Decentralized Futures

P2PLabs

Feel free to link to relevant documents, resources, or websites as part of this application. If you have any questions regarding the application process, feel free to reach out to df@web3.foundation

1. Project

1.1 Overview

Tagline:
Framework for creating client side peer-to-peer (p2p) state channels for arbitrary state machines with shared security inherited from a distributed ledger (blockchain). All the peer-to-peer networking is done between users' clients.

Brief description:

The main goal of the project is to create fully peer-to-peer applications (without any central servers) that are highly scalable, resilient and secure, but also provide the building blocks by solving common problems to enable others to do the same.

Peer-to-peer (p2p) state channels have many benefits - they're highly scalable and decentralized due to their network topology and tendency to partition sate within their group of interaction, they offer real-time communication that's 'free' (no transaction fee), but do not act like a trusted party like blockchains do.

What we aim to achieve is combine the best of both worlds, the decentralized and scalable nature of full p2p systems, while also inheriting full security from a distributed ledger.

Notice that society works in the exact same way. For most of our lives, we (peers) interact with other peers in small groups (partitions). We (peers) form agreements and commit to faithfully executing on them and in the case where those agreements fail, we go to a court system to dispute one another inorder to enforce those agreements. The courts have limited capacity and their intervention is slow and expensive, but they are trusted by all parties (that freely chose to be in that jurisdiction) to be impartial, objective and faithfully come to a verdict on the agreement/law (state machine).

Real life game theory applies here too. The courts (blockchains) have limited capacity, but most peers faithfully execute all agreements, knowing there will be consequences and enforcement if they don't. The ones that fail, the courts (blockchains) have enough capacity to deal with them.

How does this relate to Polkadot:

While most of the system is peer-to-peer, the shared security aspect is paramount and that's where DLTs come into play. The core system (infrastructure) is designed to be a public good, usable by anyone free of charge. As such it can be deployed on any blockchain that offers execution of arbitrary logic.The business model comes from projects built on top. They too can be deployed on many DLTs. We'll be starting with Poker as showcased in Hong Kong during the PBA, but we'll do other state machines soon after. We've picked Polkadot as the HQ (headquarters) where we accrue all the value, since we believe we share the same core values and mission (resilience, heterogeneous sharding, shared security and cross-chain interoperability) as the Polkadot ecosystem. Also, it's important to note that Polkadot has by far the most advanced governance and upgrade structure currently on the market and that too played a crucial role in picking Polkadot as the HQ. All the governance logic and protocol upgrades will be managed on Polkadot, with decisions beinged bridged to other chains to perform upgrades based on the outcome on Polkadot. This structure and business model is probably first of its kind in the DLT space and it makes sense since the state machines are designed to be partitioned, unlike many popular state machines that are currently deployed like liquidity pools that only work aggregated. This too resembles real life structures, where a company has an HQ in some jurisdiction (Polkadot) and still operates globally in many jurisdictions and all of the value is repatriated back to the HQ.

The value proposition for Polkadot is improving the user experience by providing a 'free' real-time execution environment which all parachains currently lack and expanding the market by capturing value from other ecosystems and repatriating it back to Polkadot - positioning Polkadot as the HQ and onboarding new users and developers into the ecosystem.

