Burrow - The Boring Blockchain

Blockchains are too exciting. Burrow wants to be boring. Then we want to be simple. Then we want to be fast. Even more boring than Git is stupid. The kind of boring that lets you sleep well at night. Burrow gives you just enough blockchain to build a strongly decentralised public permissioned network. Enough but no more.

Burrow philosophy

Burrow aims to be simple, complete, opinionated, and lightweight. We want to be 'the Redis of blockchain.' Our guiding principles are:

1. Provide single pure Go binary running as a single process with (almost) everything included
2. Support public permissioned networks as first-class citizens
3. Prefer introspection and tight coupling of components over configurability
4. Provide a single Byzantine Fault Tolerant (BFT) consensus algorithm
5. Provide a clean and simple developer experience that gets you closer to the metal

Boring deeper

Burrow is a fully-fledged blockchain and smart contract framework. That is, you ought to be able to service the same use cases as with Hyperledger Sawtooth, Fabric, and Besu (whereas Iroha takes a slightly different approach to smart contracts).

The core of Burrow is a custom, fully compliant, Ethereum Virtual Machine (EVM) implementation with an authenticated state (Merkle tree) based on Tendermint consensus (A PBFT consensus variant which we embed as a library). Burrow has a coarse-grain Unix-style model of permissioning baked directly into its EVM implementation. Permissions are for things like SEND, CREATE (a contract), and BOND (become a validator).

There are many other features wrought from our use of Burrow in the wild, both to run the Agreements Network and from years of working on use cases when it was still called eris-db.

It is an explicit focus of Burrow to support running permissioned networks that are, in some sense 'open to the public'. There are many shades of grey in terms of network participation with Burrow's permissions model; validators may be established on an invite-only basis, contract creation may be limited to autonomous contracts rather than human participants, or Burrow could be configured much like permissionless public Ethereum. We are particularly interested in modes of operation that bridge the gap between private chains and public permissionless ones. This bridge is why we value BFT consensus so highly, and why we include quorum-based governance primitives.

Burrow avoids relying on container orchestration and virtual machines as part of its basic setup. These technologies are powerful and can also hide a multitude of sins. Burrow provides an ergonomic developer experience on bare metal — on your laptop or server without needing docker or container orchestration in the first instance. We still have high-quality Kubernetes support for use in production, but you can also spin up a multiple node network directly on your laptop. We aim to give a streamlined developer UX and to help build developer intuition with our tools such as burrow examine, our JSON debug output or our curl-able HTTP info endpoint.

You can use our composable command line tools to spin up a simple chain:

```
burrow spec -v1 | burrow configure -s- | burrow start -c-
```

We provide statically linked cross-compiled binaries for Linux, macOS, and Windows.

We run over a Tendermint BFT consensus that prioritises correctness and finality and provides good throughput for networks with several validators in the low 100s. We only expose additional configuration where it is necessary and try to provide higher-level configuration and sane default to reduce the numbers of levers you need to pull on.

We provide a smart contract model exclusively in the Ethereum world view (state lives in buckets called 'accounts') and assume the structure provided by the EVM ABI.

Our entire state is modelled in Google's Protocol Buffers, which is used in our GRPC layer down to our underlying state storage.

Three faces of Burrow

Within the Hyperledger family Burrow can be seen to occupy three different niches:

1. The Hyperledger bridge to the Tendermint/Cosmos ecosystem

Burrow is tightly integrated with Tendermint via its ABCI interface for consensus. This integration means Burrow serves as an excellent way into the emerging Cosmos Network as that network and protocol emerge, Burrow will be amongst the first frameworks to join it. If you are interested in running smart contracts on Cosmos and their inter-blockchain proposals then the Burrow project has built most of what you need. We intend to continue to push the envelope on what can run on top of Tendermint/Cosmos.

2. An Ethereum side-chain and compatibility project with support for advanced smart contract languages via WASM and experimental economic systems

Since Burrow shares a smart contract language and ABI with Ethereum we have an impedance match with public Ethereum. We have plans to provide a two-way peg and support for staking on public Ethereum, which makes us a viable option for acting as an Ethereum side-chain. Since we are not pinned down by consensus, p2p, or low-level state compatibility with Ethereum mainnet, we have an excellent opportunity to innovate side-chain architectures and to help influence emerging standards in this direction.

3. A lightweight hackable EVM/Solidity execution library

Having entered Hyperledger as an independently developed Apache 2.0 licensed EVM implementation, we are most well-known for our EVM library. We successfully integrated into Fabric and Sawtooth, which helped us refine our internal interfaces and modularity. Our implementation is straightforward to read and understand and is not complicated by legacy support required for mainnet. In this respect we provide an excellent base for experimenting with the EVM - by extending it or making it run in other contexts (for example there has been recent discussion about running Burrow's EVM within processor enclaves (e.g. Intel SGX) as part of the newly established Hyperledger Trusted Compute Framework.

