

Tab 1

# Gossamer's Contributions to Polkadot-SDK v2

## [3 milestones, Q3/Q4 2025]

**Proposal Date:** 8th August 2025

**Requested Amount:** 815\_358 USDC

Payouts will be distributed in equal, milestone-based installments every two months:

- **Milestone 1:** 271,786 USDC – Payout Date: 10 September 2025
- **Milestone 2:** 271,786 USDC – Payout Date: 10 November 2025
- **Milestone 3:** 271,786 USDC – Payout Date: 10 January 2026

**Beneficiary Address:** 149mJdQjEBMHbWDjbLJ7X4e95ps6L35DZVaBzn1raR1EVQ

**Short Description:** Gossamer was originally envisioned as a Go-based implementation of the Polkadot Relay Chain Validator, contributing to the decentralization and resilience of the network through client diversity. Our team, deeply committed to the Polkadot ecosystem, built Gossamer as an alternative full node and validator client to help ensure the robustness of the protocol.

However, as the ecosystem's needs have changed, it's become clear that alternative client development is not currently a top priority. In light of this, we've made a strategic decision to shift our focus from developing an alternative client to contributing directly to the existing Rust-based SDK implementation. This change aligns with both the direction of the ecosystem and our desire to maximize our impact on Polkadot's core infrastructure.

Following consultations with Parity, we have defined a set of objectives that our team will focus on over the next six months. These efforts, in close collaboration with Parity, will leverage our experience to contribute to the Polkadot-SDK's most critical areas, helping to strengthen, optimize, and scale the protocol at its foundation.

**Project Category/Type:** Software development ▾

Website: [ChainSafe](#)



## 1. Context of the Proposal

The Gossamer team at ChainSafe is a group of technical experts engaging in protocol-level implementations for the Polkadot ecosystem. The team has formerly received funding from the Web3 Foundation and the Polkadot Treasury and has also been temporarily self-funded through ChainSafe.

This proposal is a request for six months of funding from the Polkadot Treasury for the collaboration of the Gossamer team and Parity to work on the [Polkadot protocol's](#) (polkadot-sdk) pressing needs. The ecosystem is rapidly evolving, and a lot of change is to be expected. We are therefore adopting a six-month proposal cycle to maintain agility and responsiveness to best serve the ecosystem.

In this funding period, as detailed in Section 2 (*Scope of Current Proposal*), we will dedicate resources to the needs of the **polkadot-sdk** by addressing key technical priorities identified in collaboration with Parity. Our focus will be on areas where our team's expertise can deliver the greatest impact - enhancing scalability, performance, and protocol robustness. This marks a natural evolution of our role within the ecosystem, transitioning from client diversity through Gossamer to strengthening the core infrastructure that supports the Polkadot network. Our focus will be on areas where our team's expertise can deliver the greatest impact - protocol development.

## 1.1 About ChainSafe and the Gossamer Team

At ChainSafe, we are dedicated to pioneering the development of decentralized and community-oriented technologies that empower users globally. Our mission is to advance web3 potential through open-source innovation, making it more accessible, secure, and sustainable.

Outside Gossamer, some of ChainSafe's development contributions to the Polkadot ecosystem are:

- [Multix](#)
  - An interface to easily manage complex multisigs.
- [Phala SubBridge](#)
  - Bridging data and assets from/to Dotsama and Ethereum.
- [Sygma Substrate Pallets](#)
  - Enables native connectivity for assets and messages, leveraging the Sygma protocol, between EVM and Substrate-based chains.
- [Cypress Plugin - Polkadot Wallet](#)
  - A plugin that enables integration tests with wallets using the popular testing framework, Cypress.
- [Metamask Snap](#)
  - Plugin for interacting with Polkadot dApps and other Substrate-based chains.
- [Chainlink Pallet](#)
  - Integration of Chainlink feed in Substrate-based chains.
- [Gossamer](#)
  - Alternative implementation of the Polkadot Relay Chain Validator, which was developed using Go. As an alternative client implementation, it serves decentralization and robustness of the network by being a full node and a validator node.