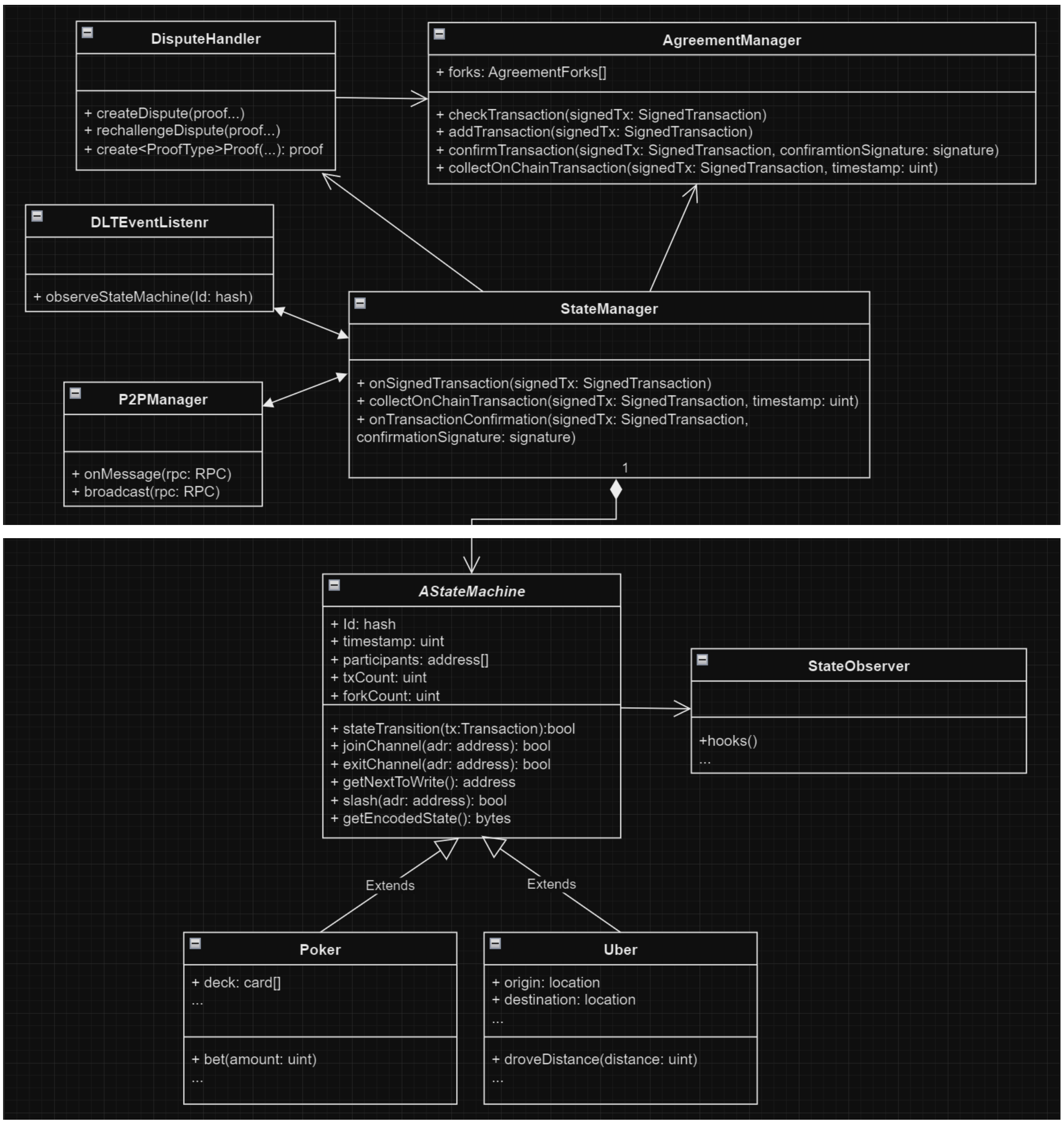
1.2 Details

This section will be used to expand on the technical solution and the architecture. It's important to note, the project is NOT in the 'idea' phase and everything that's discussed here we already have implemented. The MVP of the core tech is complete (though it may need more testing and it didn't go through an audit). I hope to cover all of the important parts, but if I miss something and/or if anyone has any additional questions and/or wants to discuss things further - feel free to reach out!

The system is composed of a client side framework/SDK and on-chain smart contracts.

The main responsibilities of the client side framework are:
1) Networking - establishing direct peer-to-peer connections and discovery over various transports (currently Holepunch, with webRTC planned later for browsers),
2) State machine replication - allowing connected peers to commit to state machines and allowing them to progress and replicate those state machines - similarly to how blockchains do it (currently only total order state machines are supported with leader election (who can write next) defined within the state machine),
3) Agreement tracking - tracking agreements on the order of transactions and state, managing forks, checking and collecting signatures, finalizing state and monitoring for byzantine behavior (double signing, incorrect state transition, failed to produce a block (tx) before timeout, spamming),
4) Virtual (logical) clock - Having a local logical clock that's synced to the DLT clock (timestamp) - allows peers to have a local perception of time that mimics the one of the DLT, but tolerates skewness to a degree. This is of course subjective and peers will never reach consensus on the clock nor with other peers nor with the DLT itself. This is very important and will be discussed further below,
5) Dispute handling - constructing fraud proofs when a peer deviates from the protocol (state machine) and submitting those proofs to a DLT to check, enforce a correct state and punish any byzantine actors,
6) Observing and notifying on state change - hooks that allow external systems to subscribe and get notified when state changes happen - analogous to on-chain events.

Here’s an abbreviated UML class diagram of the Client Side SDK/Framework. It has all the necessary components to understand how everything works and is interconnected, without being too verbose and going into implementation specifics.


DisputeHandler
+ createDispute(proof...)
+ rechallengeDispute(proof...)
+ create<ProofType>Proof(...): proof
AgreementManager
+ forks: AgreementForks[]
+ checkTransaction(signedTx: SignedTransaction)
+ addTransaction(signedTx: SignedTransaction)
+ confirmTransaction(signedTx: SignedTransaction, confiramtionSignature: signature)
+ collectOnChainTransaction(signedTx: SignedTransaction, timestamp: uint)
DLTEventListenr
+ observeStateMachine(Id: hash)
StateManager
+ onSignedTransaction(signedTx: SignedTransaction)
+ collectOnChainTransaction(signedTx: SignedTransaction, timestamp: uint)
+ onTransactionConfirmation(signedTx: SignedTransaction, confirmationSignature: signature)
P2PManager
+ onMessage(rpc: RPC)
+ broadcast(rpc: RPC)
1
AStateMachine
+ Id: hash
+ timestamp: uint
+ participants: address[]
+ txCount: uint
+ forkCount: uint
+ stateTransition(tx:Transaction):bool
+ joinChannel(adr: address): bool
+ exitChannel(adr: address): bool
+ getNextToWrite(): address
+ slash(adr: address): bool
+ getEncodedState(): bytes
StateObserver
+hooks()
...
Extends
Extends
Poker
+ deck: card[]
...
+ bet(amount: uint)
...
Uber
+ origin: location
+ destination: location
...
+ droveDistance(distance: uint)
...

