ToR: A Highly Scalable, General and Secure Cross-Chain Protocol

This supplment file contains three parts, including:

(1) Detailed evaluations in §1, which covers the latency §1.1, throughput §1.2 and parachain workload §1.3.
(2) Proof of workload optimization rate in §2, which covers Lemma 1, Lemma 2 and Theorem 1.
(3) Formalized security analysis with Universal Composition framework in §3, which covers the ideal functionalities §3.1 and security proof §3.2.

1 DETAILED EVALUATIONS

1.1 Latency

Interoperabliliy latency refers to the time taken for a cross-chain message to be initiated from the source chain, processed through the relay chain (not in NoR), and ultimately confirmed on the target chain. The experiment set $pn = [10, 60, 120, 180]$ and the $ratio$ to $[0.2, 0.4, 0.6, 0.8, 1.0]$. Fig.1 shows that the latency of ToR remains consistently low across different ratios, while the latency of AoR increases linearly with $ratio$. The NoR latency also remains relatively stable, but there is a significant increase in latency as pn increases.

The increase of latency with the $ratio$ rise in AoR is attributed to the elevated number of cross-chain messages. Under AoR, the relay chain is responsible for verifying and storing all cross-chain messages. Thus, an escalation in cross-chain messages volume augments the workload on the relay chain, leading to transaction congestion. This congestion amplifies the queuing latency of transactions on the relay chain. Higher $ratio$ exacerbates queuing delays, resulting in increased overall cross-chain interoperabliliy latency.

It can be observed that when $ratio = [0.2, 0.4, 0.6, 0.8]$ and pn=180, the latency of NoR is significantly higher than that of AoR and ToR. This is because each parachain under NoR needs to synchronize block header from other parachains, leading to high workload on the parachains. As a result, transaction congestion occurs within parachains, causing an increase in queuing delay for cross-chain transactions in the node's mempool. Ultimately, this results in higher interoperable latency.

1.2 Throughput

Throughput (TPS) is defined as N_{tx}/D, where N_{tx} represents the total number of cross-chain messages, and D denotes the total time taken for cross-chain messages from issuing on the source to confirm on the target. The Fig.2 shows that ToR has a higher throughput than NoR and AoR overall. When $pn = 10$, the difference among the three protocols are negligible. However, when the number of Parachains increases ($pn = [60, 120, 180]$), the throughput of ToR significantly outperforms that of NoR and AoR. Additionally, for AoR, the throughput tends to plateau when $pn \geq 60$ as the $ratio$ increases, reaching its performance upper limit. It is worth noting that at $pn = 180$, NoR's throughput is remarkably low, far below that of AoR and ToR.

1.3 Workload on Parachains

After significantly reducing the relay chain's workload, is the parachain's workload affected? To answer this question, this experiment monitors the its variation.

From Fig.3, it can be seen that when pn is small, the differences of parachains' workload among the three protocols are minimal and their workload gradually decrease until the end of the observation. However, when pn is large, it can be seen that the parachains' workload in NoR no longer decreases but continues to increase. It is because the parachains in NoR have to synchronize and verify the block headers from all other parachains. When pn increases, the number of block header transactions that each parachain must process increases until it exceeds the performance cap of the parachain. Therefore, even if cross-chain messages no longer exist in the system, the parachains' workload continues to increase.

The workload variation between ToR and AoR is quite similar because neither requires the parachain to process block headers from any other parachains but the relay chain. Although it is similar, the detailed figure shows that the ToR's workload on parachains is slightly higher than that of AoR. This difference is from that ToR also requires the target chain to verify the second layer proof $VP2$. Additionally, each block from the source chain only corresponds to only one $VP2$ (according to the "Reduce Verification Redundancy" strategy described in Section ??). Thus, the parachain's workload tends to be slightly higher in ToR.

2 PROOF OF WORKLOAD OPTIMIZATION RATE

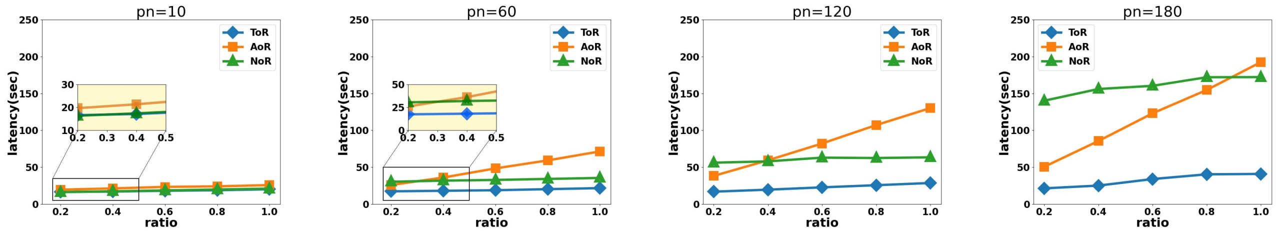
Model. We denote the relay chain's throughput as α, meaning it can process α transactions per second. Given $n \geq 2$ parachains in the cross-chain system, fully-connected, denoted as $\{P_i \mid i \in [1, n]\}$. For the parachain P_i, it can generate $\{b_i \mid i \in [1, n]\}$ blocks per second, and the number of cross-chain messages created per block is denoted as $\{x_j \mid j \in [1, b_i]\}$. We consider the average workload during the time interval D which is divided it into a discrete time series $TS = \{t_1, t_2, ..., t_d\}$ with a 1-second interval. For convenience, we define $h(x) = \frac{|x|+x}{2}$, meaning that if $x > 0$ then $h(x) = x$, and if $x <= 0$ then $h(x) = 0$.

