Treasury Proposal: Gosemble, a framework for building Substrate-compatible Runtimes in Go (Parachains & Solochains) - Phase 2

Proponent: 15reUHgnkE9QeH9zUoduCVYAbksTmQXvpMd6L95rBhTNFuGw, LimeChain
Date: 13.07.2023
Requested DOT: 63,916 DOT
Short description: Advance the Go-based Runtime into a versatile framework, marking the ongoing effort to establish a reusable toolkit for building Substrate-compatible runtimes in Go.
Project Category/Type: Software development
Previous treasury proposals:

- https://polkadot.polkassembly.io/post/1332
- https://polkadot.polkassembly.io/motion/292

---

1. Context of the proposal

This proposal is a follow-up to Framework for building Substrate-compatible Runtimes in Go. As mentioned in the previous proposal, the only current option for developing runtimes is Substrate. We believe this lack of diversity inhibits the decentralization ethos that is inherent to blockchain technology. Additionally, Rust, the language in which Substrate is written, is known for its steep learning curve.

The initial proposal describes the development process split into several phases, with the first being the building of a Substrate-compatible runtime in Go. We have written a report that describes the delivery of these milestones. The framework is now branded Gosemble, with detailed documentation found here.

With the progress made, our development team is confident that implementing **Phase 2 - Framework for runtimes in Go** is the next logical step. Modularising the codebase and developing additional modules will enhance progress and allow for an alternative framework to Substrate in Go. This will increase developers adoption by making it more accessible to the broader Go development community and contribute to the decentralization and security of the protocol.

At LimeChain, we feel confident in and have a lot of development hours invested building a framework for runtimes in AssemblyScript and the progress from the first phase. Our development in the Polkadot ecosystem has been stretched out to develop a Light client implementation in Java and Parachain Validation Conformance Testing. We have extensive

experience in building developer tooling, specifically within Rust/WebAssembly (matchstick). Apart from that, we are infrastructure contributors to Cosmos and Hedera Hashgraph.

2. Problem statement

The ultimate goal is to create a Go framework as an alternative to Substrate for runtime development. At the moment, Rust is the only programming language used for developing runtimes in Polkadot. However, developing with Rust can be quite challenging, which can make it difficult for new developers to enter the ecosystem. To address this, so far our team has taken a modular approach during the development in the first phase. The goal is to formalize these modules into a framework that other developers can reuse. By doing so, we hope to lower the learning curve for Polkadot runtime development with Golang, making it easier for new developers to get involved. Golang is a popular programming language that is known for its simplicity and ease of use. We believe that this framework has the potential to increase developer adoption and help grow the Polkadot ecosystem.

3. Proposal objective(s) or solution(s)

The primary objective of this proposal is to evolve the current codebase proof of concept into a comprehensive, modular, and adjustable framework. This is a follow-up proposal to the first phase and part of our long-term goal to gradually make a solid alternative to Substrate in Runtime development and enable wider adoption of the Polkadot ecosystem.

a. How does this proposal change the network? How do the milestones of the project achieve the ultimate goal?

The proposal does not enforce any changes to the Polkadot network in the interim timeframe. We believe that the milestones listed below are necessary for creating an alternative framework to Substrate for runtime development.

b. Who does this solution help?

As mentioned in the previous phase, this proposal offers implicit and long-term benefits to stakers, node operators, developers, network users and DOT holders. We have noticed that developer communities have expressed interest in building runtimes in languages other than Rust, which we believe will encourage developer adoption. Additionally, during the development of the first phase, the team has found inconsistencies between the Polkadot implementation and specification. Developing a second implementation to the runtime specification will better formalize it and increase the protocol security.

c. Have you seen similar solutions before?

Currently, there is no other solution for developing runtimes in Go.

d. What are the milestones, timelines, and budgets?

Team

The proposal will be executed with a team of:

- 3 full-time blockchain developers
- 1 part-time blockchain developer
- 1 part-time developer relations
- 1 part-time blockchain architect
- 1 part-time project manager

Milestones

Milestones	Tasks	Deliverables	Notes
0	Auxiliaries, Wasm compatibility		
	1.1 Compiler	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#a23130b1abb940b58889ecb959f02bf0	• Update to the latest TinyGo compiler version whenever a new stable version is released. • Prepare tests, which are executed under a wasm environment.
	1.2 Integration tests	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#4da195a255124a95b02947d1d9f3c2bd	The deliverable is spread across the other deliverables.
	1.3 CI/CD	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#395d44f38f44451fba0241fc15380c	Add Github Action that builds and exports the runtime wasm blob whenever a new version tag is created in the code repository. Extend the unit & integration workflow if necessary.