Features in detail

Governance

Burrow is built under a permissioned and open-network assumption. We expect existing participants and validators to decide who can access and validate the network. We operate a model that allows for bonded proof-of-stake or simple proof-of-authority. We support networks that are autonomous in the sense that no human key-holders can deploy contracts - only other contracts.

How to bring about governance on a blockchain remains an open question. Burrow aims to provide some necessary but not sufficient primitives for a multi-party governance of our on-chain rules.

Proposals

Proposals allow accounts with the PROPOSAL permission to propose an alteration to the network. A proposal contains a batch of normal transactions that will be executed with elevated permissions once ratified by a ballot (consisting of accounts also having PROPOSAL permission). This ratification allows a network with no direct human superusers to upgrade the contracts on that network. It is how we intend to upgrade code on the Agreements Network.

GovTx

GovTx is a transaction type that is often behind a proposal-wall with the ability to arbitrarily update an on-chain account (therefore contract in the Ethereum world view). This transaction can break the invariants of conservation of token and code immutability. As such it requires the ROOT permission. Generally expected use is within the context of a proposal; however, administrative account or contact use can be granted.

Bonding

Burrow nodes need to have the BOND permission to act as a validator (taking part in consensus via Tendermint algorithm). When a validator would like to start validating the network they can issue a BondTx which transfers some of their native tokens into validator power, which acts as a bond that can be slashed if they misbehave.

Conversely, a node can issue an UnbondTx to convert all or some of their bonded stake from validator power back into native tokens.

A variety of bonded proof-of-stake and proof-of-authority networks can be built using this model. However, they do depend on some extrinsic factors such as the value/meaning associated with Burrow native token to which Burrow itself is agnostic.

We will be providing additional crypto-economic primitives and access to Tendermint's proof-of-stake features via native function that will allow users to build more sophisticated networks, including those that hold stake on Ethereum mainnet.

GRPC interfaces

All of Burrow's functionality is available over Google's GRPC - a system that generates RPC endpoints based on Protocol Buffers definition files. These definitions can be used to generate clients in a range of languages. This functionality gives us the following benefits:

- We have a battle-tested and space-efficient binary RPC layer
- All of our core interfaces are defined in protobuf - used both for RPC and state storage - giving a clearly delineated domain model
- We are accessible from many languages
- We get nice streaming interfaces for events

Our GRPC endpoints are exposed on port 10997 (can you guess why?). Our core RPCs are documented in their protobuf definition files:

RPC Transact

The interface for sending transactions to Burrow, including the capability to sign transactions on your behalf. See tests for some example usage from Go.

RPC Query

A read-only interface for getting information about accounts, consensus, and the node instance itself.

RPC Events

A service that provides streams of execution events that are generated when transactions are included in the chain. This service includes EVM events as well as those describing blocks and transactions. They can be scanned to generate structured models of on-chain objects, which is exactly how Vent uses them.

Web3 interface

Recently landed in Burrow is complete support for Ethereum's standard web3 JSON RPC. For many years Burrow only exposed its own RPC layer; now, it is possible to use some of the excellent tooling from mainline Ethereum, including metamask, remix, and truffle!

Execution events

Each transaction that runs in Burrow is processed by an ExecutionContext that executes that transaction deterministically in a sandbox (throwing away the result if something goes wrong). We model the execution and a sequence of emitted events in which we include the EVM LOG events that are the basis of Solidity's event model. The idea of TxExecution is to provide a trace of everything your transaction did as it passed through the executor. In our state, we store a totally ordered set of StreamEvents that lend themselves to building event-driven applications over projections of these events (which is exactly how Vent works, see below).

Burrow JS

Burrow.js is our NPM module - it will have a familiar feel for anyone who has used web3.js, and it makes use of our GRPC interfaces. It can consume Solidity ABIs to dynamically generate JavaScript objects that wrap Solidity contracts on-chain. With the introduction of our on-chain ABI registry (see below), burrow.js is now able to pull these ABIs directly from chain and so can provide JavaScript objects to manipulate smart contracts via reflection.

Vent

Vent is our SQL mapping layer that runs by listening off Solidity events in our stream of execution events. It uses projections specifications that look like the snippet below and can build an object-relational mapping for Solidity contracts via Solidity vents.