1) <u>**P2PManager**</u>
Manages all peer-to-peer connections - how they’re established, which transport is used (Holepunch, webRTC,...), receiving and broadcasting messages (RPCs) to other peers and disconnecting from byzantine peers that spam or send junk messages.

The message is interpreted on the RPC level and passed through appropriate pipelines based on the type of RPC. In the case of transactions, they're forwarded to the StateManager for further processing.

2) **StateManger**
The central component of the system. It's responsible for maintaining the correct state. It receives both p2p messages (transactions) and emitted events that happened on-chain.

Received transactions pass through a verification pipeline (check signatures, is the signer part of the state channel, is it a duplicate, check timestamp, does it apply to the current state…). If all the checks pass, the transaction is applied to the state machine by invoking the stateTransition function. If everything is executed successfully, the signedTransaction is passed to the AgreementManager. If something fails, either a dispute is invoked on-chain (if a cryptographic proof can be created to prove fraud - double signing, incorrect state transition) or the P2PManager is signaled to disconnect from the peer since the message is 'junk' (doesn't have a valid signature, the signer is not part of the state channel,... - anything that can't be cryptographically proven as fraud from another participant in the channel).

Received on-chain events - Events that are of interest, at this stage, are TxCallDataPosted and SetState. TxCallDataPosted is emitted when a peer posts their transaction on-chain. This happens since other peers failed to sign the transaction confirmation within agreementTime (check Agreements and Timeout below). Transactions that are posted on-chain are assumed to be available to everyone (Data Availability) and do not need to have N-out-of-N signatures to be safe from invoking Timeout.
SetState is emitted when the DLT enforces a correct state of the state machine. This creates a fork (check Agreements and Forks) and sets the genesis state for the fork. This and everything else that's enforced through the DLT is accepted as correct unconditionally, since the DLT is the single uncontested source of truth and every honest peer continues building on the latest fork.

Timeout -
This is another responsibility of the StateManager.
There are 3 objective time parameters set for each state machine during creation:

- p2pTime - the max amount of time a peer can take to decide, execute and broadcast their Tx to other peers,
- agreementTime - accounts for reasonable network delay, execution time and skewness of clocks. All agreements on the Tx (state transition) should happen within this time,
- chainFallbackTime - if p2p agreements fail within p2pTime + agreementTime, this is the last resort allowing peers to post on chain.

When a peer fails to produce and broadcast a transaction that will progress the state machine forward within p2pTime + agreementTime in comparison to the previous transaction timestamp (each peer does this calculation in comparison to their local logical clock (this is subjective - see Clock)) - other peers won't sign the transactionConfirmation even if they receive the transaction later. The peer still has time to invoke their transaction, but is forced to post the transaction on-chain as calldata (lazy execution) within chainFallbackTime. If the peer fails to do so and does not have all N signatures after that time expires (in comparison to the DLT clock), any peer (usually the one(s) that didn't sign the confirmation) can invoke a dispute to remove the peer from the state channel (see Dispute Logic). The peer that was removed can challenge this dispute by providing all N signatures (confirmations) on the transaction and if successful the removal is reverted and the peer who initiated the dispute is removed and is fully slashed (penalty for byzantine behavior).

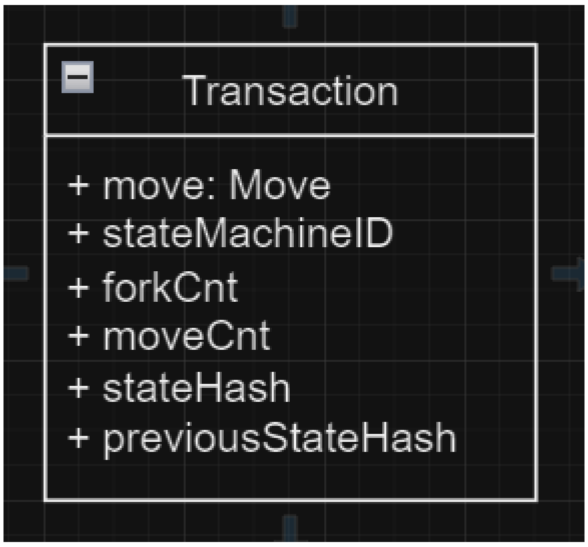
An honest peer can produce and broadcast a Tx that progresses the state machine forward within p2pTime + agreementTime, but another (byzantine) peer can refuse to sign the message confirmation and in such the issuing peer will never have all N signatures and will again be forced to post on-chain inorder to guarantee it won't be removed from the state channel. In that case the peer will refuse to sign confirmations of transactions from other peers that failed to sign its transaction, forcing them to also post calldata on-chain. Like this, both peers are at an economic disadvantage, since posting on-chain invokes transaction fees, so the optimal move is to cooperate to save on transaction fees, knowing that failure to cooperate results in economic costs. However, the DLT will guarantee correct progression of the state machine even if only 1 peer is honest and all N-1 are byzantine. (see Agreements)

