_B, which perhaps lags the current chain state considerably. Instead, our initial parachain validator V_0 constructs copaths witnesses $\bar{B}_1$ for the RW transition q_{in} to q_{out} and for the RO operations under $\{q_{\rho'}\}$. V_0 then creates $\bar{B} = \bar{B}_0 \cup \bar{B}_1$ by appending $\bar{B}_1$ to $\bar{B}_0$.

We assume temporarily that both validation of $\bar{B}$ and these two RW and RO witnessing operations succeed.

Next V runs $\mathsf{Encode}_{f+1,n}(\bar{B})$ to obtain the *unauthenticated pieces* list $\mathcal{D}'_B$ of n distinct erasure code symbols aka pieces for $\bar{B}$. V computes a Merkle root $\mathfrak{m}_B$ for the Merkle tree with leaves $\mathcal{D}'_B$.

We now define an *abridged candidate receipt* that consists of

- the parachain id $\rho.\mathsf{id}$,
- the relay chain parent hashes $H(R^1_B)$ and $H(R^0_B)$,
- the Merkle root, $\mathfrak{m}_B$,
- some some block header that contains $H(B)$, $H(M)$, and references the parachain parent B',
- parachain state commitments $r_{\mathsf{in}}, r_{\mathsf{out}}$,

- RW state commitments $q_{\mathsf{in}}, q_{\mathsf{out}}$,
- RO state commitments $\{q_{\rho'}\}$,
- the collator's id $C.\mathsf{id}$ and signature $C.\mathsf{sig}$),

We now define an *(inner) candidate receipt* $\mathfrak{C}_B$ that consists of this abridged candidate candidate receipt, along with the additional data computable by relay chain validators.

Attestation: We define an *attestation* (V, σ) for a candidate receipts to be a validator identity V together with a signature σ on the (inner) candidate receipt. We define an *attested candidate receipts* $\mathfrak{C}_{B,S} := (\mathfrak{C}_B, S)$ for B to be the abridged candidate receipt or inner candidate receipt once deserialised, along with an set S of attestations on $\mathfrak{C}_B$. We shall define more roles for attestations $(V', \sigma') \in S$ below in §4, but we say (V', σ') is a *preliminary backing attestation* when $V \in \mathcal{V}_\rho$.

We expect validators attesting in $\mathfrak{C}_{B,S}$ to provide $\bar{B}$ to other validator in $\mathcal{V}_\rho$ upon request. We shall define additional validator roles related to $\mathfrak{C}_{B,S}$ below in §4, but so far we need not provide $\bar{B}$ to validators in $\mathcal{V} \setminus \mathcal{V}_\rho$.

There are also outgoing XCMP message data produced by $\mathsf{Verify}_\rho(\bar{B}, M)$ but required by their receiving parachains, so validator attesting in $\mathfrak{C}_{B,S}$ should upon request similarly provide these outgoing messages to appropriate collators associated to the messages' receiving parachains.

Initial backing: Initially, V_0 should reject $\bar{B}$, and report it, if $\mathsf{Verify}_\rho(\bar{B}, M)$ fails or if witnessing fails for either the RW transition of q_{in} to q_{out}, or for the RO checks and computation of $q_{\rho'}$ for any sending parachain ρ'. We simply abandon B in these case where no validators sign it because invalidity claims cannot necessarily result in penalties for either ρ or C.

Assuming verification succeeds, V_0 creates a signature σ for the candidate receipt and constructs the *attested candidate receipt* $\mathfrak{C}_{B,S} := (\mathfrak{C}_B, S)$ for B by attaching its initial attestation singleton set $S = \{(V, \sigma)\}$.

Gossip: We first gossip attested abridged candidate receipts for ρ only among the parachain backing validator group $\mathcal{V}_\rho$, but not yet outside. In gossip, we further accumulates these attestation signature sets S.[1] We optionally gossip attestations on known candidate receipts separately if candidate receipts grow large.

Subsequent backing: Any subsequent parachain backing validator $V \in \mathcal{V}_\rho$ who obtains $\mathfrak{C}_{B,S}$ then downloads $\bar{B}$ from C or another validator who attests in S. After that, V judges the candidate receipt $\mathfrak{C}_B$ valid *if* validation $\mathsf{Verify}_\rho(\bar{B}, M)$ of $\bar{B}$ succeeds, our two RW and RO witnessing operations succeed on R^0_B, and it computes $\mathfrak{m}_B$ correctly. We acknowledge that V_0 constructing the candidate receipt $\mathfrak{C}_B$ remains technically a probabilistic algorithm like Prove. We do not require that V_0 rerun these checks exactly how subsequent validators do currently, but active research might change this eventually.

[1] We envision $\mathcal{V}_\rho$ being small enough that BLS signatures do not improve verification time over Schnorr signatures, although BLS might reduce the candidate receipt's signature from 640 bytes down to 50 bytes.

Announcement: We expect this validation and gossip protocol to enlarge of the attestation signature sets S for any valid candidate receipt $\mathfrak{C}_{B,S}$, under some network assumption on $\mathcal{V}_\rho \cup \{C\}$.

If S accumulates at least $\texttt{min}_\texttt{b}\texttt{ackers}$ backing attestations from distinct validators in $\mathcal{V}_\rho$, then we publish $\mathfrak{C}_{B,S}$ for relay chain block producers using relay chain gossip (mempool). If however S remains too small too long, then validators in $\mathcal{V} \setminus \mathcal{V}_\rho$ ignore $\mathfrak{C}_{B,S}$.

We think $\texttt{min}_\texttt{b}\texttt{ackers} = \frac{1}{2}n_0$ gives a reasonable choice, but our security analysis only requires that enough stake back B, so even $\texttt{min}_\texttt{b}\texttt{ackers} = 1$ suffices provided each backing validator possesses enough stake.

We archive B if another conflicting blocks gets finalised by GRANDPA, eventually deleting it. We archive but probably do not delete B if GRANDPA is stalled, probably even when the fork choice rule favours other forks.

Slashing: At any time, if any two validators disagree about a parachain block's validity then all validators shall check the block, which requires first gossiping some negatively attested candidate receipt. In this case, we accumulate votes until $f + 1$ claim validity or invalidity, and then slash the loosing side. We cannot slash if neither side reaches $f + 1$, but we still declare the block invalid in that case. We expect governance to identify software faults and manually revert slashes they cause, but governance can also manually institute slashes in this second case, or manually slash ρ for offenses like malicious code or improper non-determinism.

2.3 Relay chain authorship

We announce attested (abridged) candidate receipts $\mathfrak{C}_{B,S}$ in two relay chain transaction types. We first provide a *candidate backed transaction* that merely demonstrates enough backing attestations $\mathfrak{C}_{B,S}$ so that availability work begins, as described below in §3. AT the end of this, we post a *candidate available transaction* that begins the approval checks and attestations, as described below in §4.

We consider a **relay chain block producer** $U \in \mathcal{V}$ has an upcoming relay chain block production slot in which U shall make a relay chain block R. If the following two condition hold, then U considers adding a *candidate backed transaction* to R announcing our attested candidate receipt $\mathfrak{C}_{B,S}$.

- $\mathfrak{C}_{B,S}$ has at least $\texttt{min}_\texttt{b}\texttt{ackers}$ backing attestations in S. We note that U should continue collecting the signatures S on $\mathfrak{C}_{B,S}$ while waiting its block production slot.[2]
- Some ancestor $R' \leq R$ contains a candidate available transaction with candidate receipts $\mathfrak{C}_{B',S'}$ for the parent parachain block B' of our candidate $\mathfrak{C}_{B,S}$
- If another relay chain block R'' between R and R' that includes another candidate backed transaction for another $\mathfrak{C}_{B'',S''}$ then no relay chain block between R'' and R contains a candidate available transaction for $\mathfrak{C}_{B'',S''}$, and some expiration condition applies to $\mathfrak{C}_{B'',S''}$. We leave the expiration conditions to future work, but one example would be R'' being 64 blocks before R.

Importantly, U need not validate B itself unless $U \in \mathcal{V}_\rho$. In principle, we could make U check the RW state commitments $q_{\mathsf{in}}, q_{\mathsf{out}}$ for ρ and RO state commitments $\{q_{\rho'}\}$ for other parachains, but doing so seemingly adds nothing.

[2] See BABE for more details on block production.

2.4 Notes

...

We already handle candidate receipts off-chain among $\mathcal{V}_\rho$, although perhaps their votes could be collected on-chain too. We therefore doubt further significant optimisations exist for this subprotocol.