Outside development contributions and as part of our multifaceted approach, we are enhancing the Polkadot ecosystem through targeted infrastructure improvements:

- [Snapshot Hosting Services](#)
  - Snapshots are free and allow efficient network synchronization of both Polkadot and Kusama.
- Running Validator Nodes
  - Part of the 1K Validator Program on Kusama, with ongoing efforts to expand to Polkadot.

Additionally, we continue to be active in our non-technical contributions, such as participating and presenting at Polkadot events like Decoded ([2022](#), [2023](#)), Sub0 ([2024](#)), ParisDot, [Web3 Summit](#), Polkadot Blockchain Academy, JAM Experience, and so on. We are also organizers of Polkadot meetups, both physical and online ([Croatia meetup](#), [CSCON](#)).

## 2. Scope of Work

In each section below, you can find a set of components that will be worked on in this proposal's timeline.

### Note on Scope Flexibility and Prioritization (Candidate Items)

The initial scope of work outlined in this document was developed in **close consultation with Parity**, reflecting their strategic priorities at the time of writing. We recognize that priorities may evolve. Consequently, specific elements within this scope can be deprioritized and substituted with new tasks of equivalent value to address more urgent needs. For this specific reason, points P. 2.2, P. 2.6, and P. 2.7 are marked as **Candidate Items**, as only a selection of these will be delivered based on the final prioritization. Parity will retain the final authority to define and confirm these evolving priorities throughout the project lifecycle, ensuring the engagement remains aligned with their most critical objectives.

### 2.1 Approval-checking rewards

#### Why This Is Important

The Polkadot Approval process, a core function of the Parachain Host, is vital for ensuring only valid parachain blocks are finalized on the Relay Chain. It holds backing validators accountable and underpins the network's integrity and security.

In this process, randomly selected validators reconstruct a parachain block's Proof-of-Validity (PoV) data by downloading Data Availability chunks and performing complex erasure coding recovery. They then verify this PoV to confirm a valid state transition and aligned outputs.

Currently, the approval process lacks both rewards for participation and proactive slashing for non-participation. This omission creates a vulnerability, potentially leading to validator misbehavior and significant security risks for the network.

## **Approach**

The “Approval Rewards” mechanism relies on a decentralized, vote-based protocol where validators anonymously assess the approval participation of their peers (excluding themselves). These assessments are gossiped across the network and used to compute a **median approval count** for each validator.

Each validator independently calculates this median list, constructs a Merkle tree from it, and signs the Merkle root. If a majority of the validators sign the same root, the corresponding median list is posted on-chain and used to determine rewards. Validators who meet or exceed the median threshold are rewarded; underperformers may be penalized or slashed.

If no majority is reached - typically due to network or gossip failures - no rewards are distributed for that era, though governance may intervene in rare cases. This design ensures rewards reflect observed behavior and protects against manipulation or collusion.

## **Key deliverables**

- **Finalization of [RFC-119](#):** Completion and formal approval of the RFC defining the Approval Rewards mechanism, including scoring models, gossip protocol details, and on-chain integration.
- **Off-Chain Median Voting Protocol Implementation:** A decentralized system where validators gossip peer approval scores, compute median participation lists, and vote on Merkle root hashes for on-chain consensus.

- **On-Chain Verification and Reward Logic:** Runtime functionality to verify signed Merkle roots, establish quorum agreement, and handle reward distribution or fallback conditions in case of disagreement.

## 2.2 Parachain runtime upgrades off-chain R&D [Candidate Item]

### **Why This Is Important**

[Off-chain runtime upgrades proposal](#) is crucial for making the Polkadot network more efficient and scalable. Currently, when a parachain updates its code, the entire new code block must be sent to and stored on the relay chain. This is a very costly and slow process, as a single update can take up an entire relay chain block, impacting all other parachains. By moving this process "off-chain" the network can handle these upgrades without clogging up the main chain. This frees up valuable block space, making the network faster and more resilient, especially as more parachains are expected to join in the future.