Clock -
Every peer has a local subjective logical clock. The clock is synced with the DLT clock (timestamp) in order to provide a local perception of time that mimics the DLT time. Peers will never reach consensus on their clocks nor amongst themselves nor with the DLT itself. The protocol is designed to tolerate skewness to a degree. Peers use their clock to decide subjectively, how they want to interact with the system and other peers - are incoming transactions too far in the future, too far in the past or about in time and based on their subjective opinion they come to an agreement or abstain from signing p2p. They use this clock also to invoke Timeout. While the perception of time is subjective and is designed to tolerate skewness to a degree, once measured and recorded as a timestamp, that information is objective and comparable. While each peer will compare it to their local clock and try to form agreements, if such agreements fail p2p and the system falls back on-chain, the DLT clock will prevail as a single uncontested source of truth and its comparison of the tx timestamp to its clock will be final.

3) **AgreementManager (Agreements**)
The system does NOT implement a variation of a classical BFT (Byzantine Fault Tolerant) consensus algorithm due to lack of large economic security amongst peers in a partition and the strong design principle to allow everyone to connect freely with everyone. As such the system can't guarantee that within a partition there will not be

less than ⅓ byzantine actors as required by all classical BFT consensus algorithms. Since the DLT is used as a top level “supreme court”, we can design an optimistic consensus algorithm that tolerates N-1 byzantine actors - in other words we can design a system that’s Byzantine Fault Tolerant with at least 1 honest (non byzantine) actor by ultimately inheriting security from a DLT. (The state machine can implement a invariant check that’s always called when settling on the DLT, so even if all N actors in the state channel are colluding, once settling they still have to produce a state that satisfies the invariant (example - assets exiting the state channel can’t be more than the assets entering)).

Before explaining optimistic consensus, it’s important to understand the structure of a Transaction in this system:


Transaction

+ move: Move
+ stateMachineID
+ forkCnt
+ moveCnt
+ stateHash
+ previousStateHash

A transaction in this system references the exact state (previousStateHash) to which it is an input, and references the expected state post execution (stateHash). This resembles the logic behind UTXOs to a degree.

Under optimistic consensus we can’t force peers to collaborate, but it’s in their best interest to do so, since failure to collaborate at best invokes posting calldata on-chain and paying transaction fees and at worst results in a full slash when provable fraud can be detected. Collaboration includes signing (confirming) all valid transactions without delay (satisfying agreementTime) and producing and broadcasting a transaction when it’s the peers turn to progress the state machine forward before timeout (check Timeout).

Optimistic consensus is reached when all N signatures directly or indirectly (check Virtual Voting) reference the transaction (state). This finalizes the transaction (state). With at least 1 honest peer, an invalid state will never be finalized.

If for any reason a peer doesn’t collect all N signatures on their issued transaction after agreement time, the peer will post their transaction on-chain (see Timeout). The peer will also opt out of signing transaction confirmations from peers that didn’t sign its transaction, forcing them too to post on-chain and too being at an economic disadvantage. Peers can at any time request to exit the state channel directly on the DLT if they don't want to interact with other peers within the channel (subjective reputation and web of trust can help the peer initially decide with whom they want to connect, but is not required by the core protocol).

Virtual Voting - Same concept as with blocks in a blockchain. Since blocks are cryptographically linked with past blocks through the parent hash, a block at block height K indirectly supports and thus votes for all previous K-1 blocks that precede it. In our case, a transaction issued is connected directly to the previous transaction through the previousStateHash and since this is recursive, it votes for all previous transactions that precede it. This is really practical, since a byzantine actor not wishing to sign confirmations of other peers will at some point have to produce a valid transaction or be removed through Timeout - producing a valid transaction indirectly confirms all previous transactions. This property is very useful especially with round robin leader election (block production / who writes next) and guarantees that at most the last N transactions are enough to finalize a state even if no one wants to directly cooperate peer-to-peer.

Forks -
Forks ideally never happen and everyone cooperates fully p2p, without the need for the DLT ever to enforce a state, but in the case when the DLT has to step in, it creates a fork and sets the genesis state. The genesis state of a fork doesn't require N signatures to be considered final since it's enforced directly on the distributed ledger. Honest (non byzantine) peers always build on the latest fork.