Vent uses little bits of declarative JSON called projections to build its SQL tables. They look like this (for more details see docs linked in this section's header):

```
   json

[

  {

     "TableName": "EventTest",

     "Filter": "EventType = 'LogEvent'",

     "FieldMappings": [

     {

    "Field": "key",

    "ColumnName": "testkey",

    "Type": "bytes32",

    "Primary": true

     }

     ]

  }

]
```

Using Vent, you can build an application that runs on Postgres (a most excellent and very boring database) or sqlite using battle-tested architectures. Vent can give you tables representing entities and Burrow.js includes a feature to block until the database is synchronised with the chain.

Application developers can work with Burrow using Vent treating our Burrow transactions as the 'command' layer and a Vent-maintained query layer. Some apps may not need to directly write to the chain at all, in which case this model reduces to the familiar database-backed CRUD app model.

Natives, WASM, and EVM

Recently landed in Burrow is experimental support for the execution of Web Assembly based contracts. We have another experimental project solang, which is an LLVM-based compiler for Solidity that emits WASM that can deploy to Burrow.

Burrow also ships with a set of 'native' functions. You can think of these as our low-level syscalls, though they are not that low level. They allow you to modify permissions from smart contracts. You can call them just like any other EVM contract by calling a special address. burrow natives will emit the Solidity contract interfaces.

Natives are hard-coded in a DSL in the Burrow source code, where they are easily extensible.

Native contract domination

Burrow is moving to a model where all contracts, whether native, WASM, or EVM will be on the same footing. The same calling conventions can be used to call between any engine host, and functions can be called directly. Functionality that is currently wrapped up as separate transaction types will be migrated to a Burrow standard library of native functions.

We intend to add native functions for:

- Crypto-economic, validator, and bonded proof-of-stake primitives
- Generalised object registry (including off-chain storage)
- Governance proposals

- String manipulation
- Immutable contract upgradeability

Please plumb the depths with us in Hyperledger Burrow chat if you would like to know more or get involved.

ABI Registry

Another relatively new addition to Burrow is our ABI registry, which is optional but enabled by default.

Ethereum ABIs are JSON objects that decide the functions present on a contract, including metadata on their argument and return types. These objects are emitted by the Solidity compiler and allow you to call contracts. By hosting these with the EVM contracts themselves on chain can avoid situations where contract callers are missing the ABI or have the wrong version. Our tools use this to give a smooth experience.

In future work, we will provide additional contract reflection capabilities based on this functionality.

Name Registry

Our name registry allows a globally unique name to be registered for a specified lease period with an arbitrary data payload. It provides a kind of contract-accessible DNS.

Burrow command line tools

burrow spec is a tool for generating genesis templates; the initial state of a chain. They are templates because they describe the number and types of participants present without necessarily specifying their complete details, such as their public keys and initial balances (though these can be present). When provided to burrow configure the specific details comprising a complete genesis document can be filled in, including generating keys. A genesis spec can both function as a definition for a class of chains (at genesis) and a partially complete genesis. burrow spec can take multiple base genesis spec arguments which it will merge to support the latter.

burrow configure is a tool to make it easier to generate Burrow's configuration - including the configuration of a node usually stored in burrow.toml and the immutable genesis configuration usually stored in genesis.json (though it is possible to embed genesis in burrow.toml). It has numerous flags that translate into configuration. As mentioned above, if a genesis spec is passed with --genesis-spec, it will be used to generate a complete genesis document. For participants and validators that appear in the spec that do not have their public keys or addresses specified, then burrow configure will generate key pairs for them in the burrow keys service.

burrow deploy is a command line tool for contract deployment and testing with its YAML-embedded scripting language. You provide it with YAML scripts reminiscent of Ansible scripts called playbooks, and it takes care of reproducible deploying your contract suites. It can also formulate, sign, and broadcast all of Burrow's transaction types.

burrow examine is a forensic tool that helps you to drill down into the state of an offline burrow node for forensic audit or to debug. It's possible to look at both the Tendermint blockchain and our execution traces.

burrow tx is another command line tool that provides a low-noise interface suitable for integrating with shell scripts.

Burrow Proxy

Burrow proxy is part wallet, part cryptographic signing daemon, and part light client. It is a work-in-progress and replaces some existing functionality supported by burrow keys. It is a small local daemon that is responsible for managing your keys under your local custody, signing transactions, forwarding them to Burrow, and ensuring integrity via inclusion proofs and security via intermediate TLS.

Dump/restore

burrow dump and burrow restore support the extraction of the state of an online or offline Burrow node into a standalone JSON or Protobuf format that preserves essential state but discards the blockchain, individual transactions, and signature records. Our dump format includes only:

- Account records (balances, code, and contract metadata)
- Account storage
- EVM Log events
- Name registry entries

This format is intended to be long-term backward compatible. You can pass a Burrow dump to burrow restore on any supported version of Burrow and contain a chain that has identical smart contract state.