Lemma 1. *Based on the above model, in ToR, the average workload of the relay chain is*

$$W^{\circ}_{ToR} = \begin{cases} y_{ToR}, & \alpha \geq y_{ToR} \\ y_{ToR} + \dfrac{(d-1)}{2} * (y_{ToR} - \alpha), & \alpha < y_{ToR} \end{cases}$$

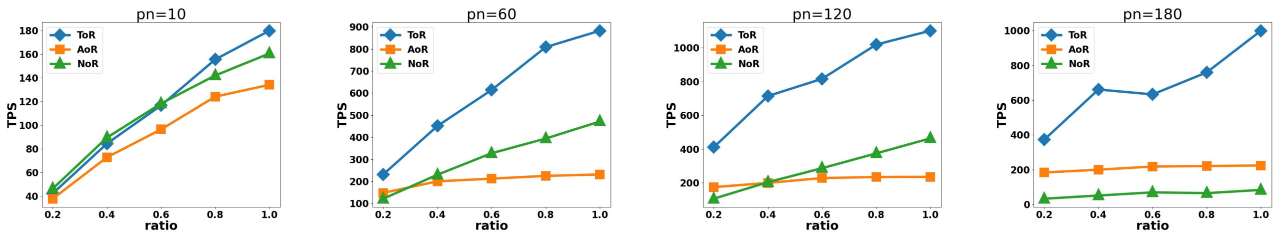
, where $y_{ToR} = \sum_{i=1}^{n} b_i$.

Proof. The relay chain can receive up to $y_{ToR} = \sum_{i=1}^{n} b_i$ transactions (i.e., block header synchronization) per second. Given that



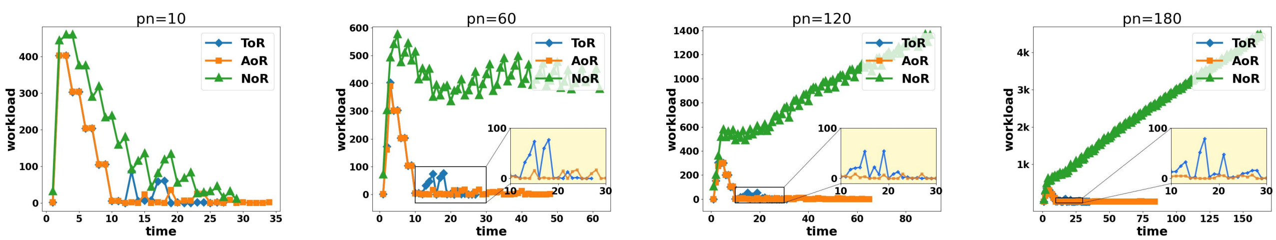
(a) (b) (c) (d)

Figure 1: In cross-chain systems with different parachain number, as increaing the ratio of cross-chain messages, it shows the latency from creating CM on the source to confirming it on the target.



(a) (b) (c) (d)

Figure 2: In cross-chain systems with different parachain number, it shows the TPS of this overall system as increaing the ratio of cross-chain messages.



(a) (b) (c) (d)

Figure 3: Parachain workload trends.

the relay chain's throughput is α, we can know W_{ToR} as follows:

$$\begin{aligned} W_{ToR} = \{W_{t_1} &= y_{ToR}, \\ W_{t_2} &= h(W_{t_1} - \alpha) + y_{ToR}, ..., \\ W_{t_d} &= h(W_{t_{d-1}} - \alpha) + y_{ToR}\}. \end{aligned}$$

