PEERD: Modular Infrastructure for Decentralized Applications

Ryan Brewer (rhunbre)

hello@peerd.io

DRAFT VERSION 0.1

February 24, 2018

This document provides a high-level overview of the PEERD architecture, reasoning, and vision, with additional technical data and specifications to be released in updated versions, or subsequent documents.

OBJECTIVE

PEERD is designed to function as an open source, modular collection of developer libraries which provides a base foundation for creating decentralized applications on top of. Such decentralized applications would utilize Bitcoin for anchoring data to the Bitcoin blockchain, as well as transferring value using the bitcoin currency. In this fashion, there is no need to create an alternative token for value transfer, nor secure an entirely separate blockchain for data validation.

PEERD is designed to offer a simple set of open and permissionless infrastructure components, and allow for relatively straightforward decentralized application development in a style familiar to developers with web development backgrounds.

OVERVIEW

PEERD has three significant design decisions: (1) the network layer exclusively uses WebRTC as its transport mechanism, (2) all libraries are expected to function in modern web browsers, however, some specific features can require Node.js, and (3) there is no platform-specific token, however, novel tokens would be possible to create on top of the system.

WebRTC has been selected as the transport mechanism because it provides a standardized, peer-to-peer communication mechanism which is supported by all major web browsers. WebRTC support and standardization are continuing to develop and increase, and can be expected to do so into the future.

Going along with the primary objective of enabling decentralized application infrastructure which is usable in, and targeted at, web browsers, all PEERD libraries are being developed in the JavaScript programming language. Usage of the recent WebAssembly standard offers potential in the future, particularly where performance improvements would be significant, but is considered outside of the scope of the current phase of development.

In the interest of facilitating ease of adoption and working with, not against, Bitcoin, there is no platform-specific token in PEERD, other than bitcoin itself. Creating an original, fungible, value routing token is viewed to be both needlessly reinventing the wheel, and antithetical to the PEERD ethos and vision since Bitcoin can be used for such purposes. By using bitcoin as its native token, or currency, the application gets access to the broad and quickly growing Bitcoin network effects, without having to waste resources attempting to reproduce what Bitcoin has spent nearly a decade creating.

As such, through the peerd-consensus library, PEERD enables bitcoin to be used as a value transfer mechanism, as well as the creation of non-fungible tokens. Non-fungible tokens can be considered a more legitimate use case than fungible, non-bitcoin tokens since they are fundamentally not value routing tokens and are therefore not attempting to compete with Bitcoin. Of note, these non-fungible tokens are not “utility tokens”, which to date have simply seen usage of the phrase as a marketing term to try to positively describe fungible tokens that are seeking to compete with Bitcoin. Such tokens

themselves offer no true utility, aside from serving as a fundraising and value transfer mechanism, which are, of course, aspects that bitcoin can readily be applied to today.

THE PEERD SOFTWARE STACK

The PEERD software stack is comprised of five components, split across three conceptual layers. Each of these five components is expected to be used as a JavaScript library, and can, and often do, rely on other stack components to enable higher-level features.

1. peerd-networking: This library functions as layer 1 of the PEERD stack, and provides the WebRTC networking components, peer management, message handling, Bitcoin integration, and other, low-level functions.

2. peerd-storage: This library, along with peerd-consensus, functions as layer 2 of the PEERD stack, and provides distributed hash table (DHT) and file management functions.

3. peerd-consensus: This library, along with peerd-storage, functions as layer 2 of the PEERD stack, and provides consensus related functions, the primary of which are: (1) an on-chain, Bitcoin based signaling and staking protocol, and (2) coordination of a virtual blockchain, or virtualchain, that anchors to the Bitcoin blockchain.

4. peerd-hashagg: This library, along with peerd-identity, functions as layer 3 of the PEERD stack, and provides general purpose hash aggregation as a decentralized service, built on top of lower-level PEERD functions, with the most notable being a peerd-consensus group which runs the library specific network and virtualchain.

5. peerd-identity: This library, along with peerd-hashagg, functions as layer 3 of the PEERD stack, and provides a decentralized service for self-identity related features, built on top of lower-level PEERD functions, with the most notable being a peerd-consensus group which runs the library specific network and virtualchain.