The off-chain upgrade mechanism also [significantly lowers the cost](#) and effort required for parachain teams to update their runtimes. The old method requires a large storage deposit, which is a major barrier to entry for new projects. The new approach reduces these costs and gives developers a smoother, more flexible way to manage their code. Ultimately, this change makes Polkadot more accessible and cheaper for developers while also improving the overall performance and stability of the network for all users.

### **Approach**

The implementation of off-chain runtime upgrades hinges on a few key changes to the node and relay chain logic. First, the process is initiated when a parachain sends a new, lightweight `RequestCodeUpgrade(Hash)` UMP message. This message contains only the hash of the new Parachain Validation Function (PVF), not the code itself, preventing relay chain bloat. Upon seeing this message, backers use a new peer-to-peer

request/response protocol to fetch the full PVF directly from the parachain's collators. This off-chain distribution method ensures the large PVF file never touches the relay chain's state.

To ensure network-wide availability without compromising liveness, the upgrade is enacted through a staged rollout. The relay chain sets a countdown (e.g., 10 blocks), communicated to validators via a runtime API. During this period, validators fetch the new PVF from backers in the background. After the countdown, the protocol logic for availability is altered: a validator will only affirm a candidate's availability if it possesses both the data chunk and the necessary PVF. This guarantees that a supermajority of validators have the new code before it's passed to the existing pre-checking and enactment process, securing the transition.

**Leveraging Existing Message Protocols:** We'll use existing UMP/DMP (Upward/Downward Message Passing) to signal runtime upgrades between collators and validators, avoiding new message types.

**Building an Off-Chain Data Availability System:** A new system will be developed to store **all versions of PVFs off-chain**, ensuring historical runtime accessibility for syncing and re-execution.

**Adapting Existing Data Availability Subsystem:** We'll partially reuse the current design for **availability distribution** and **bitfield signing** to efficiently distribute PVF data among validators and ensure they have the necessary code for validation.

### **Key deliverables**

NOTE: First phase only includes R&D with PoC and idea verification on private testnets.

- **Finalization of [RFC-102](#):** Completion and formal approval of the RFC defining the Off-chain runtime upgrades mechanism.
  - Resolve open questions, such as the mechanism for charging a deposit for off-chain code and how to handle the initial runtime.

- Refine the full design of the new UMP message and the request/response protocol.
- Complete the details for the new runtime API and the counter-based rollout mechanism.
- Document the precise behavior changes required for validators and collators.
- **Proof-of-Concept & Verification:**
  - Develop and test the new peer-to-peer communication API based on existing UMP/DMP for code distribution.
  - Implement and verify the logic for the counter mechanism and how validators use it to manage the code-fetching process.
  - Conduct internal testing to confirm that the off-chain distribution is secure and reliable, mitigating attack vectors and ensuring availability.
- **Testnet Deployment and Integration:**
  - Integrate the finalized code into a testnet implementation.
  - Run extensive tests to ensure the new upgrade process works seamlessly for parachains without disrupting other chains.
  - Verify that the new runtime update behavior works as intended.

## 2.3 Optimizing the dependency footprint

### **Why This Is Important**

The polkadot-sdk's current dependencies list size presents several challenges that impact developer experience, build efficiency, and overall security posture. Addressing these issues is crucial for:

- **Shorter Compile Times:** Excessive or unoptimized dependencies lead to significantly longer compilation times. This creates friction for developers, slows down iteration cycles, and increases the time required for CI/CD pipelines.
- **Simplified Build Process:** Dependencies on non-Rust tools and libraries (e.g., OpenSSL) introduce complexities into the build environment, requiring developers to set up additional toolchains beyond the standard Rust one. This increases the barrier to entry for new contributors and makes builds less reproducible.
- **Reduced Attack Surface for Supply Chain Attacks:** Every third-party dependency introduces a potential vulnerability point. A large and unvetted dependency tree increases the "supply chain attack" surface, where malicious code could be injected into the software through a compromised upstream package. Reducing the number of dependencies, especially high-risk ones, directly mitigates this threat.