At first glance, throwing away the tamper-resistant log of a blockchain might seem self-defeating. However, dump/restore preserves the cryptographic audit trail by including a Merkle link from a block on the dumped chain to the genesis of the restored chain in the form of the AppHash field in the genesis document. So as long as the previous chain data exists (online or offline) then the state of the restored chain can be verified.

The essential purpose of this feature is to allow Burrow to fork in a principled way and provide:

- Burrow upgradeability: Burrow chain's state can survive a breaking change to Burrow
- Validator forks: if a new or different group of validators wish to run a version of a chain with a shared history for divergent purposes
- Hacking, compliance, and experimentation: since the state can be easily manipulated by a JSON stream editor like jq then we can delete or add accounts permanently or make other arbitrary changes to state. We could compress the chain state to a zero-knowledge proof or other such larks.

Upgradeability is the most important feature at this stage in Burrow's development. We make an effort not to break state; however, we will do so before our 1.0.0 release if it meaningfully improves the features, architecture, security, or performance.

Another avenue we are interested in exploring is providing an automated way for a validator to 'follow' a fork via on-chain governance, whereby the state of a parent chain is embedded in the genesis of fork and those validators that voted for it stop validating the old chain and start validating the new.

Example architecture

It might be useful to outline the architecture we use at Monax for the Monax Platform since this is a substantial cloud deployment that has guided much of the development of Burrow and its supporting tooling.

Kubernetes

We deploy Burrow and its supporting services into Kubernetes using Burrow's helm charts.

Node.js API

We operate an API service based on Node that uses Burrow.js to call smart contracts on-chain. Vent is used to maintain some tables in Postgres that provide a queryable view onto the smart contract state.

The API provides a higher-level interface to some of the functions on-chain in smart contracts and also stores a small amount of its state relating to its direct users.

Hoard

Hoard is a content-addressed encrypted blob storage layer that can back on to most populate cloud storage and IPFS as a backend. It can deterministically encrypt documents and provide asymmetrically encrypted access grants. Together with smart contract indices, we can provide file-like storage primitives and have the option to keep certain data private to counterparties. We can also keep large files off the chain keeping down block size and avoiding congestion.

We may look to make some of Hoard's functionality native to Burrow as part of a beefed-up name registry service.

React WebApp

Our main UI is a React single-page-app and communicates our identity service and with the API to get data from the chain and Vent-maintained database.

Blackstone

Blackstone is a project containing foundational smart contracts for the Agreements Network. It includes several 'standard library' type utility contracts, a Business Process Modelling Notation (BPMN) engine, and a battery of tests.

Our core objects are designed to be upgradeable so that immutable interface contracts point at the latest version of that contract when we use proposals to update code. There is a root contract affectionately known as Doug that acts as a registry for several lower layers of factory contracts that create our base objects.

User proxies

We have smart contracts that provide a layer of indirection between external accounts sending transactions and our internal notion of users and organisations. We identify these proxy contracts as _being_ a user and require that messages are forwarded through them. This allows users to permit a particular key pair to act on their behalf, similar to SSH authorized_keys, and means we can expose a traditional username/password login to users that have authorised our API, but later on, we can switch to multi-sig or the users exclusively controlling their proxy with their keys.

Plea for help

Join us for Hacktoberfest – Burrow Issues here!!

In writing this, I have attempted to provide up-to-date coverage of Burrow's most notable features and its orientation as a project. I am acutely aware there is a gap between what is written here and what is needed to get to grips with some of the features listed above. So, I also write to appeal for any assistance in closing that gap in terms of documentation or other usability improvements. I believe the best way to allocate resources to this effort is for people to be encouraged to scratch their own itch. I or another Burrow maintainer will be happy to help you to get going with any of Burrow's features if you swing by Hyperledger Burrow chat. If you are willing to spend a little time sequestering that knowledge by contributing to our documentation, I am willing to devote special attention to your learning and the needs of your use case, including by making changes to the Burrow roadmap or fixing bugs on the spot.

If you are looking to get involved in Hyperledger, then you will find in Burrow a small and uncrowded community with much to do. We have interesting features for newcomers to work on, and we are in search of contributors and maintainers. Come and help us build!

References

- Burrow main documentation site
- Burrow Hyperledger chat
- Burrow Hyperledger wiki
- Burrow Design docs
- Blackstone – Agreements Network
- Burrow godoc

About the Author

Silas Davis is the Burrow maintainer and CTO at Monax.io. An outdoor adventurer, math fan, partner, dad, and local political advocate, Silas looks forward to collaborating on Burrow.