4) **DisputeHandler** -
Whenever a peer detects provable fraud or timeout, they initiate a dispute by going through the DisputeHandler, which creates a dispute on-chain asking the DLT to enforce a correct state. To initiate a dispute a peer provides the last known finalized transaction (state) + all the transactions till the latest one the peer signed (confirmed) (this is at most N transactions in total - see Virtual Voting). After initiating a dispute, the peer stops signing transactions for the given fork. Due to the chain not having all the data and ideally not having any, the disputes are not final and have a challenge period where they can be contested. If successfully challenged (by providing more data), the disputter is fully slashed, a correct state is enforced and the challenge window prolonged. All honest peers check the disputed data and without delay challenge it if possible, otherwise they continue building on the fork and let the challenge period expire. More details on Dispute Logic down below when we cover on-chain logic.

5) **DLTEventListener** -
Observes the DLT for relative events and passes them to the StateManager.
In V1 the blockchain interaction is done through a provider, which the peers trust. In V2 we'll explore options of making peers ultra light clients so they can access the DLT directly and increase resilience (the bootstrapping problem makes this slight

optimization not worth considering in V1, since some node has to always be trusted to provide the data).

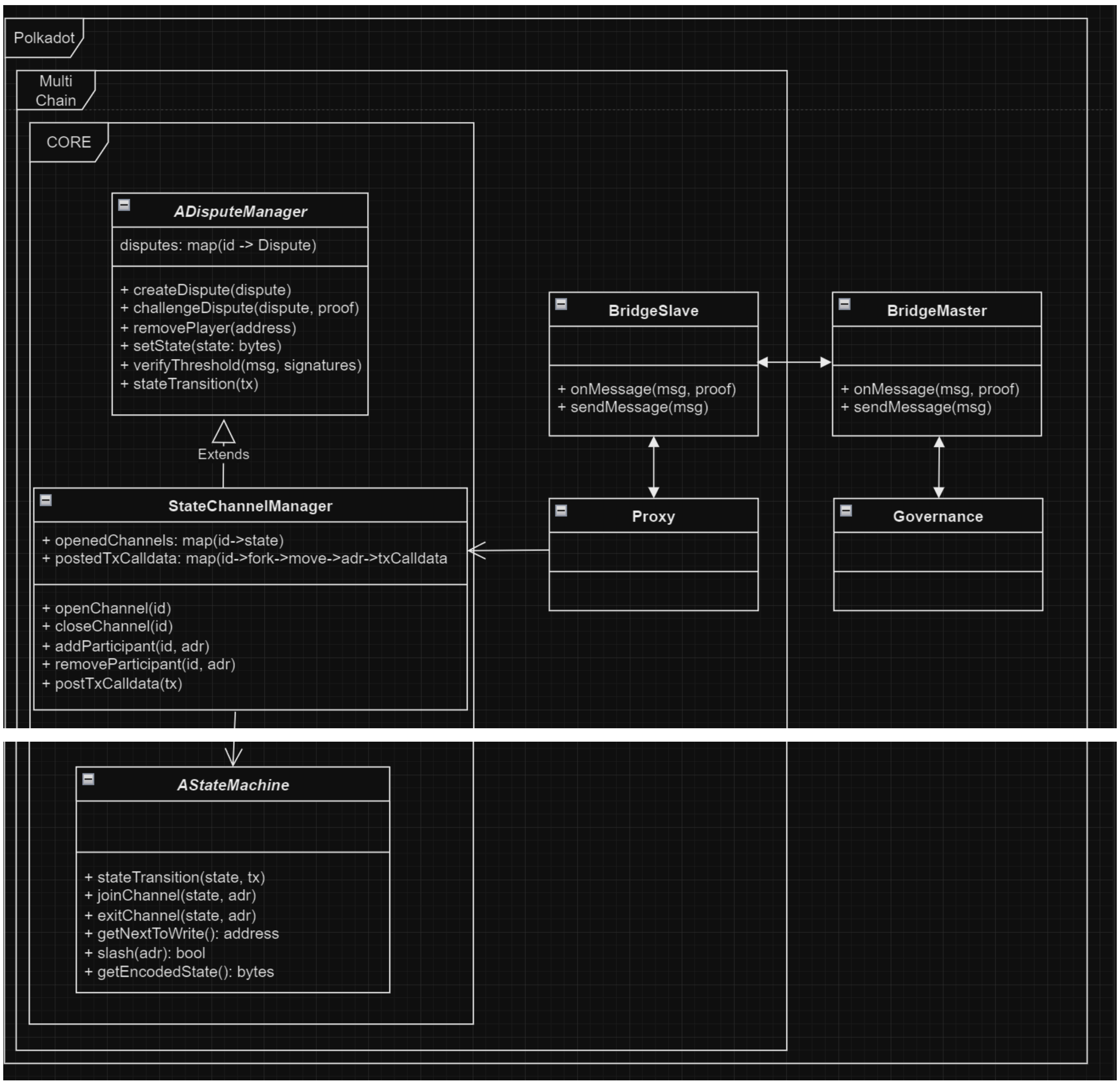
6) **AStateMachine** -
Abstract state machine that defines the base class/contract of compatible state machines. While any state and state transition can be implemented and thus supporting arbitrary state machines, the system does require a minimal interface with some minimal properties.