## **Approach**

- **Identify and Prioritize Existing Issues:** Begin by addressing known dependency-related issues within the polkadot-sdk repository (e.g., 777).
- **Leverage Rust-Specific Supply Chain Security Tools:**
  - Utilize `cargo vet` to manage and verify third-party Rust dependencies, ensuring trusted entities have audited them.
  - Employ `cargo audit` to scan for known vulnerabilities in the dependency tree.
  - Explore `cargo-auditable` to embed dependency lists into binaries, facilitating post-build auditing.
  - Use cargo `supply-chain` to analyze the overall supply chain integrity of the project's dependencies.
- **Feature Management:** Ensure that optional features in crates (e.g., WebRTC in litep2p) are correctly deselected when not required by polkadot-sdk to avoid pulling in unnecessary transitive dependencies.

- **A little copying is better than a little dependency.** Define dependencies that can be copied inside the repository instead of being imported.
- **Provide reports/issues/RFCs on what can be done.** Initiate discussions regarding dependency findings to form consensus among developers.

### **Key deliverables**

- Reduced Dependency Footprint in polkadot-sdk.
- Enhanced Supply Chain Security Workflows.
- Shorter compile times.

## 2.4 WebRTC transport for browser-based LightClients

### **Why This Is Important**

The current methods for Polkadot light clients to connect to full nodes suffer from significant drawbacks, most critically the need to trust RPC nodes. WebRTC's design for peer-to-peer (P2P) communication makes it the ideal choice for facilitating direct connectivity between browser-based light clients (like smoldot compiled to WebAssembly) and substrate-based nodes. By enabling direct P2P connections, WebRTC reduces reliance on traditional RPC endpoints, which are often centralized and can become bottlenecks or single points of failure.

### **Approach**

The project will focus on enabling robust WebRTC-based peer-to-peer (P2P) connectivity between browser-based light clients (such as smoldot compiled to WebAssembly) and substrate-based nodes via the litep2p transport layer in polkadot-sdk.

The implementation will proceed in three main phases:

### **1. Assessment and Stabilization of Existing Work**

- Investigate and build upon prior attempts to implement WebRTC transport in Substrate and polkadot-sdk.
- Collaborate with the smoldot and litep2p team to understand current architectural compatibility and integration requirements.

### **2. Litep2p Enhancements and Issue Resolution**

- Tackle critical issues identified in the litep2p GitHub tracker to ensure a stable, spec-compliant, and performant WebRTC transport.
- Maintain ongoing discussions with relevant maintainers to align efforts and validate design decisions.

### **3. Integration with Smoldot and polkadot-sdk**

- Implement and validate full WebRTC transport support in polkadot-sdk, enabling nodes to accept WebRTC connections.
- Integrate WebRTC support in smoldot, ensuring browser-based light clients can establish secure, functional connections to substrate-based nodes.
- Evaluate the feasibility and long-term value of bringing litep2p directly into the smoldot codebase to streamline integration.

## **Key deliverables**

### **● Functional WebRTC Transport in polkadot-sdk**

- A merged and stable implementation of WebRTC transport (implemented within litep2p) within the polkadot-sdk codebase.
- This will enable polkadot-sdk nodes to accept and manage WebRTC connections from light clients.

### **● Browser-Based smoldot Connectivity**

- Demonstrable capability for smoldot (compiled to WebAssembly) to connect directly to a polkadot-sdk node via WebRTC from within a web browser.

- This includes successful establishment of connections, data exchange, and basic light client functionality.
- **Reliability and Performance Improvements**
  - Measurable improvements in the reliability and performance of litep2p's libp2p-webrtc implementation, addressing the issues outlined in <https://github.com/paritytech/litep2p/issues/312>.
- **Documentation and Examples**
  - Comprehensive documentation for developers on how to enable and use WebRTC connectivity in polkadot-sdk nodes and smoldot light clients.
  - Working code examples demonstrating the WebRTC connection setup and usage.