Role churn: We again caution that parachain validator assignments $\mathcal{V}_\rho$ change somewhat frequently and worse our entire validator set $\mathcal{V}$ changes periodically too. We place no future demands upon the collators or relay chain block producer after block creation, so their churn creates no problems. We shall require that backing attesters in $\mathcal{V}_\rho$ continue to share $\bar{B}$, both in its entirety and in smaller pieces described next in §3, which makes churn problematic. Above in §2.2, we also suggested that backing attesters in $\mathcal{V}_\rho$ continue to share the outgoing XCMP messages.

As noted above, any candidate receipt $\mathfrak{C}_{B,S}$ uniquely identifies $\mathcal{V}_\rho$ because it includes $\rho.\mathsf{id}$ while the recent relay chain parent $H(R_B^0)$ determines the assignment $\rho.\mathsf{id} \mapsto \mathcal{V}_\rho$. We expect this addresses churn in $\mathcal{V}$ too because the nothing-at-stake problem imposes a long unbonding period, during which time we may continue rewarding outgoing validators for work, even if they lie in the old $\mathcal{V}_\rho$ but not the new $\mathcal{V}$. An implementations should rotate parachain validator assignments gently with heuristics that permit gossip somewhat after the assignments and even last minute backing attestations by validators in $\mathcal{V}_\rho \setminus S$.

As an aside, one could consider backing attesters $\mathcal{V}_\rho$ coming form outside the validator set $\mathcal{V}$, but doing so more requires more complex staking logic.

Para-threads: ...

3 Availability

In this section, we describe the subprotocols that provide and prove availability and thus enable our approval validity checks in §4.

We retain all definitions and notation established previously in §2. In particular, we have a candidate B for a parachain ρ for which we defined

- its unauthenticated erasure coded pieces $\mathcal{D}'_B$,
- their Merklee root $\mathfrak{m}_B$,
- its attested candidate receipt $\mathfrak{C}_{B,S}$ for B contining $\mathfrak{m}_B$ along with attestation signatures S.

3.1 Authenticated pieces

We thus far in §2 communicated only candidate receipts $\mathfrak{C}_{B,S}$ and candidate blobs $\bar{B}$ among nodes, but availability requires sending erasure coded pieces $\mathcal{D}'_B$ too.

Associated to each unauthenticated piece $d \in \mathcal{D}'_B$, there is a Merklee copath $\overrightarrow{\mathfrak{m}_B d}$ that proves d was committed to by the Merkle root $\mathfrak{m}_B$. We define *authenticated (erasure coded candidate) pieces* $(d, \mathfrak{m}_B, \overrightarrow{\mathfrak{m}_B d})$ by attaching to d the Merkle root $\mathfrak{m}_B$ and this inclusion proof

$\overrightarrow{rd}$. We transport only authenticated pieces between nodes throughout, or full blocks like $\bar{B}$, because only authenticated pieces provide an irrefutable claims about candidates.

We let $\mathcal{D}_B$ denote the list of these authenticated pieces, so

$$\mathcal{D}_B = Listst(d, \mathfrak{C}_{B,S}, \overrightarrow{\mathfrak{m}_B d}) d \in \mathcal{D}'_B,$$

with each piece signed by one member of S. We shall distribute $\mathcal{D}_B$ among the full relay chain validator set $\mathcal{V}$ with $\mathcal{D}_B[i]$ going to $\mathcal{V}[i]$ for $i = 1, \ldots, n$.

In so doing. we force the backing checker set S into making $\bar{B}$ available for testing by random approval checkers without yet revealing those approval checkers. This trick provides the core scalability advantage of Polkadot.

We might however see many competing parachain candidate blocks at this point, so we delay this distribution process until some relay chain block R contains $\mathfrak{C}_{B,S}$. We assume such an R containing $\mathfrak{C}_{B,S}$ exists throughout the remainder of this section.

3.2 Topology

We find that scalability actually depends heavily upon the topology and routing used to distribute data, but that specifics depend upon the scale. We briefly explain the specialised topology and routing requirements for our two phases of parachain data distribution. We caution however that routing almost always requires some capacity for multi-hop forwarding because it otherwise risks excluding some elected validators.

Candidate blocks: We need our network topology to permit one parachain ρ to distribute the authenticated erasure coded candidate pieces $\mathcal{D}_B$s relatively quickly, which amounts to a graph expansion property We might want a much stronger connectivity property below if doing full reconstructions turns out to be our most efficient mechanism for block retrieval, but if not then some similar expansion property permits forwarding pieces for reconstructions (see §4.1.1).

In §2, we could ask collators to send the same block to several well chosen parachain validators as well. As an example, if our parachain validators $\mathcal{V}_\rho$ form a cycle then B reaches all parachain validators in two hops if the collators send B to one third of $\mathcal{V}_\rho$, but adding well chosen chords reduces our connections with collators and improves our connectivity.

Candidate pieces: All parachain validators in $\mathcal{V}_\rho$ must compute all of $\mathcal{D}'_B$ to compute $\mathfrak{m}_B$, from which computing $\mathcal{D}_B$ too costs nothing. We should therefore divide the distribution burden as equally as possible among parachain validators in $\mathcal{V}_\rho$. We also prefer if the topology is symmetric in the sense that the links over which ρ_1 sends to validators in ρ_2 are the same as the links over which ρ_2 sends to validators in ρ_1.

There are simple topologies that satisfy these requirement:

We start with some fixed symmetric topology $\bar{\mathcal{T}}$ on the set of parachains, preferably a complete graph, i.e. diameter one. In $\bar{\mathcal{T}}$, we equip each edge $\overline{\rho_1\rho_2} \in E(\mathcal{T})$ with a regular bipartite graph $\mathcal{T}_{\overline{\rho_1\rho_2}}$ between its two parachains ρ_1 and ρ_2. We then define the edge set of $\mathcal{T}$ to be the union of the edge sets of all $\mathcal{T}_{\overline{\rho_1\rho_2}}$ for $\overline{\rho_1\rho_2} \in E(\mathcal{T})$

$$E(\mathcal{T}) = \cup \{ \mathcal{T}_{\overline{\rho_1\rho_2}} \mid \overline{\rho_1\rho_2} \in E(\mathcal{T}) \}$$

We get symmetry from the $\mathcal{T}_{\overline{\rho_1\rho_2}}$ being undirected. We distributes work equally by regularity of the $\mathcal{T}_{\overline{\rho_1\rho_2}}$. We could weaken this to semiregular if we assign different numbers of validators to different parachains.

We note one simple example for $\mathcal{T}_{\overline{\rho_1\rho_2}}$: Assume each parachain ρ has an associated value $\rho.\mathsf{seed}$. We let $\mathsf{parashuffle}(\rho_i, \rho_{3-i})$ denote the Fisher-Yates shuffle of $\mathcal{V}_{\rho_i}$ seeded by $H(\rho_i.\mathsf{seed}, \rho_{3-i}.\mathsf{seed})$. We define $\mathcal{T}_{\overline{\rho_1\rho_2}}$ on the $\mathcal{V}$ by connecting $\mathsf{parashuffle}(\rho_1, \rho_2)[j]$ to $\mathsf{parashuffle}(\rho_2, \rho_1)[j]$ for $j = 1, \ldots, n_0$.

Any block production epoch e contains many shorter epochs (e, k) in which we compute some parachain validator assignment $\rho.\mathsf{id} \mapsto \mathcal{V}_\rho$. We could however leave this topology abstract and reposition the actual assignments indices to validators and/or parachains.

We spoke of ρ_i being a parachain, but really ρ_i acted like an arbitrary validator grouping, so our actually parachains could be reshuffled freely among these groups. We thereby supports fast churn without disrupting validator groupings.

We can similarly reshuffle the mapping from actual validators to validator indices: We let $\mathcal{V}_{e,k}$ denote the Fisher-Yates shuffle of $\mathcal{V}$ seeded by by $r_e||k$. We then simply apply some fixed mapping from indices to parachains, such as $\lfloor k/n_0 \rfloor$.

We can adapt this scheme to varying $|\mathcal{V}_{\rho_i}| \geq n_0$ quite easily if not all parachains have the same number of assigned validators, i.e. if n_0 is not tight. We can also do additional shuffles if more than one link is desired. If two linked nodes cannot connect, then any still online attempt connections with some random other nodes from the other parachain, or perhaps use some smarter scheme.

As an unoptimised example, assume $\mathcal{T}$ is a complete graph: After our parachain validator V of ρ observes some relay chain block R containing $\mathfrak{C}_{B,S}$ then, for all other parachains $\rho' \neq \rho$, V computes the $i_{\rho'}$s such that $V = \mathsf{parashuffle}(\rho, \rho')[j]$ and $\mathcal{V}[i_{\rho'}] = \mathsf{parashuffle}(\rho', \rho)[j]$ for some $j \leq n_0$, and V send $\mathcal{D}_B[i_{\rho'}]$ to $\mathcal{V}[i_{\rho'}]$ directly using QUIC. We expect $\mathcal{V}[i_{\rho'}]$ might have some piece from ρ' for V too, thanks to the symmetry of our $\mathsf{parashuffle}$ criteria. We expect this symmetry to reduce the required connections by almost a factor of two.

