12.10 Overlay Schedule

The transition from the bootstrap era to a fully decentralized network is explained in section 3.9.2 of [SL-D1]. Key to this transition is a protocol parameter d which controls how many slots are governed by the genesis nodes via OBFT, and which slots are open to any registered stake pool. The transition system introduced in this section, OVERLAY, covers this mechanism.

This transition is responsible for validating the protocol for both the OBFT blocks and the Praos blocks, depending on the overlay schedule.

The actual overlay schedule itself it determined by two functions, isOverlaySlot and classifyOverlaySlot, which are defined in Figure 68. The function isOverlaySlot determines if the current slot is reserved for the OBFT nodes. In particular, it looks at the current relative slot to see if the next multiple of d (the decentralization protocol parameter) raises the ceiling. If a slot is indeed in the overlay schedule, the function classifyOverlaySlot determines if the current slot should be a silent slot (no block is allowed) or which core node is responsible for the block. The non-silent blocks are the multiples of $1/\mathsf{ActiveSlotCoeff}$, and the responsible core node are determined by taking turns according the lexicographic ordering of the core node keyhashes. **Note** that $1/\mathsf{ActiveSlotCoeff}$ needs to be natural number, otherwise the multilples of $\lfloor 1/\mathsf{ActiveSlotCoeff} \rfloor$ will yield a different proportion of active slots than the Praos blocks.

The environments for the overlay schedule transition are:

- The decentralization parameter d.
- The epoch nonce η_0.
- The stake pool stake distribution pd.
- The mapping $genDelegs$ of genesis keys to their cold keys and vrf keys.

The states for this transition consist only of the mapping of certificate issue numbers.

This transition establishes that a block producer is in fact authorized. Since there are three key pairs involved (cold keys, VRF keys, and hot KES keys) it is worth examining the interaction closely. First we look at the regular Praos/decentralized setting, which is given by Equation 40.

- First we check the operational certificate with OCERT. This uses the cold verification key given in the block header. We do not yet trust that this key is a registered pool key. If this transition is successful, we know that the cold key in the block header has authorized the block.
- Next, in the vrfChecks predicate, we check that the hash of this cold key is in the mapping pd, and that it maps to $(\sigma,\ hk_{vrf})$, where $(\sigma,\ hk_{vrf})$ is the hash of the VRF key in the header. If praosVrfChecks returns true, then we know that the cold key in the block header was a registered stake pool at the beginning of the previous epoch, and that it is indeed registered with the VRF key listed in the header.
- Finally, we use the VRF verification key in the header, along with the VRF proofs in the header, to check that the operator is allowed to produce the block.

The situation for the overlay schedule, given by Equation 39, is similar. The difference is that we check the overlay schedule to see what core node is supposed to make a block, and then use the genesis delegation mapping to check the correct cold key hash and vrf key hash.

The OVERLAY rule has nine predicate failures:

- If in the decentralized case the VRF key is not in the pool distribution, there is a *VRFKeyUnknown* failure.

Overlay environments

$$\mathsf{OverlayEnv} = \left(\begin{array}{rcll} d & \in & \{0,\ 1/100,\ 2/100,\ \ldots,\ 1\} & \text{decentralization parameter} \\ pd & \in & \mathsf{PoolDistr} & \text{pool stake distribution} \\ genDelegs & \in & \mathsf{GenesisDelegation} & \text{genesis key delegations} \\ \eta_0 & \in & \mathsf{Seed} & \text{epoch nonce} \end{array} \right)$$

Overlay Transitions

$$\_ \vdash \_ \xrightarrow[\text{OVERLAY}]{} \_ \subseteq \mathbb{P}\ (\mathsf{OverlayEnv} \times (\mathsf{KeyHash}_{pool} \mapsto \mathbb{N}) \times \mathsf{BHeader} \times (\mathsf{KeyHash}_{pool} \mapsto \mathbb{N}))$$

Overlay Schedule helper functions

$$\begin{aligned} &\mathsf{isOverlaySlot} \in \mathsf{Slot} \to [0,\ 1] \to \mathsf{Slot} \to \mathsf{Bool} \\ &\mathsf{isOverlaySlot}\ fSlot\ dval\ slot = \lceil s \cdot dval \rceil < \lceil (s+1) \cdot dval \rceil \\ &\quad \textbf{where}\ \ s := slot\ -_s\ fslot \end{aligned}$$

$$\begin{aligned} &\mathsf{classifyOverlaySlot} \in \mathsf{Slot} \to \mathbb{P}\ \mathsf{KeyHash_G} \to [0,\ 1] \to (0,\ 1] \to \mathsf{Slot} \to \mathsf{Bool} \\ &\mathsf{classifyOverlaySlot}\ fSlot\ gkeys\ dval\ asc\ slot = \\ &\quad \begin{cases} \mathsf{elemAt}\ \left(\frac{position}{ascInv} \bmod |gkeys|\right)\ gkeys & \text{if } position \equiv 0 \pmod{ascInv} \\ \diamond & \text{otherwise} \end{cases} \\ &\quad \textbf{where} \\ &\quad ascInv := \lfloor 1/asc \rfloor \\ &\quad position := \lceil (slot\ -_s\ fSlot) \cdot dval \rceil \\ &\quad \mathsf{elemAt}\ i\ set := i'\text{th lexicographic element of } set \end{aligned}$$