## 2.5 Speculative Availability

### Why This Is Important

This improvement is important for significantly enhancing the efficiency and user experience of the Polkadot network, particularly in the context of its Asynchronous Backing mechanism. By addressing delays in core availability, it aims to unlock greater throughput and improve transaction reliability.

**Maximizing Core Utilization for Increased Throughput:** Polkadot's core computational units (cores) are temporarily occupied while a parachain block (parablock) is undergoing availability checks after being backed. Asynchronous Backing, a key advancement for Polkadot 2.0, already reduces block time from 12 to 6 seconds and allows for pipelining of backing and inclusion. However, timely core release is still vital. Finishing availability checks sooner means a core can be reused earlier for new parachain blocks, directly increasing the overall system throughput and leveraging the benefits of Asynchronous Backing more fully. This is especially important

as parachains aim to produce blocks more frequently and with larger execution times (up to 2 seconds compared to 0.5 seconds previously) as enabled by Asynchronous Backing.

**Delayed Inclusion possibly can lead to "Lost Transactions"**: A significant pain point for users is the occasional "[disappearance](#)" of *parachain* transactions, often requiring resubmission. One leading hypothesis for this is that a parachain candidate might be backed on the Relay Chain but then fail to be included because its availability distribution process (where validators collect erasure chunks and Proofs-of-Validity (PoVs)) hasn't completed within the required timeframe (currently limited to 1.5 seconds, starting *after* backing).

### **Approach**

- The primary goal is to empirically prove or disprove the hypothesis that "lost transactions" are indeed caused by parachain candidates being backed but *not* included due to availability distribution failures.
- Proceed with implementing the speculative availability enhancements, largely based on the [suggestions](#)
  - Make the code for fetching *backable* candidates from the prospective parachains subsystem reusable from subsystems other than provisioner, [where it currently lives](#).
  - Modify the availability distribution subsystem to call this refactored code. This will enable it to proactively provide erasure chunks and PoVs for backable candidates to non-backing validators *earlier* in the process.
  - Implement logic for non-backing validators to speculatively fetch these erasure chunks and PoVs as soon as a candidate is backed off-chain, rather than waiting for the block author to select it. This pre-fetching is the core of "speculative" availability.

### **Key deliverables**

- **Speculative Availability Implementation**

- Implement the core logic for speculative availability, allowing non-backing validators to proactively fetch erasure chunks and Proofs-of-Validity (PoVs) as soon as a seconded statement has been received for a candidate.
- Modify the Availability Distribution subsystem to leverage this logic, initiating availability and inclusion processes *before* a candidate is officially backed on-chain.
- **Empirical Validation and Metrics**
  - Conduct an analysis to validate/invalidate the hypothesis that "lost transactions" stem from availability distribution delays post-backing.
  - Demonstrate measurable improvements in the Availability Distribution, and Inclusion processes.

## 2.6 Polkadot network chaos testing and DDoS protection

### [Candidate Item]

#### **Why This Is Important**

Testing decentralized systems is an inherently complex endeavor, characterized by a vast array of potential attack vectors and unique operational challenges. A foundational understanding here is critical: all decentralized systems are, by their very nature, distributed, relying on interconnected nodes operating without a central authority. This inherent distributed architecture means that a significant subset of potential issues in decentralized networks directly stems from their underlying distributed nature. Our focus is precisely on this intersection, aiming to identify issues common to all distributed systems while also diving deeper to test for internal consensus problems by attacking the system from within. This involves simulating consensus-level failures by introducing faulty or malicious validators to analyze how the network withstands coordinated attacks

where trusted participants turn hostile. Luckily, this is a long-standing issue, and we have a variety of [tools](#) to utilize, plus our deep knowledge of the protocol.

## **Approach**

The approach will involve leveraging the Gossamer codebase (or existing Parity validator) to build what we called the Chaos Gossamer tool to simulate and test various DDoS and consensus attack scenarios.