We have described this as V initiating the connection, but a similar procedure works for V requesting its piece for some ρ' block, or symmetrically $\mathcal{V}[i_{\rho'}]$ requesting $\mathcal{D}_B[i_{\rho'}]$. In fact, an initial implementation should focus upon requests because as noted above we shall request from other validators when our first choice fails.

If $\mathcal{V}[i_{\rho'}]$ cannot reach V then $\mathcal{V}[i_{\rho'}]$ must select some backup node to replace V. Assuming $\mathcal{T}$ is complete, we should distribute these evenly among $\mathcal{V}_\rho \setminus \{V\}$, so as one option $\mathcal{V}[i_{\rho'}]$ could perform a Fisher-Yates shuffle of $\mathcal{V}_\rho \setminus \{V\}$, seeded by its own identity $\mathcal{V}[i_{\rho'}].\mathsf{pk}$ and $\rho.\mathsf{seed}$, and then contact those remaining parachain validators in the resulting order. We caution this option breaks the topology's symmetry, so as noted above nodes might exploit whatever links work first, and only take guidance from this shuffle when creating new links. If $\mathcal{T}$ is not complete then alternative approaches that choose another parachain work too.

If $\mathcal{T}$ is not complete then intermediate nodes must forward authenticated pieces for other nodes. In fact, the diameter of $\mathcal{T}$ equals the maximum number of hops required for $\mathcal{T}_e$ to distribute each piece. In practice, these edges in $\mathcal{T}_e$ ro be the two nodes maintaining a UDP protocol like QUIC connection because UDP should permit higher valency than TCP and hence permit a lower diameter $\mathcal{T}$. We should evaluate other topologies besides $\mathcal{T}_e$ before going beyond complete $\mathcal{T}$ proves necessary, but our symmetry property provided by $\mathcal{T}_e$ remains important.

We admitted adversarial manipulation of our network topology here, but it sounds acceptable for our availability scheme, at least with $\mathcal{T}$ complete. We shall consider whether this

impacts gossip protocols in future work.

3.3 Votes

We consider a candidate receipt $\mathfrak{C}_{B,S}$ backed in some recent relay chain block R.

An honestly erasure coded $\bar{B}$ can be recovered from any $f+1$ pieces in $\mathcal{D}_B$. If not honestly encoded, then our reconstruction algorithm $\mathsf{Decode}_{k,n}$ still produces data from any $f+1$ symbols.

We assume throughout that $2f+1$ validators behave correctly and at most f validators behave byzantine. It follows that, if $2f+1$ validators claim they possess their piece, then anyone could later reconstruct an honestly erasure coded $\bar{B}$ from the remaining $f+1$ held by validators who claimed correctly, much like other byzantine fault tolerant voting processes.

At this point, our validators votes that they believe candidate to be available once they received their authenticated piece $(d, \mathfrak{C}_{B,S}, \overrightarrow{\mathfrak{m}_B d}) \in \mathcal{D}_B$ and validate the signature and the witness copath $\overrightarrow{\mathfrak{m}_B d}$ to the merkle root $\mathfrak{m}_B$ in $\mathfrak{C}_{B,S}$. We shall broadcast these *availability votes* via gossip and accumulate them in relay chain blocks, almost like transactions, but we briefly explain their implications first before explaining the voting mechanics.

Security: We wait upon these availability votes before our approval checkers announce their assignments in §4.2 or run their assigned checks in §4.1, which must happen before our finality gadget GRANDPA considers R in §4.2.8.

It follows that approval checkers should all receive enough pieces to run the reconstruction algorithm $\mathsf{Decode}_{k,n}$. At this point, an approval reruns the erasure coding algorithm $\mathsf{Encode}_{k,n}$ to recomputes all $\mathcal{D}'_B$ and $\mathfrak{m}_B$. If this fails, they declare the candidate receipt $\mathfrak{C}_{B,S}$, which eventually results in slashing S.

An invalid erasure coding should therefore by detected slashed, assuming both that $2f+1$ validators behave honestly and that some honest validator approval checks $\mathfrak{C}_{B,S}$. We shall address this second assumption in later sections. We note here however that approval checkers must reveal themselves to obtain the block, but any who announce early provide far less security, which makes it critically importnat they wait until the availability votes. In concrete term, any adversary could adaptively choose not to make their block available if they observed an approval checker they believe honest or unassailable.

Execution: We implement some of the protocol described here within the relay chain's state machine, which carries out tasks like record backed candidates and availability votes. There are also actions in the relay chain associated to execution via $\mathsf{Verify}_\rho(\bar{B}_0, M)$ of the parachains candidate, at minimum updating the parachain's head and state root q_ρ. We discussed in §2.2 that V_0 computes these actions on the relay parent before the block producer even creates R, which makes the relay chain into a lite client of its past state. We ensure these updates can be applied later, but we must apply them eventually however.

We expect availability votes could pass for some candidate receipts in R, while failing for other candidate receipts in R. We do not delay the available parachain while waiting for the unavailable one. As some parachain's availability might stall indefinitely, we impose some time limit, or other criteria, after which backed candidate receipts expire, so the relay chain can forget about the unsuccessful candidate, and new candidates can be accepted from the parachain and its validators.

In other words, we freely replace backed candidates that do not achieve availability quickly enough. We therefore cannot regard the candidate receipt $\mathfrak{C}_{B,S}$ as acting in the relay chain block R in which its backing transaction appears.

Instead we have $\mathfrak{C}_{B,S}$ act only in the relay chain block $R' > R$ that finally declares it passed the $2f + 1$ availability votes, meaning the relay chain block producer for R' applies the update to the parachain's head, state root q_ρ, and possibly other data. At minimum R lies strictly between R' and the relay parent of $\mathfrak{C}_{B,S}$, so good pipelining for parachain blocks depends upon them building upon one another correctly before they appear on chain.

Mechanics: We should "aggregate" the signatures for availability votes because most validators vote for most candidates. As a simple but efficient scheme, validators' signed votes contain a bitfield with one bit for each candidate and indicate the relay chain height that maps bits to candidates.

We could indicate the map from bits in the bitfield to candidates with a relay chain block hash of course. We might however do so more efficiently using a relay chain state root directly, so then relay chain block producers include the availability vote by providing updated witnesses. ... In short, we yet again make the relay chain a lite client of its own past state.

We must rotate which parachain validator groups back which parachains, so there exist two natural strategies foe allocating bits in this bitfield to represent candidates:

We want parachains that produce candidates for many relay chain blocks. Any such busy parachain might control one bit in this bitfield for an entire epoch, so its currently backed candidate receipt can be looked up directly in the relay chain state.

We also want parachains called parathreads that produce candidates only very rarely after winning some auction. We cannot permanently allocate one bit for every parathread, so instead we temporarily assign parathread candidates a bit tied to their parachain validator group.

We think this second parathread mapping would impose unneeded waiting for full parachains when they rotate between parachain validator groups, so implementing both techniques sounds easiest.

3.4 Notes

Initially, we explored designs in which availability votes happened primarily off-chain, but these incur significant complexity and demand more subtle security assumptions. Also, these ideas roughly parallel existing side chain work so some provide only slow on-chain resolution.

We expect fine tuning the signature aggregation mechanism might yield better results:

We could aggregate using BLS signatures, so that verifiers only check one signature per candidate. We want roughy one tenth as many candidates as validators however, so the overall higher cost form BLS overuns any savings here. We do only require $2f + 1$ signers, so perhaps clever optimisations could gives extremely efficient BLS signatures that signers who complicate verification, but this sounds like subtle future optimisation work.

We expect most direct optimisation would be Rabin-Williams signatures that increased total on-chain vote size tenfold, but only double voting bandwidth, but improved verification performance by a larger 10-100 factor, so Rabin-Williams signatures warrant further investigation.

There could even be validator groupings that negotiates Schnorr multi-signatures too, but this remains soundly future work.

4 Approval

We now describe the subprotocol that provide the approval validity checks required before finality.

We preserve the previous notation from §2 and §3:

Any parachain block B on a parachain ρ has a candidate proof-of-validity blob $\bar{B}$ that contains the witness proofs for any parachain state interactions, as well as an associated candidate receipt $\mathfrak{C}_B$ that contains the Merkle root $\mathfrak{m}_B$ for the Merkle tree with leaves consisting of the erasure coded pieces $\mathcal{D}'_B$ of $\bar{B}$. We distributed the authenticated pieces $\mathcal{D}_B$ that consist of one piece from $\mathcal{D}'_B$ and its witness copath for $\mathfrak{m}_B$.