In terms of varying values of α,

i. if $\alpha \geq y_{ToR}$ then

$$W^{\circ}_{ToR} = W_{t_k} = W_{t_1} = y_{ToR},\ where\ k \in [1, d]$$

ii. if $\alpha < y_{ToR}$ then

$$W_{t_{k+1}} = W_{t_k} - \alpha + y_{ToR},\ where\ k \in [1, d-1]$$

Thus, W_{ToR} forms an arithmetic sequence with a common difference of $\Delta = y_{ToR} - \alpha$. Therefore, it can be derived that:

$$W^{\circ}_{ToR} = y_{ToR} + \frac{(d-1)}{2} * (y_{ToR} - \alpha)$$

□

Lemma 2. *Based on the above model, in AoR, the average workload of the relay chain is*

$$W^{\circ}_{AoR} = \begin{cases} y_{AoR}, & \alpha \geq y_{AoR} \\ y_{AoR} + \frac{(d-1)}{2} * (y_{AoR} - \alpha), & \alpha < y_{AoR} \end{cases}$$

$$,\ where\ y_{AoR} = \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} \sum_{j=1}^{b_i} x_{i,j}$$

Proof. the relay chain can receive up to $y_{AoR} = \sum_{i=1}^{n} b_i + \sum_{i=1}^{n}\sum_{j=1}^{b_i} x_{i,j}$ transactions (i.e., block header synchronization and cross-chain messages) per second. We can know W_{AoR} as follows:

$$\begin{aligned} W_{AoR} = \{W'_{t_1} &= y_{AoR}, \\ W'_{t_2} &= h(W'_{t_1} - \alpha) + y_{AoR}, ..., \\ W'_{t_d} &= h(W'_{t_{d-1}} - \alpha) + y_{AoR}\}. \end{aligned}$$

In terms of varying values of α,

i. if $\alpha < y_{AoR}$ then

$$W'_{t_{k+1}} = W'_{t_k} - \alpha + y_{AoR}, \ where \ k \in [1, d-1].$$

Thus, W_{AoR} forms an arithmetic sequence with a common difference of $\Delta = y_{AoR} - \alpha$. Consequently, it follows that:

$$W^{\circ}_{AoR} = y_{AoR} + \frac{(d-1)}{2} * (y_{AoR} - \alpha).$$

ii. if $\alpha \geq y_{AoR}$ then

$$W^{\circ}_{AoR} = W'_{t_k} = W'_{t_1} = y_{AoR}, \ where \ k \in [1, d-1]$$

Hence,

$$W^{\circ}_{AoR} = \begin{cases} y_{AoR}, & \alpha \geq y_{AoR} \\ y_{AoR} + \dfrac{(d-1)}{2} * (y_{AoR} - \alpha), & \alpha < y_{AoR} \end{cases}$$

□

Thereom 1. *Based on the above model, we have the average workload optimization ratio*

$$\tau \geq 1 - \frac{1}{\varsigma}$$

, where $\varsigma = \overline{x} = \dfrac{\sum_{i=1}^{n}\sum_{j=1}^{b_i} x_{i,j}}{\sum_{i=1}^{n} b_i}$*, which refers to the average number of cross-chain messages in a parachain block.*

Proof. From Lemma 1 and Lemma 2, we can know τ as follows:

$$\begin{aligned} \tau &= 1 - \frac{W^{\circ}_{ToR}}{W^{\circ}_{AoR}} \\ &= 1 - \begin{cases} \dfrac{y_{ToR} + \frac{(d-1)}{2} * (y_{ToR} - \alpha)}{y_{AoR} + \frac{(d-1)}{2} * (y_{AoR} - \alpha)}, & \alpha < y_{ToR} \\ \dfrac{y_{ToR}}{y_{AoR} + \frac{(d-1)}{2} * (y_{AoR} - \alpha)}, & y_{ToR} \leq \alpha < y_{AoR} \\ \dfrac{y_{ToR}}{y_{AoR}}, & y_{AoR} \leq \alpha \end{cases} \end{aligned}$$

Discuss by different conditions.

i. if $\alpha < y_{ToR}$ then

$$\begin{aligned} \tau &= 1 - \frac{W^{\circ}_{ToR}}{W^{\circ}_{AoR}} \\ &= 1 - \frac{y_{ToR} - \frac{d-1}{d+1} * \alpha}{y_{AoR} - \frac{d-1}{d+1} * \alpha} \end{aligned}$$

As the time series length d approaches sufficiently large, given $\lim_{d \to +\infty} \frac{d-1}{d+1} = 1$, we can consequently deduce that

$$\tau = 1 - \frac{y_{ToR} - \alpha}{y_{AoR} - \alpha}$$

Let $g = \dfrac{y_{ToR} - \alpha}{y_{AoR} - \alpha}$, then

$$\begin{aligned} g &= \frac{1 - \frac{\alpha}{n\overline{b}}}{1 + \frac{\sum_{i=1}^{n}\sum_{j=1}^{b_i} x_{i,j}}{n\overline{b}} - \frac{\alpha}{n\overline{b}}} \\ &= \frac{1 - \frac{\alpha}{n\overline{b}}}{1 + \overline{x} - \frac{\alpha}{n\overline{b}}} \end{aligned}$$

Let $v = 1 - \dfrac{\alpha}{n\overline{b}}$, then $g = \dfrac{1}{1 + \frac{\overline{x}}{v}}$.