		92	
1.	Modularisation	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#5d22fe22da8a435381b5923c878db7c3	Abstract the entire codebase into different modules. Be able to easily create a main file, which describes the whole API and configuration of the runtime.
2.	Fixes and Improvements	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#712fd2174b444d00a84dfaec375cacb9	• Fix TinyGo tests that file due to changes of the wasm build target. • Include SCALE in TinyGo testing • Develop a new GC implementation that uses the host's allocator ext_alloc_free function • Try to integrate Gossamer's SCALE • Refactor metadata functionality and implement V15 support.
3.	Benchmark & Optimisations	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#7b0c57d454e645d384a1cf68a6176444	• Execute benchmarking on runtime functionality and generate weights • Be able to build the runtime with optimisations: compactability & compression.
4.	Repository	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#a451e3cf02d84f46a5f965ebc3e704c8	In the code repository, document the: • Code of conduct • Contribution guidelines • Issue and pull request templates • Changelog • Dependabot
5.	Documentation & Guides	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#32332d79b1424dcfbb7ae103c755c0c9	Revise all the documentation from the other milestones. Add guides on how to: • Add a new module to Gosemble • Modification of configuration parameters • Run a parachain

6.	Additional Modules	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#87f697f016ca47618ba520b0fed4a8c9	This milestone aims to update existing modules to their latest implementation and add new ones. ● Update **balances** module ● Implement **system** module ● Implement **session** module ● Implement **babe** module ● Implement **grandpa** module ● Implement **staking** module
7.	Run a Parachain and connect to the Testnet relay chain	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#a239226339384484bcf782a48feb0be1	This milestone aims to demonstrate the capability of running a Parachain using Gosemble and connecting it to a Testnet Relay Chain.
8.	Developer Relations	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#49a6bf6522244e0ca1d186750de0fa66	The goal of this milestone is to position Gosemble as a fully-fledged runtime framework. This will involve producing educational content, gathering community and developer feedback through customer development.
9.	Research	https://limechain.notion.site/Phase-2-Milestones-fc39245e4dd34c24b28ec076fe189a08#9610e1ee589944eb9cbe3dca01ecb2dc	The purpose of this milestone is to allocate a small amount of effort towards researching and exploring different methods for improving the framework.

Timelines

Tasks	FTE #1	FTE #2	FTE #3	PTE #1	Architect	Dev Relations	PM	Total time
0. Auxiliaries, Wasm compatibility	15.5 days	15.5 days	15.5 days	7.75 days	1.75 days	-	1.75 days	57.75 days
1. Modularisation	27 days	27 days	-	-	2 days	-	1.75 days	57.75 days
2. Fixes and Improvements	20.5 days	20.5 days	20.5 days	10.25 days	2 days	-	2 days	75.75 days
3. Benchmark & Optimisations	19.5 days	19.5 days	19.5 days	9.75 days	2 days	-	2 days	72.25 days
4. Repository	2 days	2 days	2 days	-	2 days	-	1.75 days	9.75 days
5. Documentation & Guides	6 days	6 days	6 days	-	2 days	-	1.75 days	21.75 days
6. Additional Modules	43 days	43 days	43 days	21.5 days	2 days	-	1.75 days	154.25 days
7. Run a Parachain and connect to the Testnet relay chain	12 days	12 days	12 days	6 days	2 days	-	1.75 days	45.75 days
8. Developer Relations	-	-	-	-	2 days	25 days	1.75 days	28.75 days
9. Research	4.5 days	4.5 days	4.5 days	2.25 days	2 days	-	1.75 days	19.5 days
TOTAL TIME	150 days	150 days	123 days	57.5 days	19.75 days	25 days	18 days	543.25 days

Budgets

Tasks	Hourly rate	Costs
0. Auxiliaries, Wasm compatibility	$77	$35,574
1. Modularisation	$77	$35,574
2. Fixes and Improvements	$77	$46,662
3. Benchmark & Optimisations	$77	$44,506
4. Repository	$77	$6,006
5. Documentation & Guides	$77	$13,398
6. Additional Modules	$77	$95,108
7. Run a Parachain and connect to the Testnet relay chain	$77	$28,182
8. Developer Relations	$77	$17,710
9. Research	$77	$12,012
TOTAL COSTS		**$334,732**

e. Do you have other resources or additional information?

Github Repository

4. Proposal report

Evaluating the 'success' of this proposal will be by meeting the expectation targets, specified in the Success metrics table below. We intend to fill out a Treasury proposal report with our progress and send it to the necessary communication channels.

a. What are the success metrics and targets linked to the deliverables?

Success metrics

Milestones	Deliverables	Targets	Notes
0.	Auxiliaries, Wasm compatibility		
	1.1 Compiler	Git branches in LimeChain's TinyGo fork.	