We handle attested candidate receipts $\mathfrak{C}_{B,S}$ that consist of its abridges candidate receipt $\mathfrak{C}_B$ and validity attestations. We first in §2 posted candidate receipts $\mathfrak{C}_{B,S}$ on chain once they received sufficient backing. We then in §**??** accumulated on-chain the availability attestations for them. We finally recognized availability on-chain once this reached $2f + 1$, which determines when candidate parablocks actually interacts wit the relay chain.

In this section, we consider a relay chain block R in which several candidate receipts $\mathfrak{C}_{B,S}$ were declared available and executed. In §2, we freely announced $\mathfrak{C}_{B,S}$ on chain without impacting relay chain validity because doing so interacted with nothing. As B executes in R however, we cannot consider R valid unless we consider B valid, but so far we have validity attestations for B only from $S \subset \mathcal{V}_\rho$, and $\mathcal{V}_\rho$ is a tiny and dangerously foreseeable set of validators. We discuss here the approval phase for R in which an unforeseeable selection of approval checkers provide attestations.

The approval phase for R has two components, approval checker assignments and actually performing the checks. We have three flavours of approval checker assignments, which all roughly follow the pattern of

- first VRF announcements via gossip, and
- then computation of approval checker assignments.

All approval checker assignments trigger the same approval check protocol, which consists of

- retrieval and reconstruction of full candidate proof-of-validity blobs,
- running approval checks on the candidate proof-of-validity blobs, and
- announcing these candidates validity or invalidity.

We describe this second component first because ???why???.

In spirit, candidate validity attestations could come from any validator, and might ideally be saved, due to being slashable, but we only interpret these signatures as relevant towards specific goals depending upon the validator's role, like backing or approval, when they come from nodes with the appropriate role assigned. We shall discusses invalidity claims and unavailability complaints by non-validators, like collators and fishermen, in the next section.

4.1 Checking

In this subsection, we consider a validator V who passed some approval checker assignment criteria subphase: First, V has constructed and announced via gossip some approval checker VRF signature $\pi_{V,\cdot}$. Second, there exists a parachain ρ determined by the specific approval checker assignment criteria applied to $\mathsf{VRF.Out}(\pi)$. We let B denote the parachain candidate B for ρ in R and $\mathfrak{C}_{B,\cdot}$ its candidate receipt.

We need only information posted on-chain to validate the VRF signatures $\pi_{V,\cdot}$ for approval checker assignment criteria, but some assignment criteria employ some off-chain information. We employ absence from the chain occasionally, which seems unproblematic. We might package equivocation proofs (see below) with the VRF signatures, depending upon their size. We always prevent late stage adversarial influence from shifting approval checker assignments by making assignments strictly additive, and always in batches.

4.1.1 Retrieval

We never consider substructure of B to be meaningful, so V must *retrieve* the full *candidate proof-of-validity blob* $\bar{B}$ before checking. Now V knows which which nodes have their individual pieces, thanks to their availability announcements. It thus follows from our $2f + 1$ honesty assumption that V could always reconstruct $\bar{B}$ by obtaining $f + 1$ pieces $\mathcal{D}_B$ from nodes known to posses them, with at most f dishonest nodes not responding.

We note however that V also knows that all pieces are known by the preliminary backing validity checkers aka parachain validators who approved $\bar{B}$, as well as approval checkers who already approved $\bar{B}$. So V could first contact some node that possesses all of $\bar{B}$, and only then begin a full reconstruction process.

In both cases, V must recompute $\mathcal{D}_B$ to verify $\mathfrak{C}_{B,\cdot}$. We therefore cannot see much computational difference between V reconstructing $\mathcal{D}_B$ from arbitrary pieces or from $\bar{B}$ itself. It remains plausible V avoids some networking overhead by asking for $\bar{B}$ though. We think a first implementation should reconstruct $\mathcal{D}_B$ from arbitrary pieces, while leaving requests for the full $\bar{B}$ as future optimisations.

Ideally V might retrieve the pieces in $\mathcal{D}_B$ only using its existing connections in our topology specified above, except these intentionally do not include 1/3rd of validators. Also, V need not connect to any node with all of $\mathcal{D}_B$. Interestingly, V already connects to at least one prachain validator in $\mathcal{V}_\rho$ who often checked B first, which indicates that transferring the full blob $\bar{B}$ may prove optimal.

We caution against abandoning approval checkers over topology concerns however because then adversarial influence over the topology could wreck our assignment criteria below.

Also, our retrieval component could be engineered to avoid requests entirely: After obtaining $\pi_{V,\cdot}$, another validator V' could simply compute its own priority for sending its piece from $\mathcal{D}_B$ to V. We caution that doing do might become inefficient, either because V winds up rejecting sends, or when many nodes go offline.

4.1.2 Announcement

After V obtains enough pieces to recompute and verify $\mathfrak{C}_{B,\cdot}$ from $\bar{B}$, then V verifies $\bar{B}$ itself by running the execute block function of ρ. If both these checks pass, then V signs $\mathfrak{C}_{B,\cdot}$ and announces its signature via gossip. If either check fails, then V claims invalidity for $\mathfrak{C}_{B,\cdot}$, which results in all validators checking B.

4.2 Assignments

We consider self-assignment schemes determined by several paramaters:

- our target number ℓ of eventual checkers,
- our total number n' of candidate checkers, normally $n' = n - n_0$,
- the weight θ of a no-show checker with $1 \leq \theta \leq 2$,
- a network delay bound Δ_{net} for gossip messages, and
- a block checking delay bound Δ_{check} that covers both retrieval and validation.

We never split the check bound Δ_{check} into separate retrieval and validation bounds, because doing so could only improves latency in extremely niche situations, and maybe the extra gossip costs more.

4.2.1 Verifiable random sampling

We propose self-assignment schemes based on *verifiable random sampling* using a VRF:

Any protocol run requires some seed z and some associated time T_0 that marks the protocols' start time. As a rule, we select z to satisfy some freshness assumption, meaning our adversary learns z before some time T_0 with odds less than our ambiant security treshold f/n.

Any validator V with key pair $(\mathsf{pk}, \mathsf{sk})$ computes its VRF signature $\tilde{\pi}_{V,z} := \mathsf{VRF.Sign}_{\mathsf{sk}}(z)$, which it announces when specified by the specific protocol, but not before T_0. Anyone else verifies $\tilde{\pi}_{V,z}$ with pk. All participants seed their sampling procedure for V's assignment with the verifiable randomness $\tilde{\omega}_{V,z} := \mathsf{VRF.Out}(\pi_{V,z})$.

We prefer z already be known to verifiers, but our protocols run far too fast to enforce this. We expect premature gossip messages that arrive before the verifier knows z require subtle caching and expiration policies, but some scenarios provide a short proof for z, which may or may not be worth including in gossip messages.

We remark that gossip messages could be signed with π itself by incorporating the output into the full gossip message, which gets signed as extra message data. If done, this reduces the computational cost for signature verification by a third to a half, but may violate protocol layering and complicate validating protocol security and correctness, audits, etc.

We always need a target number ℓ of eventual checkers when sampling assignments based on $\tilde{\omega}_{V,z}$. We shall often alter ℓ during protocol runs though, so assignments must be covariant in ℓ, meaning increasing ℓ should not invalidate assignments made with smaller ℓ. We avoid rejection sampling after ℓ enters the sampling computation for this reason.

4.2.2 Generic one-shot delay

We first define a generic "one-shot" self-assignment subprotocol in which the seed z uniquely identifies a parachain candidate B on some parachain ρ. We term this a "one-shot" protocol in that validators reveal their assignment at the last possible moment before checking, which costs us efficiency but gives code simplicity and completeness. It also gives us a fig leaf of defense against some adaptive adversary classes. We describe this subprotocol generically

because it acts as a basis for several self-assignment subprotocols, thanks to being one-shot, and should enable considerable code reuse.

All validators track the set A_z of valid announced checker claims, as well as the subset S_z of all validity claims S_B for B generated by nodes in A_z. We identify here the generic one-shot protocol run by z because multiple z may imply the same B, except we emphasise that messages exist only for S_B not S_z, and that our protocol run ends by considering S_z sufficient.

We uniformly sample discrete time delays from $\{\Delta_{\mathsf{net}}, \Delta_{\mathsf{net}} \frac{n'}{\ell}\}$, so that an expected ℓ receive delay Δ_{net}. We use verifiable sampling here of course, meaning we compute this delay $d_{V,z}$ from $\tilde{\omega}_{V,z}$.