It can be easily deduced that for $v \in (0, 1)$, $g(v)$ is monotonically increasing, and τ is monotonically decreasing.

Given $\alpha \longrightarrow 0$, τ approaches its minimum value $\dfrac{1}{1 + \overline{x}}$.

Therefore, if $\alpha < y_{ToR}$, then $\tau > 1 - \dfrac{1}{\overline{x}}$.

ii. if $y_{ToR} \leq \alpha < y_{AoR}$, then

$$\begin{aligned} \tau &= 1 - \frac{W^{\circ}_{ToR}}{W^{\circ}_{AoR}} \\ &= 1 - \frac{2 * y_{ToR}}{(d+1) * y_{AoR} - (d-1) * \alpha} \end{aligned}$$

Let

$$g(a) = \frac{2 * y_{ToR}}{(d+1) * y_{AoR} - (d-1) * \alpha}$$

, we can know $g(a)$ is monotonically increasing, andτ is monotonically decreasing. Given $\alpha \longrightarrow y_{AoR}$, τ approaches its minimum value $\dfrac{y_{ToR}}{y_{AoR}}$.

Let $\gamma = \dfrac{y_{ToR}}{y_{AoR}}$, then

$$\begin{aligned} \gamma &= \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} b_i + \sum_{i=1}^{n}\sum_{j=1}^{b_i} x_{i,j}} \\ &= \frac{1}{1 + \overline{x}} \end{aligned}$$

Therefore, if $y_{ToR} \leq \alpha < y_{AoR}$, then $\tau > 1 - \dfrac{1}{\overline{x}}$.

iii. if $y_{AoR} \leq \alpha$ then $\tau = 1 - \dfrac{1}{\overline{x}}$.

In summary, from (i.,ii.,iii.), we have $\tau \geq 1 - \dfrac{1}{\overline{x}}$. □

3 FORMALIZED SECURITY ANALYSIS

3.1 Ideal Functionalities

In order to model the ideal protocol of ToR in the ideal world, it is necessary to construct the ideal functionality $\mathcal{F}_{ToR}$. $\mathcal{F}_{ToR}$ relies on $\mathcal{F}_{rc}$, and $\mathcal{F}_{pc}$, where $\mathcal{F}_{rc}$ is responsible for the cross-chain service contract R_0^{csc} instance on the relay chain, and $\mathcal{F}_{pc}$ is for that on the parachain. appendix **??** shows details of ideal functionalities.

$\mathcal{F}_{pc}$ (see appendix Fig.4) maintains a set of active contract instance deployed on the source chain. It includes the interfaces for creating CM and updating BCR on the source, verifying block headers from the relay chain by LC_0 and using DLV proofs to verify CM on the target by MTA_0. Before verifying the block header or BCR, $\mathcal{F}_{pc}$ first needs to check whether it has already been stored. Especially for BCR, $\mathcal{F}_{pc}$ also needs to check whether the block header in which BCR is included on the relay chain has been stored.

$\mathcal{F}_{rc}$ (see appendix Fig.5) maintains a set of active contract instance deployed on the relay chain. It focuses on verifying and storing trust roots from parachains. Before verifying by LC_{src} and MTA_{src}, it first checks whether the trust root has been stored ever. Both the light client and merkle tree verification processes are executed as subroutines.

$\mathcal{F}_{ToR}$ (see appendix Fig.6) interacts with the environment $\mathcal{Z}$, P_{src}, P_{dst}, and R_0, and relies on $\mathcal{F}_{pc}$ and $\mathcal{F}_{rc}$ to implement three interfaces: issue cross-chain message (denoted by ISSUE), synchronize trust root (denoted by SYNC), and transmit cross-chain message with DLV proof (denoted by TRANSMIT). $\mathcal{F}_{ToR}$ receives interoperable requests from environment $\mathcal{Z}$ through ISSUE and sends the request to P_{src}, which then calls $\mathcal{F}_{pc}$ to create CM and include it in a block. After P_{src} reaches consensus on the block, it notifies $\mathcal{F}_{ToR}$ to query the latest block header, BCR, and CM. Then $\mathcal{F}_{ToR}$ packs them as the trust root and forward it to R_0 through SYNC. R_0 calls $\mathcal{F}_{rc}$ to verify the trust root. After R_0 generates a new block, it notifies $\mathcal{F}_{ToR}$ for synchronizing the latest block header to P_{dst} through SYNC. Then P_{dst} calls $\mathcal{F}_{pc}$ to verify and store it. Simultaneously, $\mathcal{F}_{ToR}$ query the first-level proof $VP_1 = \Pi_{cm}^{BCR} = MTA_0.proof(CM, BCR)$ from P_{src}, and the second-level proof $VP_2 = \Pi_{BCR}^{0} = MTA_0.proof(BCR, stateRoot)$ from R_0. Then $\mathcal{F}_{ToR}$ sends (BCR, VP_2) and (CM, VP_1) to P_{dst} through TRANSMIT in a strict order. P_{dst} calls $\mathcal{F}_{pc}$ for validations, and if valid it delivers CM to the corresponding DApp.