The core strategy is as follows:

- A specialized Gossamer node will be developed where its internal state and network messaging behavior can be precisely controlled via an API.
- This "Chaos Gossamer" node will serve as the primary tool for generating controlled disruptive traffic and testing network resilience under adversarial conditions.
- **A primary focus** will be on identifying ways to cause disruptions *without* running a validator in the active set. This is critical because the absence of a staked economic deterrent (risk of slashing) makes such attacks potentially more damaging and accessible.
- **Additional focus** will include, but is not limited to, overwhelming network resources (e.g., sending a million packets and reading them very slowly to induce TCP/IP congestion) to the validators in active sets.
- **In the future** this tool can also be used for "attacking" a network from within an active validator set (disputes, wrong backings, approvals, etc.).

## **Key deliverables**

- Build a Chaos Gossamer tool using Go, deliberately chosen to leverage the rich ecosystem of distributed systems tooling available in the language. This tool will integrate with the Gossamer codebase - which already implements most of the Polkadot protocol primitives and networking architecture - and connect it with existing tools for testing distributed networks. The goal is to provide comprehensive and definitive testing capabilities for Polkadot-based systems.

- Design and implement a comprehensive suite of test scenarios to evaluate the behavior, resilience, and performance of the network under a wide range of distributed conditions and failure modes.
- Generate detailed reports on scenario execution, including metrics, outcomes, anomalies, and insights, to support debugging, validation, and continuous improvement of the protocol and network stack.
- Initially, the report and codebase will most likely be privately shared only with Parity and the Technical Fellowship for security reasons.

## 2.7 Smart Contracts Infrastructure Integration (Foundry-Polkadot) [Candidate Item]

### **Why This Is Important**

This initiative is focused on directly assisting Parity in building the Polkadot Foundry, a critical piece of infrastructure designed to unlock the full potential of smart contracts on AssetHub. By providing a powerful and familiar development toolkit analogous to Ethereum's popular Foundry, we will empower developers to transform AssetHub from a simple asset ledger into a dynamic platform for programmable assets, enabling sophisticated DeFi and NFT applications. This work is strategically vital for lowering the barrier to entry for the vast community of existing blockchain developers, accelerating innovation and user adoption across the entire Polkadot ecosystem by delivering a best-in-class developer experience.

### **Approach**

The general approach will focus on delivering the **Anvil** part of the Polkadot-Foundry based on the task scope that is outlined in the [Epic](#).

### **Key deliverables**

- Functional, standalone, Anvil-compatible Polkadot test node that integrates pallet-revive for local smart contract testing.
- Implementation of the required RPC methods to ensure full compatibility with the Foundry toolkit and other developer tools.
- Successful integration of the Anvil Polkadot node into the main Foundry framework, allowing developers to use it as a standard testing environment.
- Support for executing deployment scripts via the forge script command, enabling a complete development and testing lifecycle.

## **3. Budget**

### **3.1 Current Proposal Budget Description**

The Gossamer core team currently consists of:

- 1x Full-Time Protocol Product Manager/Senior Engineering Manager
- 8x Full-Time Protocol Engineers

The total funding request covers a six-month period (**July 2025 – December 2025**) and includes all salary and ancillary (infrastructure, tooling, operations, and taxes) costs for the team.

### **Compensation logic**

- Average engineering rate: **\$94.37/hour**, reflecting the team's expertise and responsibilities.

- Full-time assumption: **160 working hours/month** per engineer, accounting for holidays and leave (averaged to 24 days off per year).
- Some engineers also support QA and DevOps as part of their protocol roles.

This results in the following cost breakdown for the **six-month proposal period** (July 2025 - December 2025):

Role	# of FTE	Hourly Rate (USD)	Monthly Hours	Months	Subtotal (USD)
Protocol Eng. Manager / Sr. Eng. Manager	1	\$94.37	160	6	\$90,595
Protocol Engineer	8	\$94.37	160	6	\$724,765
Ancillary Costs (infra, ops, etc.)	—	—	—	6	Included
<b>Total (USD)</b>					<b>\$815,358</b>

We are requesting a total of **\$815,358 USD for 6 months of development** work for a 9-person team. This includes all salary and ancillary costs, resulting in an average team cost of **~\$135,000 per month**.