As noted above, we shall alter ℓ during protocol runs, which prevents involving it too directly. Instead, we let ℓ_0 denote the initial value of ℓ to define

$$d_{V,z} = \Delta_{\mathsf{net}} \lfloor \frac{n'}{\ell_0 m} \omega_{V,z} \rfloor$$

where m is the upper bound on $\omega_{V,z}$, i.e. $\omega_{V,z} \in [0, m)$. Integer division without rejection sampling suffices because checker with high delays never check or even announce their values. In this, we implicitly condense checking assignments into discrete delay tranches. In particular, all checkers with $d_{V,z} = 0$ check at time T_0, which prevents powerful adversaries from forcing nodes before them. In Rust, we might sample with code resembling

```
fn generic_oneshot_assigned_delay(
    ctx: &'static [u8],
    network_delay: u32,
    candidates: u32,
    target: u32,
    omega: ::schnorrkel::vrf::VRFInOut,
) -> u32 {
    assert!(network_delay > 0); // Unrealistic network otherwise
    assert!(candidates > 0 && target > 0); // No candidates, not even us.
    let mut s = u32::from_le_bytes( omega.make_bytes(ctx) ) as u64;
    s *= (candidates as u64) * (network_delay as u64);
    s /= (u32::value_max() as u64) + 1;
    s.saturating_sub(network_delay as u64)
    .try_into::<u32>().expect("candidates * network_delay > u32::value_max()")
    / target
}
```

Consider a validator V with current time t. We describe how V tracks the announcements A_z and their answering validity claims S_z, and deals with no-shows.

V records the time t_a it receives each checker announcement $a \in A_z$. Also V defines its no-show count ℓ' to be the number of $a \in A_z$ for which both $t_a + \Delta_{\mathsf{check}} < t$ and the owner $V' \in \mathcal{V}$ of a does not yet appear in S_z. As input, we take a no-show suspicion factor θ' with $0 \le \theta' \le 2$, by which we multiply ℓ' below.

Any validator V announces via gossip their $\tilde{\pi}_{V,z}$, and begins retrieving and checking B, whenever

1. $T_0 + d_{V,z} + \Delta_{\mathsf{net}} < t$,

2. $|A_z| < \ell + \theta\ell'$, and

3. $V \notin \mathcal{V} \setminus \mathcal{V}_\rho$.

In some protocols, we do increase ℓ to compensate for absent parachain validators in $\mathcal{V}_\rho$, so revealing a validity claim from a $V \in \mathcal{V}_\rho$ may reduce ℓ, but such a claim never counts towards S_z.

Also V considers the block B valid whenever $|S_z| \geq \ell + \theta\ell'$, but one-shot subprotocol cannot terminiate themselves because their caller might later increase ℓ. We ask the caller manage prioritisation for work on z and B.

4.2.3 Initial:

We recall that a relay chain block R always has an associated VRF signature $\hat{\pi}_R$ whose output $\hat{\omega}_R := \mathsf{VRF.Out}(\hat{\pi}_R)$ gets recycled for the random beacon, and perhaps seals the block R itself. In Praos [**?**] or BABE [**?**], $\hat{\omega}_R$ equals the VRF output ω_R that justifies creating R, but secret single leader elections like Sassafras [**?**] might reveal ω_R early. We want $\hat{\omega}_R$ to be the random beacon source that only the block producer of R knows in advance, and that even they can only influence by choosing wether or not to make R.

We run the initial approval checker assignments that apply to a candidate receipt $\mathfrak{C}_B$ based on $\hat{\omega}_R$ where R declares $\bar{B}$ available, as well as the parachain ρ of B. In this way, we minimize adversaries' influence the $\hat{\pi}_R$ used. In fact, adversaries could always withhold availability reports until their planned $\hat{\pi}_R$, but delaying approvals sounds more complex than delaying backing votes.

An adversary wants to influence $\hat{\pi}_R$ to give their B at least ℓ?? dishonest approval checkers. We determine assignments using the assigned parties' VRFs, so adversaries cannot prevent honest checkers being assigned too, but if they force many low delays then approval checkers with higher delays never check.

4.2.4 Initial one-shot:

We describe using the one-shot self-assignment subprotocol: All validators run the one-shot self-assignment subprotocol for each parachain ρ declared available in R using the seed $z_B := \hat{\omega}_R || \rho.\mathsf{seed}$.[3] We outline computing the associated target number ℓ_B of eventual checkers for the one-shot assignment subprotocol:

As discussed previously, our approval phase begins only after acquiring enough preliminary backing parachain validators from ρ. We suppose here that n_0 preliminary backing checks to always be realistic, making fewer than n_0 preliminary backing checks suspicious. We set ℓ'_B to be the number $|\mathcal{V}_\rho|$ of parachain validators for ρ minus $|S \cap \mathcal{V}_\rho|$.

We set the no-show suspicion factor θ' in the range $0 \leq \theta' \leq 2$, as noted above. In additions, we accumulate several dynamic report factors, like an unavailability report factors ℓ_v from validators and ℓ_a from collators, and an invalidity report factor ℓ_f from both collators and untrusted fishermen. We set the target number ℓ of approval checkers to be the sum of these, along with any missing preliminary backers $n_0 - \texttt{min}_\texttt{b}\texttt{ackers}$. So

$$\ell_B := \ell_b + \lceil \theta' \ell'_B + \ell_v + \ell_a + \ell_f \rceil .$$

[3] We write z_B instead of $z_{R,\rho}$ because parachains have only one block declared available per relay chain block.

Implementations might slightly delay accounting for $\theta'\ell_\rho$, and possibly other factors, to prevent ℓ from being overly large initially.

4.2.5 Initial modulo:

In one-shot schemes, all validators have separate z_B for each parachain ρ represented in R, which results in statistically independent VRF signatures $\hat{\pi}_{V,B}$ for each ρ, which remain secret until R declares $\mathfrak{C}_{B,S}$ available. Independence here protects against adversaries arranging that all checks of B be run by their own validators.

As a cost to independence, our candidate B might receive too few or too many approval check assignments. We partially address the too few case with our delay system, which improves the too many case indirectly, but delays admits some adversarial manipulation. We avoid more complex prioritization rules because reducing independence between validators admits adversarial manipulation.

In the same vein, any individual validator could similarly receive too few or too many approval check assignments. We gain little from individual validator's initial assignments being independent across parachains. We therefore sacrifice independence across parachains for a more balanced distribution.

We ask that each V announce one $\tilde{\pi}_{V,R} := \mathsf{VRF.Sign}_{V.\mathsf{sk}}(\hat{\omega}_R)$ via gossip for R as soon as it considers R to be available. In other words, we make $\hat{\omega}_R$ alone the seed to obtain an output $\tilde{\omega}_{V,R} := \mathsf{VRF.Out}(\tilde{\pi}_{V,R})$ associated to R, not any particular B.

Now let n_A denote the number of availability cores. Also let A_R be the map from $\{1, \ldots, n_A\}$ to parachains declared available in R. We assign V to check the parachain $A_R(\tilde{\omega}_{V,R} \mod n_A)$ with a delay of zero.

In other words, any validators V assigned by their $\tilde{\pi}_{V,R}$ check before any nodes assigned by their delay. We should retain the delay checkers with a delay of zero, primarily because we cannot prevent availability cores from going unused, so that A_R returns nothing for most checkers.

All nodes should announce their $\tilde{\pi}_{V,R}$, and then run their assigned check. Any abdications signal nodes being offline as a useful side effect.

4.2.6 Equivocation:

We say two distinct relay chain blocks R and R' are *equivocation* for one another if they use the same VRF output $\hat{\omega}_{R'} = \hat{\omega}_R$ for the relay chain randomness. We say R has an equivocation R' in this case, or just has equivocations to merely express the existence of R'. We distrust relay chain blocks with equivocations of course, and must mark them as having equivocations, but we sadly cannot invalidate them because doing so creates attacks on finality.

We therefore fear an adversary might create an R' with an innocent B' on ρ merely to learn the approval checkers determined by $\hat{\omega}_R$ for ρ, but then equivocate with another R that contains a malicious B on ρ. In this scenario, our equivocation R triggers exactly the same approval checks for B as R' did for B', which wrecks our advantage over the adversary.

We say a candidate receipt $\mathfrak{C}_{B,S}$ has *equivocations* if a relay chain block R that declares $\mathfrak{C}_{B,S}$ available has an *equivocation* R' that declares a (conflicting) candidate receipt $\mathfrak{C}_{B',S'}$ available.