3.2 Security Proof

Our analysis focuses on two aspects: (1)Synchronize Trust Root, (2)Transmit Cross-chain Message and DLV proof.

(1) **Synchronize Trust Root**

From P_{src} **to** R_0. When P_{src} sends a sequence of transactions $[TX_{pch2rc} = (P_{src}.id, Hdr_{src}), TX_{pcBCR2rc} = (P_{src}.id, H_{BCR}, BCR, \Pi_{BCR}^{src})]$ to R_0 in a strict order, $\mathcal{S}$ sends messages $[msg_1 = (Hdr_{src}), msg_2 = (BCR, \Pi_{BCR}^{src}, H_{BCR})]$ to $\mathcal{F}_{ToR}$ in the name of P_{src} . (Note: since blockchains lack the capability to send information externally, the actual senders are one/group of forwarding nodes, namely G_{pr}). If G_{pr} is corrupted, then four scenarios may occur:

(a) If G_{pr} sends to R_0 a forged transaction $TX_pch2rc' = (P_{src}.id, Hdr_{src}')$ created by $\mathcal{A}$, then R_0 will return a termination message. Similarly, $\mathcal{S}$, in the name of P_{src}, sends to $\mathcal{F}_{ToR}$ the message $msg_1' = (Hdr_{src}')$, and $\mathcal{F}_{ToR}$ constructs the transaction TX_{pch2rc}' to send to $\mathcal{F}_{rc}$. $\mathcal{F}_{rc}$ verifies Hdr_{src}' as invalid based on the light client algorithm of P_{src}, and returns $terminate_7$, causing $\mathcal{S}$ to terminate the operation.

(b) If G_{pr} sends to R_0 a forged transaction $TX_{pcBCR2rc}' = (P_{src}.id, H_{BCR'}), BCR', \Pi_{BCR'}^{src})$ created by $\mathcal{A}$, then R_0 will return a termination message. Similarly, $\mathcal{S}$, in the name of P_{src}, sends to $\mathcal{F}_{ToR}$ the message $msg_2 = (BCR', \Pi_{BCR'}^{src}, H_{BCR'})$, and $\mathcal{F}_{ToR}$ constructs the transaction $TX_{pcBCR2rc}'$ to send to $\mathcal{F}_{rc}$. $\mathcal{F}_{rc}$ verifies BCR' as invalid and returns the interruption message $terminate_{10}$, causing $\mathcal{S}$ to terminate the operation.

(c) If G_{pr} sends the transaction sequence $[TX_{pcBCR2rc}, TX_{pch2rc}]$ (in the wrong order) to R_0, then R_0 will return a termination message (corresponding block header not found). Similarly, $\mathcal{S}$, in the name of P_{src}, first submits the transaction $TX_{pcBCR2rc}$ to $\mathcal{F}_{ToR}$, $\mathcal{F}_{rc}$ will return the interruption message $terminate_{11}$, causing $\mathcal{S}$ to terminate the operation.

(d) If G_{pr} crashes, then $\mathcal{S}$ terminates.

From R_0 **to** P_{dst}. When R_0 sends the transaction $TX_{rch2pc} = (Hdr_0)$ to P_{dst} (where G_{rp} is the actual sender), $\mathcal{S}$ sends the message $msg = (Hdr_0)$ to $\mathcal{F}_{ToR}$ in the name of R_0. If G_{pr} is corrupted, then four scenarios may occur:

(a) If G_{rp} sends to P_{dst} a forged transaction $TX_{rch2pc}' = (Hdr_0')$ created by $\mathcal{A}$, P_{dst} will return a termination message. Similarly, S, sends the message $msg' = Hdr_0'$ to $\mathcal{F}_{ToR}$ in the name of R_0. $\mathcal{F}_{ToR}$ will construct the transaction $TX_{rch2pc}' = (Hdr_0')$ and send it to $\mathcal{F}_{pc}$, and $\mathcal{F}_{pc}$ will use the light client of the relay chain to verify that Hdr_0' is invalid, and return $terminate_1$, causing $\mathcal{S}$ to terminate the operation.

(b) If G_{rp} crashes, then $\mathcal{S}$ terminates.