*The proposed monthly salary costs for our protocol engineers remain below **\$150,000 USD per engineer per year**, which is competitive with current market rates. This is supported by industry benchmarks such as:*

- <https://web3.career/web3-salaries> (Go Engineer table entry)
- <https://metana.io/blog/web3-developer-salary-2025-what-you-can-earn-in-blockchain/>
- <https://web3.career/protocol-engineer-jobs>

## 3.2 Payout Structure

Payouts will be distributed in equal, milestone-based installments every two months:

- **Milestone 1:** 271\_786 USDC – Payout Date: 10 September 2025
- **Milestone 2:** 271\_786 USDC – Payout Date: 10 November 2025
- **Milestone 3:** 271\_786 USDC – Payout Date: 10 January 2026

This structure ensures a retroactive compensation model while also establishing a security mechanism that allows the community to halt future payouts if ChainSafe fails to meet its milestone obligations.

## 3.3 Total Budget of This Proposal

**Beneficiary account:** 149mJjdQjEBMHHbWDjbLJ7X4e95ps6L35DZVaBzn1raR1EVQ

**Cost (in USDC):** 815\_358

## 4. Progress, Updates, and Delivery

### 4.1 Milestones

**Note:** As outlined in [Section 2](#), milestone details are subject to change, provided that any modifications are formally agreed upon in writing with Parity and are publicly communicated to the Polkadot community.

Currently, we outline three milestones:

#### Milestone 1: Jul - Aug 2025

Initiative	Deliverables
Approval-checking rewards	<ul style="list-style-type: none"><li>- Initial work design and active RFC drafting participation</li><li>- Off-chain Approval rewards calculations (next <i>ApprovalsTally</i>) based on <code>approval_usages</code> and <code>noshows</code>.</li><li>- Off-chain <i>ApprovalsTally</i> distribution among active validators</li><li>- Computing rewards medians across validators</li></ul>
Speculative Availability	<ul style="list-style-type: none"><li>- Initial research and spec drafting</li><li>- Implement speculative pulling of backable candidates from prospective parachain</li></ul>
Parachain runtime upgrades off-chain R&D	<ul style="list-style-type: none"><li>- Initial feasibility research</li></ul>
Optimizing the dependency footprint	<ul style="list-style-type: none"><li>- Remove unused dependencies</li><li>- Remove curve-25519-dalek dependency</li><li>- Investigation on <a href="#">Relocation</a> of Cryptographic Implementations</li></ul>
WebRTC transport for browser-based LightClients	<ul style="list-style-type: none"><li>- Research on WebRTC-Direct</li><li>- Research on the <a href="#">current state of litep2p</a> WebRTC support and integration into polkadot-sdk.</li><li>- Research Smoldot libp2p WebRTC integration</li><li>- Align with litep2p developers on a roadmap to improve robustness and stability of WebRTC support.</li></ul>
DDoS protection	<ul style="list-style-type: none"><li>- Initial research</li></ul>

<b>Smart Contracts Infrastructure Integration (Foundry-Polkadot)</b>	Getting familiar with the codebase/docs, having a sync meeting, and identifying the best area for contribution.
--	---