7) **StateObserver** -
Observes the concrete state machine and allows for other systems to subscribe for updates when the state changes. Analogous to on-chain events.

The main responsibilities of on-chain smart contracts are:
1) Agreement verification - Verifies that all participants agree on a given state,
2) Dispute resolution (shared security) - Enforcing a correct state of the state machine if peer-to-peer agreements fail,
3) Settlement - final settlement of any assets that entered into the state channel,
4) Data Availability - Guaranteeing data availability for both state and transaction calldata if peer-to-peer agreements fail,
5) Liveness guarantees - In the case that the state machine can't progress peer-to-peer, the DLT can progress it and remove any participants who failed to progress it within timeout,
6) Adding/Removing participants - adding and/or removing participants from the state channel,
7) Objective clock - provides a single global clock (timestamp) that's uncontested and accepted as correct/true unconditionally. Peers sync their local logical clock to the DLT clock to mimic the DLT perception of time,
8) Governance - HQ logic, managing funds, adding state machines, upgrading existing ones, arbitrary governance execution with all of the voting and decision making happening on Polkadot,
9) Bridging - bridging governance outcome on Polkadot (HQ) to other deployment on other chains, repatriating funds from other chains back to Polkadot, arbitrary message passing (execution).

NOTE: Governance and bridging isn't implemented in the code yet.

Here's an abbreviated UML contract diagram of the Smart Contracts. It has all the necessary components to understand how everything works and is interconnected, without being too verbose and going into implementation specifics.


Polkadot
Multi Chain
CORE
ADisputeManager
disputes: map(id -> Dispute)
+ createDispute(dispute)
+ challengeDispute(dispute, proof)
+ removePlayer(address)
+ setState(state: bytes)
+ verifyThreshold(msg, signatures)
+ stateTransition(tx)
Extends
StateChannelManager
+ openedChannels: map(id->state)
+ postedTxCalldata: map(id->fork->move->adr->txCalldata
+ openChannel(id)
+ closeChannel(id)
+ addParticipant(id, adr)
+ removeParticipant(id, adr)
+ postTxCalldata(tx)
AStateMachine
+ stateTransition(state, tx)
+ joinChannel(state, adr)
+ exitChannel(state, adr)
+ getNextToWrite(): address
+ slash(adr): bool
+ getEncodedState(): bytes
BridgeSlave
+ onMessage(msg, proof)
+ sendMessage(msg)
BridgeMaster
+ onMessage(msg, proof)
+ sendMessage(msg)
Proxy
Governance

Fraud proofs - fraud proofs in our system vastly differ from the ones of Optimistic Rollups such as Arbitrum/Optimism. The fraud proofs in their system are designed to work for servers that are handling transactions on behalf of other users and as such they have to post all the calldata on-chain for everyone to see and verify correctness or detect fraud. This is more expensive and less scalable, since more of the on-chain limited resources are being used. There are also no liveness guarantees, if the sequencer stops producing blocks.

In our system, all the transactions being executed within the channel belong solely to the participants of the channel and as such they don't need to guarantee data availability for external parties. Data Availability comes into play only when peers fail to cooperate and as such its main goal is to guarantee data availability to other peers and provide a way to progress the state without cooperation or remove peers through Timeout.

1) **ADisputeManager** (Dispute Logic):
   ADisputeManager is an abstract contract that implements all the dispute logic and provides an interface that the StateChannelManager contract must implement.

   When a peer detects fraud locally, they initiate a dispute on-chain. In order to do so, they have to provide a proof of the last finalized state and all the transactions up to the last one the peer confirmed (this all can be done with at most the last N transactions and ideally with just 1 or 2). The peer also has to provide a fraud proof.

   Fraud proof types:
   - FoldRechallengeProof - when a peer invokes Timeout on another peer, the other peer can counter the timeout by providing all N signatures for the timedout transaction.
   - DoubleSignProof - a peer provides 2 conflicting transactions signed by the same peer.
   - IncorrectDataProof - this requires re-execution on-chain and checking if the submitted data is correct.
   - NewerStateProof - Since the system is designed to try and not post data on chain, it assumes the peers will be honest and post the latest correct state that they signed when creating the dispute. If a peer posts an older state, providing a newer state with the peers signature validates the proof.
   - FoldPriorTxProof - when players collude and eclipse other players, multiple transactions can be available for timeout - FoldPrior allows the player which was timedout to request a transaction that preceded his to be timedout instead. This has a risk of slashing if the peer lied and signed the confirmation transaction.
   - TxTooFarInFutureProof - if a peer receives a timestamp too far into the future (subjective) they can challenge it on chain and if tx.timestamp>block.timestamp the issuer is slashed.

2) **StateChannelManager:**
   There can exist many state channels in parallel (open channels) and as such this contract manages all state channels - opening/closing the channel, adding/removing participants from the channel, keeping track of state (bytes) without interpreting it, allowing multiple state machines to be linked and invoking the correct one when needed, allowing peers to post their Tx as calldata and guaranteeing Data Availability to all.