(2) **Transmit Cross-chain Message and DLV proof** When P_{src} sends the transaction sequence $[TX_{L2syncBCR} = (P_{src}.id, H_{BCR}^0, H_{BCR}, BCR, \Pi_{BCR}^0), TX_{L1cm} = (P_{src}.id, cm, H_{cm}, \Pi_{cm}^{BCR})]$ to P_{dst} in a strict order (with the actual sender being G_{pp}), $\mathcal{S}$ sends messages $[msg_1 = (H_{BCR}^0, H_{BCR}, BCR, \Pi_{BCR}^0), msg_2 = (cm, H_{cm}, \Pi_{cm}^{BCR})]$ to $\mathcal{F}_{ToR}$ in the name of P_{src}. If G_{pp} gets corrupted, then five scenarios may occur:

(a) If G_{pp} sends to P_{dst} a forged transaction $TX_{L2syncBCR}'$ created by $\mathcal{A}$, then P_{dst} will return a termination message. Similarly, $\mathcal{S}$ sends the message $msg_1' = (H_{BCR'}^0, H_{BCR'}, BCR', \Pi_{BCR'}^0)$ to $\mathcal{F}_{ToR}$ in the name of P_{src}. $\mathcal{F}_{ToR}$ constructs the transaction $TX_{L2syncBCR}'$ and sends it to $\mathcal{F}_{pc}$, which will return $terminate_4$, causing $\mathcal{S}$ to terminate the operation.

Ideal functionality $\mathcal{F}_{pc}$

Session Setup

- Each session manages block-header synchronizing process from R_0 and cross-chain message verifying process from P_{src} to P_{dst}.
- Initialize $LCStorage = \{\}$ which is used to store block header from R_0 by block height.
- Set $session.srcid = P_src.id$, $session.dstid = P_{dst}.id$, $session.height = P_{src}.blockHeight$.

Issue Cross-chain Message

Upon receiving $req = (srcid, dstid, payload)$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{pc}$ calls related source DApp according to payload, gets response and creates cross-chain message cm. Then $\mathcal{F}_{pc}$ calls $BCR = A^u_{BCR}(session.height, cm)$ and stores (BCR, cm) on chain.

Synchronize Relay-chain Block Headers

Upon receiving $TX_{rch2pc} = (Hdr_0)$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{pc}$ calls $LCExist(Hdr_0.height)$ to check whether Hdr_0 exists or not, if it exists then $\mathcal{F}_{ToR}$ calls $LCVerify(Hdr_0)$ to verify Hdr_0 and calls $LCStore(Hdr_0)$ to store it if verified, else $terminate_1(\text{"}verifyR_0headerfailed\text{"})$.

Verify Cross-chain Message

- Upon receiving $TX_{L2syncBCR} = (P_{src}.id, H^0_{BCR}, H_{BCR}, BCR, \Pi^0_{BCR})$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{pc}$ calls $L02Exist(P_(src).id, H_{BCR})$ to check: (1) whether BCR of H_{BCR} has not been stored, and calls $LCExist(H^0_{BCR})$ to check: (2) whether block header of H^0_{BCR} has been stored in light client of R_0. If condition (1) not satisfied, then $terminate_2(\text{"}BCRhasbeenstored\text{"})$. If condition (2) not satisfied, then $terminate_3(\text{"}norelatedR_0header\text{"})$. If both are satisfied then $\mathcal{F}_{pc}$ gets Hdr_0 from $LCStorage[H^0_{BCR}]$ and calls $L02Verify(Hdr_0.stateRoot, BCR, \Pi^0_{BCR})$ to verify whether BCR has been confirmed on relay-chain. If verified then $\mathcal{F}_{pc}$ calls $L02Store(P_{src}.id, H_{BCR}, BCR)$ to store it, else $terminate_4(\text{"}L02verifyfailed\text{"})$.
- Upon receiving $TX_{L1cm} = (P_{src}.id, cm, H_{cm}, \Pi^{BCR}_{cm})$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{pc}$ calls $L02Exist(P_{src}.id, H_{cm})$ to check whether BCR exists, if it exists then F_{pc} gets BCR from $L02Storage[P_{src}.id][H_{cm}]$ and calls $L01Verify(BCR, cm, \Pi^{BCR}_{cm})$ to verify whether cm is valid. If cm is valid then execute related destination DApp. If BCR not exists then $\mathcal{F}_{pc}$ replies $terminate_5(\text{"}norelatedBCR\text{"})$. If cm is not valid, then $\mathcal{F}_{pc}$ replies $terminate_6(\text{"}verifycmfailed\text{"})$.