When considering the PEERD software stack, layers 1, 2, and 3 comprise the five core PEERD libraries. Each of the first three layers is a developer-focused layer, and they are implemented in the form of JavaScript libraries and intended for developer use when creating decentralized applications. This is in contrast with layer 4, the application layer, which is abstractly made up of any arbitrary user-facing applications developers build using PEERD.

Five examples of decentralized applications are presented, with each example serving to show a possible use case that could be built using PEERD for most of the significant infrastructure.

Layer 3 is the most likely to be expanded with additional libraries, and speculative libraries such as a decentralized exchange as a developer library, can readily be imagined. This layer can, at least partially, be thought of as a virtualchain, or sidechain, layer offering arbitrary functions as an incentivized network service. However, additional network service libraries would likely be built on top of peerd-hashagg and peerd-identity, potentially splitting the third layer into two internal sublayers.

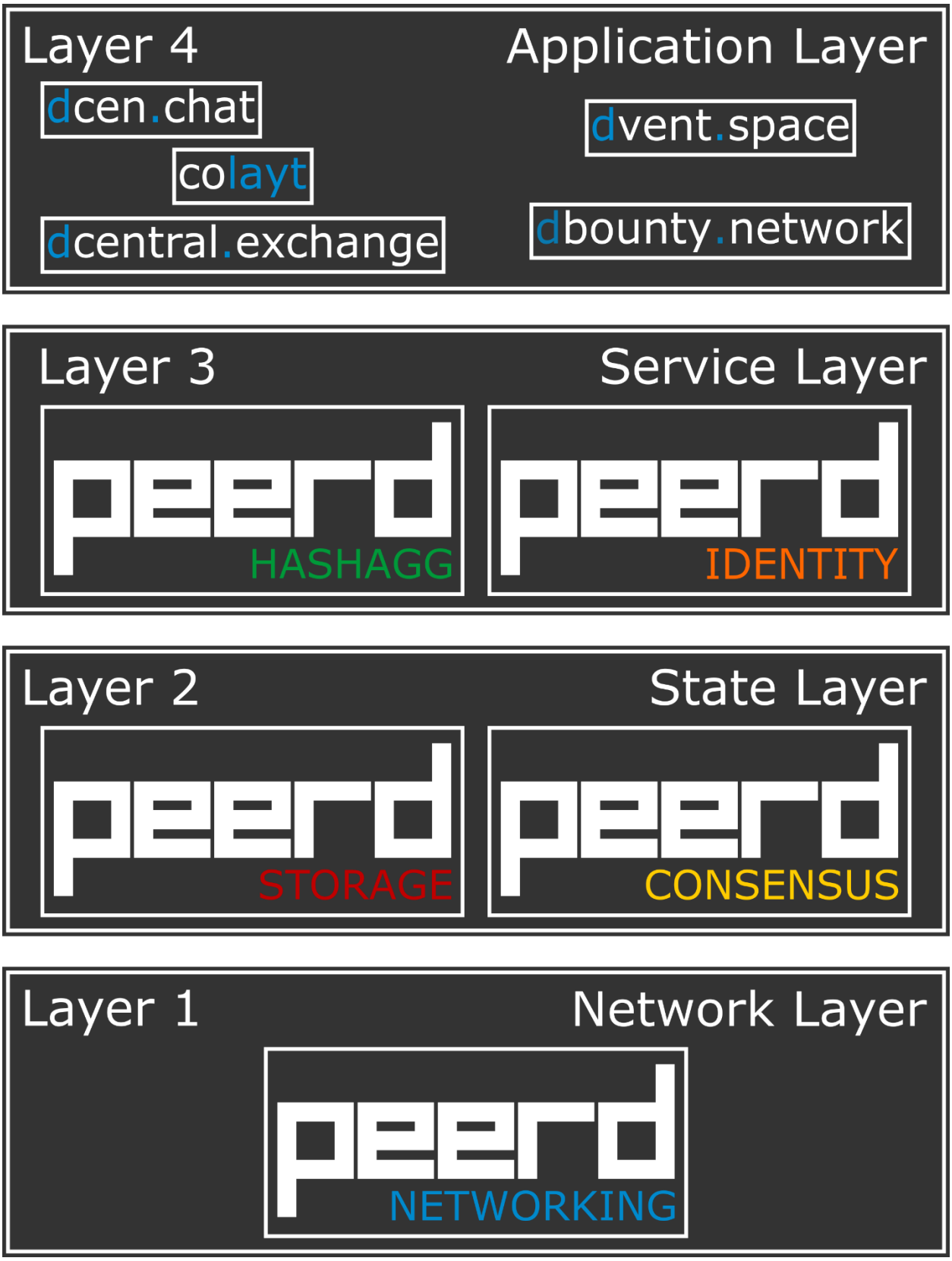

Figure 1: An illustration of the PEERD software stack, split into four conceptual layers.

BRIEF EXAMPLES OF POSSIBLE DECENTRALIZED APPLICATIONS

These five examples are meant to serve as starting points to show the types of desirable applications that could be built on PEERD. They are outside of the scope of the PEERD project, but are what PEERD hopes to ultimately enable.

dcentral.exchange: As mentioned, a decentralized cryptocurrency-to-cryptocurrency exchange would be an excellent candidate for a third service library in layer 3 of the PEERD stack, constructed in a similar fashion to the peerd-hashagg and peerd-consensus libraries, with a user-facing front end potentially built on top of it.

dbounty.network: An example of a potential user-facing software development bounty network, where users can pledge monetary bounties, in bitcoin, to incentivize open source software development.

dvent.space: An example of a potential user-facing group and event information network, that would function as a decentralized alternative to currently centralized services.

dcen.chat: An example of a truly decentralized peer-to-peer messaging application, which is relatively low-hanging fruit for demonstrating usability of the PEERD software stack.

colayt: An example of a more significant potential application that would function as a decentralized annotation and commenting layer for the web. This would be in contrast to centralized commenting services, and the entire platform would be permissionless, which no commenting platform seems to currently offer.

PEERD-NETWORKING LIBRARY