Figure 67: Overlay transition-system types

$$\frac{\begin{array}{c} bhb := \mathsf{bheader}\ bh \qquad vk := \mathsf{bvkcold}\ bhb \qquad vkh := \mathsf{hashKey}\ vk \\ slot := \mathsf{bslot}\ bhb \qquad fSlot := \mathsf{firstSlot}\ (\mathsf{epoch}\ slot) \\ gkh \mapsto (vkh,\ vrfh) \in genDelegs \\ \mathsf{isOverlaySlot}\ fSlot\ (\mathrm{dom}\ genDelegs)\ slot \\ \mathsf{classifyOverlaySlot}\ fSlot\ (\mathrm{dom}\ genDelegs)\ d\ \mathsf{ActiveSlotCoeff}\ slot = ghk \\ \\ \mathsf{pbftVrfChecks}\ vrfh\ \eta_0\ bhb \\ \\ \begin{array}{c}\mathrm{dom}\ pd \\ \mathrm{range}\ genDelegs\end{array} \vdash cs \xrightarrow[\text{OCERT}]{bh} cs' \end{array}}{\begin{array}{c} d \\ \eta_0 \\ pd \\ genDelegs \end{array} \vdash cs \xrightarrow[\text{OVERLAY}]{bh} \mathbf{cs'}}\ \text{Active-OBFT} \tag{39}$$

$$\frac{\begin{array}{c} bhb := \mathsf{bheader}\ bh \qquad slot := \mathsf{bslot}\ bhb \qquad fSlot := \mathsf{firstSlot}\ (\mathsf{epoch}\ slot) \\ \neg\mathsf{isOverlaySlot}\ fSlot\ (\mathrm{dom}\ genDelegs)\ slot \\ \\ \vdash cs \xrightarrow[\text{OCERT}]{bh} cs' \\ \\ \mathsf{praosVrfChecks}\ \eta_0\ pd\ \mathsf{ActiveSlotCoeff}\ bhb \end{array}}{\begin{array}{c} d \\ \eta_0 \\ pd \\ genDelegs \end{array} \vdash cs \xrightarrow[\text{OVERLAY}]{bh} \mathbf{cs'}}\ \text{Decentralized} \tag{40}$$

Figure 68: Overlay rules

- If in the decentralized case the VRF key hash does not match the one listed in the block header, there is a *VRFKeyWrongVRFKey* failure.
- If the VRF generated nonce in the block header does not validate against the VRF certificate, there is a *VRFKeyBadNonce* failure.
- If the VRF generated leader value in the block header does not validate against the VRF certificate, there is a *VRFKeyBadLeaderValue* failure.
- If the VRF generated leader value in the block header is too large compared to the relative stake of the pool, there is a *VRFLeaderValueTooBig* failure.
- In the case of the slot being in the OBFT schedule, but without genesis key (i.e., *Nothing*), there is a *NotActiveSlot* failure.
- In the case of the slot being in the OBFT schedule, if there is a specified genesis key which is not the same key as in the bock header body, there is a *WrongGenesisColdKey* failure.
- In the case of the slot being in the OBFT schedule, if the hash of the VRF key in block header does not match the hash in the genesis delegation mapping, there is a *WrongGenesisVRFKey* failure.

- In the case of the slot being in the OBFT schedule, if the genesis delegate keyhash is not in the genesis delegation mapping, there is a *UnknownGenesisKey* failure. This case should never happen, and represents a logic error.

Protocol environments

$$\mathsf{PrtclEnv} = \left(\begin{array}{rcll} d & \in & \{0,\ 1/100,\ 2/100,\ \ldots,\ 1\} & \text{decentralization parameter} \\ pd & \in & \mathsf{PoolDistr} & \text{pool stake distribution} \\ dms & \in & \mathsf{KeyHash}_{\mathsf{G}} \mapsto \mathsf{KeyHash} & \text{genesis key delegations} \\ \eta_0 & \in & \mathsf{Seed} & \text{epoch nonce} \end{array} \right)$$

Protocol states

$$\mathsf{PrtclState} = \left(\begin{array}{rcll} cs & \in & \mathsf{KeyHash}_{pool} \mapsto \mathbb{N} & \text{operational certificate issues numbers} \\ \eta_v & \in & \mathsf{Seed} & \text{evolving nonce} \\ \eta_c & \in & \mathsf{Seed} & \text{candidate nonce} \end{array} \right)$$

Protocol Transitions

$$\_ \vdash \_ \xrightarrow[\text{PRTCL}]{} \_ \subseteq \mathbb{P}\ (\mathbb{P}\ \mathsf{PrtclEnv} \times \mathsf{PrtclState} \times \mathsf{BHeader} \times \mathsf{PrtclState})$$

Figure 69: Protocol transition-system types

12.11 Protocol Transition

The protocol transition covers the common predicates of OBFT and Praos, and then calls OVERLAY for the particular transitions, followed by the transition to update the evolving and candidate nonces.

The environments for this transition are:

- The decentralization parameter d.
- The stake pool stake distribution pd.
- The mapping dms of genesis keys to their cold keys.
- The epoch nonce η_0.

The states for this transition consists of:

- The operational certificate issue number mapping.
- The evolving nonce.
- The canditate nonce for the next epoch.

The PRTCL rule has no predicate failures, besides those of the two sub-transitons.