3) **AStateMachine:**
   Abstract interface of a library (stateless contract) that only bears the state machine logic. All the library methods take raw state (bytes) as input and interpret it as

concrete state for the underlying state machine. Mostly used to perform state transitions during disputes by invoking stateTransition(state,tx).

Everything thus far we have implemented - the core.

4) **Governance, Proxy, Bridge Master/Slave**:
These contracts are currently not implemented, but are well known.

Governance - voting on protocol upgrades and treasury expenditure based on voting power in proportion to the amount of governance tokens. The governance outcome is passed to the Bridge Master.

Bridge Master - only deployed on Polkadot and is used in the message exchange to other chains where the governance outcome has to be relayed. Not much different than token bridging in the sense that the message relayed is an arbitrary message with the same security assumptions.

Bridge Slave - deployed on all blockchains, verifies messages received (relayed) from Bridge Master and executes on them - upgrading the proxy if needed.

Proxy - classical proxy contract for the proxy-upgrade pattern.

1.3 Ecosystem Fit

Help us locate your project in the Polkadot/Substrate/Kusama landscape and what problems it tries to solve by answering each of these questions:

- Where and how does your project fit into the ecosystem?
    - On the core tech it fits as a public good / infrastructure by allowing everyone to build secure real-time p2p dApps. On the concrete state machines that we do/build, those are clearly dApps and provide a direct benefit to the end user. On the governance and upgrade side, Polkadot is the best fit by a long shot considering how it handles upgrades in contrast to others.
- Who is your target audience (parachain/dapp/wallet/UI developers, designers, your own user base, some dapp's users, yourself)?
    - The end user that uses the dApps is the target audience. That’s where the business model is.
- What need(s) does your project meet?
    - Web2 user experience through real time communication that’s ‘free’, with web3 security guarantees.
- Are there any other projects similar to yours in the Substrate / Polkadot / Kusama ecosystem?

- Not that we're aware of, not only in Polkadot / Kusama, but web3 in general since the state channels that are adopted are mostly payment channels and those are servers and in general all the scaling in web3 has been done on the 'server' side, while we focus fully on the client side. Since we're doing poker as the first state machine, there are few competitors that are building poker, but fully on-chain and the user experience isn't anywhere near acceptable.

1.4 Future Plans

Everything we build, we build with resiliency, decentralization and scalability in mind. We never want to introduce a centralized server in our design or run any infrastructure. This offers a clear benefit to our users, but also to us, since we don't have to maintain servers and our operational costs converge to 0 overtime. Our only long term costs should be development of new state machines and improving existing ones. We plan on picking and developing state machines that provide real utility, without any ponzi (unsustainable) economics. We want them to be useful and a better alternative to existing solutions today. We believe the systems we create can offer the good user experience of web2 (centralized) systems, with all of the web3 (decentralized) security and permissionless guarantees.

While the core infrastructure will always remain free as a public good and accessible to everyone, the concrete state machines will have a business model that's sustainable.

Our first state machine will be poker, a popular game without any practical web3 implementation. The state machine will charge a fraction of the fee of what other centralized web2 providers charge and only capturing a small percentage of the poker market should be more than enough to fund all our future costs. We don't only provide a better alternative to the end user, but we also solve a problem for most casinos. Most of the poker market is captured by 2 big centralized providers and smaller providers can't compete since the big guys have all the users (a popular phrase in finance "liquidity attracts liquidity"). With this solution, existing casinos can pool their player base with other casinos and web3 native users without any intermediaries to handle funds and guarantee fairness, while at the same time having the user experience of web2. A few local and a few representatives from bigger international casinos already showed interest.

2. Team

2.1 Team members

Partially grant supported:

- Co-founder - Luka Stanisic
  - https://github.com/lukastanisic99
  - https://www.linkedin.com/in/luka-stanisic-712907251/
- Engineer - Dragan Basta
  - https://github.com/thepeaknick
  - https://www.linkedin.com/in/dragan-basta/
- Designer - Aleksa Krstic
  - https://www.linkedin.com/in/aleksa-krstic-124880112/
- Business development - Aleksa Gradinski
  - https://www.linkedin.com/in/aleksa-gradinski/
- Business development - Gerard Murphy
  - https://www.linkedin.com/in/gerard-murphy-8939b520/

Not grant supported:

- Cryptography - Jeffrey Burdges
  - https://github.com/burdges
  - https://www.linkedin.com/in/jeffreyburdges/
- Economics & legal - Kasper Mai Jørgensen
  - https://www.linkedin.com/in/kmjoergensen/
- Engineering - Andrew Burger
  - https://github.com/coax1d
  - https://www.linkedin.com/in/andrew-b-8a628870

2.2 Contact

- **Contact Name:** Luka Stanisic
- **Contact Email:** lukastanisic2@gmail.com

2.3 Legal Structure

Plan to form a Swiss non-profit foundation with an approval from FINMA on the token and business model.