The peerd-networking library functions as the core of the PEERD system, with connection bootstrapping consisting of an open-membership group of public PEERD nodes, hereafter called gateway nodes, that run a PEERD node using Node.js and offer bootstrapping services. This is analogous to the DNS seed nodes utilized as bootstrapping points in Bitcoin, however, PEERD nodes use WebSockets rather than DNS for this purpose since WebSockets are more applicable to the browser-focused nature of the PEERD architecture.

In practice, at network launch, there will be one official PEERD gateway node. However, others are highly desired and gateway node operators are encouraged to get in contact for source-list inclusion. PEERD gateways also inform connected peers of any other known gateway nodes, and previously identified gateway nodes can be included, as a fallback mechanism, in each release of the peerd-networking library.

A previously unconnected PEERD node first connects to the gateway nodes specified in the gateway list, using WebSockets, and requests a list of all known peers from each gateway node. Upon receipt of the peer lists, the previously unconnected node begins creating WebRTC offers for each peer, including the gateway nodes, and broadcasts all of its created offers to each of the gateway nodes that it is communicating with, using WebSockets. Once a WebRTC channel has been opened with a gateway

node, all further communication between the previously unconnected node and gateway node takes place over WebRTC, rather than the initial WebSocket connection, and the gateway node begins offering network message data to the newly connected node.

From this point on, the newly connected node inspects network messages for messages intended for itself, and offers all messages it receives to all connected peers.

Upon receiving WebRTC acceptance messages in response to the offer messages that the previously unconnected node broadcast, it finishes creating, and then maintains, a WebRTC channel with the responding peer. Gateway nodes are deprioritized once enough other peers are connected, and the node stores peer information to use when attempting connections in the future.

PEERD-STORAGE LIBRARY

The peerd-storage library exposes APIs for creating and maintaining distributed hash tables (DHTs) and participating in torrent swarms using WebTorrent, an open source, browser compatible BitTorrent implantation that uses WebRTC.

Magnet links are created, stored, and forwarded by peers, along with additional metadata. This is particularly useful when the peer group wants to maintain a set of specified, public files, such as application resources. Files can be identified by their cryptographic hash, serving as the key, with the magnet link serving as the value.

A more advanced chunking mechanism that allows small chunks of files to be sold for micropayments, using hash trees, also called Merkle trees, for data validation, and either a peerd-consensus validator group or Lightning Network for payments is one example of additional functionality that could be built on top of the base peerd-storage library. However, this is considered outside of the scope of the current phase of development.