An equivocation among candidates requires both a relay chain fork and a relay chain equivocation, which makes them extremely suspicious. We thus want additional checks that

remain unforeseeable to our adversary. At the same time, we cannot prevent validator operators from running poorly designed custom key management infrastructure, either to mitigate internal threats or for high availability, which makes relay chain equivocations and even candidate equivocations plausible. We should therefore address them without triggering excessive checks against innocent parachain candidates under repeated equivocations.

We find this compromise as follows: If $\mathfrak{C}_{B,S}$ has equivocations, then we instantiate the generic one-shot self-assignment subprotocol with seed $z_B = H(B)$[4], so that each parachain candidate gets assigned only an extra checkers group only once. In other words, we work with VRF signatures $\tilde{\pi}_{V,B} := \mathsf{VRF.Sign}_{V.\mathsf{sk}}(H(B))$ and do verifiable sampling with the VRF outputs $\omega_{V,B} := \mathsf{VRF.Out}(\tilde{\pi}_{V,B})$, as opposed to the $\tilde{\pi}_{V,R,\rho}$ or $\tilde{\pi}_{V,R}$ from our initial checks.

In this one-shot application, we could choose our target number ℓ of approval checkers to be a constant ℓ_{eqiv}, perhaps equal to ℓ_b, or at the other extreme reuse our target number ℓ_B of approval checkers as defined for the initial one-shot variant. We suggest however that report factor from ℓ_B by reconsidered under the equivocations attack scenario.

In this way, an adversary who equivocates always faces ℓ extra checks against their new parachain block B, without impacting the innocent parachains too much when faced with numerous equivocations.

Implementers might suppress this check if parachains always carry less value than gets slashed when relay chain block producers equivocate. Attacks could impact multiple parachains simultaneously but doing so requires significantly more resources.

4.2.7 Rewards

We reward validators for both their assignment announcements as well as actually running their assigned approval checks. Incidentally, the initial modulo assignments strategy produces more uniform rewards than the one-shot strategy, although amortized rewards work out similarly.

We do not slash validators for not delivering their assigned checks, but ...

4.2.8 Finality

As noted above, all validators accumulate approval checks in the checks S of an attested candidate receipt $\mathfrak{C}_{B,S}$. All our assignment schemes come with applicability criteria, like R having equivocations, as well as fulfilment criteria, like $|S_z| \geq \ell + \theta\ell'$.

All validators would eventually get every approval assignment, which ensures we have enough assignments, and that no-shows eventually get replaced. As a result, any individual validator faces several obstacles to approving R, either

- R contains an available but invalid parachain candidate receipt $\mathfrak{C}_{B,S}$, which results in slashing its backers in S, or else
- R contains an unavailable candidate receipt $\mathfrak{C}_{B,S}$, which eventually redirects block production in §**??**.

We impose one further assignment scheme mentioned previously in §**??**: If any two relay chain validators ever give opposing validity attestations for some candidate B included in R, then all relay chain validators should produce approval validations for B. In this case, we

[4]If desired, replace $H(B)$ by the Merkle root $\mathfrak{m}_B$.

accumulate votes until $f+1$ claim validity or invalidity, and then slash the loosing side. We cannot slash if neither side reaches $f+1$, but we still declare the block invalid in that case. We expect governance to identify software faults and manually revert slashes they cause, but governance can also manually institute slashes in this second case, or manually slash ρ for offenses like malicious code or improper non-determinism.

We judge R approved when all applicabile assignment schemes appear fulfilled. Any validator V judges R to be *strongly available* when $2f+1$ validators including V itself announced availability claim for every candidate in R, meaning they claim they possess their own piece from $\mathcal{D}_B$. An approved and strongly available relay chain block R becomes eligible for consideration by our finality gadget GRANDPA.

5 Non-validators

UNFINISHED, ALL TEXT HERE LIFTED FROM PREVIOUS VERSION, SOME OF IT MIGHT BE WRONG.

5.1 Validators

Slashing: If at least $f+1$ validators sign for the invalidity of a block and less than $f+1$ validators sign for the validity of the block, then we slash all validators who signed for the validity.

If at least $f+1$ validators sign for the validity of a block and less than $f+1$ validator sign for invalidity, we slash the ones who signs for the invalidity.

We note that $f+1$ validity signatures and $f+1$ invalidity signatures are not possible given that at most f validators are malicious and proof of block algorithms are correct (See Definition 1).

Rewarding: All parachain validators and extra validators who signed for the final decision (either valid or invalid) of a block receive a reward.

Eligibility of the reward can be easily checked for the parachain validators and extra validators who satisfies the conditions when $\tau = 0$. Latter, we just need to verify the VRF proof. However, it is not easy to verify the condition for $\tau > 0$, because this value is subjective. Therefore, we may not need to be very precise for this τ value, if the signature of the validator is correct.

5.2 Fishermen

All fishermen collect blobs from collators in order to check the validity all the time. Whenever they discover an invalid block, they announce it to the validators by signing their claim. These claims are added to the block list by the validators. So, these claims are going to be in the relay chain. They do not provide any proof. If their claims are wrong then they are slashed. Otherwise, they are rewarded. The correctness of the fishermen claims is determined with the validity check protocol.

...

Slashing: If at least $\ell_{\text{TOTALREMOVED}}$ validators sign saying that the block is valid, then fishermen with the claim saying the block is invalid are slashed. All the claims from these fishermen are ignored in future.

Rewarding: If $f + 1$ validators sign to say the block is invalid, the fishermen with the claim saying that the block is invalid are rewarded.

5.3 Collators

We expect collators are free to act as fishermen .. except collator authority should probably be additive with fishermen authority

... unavailability reports from collators ...

6 Security arguments

Theorem 1 (Availability)**.** *Assuming the Merkle tree constructed from collision resistant hash functions and a block in GRANDPA can be finalized with at least* $2f + 1$ *votes, then the availability protocol is secure.*

Proof. Assume that an unavailable block is finalized. It means that this finalized block includes a block header of which at most f honest validators has the correct erasure code piece (See Definition **??**). If this block is finalized, it means that either

- not all honest validator who do not have the erasure code piece announce the unavailability. If they would announce it, since their number is at least $2f + 1 - f = f + 1$ (i.e., honest parties - honest parties who do not have the erasure code), the other honest parties do not finalize it and malicious parties do not have enough number to finalize it. Therefore, we can assume that there is at least one honest party who does not announce the unavailability of its erasure code in this case, because he thinks that he has one but actually it is not correct. If there exists one honest party who has incorrect erasure code, but having the proof that it is correct (provided by parachain validators), then it means that the collision resistance assumption of Merkle tree is broken.

- or $f + 1$ parties announce unavailability meaning that GRANDPA finalized the block with at most $2f$ parties. So, this implies that the security of GRANDPA is broken by finalizing a block with less than $2f + 1$ votes.

Since the GRANDPA is secure and the Merkle tree is collision resistant, the unavailability protocol is secure. □

Theorem 2 (Validity)**.** *Assuming that the availability protocol is secure, the signature scheme is EF-CMA secure, the hash function H in Conditions (**??**) and (**??**) is a random oracle, the PoV protocol is secure and correct (Definition 1), the VRF is secure and a block in GRANDPA can be finalized with at least* $2f + 1$ *votes then the invalid block is finalized in the relay chain with the probability less than risk probability.*

Proof. (Sketch)

If there is one honest PV and there is not any $f + 1$ unavailability report, then at least $2f$ of the honest parties has the piece. It means that PV can collect all the pieces from the honest parties in order to reconstruct the blob and check its validity. Therefore, if there is at least one honest party in PV, it can detect the invalidity and announce it. In the end, we will have more than f invalidity signatures since the other parties also check invalidity after seeing the signature saying that the block is invalid.

Therefore, all parachain validators have to be malicious not to be caught. If there is at least one honest validator who is assigned for an extra validity check with either of the conditions, then the same case happens as having at least one honest parachain validator. Therefore, an invalid block cannot be detected (the attack succeeds) if all parachain validators are malicious and all extra check validators are malicious as well.

Remark that the extra check conditions (**??**) and (**??**) with $\tau = 0$ and (**??**) and (**??**) guarantees that the expected number of checks by parachain validators and the extra check validators is $\ell_{\text{TOTALREMOVED}}$. Therefore, we can consider that the parachain validator and extra-validator selection mechanism actually randomly samples $\ell_{\text{TOTALREMOVED}}$ validators for each parachain. So, the probability of all selected validators for a particular parachain being malicious is bounded by $(\frac{1}{3})^{\ell_{\text{TOTALREMOVED}}}$. See 1 for the details.