Description of the subroutines:

- **LCVerify:** On input Hdr_0, execute light client algorithm of R_0 to verify whether Hdr_0 is valid. If Hdr_0 is invalid return false else true.
- **LCStore:** On input Hdr_0, store Hdr_0 in $LCStorage[Hdr_0.height] = Hdr_0$.
- **LCExist:** On input $height$, check whether Hdr_0 of $height$ exists in $LCStorage[height]$. If it does then return true else return false.
- **L02Verify:** On input a tuple $(root, BCR, \Pi^{root}_{BCR})$, execute Merkle tree verification algorithm of R_0 to verify whether BCR is relevant with root.
- **L02Store:** On input a tuple $(paraid, H_{BCR}, BCR)$, store BCR in $L02Storage[paraid][H_{BCR}] = BCR$.
- **L02Exist:** On input a tuple $(paraid, H_{BCR})$, check whether BCR of H_{BCR} exists in $BCRStorage[paraid][H_{BCR}]$. If it does then return true else return false.
- **L01Verify:** On input a tuple $(root, cm, \Pi^{BCR}_{cm})$, execute Merkle tree verification algorithm of R_0 to verify whether BCR is relevant with root.

Figure 4: Ideal functionality $\mathcal{F}_{pc}$ for CSC on parachain

(b) If G_{pp} sends a forged transaction $TX^{'}_{L1cm}$ created by $\mathcal{A}$ to P_{dst}, then P_{dst} will return a termination message. Similarly, $\mathcal{S}$ sends the message $msg^{'}_2 = (cm^{'}, H_{cm^{'}}, \Pi^{BCR^{'}}_{cm^{'}})$ to $\mathcal{F}_{ToR}$ in the name of P_{src}. $\mathcal{F}_{ToR}$ constructs the transaction $TX^{'}_{L1cm}$ and sends it to $\mathcal{F}_{pc}$, which will return $terminate_6$, causing $\mathcal{S}$ to terminate the operation.

(c) If G_{pp} sends the transaction TX_{L1cm} to P_{dst} before sending $TX_{L2syncBCR}$, then P_{dst} will return a termination message. Similarly, $\mathcal{S}$ sends the message $msg_2 = (cm, H_cm, \Pi_cm^BCR)$ to $\mathcal{F}_{ToR}$ in the name of P_{src}. $\mathcal{F}_{ToR}$ will then return the $terminate_{13}$, causing $\mathcal{S}$ to terminate the operation.

(d) If G_{pp} sends $TX_{L2syncBCR}$ to P_{dst} before the block header(height if H^0_{BCR}) of relay chain is confirmed by P_{dst}, then P_{dst} will return a termination message. Similarly, $\mathcal{S}$ sends the message $msg_1 = (H^0_{BCR}, H_{BCR}, BCR, \Pi^0_{BCR})$ to $\mathcal{F}_{ToR}$ in the name of P_{src}. $\mathcal{F}_{ToR}$ will then return $terminate_{12}$, causing $\mathcal{S}$ to terminate the operation.

(e) If G_{rp} crashes, then $\mathcal{S}$ terminates.

Ideal functionality $\mathcal{F}_{rc}$

Session Setup

- Each session manages a trust root synchronizing process from P_{src}.
- Initialize $LCStorage = \{\}$ which is used to store block header from P_{src} by block height, $BCRStorage = \{\}$ which is used to store BCR from P_{src} by block height.

Synchronize Trust Root

- Upon receiving $TX_{pch2rc} = (P_{src}.id, Hdr_{src})$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{rc}$ calls $LCExist(P_{src}.id, Hdr_{src})$ to check whether Hdr_{src} exists, if it does then $\mathcal{F}_{rc}$ calls $LCVerify(P_{src}.id, Hdr_{src}.height)$ of P_{src} to verify Hdr_{src} and calls $LCStore(P_{src}.id, Hdr_{src})$ if verified, else $terminate_7(\text{"}verifyP_{src}headerfailed\text{"})$.
- Upon receiving $TX_{pcBCR2rc} = P_{src}.id, H_{BCR}, BCR, \Pi^{src}_{BCR}$ from $\mathcal{F}_{ToR}$, $\mathcal{F}_{rc}$ calls $BCRExist(P_{src}.id, H_{BCR})$ to check whether BCR of H_{BCR} has not been stored. If exist then $terminate_8(\text{"}BCRhasbeenstored\text{"})$ else $\mathcal{F}_{ToR}$ calls $LCExist(P_{src}.id, H_{BCR})$ to check whether Hdr_{src} of H_{BCR} has been stored. If not stored then $terminate_9(\text{"}norelatedHdr_{src}\text{"})$ else $\mathcal{F}_{rc}$ gets Hdr_{src} from $LCStorage[P_{src}.id]$ and calls $BCRVerify(Hdr_{src}.stateRoot, BCR, \Pi^{src}_{BCR})$. If verified then $\mathcal{F}_{rc}$ calls $BCRStore(P_{src}.id, H_{BCR}, BCR)$ to store BCR else $terminate_10(\text{"}verifyBCRfailed\text{"})$.