PEERD-CONSENSUS LIBRARY

The peerd-consensus library exposes APIs for creating and maintaining virtualchains that anchor to the Bitcoin blockchain. Ideally, this library will expose multiple protocol methods, beginning with a relatively simple multisignature scheme that starts with an on-chain, Bitcoin based signaling and staking protocol, enabling a staked validator group with a configurable number of validator slots and a blinded bidding process for bidding on the slots. This represents an open-membership, multi-party peg.

Alternatively, validator slots could be handed out centrally, even without stake, for models that wish to rely on a closed-membership, authoritative peg.

In the open-membership model, validator set size limitations arise from Bitcoin script limitations. Currently, 67 appears to be a practical maximum, with lower numbers being easier to implement, and less costly, while likely still providing a practical level of security to enable a variety of interesting use cases. In the future, this upper limit can be expected to improve by way of various signature aggregation

proposals, as well as increasing current Bitcoin script limitations. Such improvements would allow larger validator sets, cheaper transaction costs, or both.

This library allows for arbitrary consensus rules to be set for a given subnetwork inside of the PEERD network. For example, a decentralized application can have its own consensus rules for how data is processed and what data is considered valid.

Additionally, the validator set can require non-validator nodes to maintain a stake with them, or follow consensus rules, in order to access network services. Such consensus rules can include, for example, a requirement that non-validator nodes vote, or submit accurate data, at regular intervals, or otherwise be banned by consensus following nodes.

In practice, this would have a cost in order to prevent Sybil attacks, either via an on-going, refundable stake, or by charging for some required resource, such as a username. Charging for a resource also has the added benefit of ensuring validator revenue in scenarios where transaction fees would not be sufficient.

All validator group messages are expected to be broadcast to the entire network in a timely fashion. In this way, validators cannot effectively censor other validators (both during multisignature construction and after) because a censored validator simply has to get their message out to the overall network for the network to realize that they should have been heard.

Another potentially valuable feature is the notion of a generalized option of allowing “UTXO votes”, by leveraging unspent Bitcoin transaction outputs, or otherwise allowing some form of participation to be indirectly “purchased” with UTXOs, where an on-chain Bitcoin UTXO could be used to perform up to X number of separate votes, at which point the UTXO is permanently exhausted of voting capacity. Validator nodes would maintain a list of observed UTXOs used for such purposes, along with how many times the UTXO has been used for voting, and what the votes were. This would be implemented by using UTXOs to sign voting messages.

PEERD-HASHAGG LIBRARY

The peerd-hashagg library utilizes the peerd-consensus library to expose APIs for creating and maintaining a staked virtualchain and enabling hash aggregation transactions by way of a hash tree. Service users deposit funds into the validator multisignature address, to be used for payment of future hash inclusion transactions.

This is similar to commercial services that attempt to offer hash aggregation into the Bitcoin blockchain, but done in a decentralized manner without a trusted company or service-specific value token in the middle.

PEERD-IDENTITY LIBRARY

The peerd-identity library utilizes the peerd-consensus library to expose APIs for creating and maintaining a staked virtualchain and enabling self-identity features and transactions. This includes identities controlled by a single owner, as well as multi-ownership models.

Also included is a method for signing legacy domain names and other legacy assets, such as Twitter accounts, by way of hosting a signed message on the domain or account, and making a complimentary transaction to the peerd-identity network.

A PEERD node wishing to resolve a domain name that has been signed in this fashion would query the network for the latest ownership data, verify it over HTTP or HTTPS, and then continue to load whatever resources the effectively repurposed domain name specifies in the context of the decentralized application.

In this way, domain names can be repurposed for use as supplementary, human-readable identifiers for decentralized resources, and would allow owners of existing domain names to utilize the value such domain names represent, when constructing a newly created decentralized system.

Alternatively, internal namespaces can be used, where the registration and renewal of the name itself happens internally on the peerd-identity network.

CONCLUSION

PEERD proposes a browser-focused, modular system of scalable infrastructure for building decentralized applications, providing the necessary foundational components to allow decentralized applications to be created in a method familiar to web developers.