□

Theorem 3. *Given that $\ell_{\text{TOTALREMOVED}}$ malicious validators are eligible to check the validity of a block header, the invalidity attack is bounded by the probability $\exp(-\frac{2}{3}(\mu+r))$ if $\mu+r < n/m$ and the probability $\exp(-\frac{2n}{3m})$ if $\mu + r \geq n/m$.*

Proof. Below, we give the probability of any validator being in conditions (**??**), (**??**), (**??**), (**??**):

$$\Pr[\text{Cond. (??)}] = \Pr[\text{Cond. (??)}] = p_{vrf} = \frac{1}{m}$$

$$\Pr[\text{Cond. (??) at time } \tau] = p_\tau = \frac{\mu'}{n - nc} + \tau$$

$$\Pr[\text{Cond. (??) }] = \frac{\mu'}{n - nc}$$

We assume that $n_0 = \frac{n}{m}$. The number of $\ell_{\text{TOTALREMOVED}} - n_0 = \mu + r$ malicious validators can be selected for the extra check with the following cases:

1. $\mu + r \geq \frac{n}{m}$ and at least $\mu + r$ malicious validators are in condition (**??**) and condition (**??**) until the time τ_k for the parachain whose validators are malicious. The attack probability is highest if at least $\mu + r$ malicious validators are in (**??**). In this case, the attack succeeds if no honest validator satisfies the condition (**??**)

$$\Pr[\mathsf{attack1}] = (1 - \frac{1}{m})^{2n/3} < e^{-2n/3m}$$

2. $\mu + r < \frac{n}{m}$ and at least $\mu' = \mu + r$ malicious validators are in condition (**??**) until the time τ_k. For simplicity, we divide time in discrete values $[\tau_0 = 0, \tau_1 = u, \tau_2 = 2u, ..., \tau_k = ku]$. τ_k can be computed by malicious validators before the validation process begins. The probability of this attack is the following

$$\begin{aligned}
\Pr[\mathsf{attack2}] &= \prod_{i=0}^{k}(1-p_{\tau_i})^{\frac{2n}{3}} \\
&\leq \prod_{i=0}^{k}(1-\frac{\mu'+\tau_i n(1-c)}{n(1-c)})^{\frac{2n}{3}} \\
&= \exp(\sum_{i=0}^{k}-\frac{2}{3}\frac{\mu'+\tau_i n(1-c)}{1-c}) \\
&= \exp(-2/3(\frac{(k+1)\mu'}{1-m/n}+nu\frac{k(k+1)}{2})
\end{aligned}$$

Remark that $\Pr[\mathsf{attack3}]$ is maximum when $k=0$. When $k=0$, $\Pr[\mathsf{attack3}] \leq \exp(-\frac{2\mu' n}{3(n-m)}) \leq \exp(-\frac{2}{3}(\mu+r))$.

3. This case considers equivocation: Here, we can assume that the adversary knows who satisfies the condition (**??**) and (**??**) since if the malicious validator is the block producer, he first produces the block and sees who validates this block. If no honest validator checks it with condition (**??**), he equivocates it. When this happens, the attack succeeds if no honest party satisfies the condition (**??**) or (**??**). Hence, the probability of attack 4 is

$$\begin{aligned}
\Pr[\mathsf{attack4}] &= \Pr[\text{no honest check in cond. (??) and (??)}] \\
&\leq \Pr[\mathsf{attack1}](1-\frac{\mu'}{n-nc})^{\frac{2}{3}n} \\
&\leq \exp(-2/3(\frac{n}{m}+\mu+r))
\end{aligned}$$

□

7 Practical Results

As it can be seen from the proof of theorem 2, malicious validators can attack the validity protocol with probability $(\frac{1}{3})^{\ell_{\text{TOTALREMOVED}}}$. Remember that $\ell_{\text{TOTALREMOVED}} = n_0 + \mu + r$ where $r = \lceil r_a + r_f \rceil$. If we assume that parachain validators change in every x minutes, the malicious parties need to wait $x3^{\ell_{\text{TOTALREMOVED}}}$ minutes in order to succeed the attack. For example, if $x = 5$ minutes, then the total time for an attack for each $\ell_{\text{TOTALREMOVED}}$ is given in Figure 1:

As it can be seen, if $\ell_{\text{TOTALREMOVED}}$ is more than 14, the adversary needs to wait more than 50 years for the attack. Therefore, in terms of security, it is important to have minimum $\ell_{\text{TOTALREMOVED}}$ validators.

The parameter μ: The possible μ (minimum number of extra check) values:

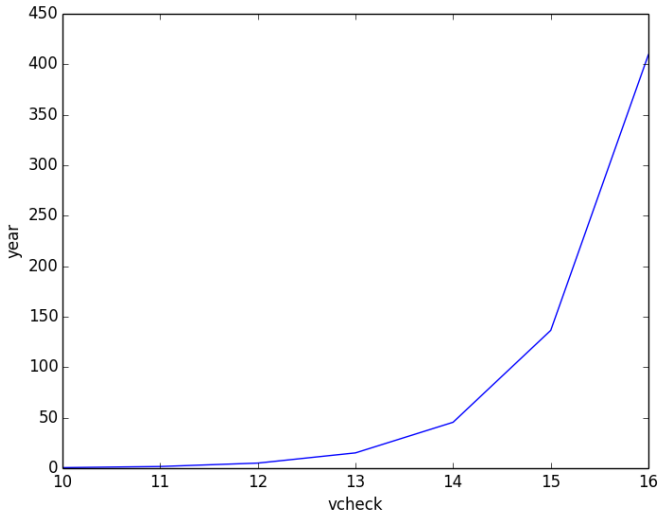

Figure 1: It shows the waiting time for an adversary for a successful attack given that every parachain validators changes every 5 minutes

- If $\mu = 0$, then $|pv| = 14$ in order to make sure that minimum $\ell_{\text{TOTALREMOVED}}$ is always equal to 14. Given that $|pv| = n/m$ where m is the number of parachain validators $n = 14 * m$.
- If $\mu > 0$, then $|pv| < 14$. It means that we need less validators for the security than in the case where $\mu = 0$. Less validators imply less network delay and less network delay implies more secure BABE. On the other hand, if μ increases, it means that more validators need to reconstruct a blob in order to check the validity. So, validators need to do more work. In terms of scalability of the relay chain, we need to decide μ.

As rule of thumb, if we don't care about the number of validators as far as it is not the maximum number of validators (N) that Polkadot can handle, then if $14 * m < N$, $\mu = 0$. Otherwise, $\mu = 14 - N/m$.

Another disadvantage of having $\mu = 0$ is that parachain validators have a less risky attack when all of them are malicious. Imagine a parachain with a few collators. We can assume that they may be malicious and collaborate with the malicious validators. In this case, the validators will not have any report. So, there will be 0 extra checks. As soon as all parachain validators are malicious in this malicious parachain, they can add invalid block headers and cannot be caught. The security argument says here that they need to wait around 50 years for this, so the attack is not possible. However, in the real life, since the attack does not have any risk, the collators can bribe parachain validators with their stake and parachain validators validate an invalid blob.

Theorem 3 gives the risk that malicious validators take at when they put an invalid report. If $\mu + r \leq n/m$, the invalid block will not be detected with probability $\exp(-\frac{2}{3}(\mu + r))$. In the worst case scenario, if no report is received then the attack probability is $\exp(-\frac{2}{3}(\mu))$. We give in Figure 2 that the probability of a successful attack in the case of $\ell_{\text{TOTALREMOVED}}$ malicious validators are selected. As it can be seen in Figure 2, their risk is exponentially

increasing when μ increases. In order to make the risk close to 0 even if no report received, μ should be greater than 4.