2.4 Team's experience

Luka - Background in engineering, protocol level of different DLTs (DAGs and blockchains), distributed systems at Microsoft Azure, Polkadot Blockchain Academy alumni with distinction.

Dragan - Software engineer with 10+ years of experience.

Aleksa Krstic - Senior UX/UI designer at another major DLT.

Aleksa Gradinski - In crypto since 2017, CEO of Agroblock ($1M annual profit), helped in negotiations and expansion of leading web3 companies in Serbia.

Gerard - Engineering degree holder with over a decade in high-end business development, overseeing multiple offices in the London property market. 15 years of professional poker playing and game theory expertise, bringing a strategic and analytical approach to business growth and a specific understanding of peer-to-peer commercial opportunities.

Jeff - Applied cryptographer and mathematician at the Web3 Foundation. One of the lead protocol designers behind Polkadot and Kusama.

Kasper - ex CFO Web3 Foundation and co-founder Polimec.

Andrew - Researcher/Engineer at the Web3 Foundation.

2.4.1 Team Code Repos (if available and applicable)

Closed source for now (until launch) to have a competitive advantage and be first, but all of the important things are described here already. We’re more than happy to give full access to the code, even at this stage, to strategic partners where common interest aligns.

3 Milestones/Cost Breakdown

3.1 Overview

While projects of similar complexity raise millions, we don't believe we need that much or at least not at this stage. Most of the core team will only take tokens, no salary. That resembles our conviction and commitment on delivering and willingness to take on risk. However, we do need to raise some funds to cover external costs and additional hires. For this we're looking for strategic partners that align with the mission. We understand that all investors have their own requirements and we're carefully picking and engaging with the ones that we believe align with us. The public good (infrastructure) is not designed to generate revenue, hence VC funding is not ideal for the project as the incentives are not fully aligned. With grant support from DF we can get the project launched, finish the infrastructure and later develop concrete businesses on top that will sustain the project.

- **Total Estimated Duration:** 9 months for everything as described in the sheet below..

- **Full-Time Equivalent (FTE):** 7 after additional hires.
- **Total Costs:** $483,450 as described in the sheet below.

3.2 Milestones

The cost breakdown can be found in full in the sheet below and the deliverables are the concrete tasks listed.

https://docs.google.com/spreadsheets/d/1ojsPfe-x1vzh0IO-eoXe_ifUs_JJR_PGY2wi2UFMqF0/edit?usp=sharing

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (DAYS)	COST
1	**Setup, Legal and Compliance (Switzerland)**					**$50,000**
1.1	Legal structure setup	Luka, K	1/7/24	1/9/24	60	$20,000
1.1.1	Legal advice and compliance	Luka, K	1/7/24	1/9/24	60	$15,000
1.2	FINMA approval letter	Luka, K	1/7/24	1/9/24	60	$15,000
2	**System Design - Architecture**					**$60,000**
2.1	Client Side SDK/Framework: Networking, State machine replication, Agreement tracking, Dispute handling, Virtual clock, Observing and notifying	Luka, J	1/7/24	16/8/24	45	$20,000

2.2	Blockchain Components: Shared Security, Data Availability, Liveness guarantees, Cryptographic primitives	Luka, J	1/7/24	16/8/24	45	$40,000
3	**Development - Core Tech**					**$228,000**
3.1	Client Side SDK/Framework: as defined in 2.1	Luka	16/8/24	16/2/25	180	$100,000
3.2	Blockchain Components: as defined in 2.2	Luka	16/8/24	16/2/25	180	$80,000
3.2.1	Cryptography integration	Luka, J	16/8/24	16/2/25	180	$48,000
4	**Testing**					**$30,000**
4.1	Client SDK/Framework: Unit tests, Integration tests, System tests, Security tests, Distributed system tests	Luka	16/2/25	1/4/25	45	$15,000
4.2	Blockchain Components & Cryptography: Unit tests, Integration tests, System tests, Security tests	Luka, J	16/2/25	1/4/25	45	$15,000

5	**Design**					**$20,000**
4.1	Website design (Desktop, Mobile)	Aleksa	16/8/24	1/10/24	45	$3,800
4.2	Game design (Desktop, Mobile)	Aleksa	16/8/24	1/10/24	45	$11,900
4.3	Branding full design: Logo, Presentations, Promotional materials, Pitch decks, Colors, type, imagery, graphics, icons, Social media templates & materials	Aleksa	16/8/24	1/10/24	45	$4,300
6	**Frontend - UI**					**$91,500**
4.1	Website implementation	Aleksa	1/10/24	1/2/25	120	$12,600
4.2	Game implementation	Aleksa	1/10/24	1/2/25	120	$78,900
6	**Audit**					**$50,000**
4.1	Audit 50k+	Luka, J			0	$50,000

Sum		$469,500
Unforeseen costs	10%	$46,950
Total		$516,450

J - Advised by Jeff

K - Advised by Kasper