Description of the subroutines:

- **LCVerify:** On input a tuple $(P_{src}.id, Hdr_{src})$, execute light client algorithm of P_{src} to verify whether Hdr_{src} is valid. If Hdr_{src} is invalid return false else return true.
- **LCStore:** On input a tuple $(P_{src}.id, Hdr_{src})$, store Hdr_{src} in $LCStorage[P_{src}.id][Hdr_{src}.height] = Hdr_{src}$.
- **LCExist:** On input $height$, check whether Hdr_0 of $height$ exists in $LCStorage[height]$. If it does then return true else return false.
- **L02Verify:** On input a tuple $(P_{src}.id, height)$, check whether Hdr_{src} of height exists in $LCStorage[P_{src}.id][height]$. If it does then return true else return false.
- **BCRVerify:** On input a tuple $(root, BCR, \Pi^{root}_{BCR})$, execute Merkle tree verification algorithm of P_{src} to verify whether BCR is relevant with root.
- **BCRStore:** On input a tuple $(P_{src}.id, H_{BCR}, BCR)$, store BCR in $BCRStorage[P_{src}.id][H_{BCR}] = BCR$.
- **BCRExist:** On input a tuple $(P_{src}.id, H_{BCR})$, check whether BCR of H_{BCR} exists in $BCRStorage[P_{src}.id][H_{BCR}]$. If it does then return true else return false.

Figure 5: Ideal functionality $\mathcal{F}_{rc}$ for CSC on relay chain

Ideal functionality $\mathcal{F}_{ToR}$

Session Setup

- Each session manages a interoperability process from P_{src} to P_{dst}.
- Initialize $pcHdr2rc = \{false\}$, $pcBCR2rc = \{false\}$, $rcHdr = \{false\}$, $rcBCR = \{0\}$, $pcRh = \{0\}$, $rcBCR2pc = \{false\}$.
- Set $session.srcid = P_{src}.id$,$session.dstid = P_{dst}.id$.
- Start monitoring daemon for the parachain and relay chain in this session

Issue Cross-chain Message

Upon receiving cross-chain request $req = (srcid, dstid, payload)$ from environment $\mathcal{Z}$ where $srcid$ equals $session.srcid$, $dstid$ equals $session.dstid$ and $payload$ denotes specific cross-chain request content, $\mathcal{F}_{ToR}$ sends req to P_{src}.

Synchronize Trust Root

- Upon receiving Hdr_{src} from P_{src}, $\mathcal{F}_{ToR}$ constructs transaction $TX_{pch2rc} = (P_{src}.id, Hdr_{src})$, submits it to R_0 and sets $pcHdr2rc[Hdr_{src}.height] = true$.
- Upon receiving $msg = (BCR, \Pi_{BCR}, H_{BCR})$ from P_{src} where Π_{BCR} denotes state proof of BCR and H_{BCR} denotes height of BCR on P_{src}, if $pcHdr2rc[H_{BCR}]$ is true, $\mathcal{F}_{ToR}$ constructs transaction $TX_{pcBCR2rc} = P_{src}.id, H_{BCR}, BCR, \Pi^{src}_{BCR}$, submits it to R_0 and sets $pcBCR2rc[H_{BCR}] = true$, else $terminate_{11}$.
- Upon receiving Hdr_0 from R_0, $\mathcal{F}_{ToR}$ constructs transaction $TX_{rch2pc} = Hdr_0$, submits it to P_{dst} and sets $rcHdr2pc[Hdr_0.height] = true$.

Transmit Cross-chain Message and DLV proof

- Upon receiving $receipt_{BCR}$ of transaction TX_{BCR2R} from R_0, set $rcBCR[H_{BCR}] = receipt_{BCR}.height$.
- Upon receiving $receipt_{Hdr_0}$ of transaction TX_{Rh2P} from P_{dst}, set $pcRh[Hdr_0.height] = receipt_{Hdr_0}.height$.
- Upon receiving $msg = H^0_{BCR}, H_{BCR}, BCR, \Pi^0_{BCR}$ from R_0, if $pcRh[H^0_{BCR}]$ is true, then $\mathcal{F}_{ToR}$ constructs transaction $TX_{L2syncBCR} = P_{src}.id, H^0_{BCR}, H_{BCR}, BCR, \Pi^0_{BCR}$ and submits it to P_{dst}, sets $rcBCR2pc[H_{BCR}] = true$, else $\mathcal{F}_{ToR}$ $terminate_{12}$.
- Upon receiving $msg = (cm, H_{cm}, \Pi^{BCR}_{cm})$ from P_{src}, if $rcBCR2pc[H_{cm}]$ is true, then $\mathcal{F}_{ToR}$ constructs transaction $TX_L 1cm = P_{src}.id, cm, H_{cm}, \Pi^{BCR}_{cm}$ and submits it to P_{dst}, else $\mathcal{F}_{ToR}$ $terminate_{13}$.

Figure 6: Ideal functionality $\mathcal{F}_{ToR}$ for ToR