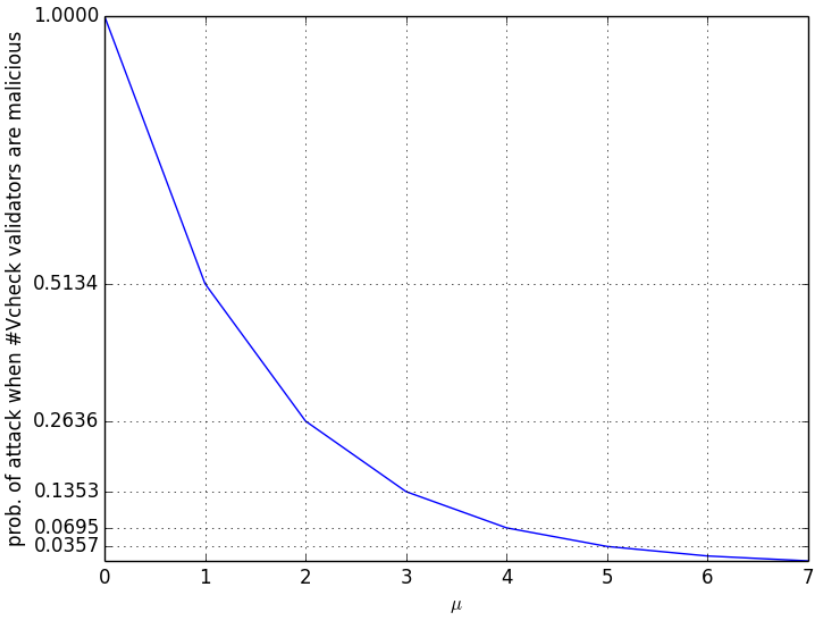

Figure 2: The risk of malicious validators when they attack even if $\ell_{\text{TOTALREMOVED}}$ validators are malicious

Fisherman and Collator Reports: In the validity scheme, we rely on the invalidity reports of fishermen and unavailability reports of collators to find the parameter $\ell_{\text{TOTALREMOVED}}$. However, we need to bound $\ell_{\text{TOTALREMOVED}}$ so that these reports do not make too many validators check the validity. Let us call this bound μ_{max} (i.e, $\mu + \lceil r_f + r_a \rceil \leq \mu_{max}$). This bound is necessary because regular malicious reports can slow down the process easily.

When a fisherman sends a report of invalidity, but later on it is found to be valid, the fisherman is slashed. Therefore, the reliability of a fisherman report can be measured by how much it is staked. Considering this, r_f can be computed as follows:

$$r_f = \frac{\sum_i \mathsf{stake}_{f_i}}{v_{gain}}$$

Here, stake_{f_i} is the stake that a fisherman f_i puts for this report and v_{gain} is the DOT value that a validator receives in each block. In a nutshell, a fisherman needs to stake at least the same amount that the validator earns for each block in order to make a validator to execute an extra validity check. If the fisherman report is not valid, then the fisherman pays for this extra check. So, a malicious fisherman has to spend $\mu_{max} v_{gain}$ for a report to slow down the validator network. We believe that this model discourages fishermen from sending false reports.

However, we cannot measure the reliability of unavailability reports as fisherman's reports since it is not possible to show the correctness or incorrectness of any unavailability reports.

Malicious collators do not lose anything by just sending fake unavailability reports. In order to solve this issue, we assign a parameter $\alpha \in (0,1)$ for a parachain that defines the proportion of honest collator assumption. Depending on α, we have defined r_a for two cases:

- if $\alpha > 1/2$, $r_a = \mu_{max}^{x/\alpha}$

$$r_a(x) = \begin{cases} \mu_{max}^{x/\alpha} & \text{if } x \leq \alpha \\ \mu_{max} & \text{otherwise} \end{cases}$$

where

$$x = \frac{\text{total unavailability reports}}{\text{all collators}} = \frac{\sum_{c_i \in C_P} st_{c_i}}{|C_P|}.$$

Here, $st_{c_i} \in \{0,1\}$ and it is 0 if the block is available. Otherwise, it is 1. The reason of using a function such as $\mu_{max}^{x/\alpha}$ is to make sure that we have extra checks close to maximum check only if the number of unavailability reports are close to the number of honest collator assumption. Thanks to this, if the number of honest collators are majority, the malicious collators who want to slow down Polkadot cannot achieve to make always maximum number of checks with only unavailability reports. In general, these type of parachains can be investigated by fishermen as far as the blocks are available to honest collators. Therefore, we expect that if there is any invalid block, the fishermen catch it and send a report. If the blocks are unavailable to honest collators (so fishermen too), validators check the invalidity with the maximum capacity.

- if $\alpha < 1/2$, then it is critical to give more importance to the unavailability reports from this parachain. Therefore, we use the following linear function:

$$r_a(x) = \begin{cases} \mu_{max} \frac{x}{\alpha} & \text{if } x \leq \alpha \\ \mu_{max} & \text{otherwise} \end{cases}$$

So, $\mu' = \mu + \lceil r_a + r_f \rceil$ if $\mu + \lceil r_a + r_f \rceil \leq \mu_{max}$. Otherwise, $\mu' = \mu_{max}$.

The parameter μ_{max}: Given the fact that we have maximum number of extra checks when there are many invalidity or unavailability reports, the parameter μ_{max} needs to be big enough so that the probability of having at least one honest extra check is almost 1. This probability can be bounded by $1 - (\frac{1}{3})^{\mu_{max}}$. Therefore, we can have $\mu_{max} = 15$.

In Figure 3, we compute the number of extra checks required depending on the α-parameter of a parachain. As it can be seen, the parachain that we trust less requires more extra-checks. The parachain where we assume honest majority reaches the maximum check when the unavailability reports are close to the number of honest collators.

A Proof Details

Lemma 1. *For any slot number s, block producer bp, epoch randomness r for the epoch that includes s, integer $k > 0$, and parachain P, with probability at least $1-(f/n)^{-k}$ over the epoch randomness and the random oracles for the hashes and VRFs, for any block that bp produces in slot s that includes a parachain header from P, if we eventually have $\ell_{\text{TOTALREMOVED}} \geq k$ for this parachain header, then some honest validator checks the parachain block.*

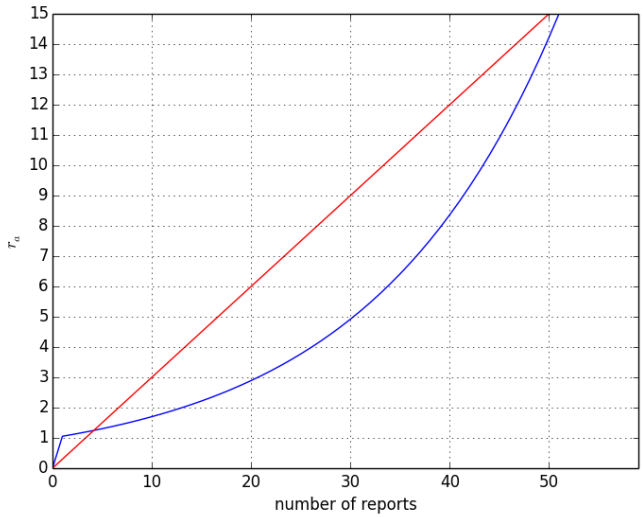

Figure 3: The red and blue graph shows the number of required extra checks when $\alpha \leq 1/2$ and $\alpha > 1/2$. Here, we assume that the total number of collators are 100.

Proof. Consider generating an ordering $\mathcal{O}$ of validators as follows, first we have the parachain validators in a uniformly random order, then all validators who satisfy condition (**??**) (using *bp*'s VRF for slot s) in a uniformly random order, then we order all remaining validators in increasing order of the hash $H(ID_{PC}||\mathtt{VRF}_{\mathtt{sk}_j^v}(V))$ used in condition (**??**), breaking ties due to collisions in a uniformly random order. We claim that distribution of $\mathcal{O}$ is as a uniformly random permutation of the validator set. To see this note that the only operation for which the different validators are treated differently are the VRFs and we assume that they are random oracles. The parachain validators are a set of n/m validators chosen uniformly at random using r. Each validator satisfies condition (**??**) with the same probability independently and so the distribution of the set of validators satisfying (1) conditioned on its size is a set of that size chosen uniformly at random from the non-parachain validators. Also the hashes in (2) are independently and identically distributed.

Next we show that for any block *bp* produces in slot s which includes a parachain header for P, if any honest validator ever sees that $\ell_{\text{TOTALREMOVED}} \geq k$, then any honest validators in the first k validators in $\mathcal{O}$ eventually check that block. All honest parachain validators and honest validators who satisfy (**??**) check. If all honest validators in the fist k validators in $\mathcal{O}$ satisfy these conditions we are done. So suppose that v is an honest validator in the first k validators in $\mathcal{O}$ who doesn't satisfy either of these conditions. v will still check if τ is large enough and they don't see attestation from $\ell_{\text{TOTALREMOVED}}$ validators who either are parachain validators, satisfied condition (**??**) or had a smaller hash in condition (**??**). But noting that all such validators are before v in $\mathcal{O}$ so there can be at most $k-1$ of them. Thus v only ever counts $k-1$ attestations towards the number needed $\ell_{\text{TOTALREMOVED}}$ which is eventually $\geq k$. So when $\ell_{\text{TOTALREMOVED}} \geq k$ and τ is large enough, v checks.

Finally we need to show that the probability that the first k validators in $\mathcal{O}$ are honest is at least $(f/n)^k$. Since the first k validators in $\mathcal{O}$ are distributed as a set of k validators chosen uniformly at random, this probability is $\prod_{i=0}^{k-1}(f-i)/(n-i) \leq (f/n)^k$. □