		These branches must include the addition of the Gosemble wasm target based on the TinyGo versions (f.e. 0.26, 0.27, 0.28), with the corresponding garbage collector implementations. An option would be to have docker images for each of those branches, deployed in Limechain Gosemble docker organization. Addition of tests, which validate that functionality used in Gosemble runs as expected after compilation to wasm environment.	
	1.2 Integration tests	Integration tests for all Runtime API, including new modules. The tests should be executed against a build of the runtime in wasm environment.	
	1.3 CI/CD	• New Github Action workflow, which triggers whenever a new version tag is released. The workflow will create a build of the runtime from the tag commit hash. • Extended or modified github workflow for integration tests and unit tests.	
1.	Modularisation	• Refactor how modules are written. Each module must have a configuration where all parameters can be modifiable. • Code coverage from unit tests must be close to 100%. • Include inline documentation and description on how to create a module. • Provide an example template main file, which describes the minimal necessary interface of a runtime.	
2.	Fixes and Improvements	• Fix TinyGo tests that are currently broken due to our changes. • A GC implementation for TinyGo, which uses the host allocation function called ext_allocator_free. • After upgrading TinyGo to the new versions, where Go reflection support	

		is better, try to integrate Gossamer's SCALE. • Refactor metadata implementation to use reflection and add support for V15. Include unit tests and inline documentation. • Remove unnecessary submodules and convert them into Go dependencies.	
3.	Benchmark & Optimisations		
	Execute benchmark tests and generate weights	• Create benchmarking tests, which includes the formula of calculating weights. • Benchmark all the functionality, including module hooks. • Document the flow on benchmarking a module. • Create a template file for weights, which once weights are benchmarked, it fills the weight values and imports the file into the given module.	
	Optimisations	Have a build parameter, which whenever called builds Gosemble runtime with optimized compactability and compression.	The description of the parameter must be in the build section of the documentation.
4.	Repository		
	Code of conduct	A markdown file, which describes the rules, responsibilities, standards, scope and enforcements. File must be situated in the code repository and referenced in the README.	
	Contributor guidelines	A markdown file, which describes the contributor guidelines. File must be situated in the code repository and referenced in the README.	
	Issues and pull request templates	Issue and pull request templates, used whenever someone wants to open a Github issue or pull request.	Must be situated in the .github folder in the code repository.

	Changelog	Create a CHANGELOG markdown file, which lists the changes made in each version of the runtime repository. File must be situated in the code repository and referenced in the README.	

5.	Documentation & Guides	• Add missing documentation and inline documentation throughout milestones 1-7.
		• Create the following guides:
		  ○ Adding a new module
		  ○ Modifying module configuration and properties of a runtime
		  ○ Run a parachain locally using Substrate and Gossamer.	

6.	Additional Modules	• Update the **balances** module • Implement the **system** module. • Implement the **session** module. • Implement the **babe** module. • Implement the **grandpa** module. • Implement the **staking** module.	All modules include the following: • Implementation • Unit tests • Integration tests • Benchmarking and weight values.

7.	Run a Parachain and connect to the Testnet relay chain	• Implement CollectCollationInfo API.
		  ○ Unit tests
		  ○ Integration tests
		• Implement the cumulus pallet module.
		  ○ Unit tests
		  ○ Integration tests
		  ○ Benchmarking tests and weights
		• Run a parachain in local development
		  ○ Document the whole flow on starting a relay chain and a parachain.
		• Run a parachain on testnet.	

8.	Developer Relations	• Establish a community channel for support, questions and feedback. • Host a workshop • Short series of educational content • A document, which reviews the feedback from the community and parachain teams.	

9.	Research	A document, which lists the options in enhancing Gosemble.	

b. How will you report on the delivery of your milestones?

We intend to fill a Treasury proposal report with the progress of the development and will share it with the Polkadot community, developers and w3f.

5. Payment conditions

Please specify any special conditions regarding the payment of this proposal.

a. **Requests**: **63,916 DOT**, amount to be recalculated at **7 days EMA** on day of on-chain submission. Base amount is **$334,732**.
b. **Accounts**: chrislime, **15reUHgnkE9QeH9zUoduCVYAbksTmQXvpMd6L95rBhTNFuGw**
c. **Communications**: Zhivko Todorov (zhivko@limechain.tech) and Chris Veselinov (chris@limechain.tech)

6. Comments, Qs&As

The discussion during the proposal approval - https://polkadot.polkassembly.io/referenda/90.

7. Why Polkadot Network?

At LimeChain, we believe in a multi-chain future, based on interoperability and decentralization. Principles that are built into the core of the Polkadot network. We have been firm believers in the Polkadot ecosystem for the last 3+ years. As such, we have been exploring different ways to add value to its developer community, starting with Subsembly - a **framework used to design and implement Substrate runtimes in Assembly Script**, the current development of **Gosemble**, a **light client implementation in Java** and building a **Parachain Validation Conformance Testing suite**.

8. We'd love to hear about how you got to know about the Polkadot on-chain treasury.

We have been working closely with David Hawig for our Subsembly grant, including the first phase of Gosemble. David introduced us to Otar Shakarishvili and Raul Romanutti. They have been so kind to walk us through the whole process and spending mechanism of the on-chain treasury.