## Milestone 2: Sep - Oct 2025

Initiative	Deliverables
<b>Approval-checking rewards</b>	<ul style="list-style-type: none"> <li>- Implementation of on-chain submission of <i>ApprovalsTally</i></li> <li>- On-chain computation of participation rewards for the approval process, utilizing <i>approval_usages</i> and <i>noshows</i> metrics</li> <li>- Private testnet deployment for functional validation and metrics collection</li> <li>- Off-chain implementation of rewards calculations based on <i>Availability chunks uploads and downloads</i></li> </ul>
<b>Speculative Availability</b>	<ul style="list-style-type: none"> <li>- Private testnet deployment for functional validation and metrics collection</li> </ul>
<b>Parachain runtime upgrades off-chain R&amp;D</b>	<ul style="list-style-type: none"> <li>- Finalising RFC</li> <li>- Work on PoC</li> </ul>
<b>Optimizing the dependency footprint</b>	<ul style="list-style-type: none"> <li>- Removal of unused features</li> <li>- Simplification of Dependency Tree</li> <li>- Work on Relocation of Cryptographic Implementations.</li> </ul>
<b>WebRTC transport for browser-based LightClients</b>	<ul style="list-style-type: none"> <li>- Complete development efforts to improve robustness and stability of WebRTC in litep2p.</li> <li>- Revisit existing attempts to integrate litep2p WebRTC into polkadot-sdk.</li> <li>- PoC WebRTC communication between node and lightclient.</li> <li>- Private testnet deployment preparation for testing of Substrate-based chains utilizing WebRTC.</li> </ul>
<b>DDoS protection</b>	<ul style="list-style-type: none"> <li>- Implement basic traffic filtration rules based on Polkadot GridTopology</li> <li>- Conduct a security analysis of the consensus networking module to identify potential vulnerabilities.</li> </ul>

<b>Smart Contracts Infrastructure Integration (Foundry-Polkadot)</b>	Deliverables will be announced based on initial specification work.
--	---

### Milestone 3: Nov - Dec 2025

<b>Initiative</b>	<b>Deliverables</b>
<b>Approval-checking rewards</b>	<ul style="list-style-type: none"> <li>- On-chain implementation of <i>ApprovalsTally</i> rewards calculations based on <i>Availability chunks, uploads and downloads</i></li> <li>- Support phase *</li> </ul>
<b>Speculative Availability</b>	- Support phase*
<b>Parachain runtime upgrades off-chain R&amp;D</b>	- Finalising PoC for private testnet deployment
<b>Optimizing the dependency footprint</b>	<ul style="list-style-type: none"> <li>- Ongoing Removal of Unused Features</li> <li>- Ongoing Simplification of Dependency Tree</li> </ul>
<b>WebRTC transport for browser-based LightClients</b>	<ul style="list-style-type: none"> <li>- Address issues that arise from private testnet utilizing WebRTC.</li> <li>- Work towards inclusion in polkadot-sdk and inclusion in a Polkadot release.</li> <li>- Support phase*</li> </ul>
<b>DDoS protection</b>	<ul style="list-style-type: none"> <li>- Private testnet deployment of GridTopology filtration rules</li> <li>- Compile and share an initial confidential report detailing identified security vulnerabilities within the consensus networking module</li> <li>- Evaluate potential mitigation strategies and develop a structured execution plan for the upcoming project milestones.</li> </ul>
<b>Smart Contracts Infrastructure Integration (Foundry-Polkadot)</b>	Deliverables will be announced based on initial specification work.

*\*By the support phase, we refer to facilitating the deployment process to a public testnet or mainnet\*\*, including metering and tracing of the solution to ensure correct execution within a decentralized network.*

*\*\*ChainSafe cannot solely guarantee public mainnet/testnet deployment, as we do not control the full integration and deployment pipelines. Additionally, certain updates require community approval through RFCs and runtime upgrade proposals.*

## 4.2 Progress and Updates:


To maintain transparency, reports on milestone progress will be continuously shared via:

- Monthly progress reports (GitHub [discussions](#), [X](#), [polkadot forum](#))
- GitHub code progress (see [GitHub project board](#))
- Presentations at Polkadot events

## 5. Contact

- Kyrylo Pisarev, Senior Engineering Manager, [kyrylo@chainsafe.io](mailto:kyrylo@chainsafe.io), @p1sar:[matrix.org](https://matrix.org)
- Peter Kalambet, Head of Protocol, [peter@chainsafe.io](mailto:peter@chainsafe.io), @peter:[dod.ngo](https://dod.ngo)
- Belma Gutlic, VP of Engineering, [belma@chainsafe.io](mailto:belma@chainsafe.io), @morrigan.iv:[matrix.org](https://matrix.org)

## 6. FAQ

 FAQ - Gossamer's Contributions to Polkadot-SDK v2