Peernet

A decentralized internet.

info@peernet.org

Version 0.8.2 from 23.05.2021

Abstract: Peernet is creating a new peer-to-peer internet and browser in which individuals are in full control of their data. They are no longer subject to the censorship and restrictions imposed by companies or governments. This network is fully controlled and owned by its users. Data is shared freely.

Introduction

Peernet is a public peer-to-peer network that is made accessible to the user via the Peernet Browser, which allows for a seamless integration and user experience. It gives autonomy back to the user: each user holds their private key which gives them full control over their data. This private key manages the user's personal blockchain which stores metadata of shared files and folders, as well as indicating what the user likes via stars (☆) and subscriptions (📄), and spam reports. Users may publish, share, monetize, and remove their data at their discretion.

The network hierarchy is organized in a Kademlia-like fashion. The distance between two peers is measured as XOR of their peer ID. The peer ID is the user's public key (secp256k1 is used). Distributed Hash Tables (DHTs) are used to discover peers, blockchain data, and to inform who stores files. There is no central authority; each peer makes its own decisions on the extent to rely on and trust other peers.

Clients are encouraged to let users choose whether to use predefined seed lists for root peers and whether to use any content blacklists. At any time, the user is in control of what data he publishes and what data he consumes.

The challenges and failures faced by previous decentralized network efforts can be categorized into:

a. Poor user experience by using regular web browsers as the user interface.
b. Ghost-towns and too many leechers due to missing incentives or lack of enforcement.
c. Spam
d. Legal challenges

The use of a dedicated browser for this new peer-to-peer (P2P) network is emphasized as it is directed to solve these challenges and pave the way for mass user adoption.

The Peernet Browser is available on the legacy Internet at https://peernet.org/.[1]

Benefits to Users

The problem with today's internet services is that they are highly centralized. Users lack control of their data, face censorship, and are often subject to monetization of their data without meaningful consent and without receiving royalties. The Big 5 – Amazon, Apple, Facebook, Google, and Microsoft

[1] Available with the first beta version, expected Summer 2021.

– control large parts of the internet… and large chunks of the world’s data. By extension of this immense control these companies exercise power without meaningful accountability.

On the contrary, Peernet solves these issues and has many benefits:

1. Free: The network itself does not enforce monetary costs for sharing data. Users may choose to ask for payment at their own discretion.
2. Content remains online: Peers are incentivized to keep popular data online, even if the original peer who shared the content is offline.
3. No intermediary
    a. No censorship and restrictions by a central authority.
    b. Users are in full control of what data they publish and consume.
    c. Any royalties and income go directly to the content creator.

Terminology

Client	A client is a software program that participates in the Peernet.
Decentralized Internet	In this phrase, “internet” refers to application-level services recognized by a typical user. Peernet connects over the internet as well as LANs.
Root Peer	A peer operated by a known trusted entity. They allow to speed up the network including discovery of peers and data. The chain of trust is established by hard coding such peers into the client. Users may choose to remove those peers from the trust list and add their own curated list.
User	Human.
Peer	An endpoint participating in the peer-to-peer network. Usually a Computer.
Bootstrapping	The process of discovering the initial set of peers to connect to the network.
Message	A packet sent from one peer to another.
Offset and Size	In bytes unless otherwise specified.
	All mentioned forms in this paper apply to both genders equally.

Trust

Trust must be established to prevent spam (and on the flipside, encourage high quality data) and encourage peers to remain honest (i.e., not violate the protocol by, for example, keeping data that was supposed to be deleted). Dishonest and spam peers shall be blacklisted in a peers’ blockchain. Since all blockchains are public via a DHT, other peers may choose to use that information to isolate bad peers.

The tricky part is to know in the first place whether a peer and its data can be trusted. Peernet uses the PKI-based method described in the paper “Scalable Methods for Spam Protection in Decentralized Peer-to-Peer Networks” [Chapter 4.1, Spam Protection Methods][2]. It starts by designating root peers as pretrusted peers. Any peers for which the user subscribes or stars its data shall also automatically establish trust. Based on the proposed model, a dynamic trust model between peers is established. By using proof of work (such as a captcha or hashcash), it will disproportionally increase the cost for spammers vs. honest peers.

[2] https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8873623&fileOId=8873651

Virtual Folders

Anyone can create a new “virtual folder”. Its metadata is stored on the users’ blockchain who creates it; therefore, the creator will be the sole owner of the folder by holding its private key. It is a means for sharing data.

Virtual folders are addressed by the hash of the public key. The format of native URLs is:

> peer://PUBLICKEY/folder

In the future, users will be able to link other folders into their virtual folder. This enables the creation of a “collection” of other people’s published data (for example, linking other people’s food websites).

In the future, encryption of blockchains (or parts thereof) might be used to allow users to limit permissions (for example only allow people who have a password or key to access the content).

Discovery of Content

Discovery of content can be accomplished in a decentralized way via distributed hash tables; offline by users sharing their public keys, for example, via email or chat; semi-centralized via infrastructures such as DNS; and via centralized discovery services.

In the case of hosting websites on Peernet, it makes sense to use DNS for simplicity. Website owners only need to add their public key as TXT record to the domain and website visitors only need to know the domain name. This is a powerful alternative to the existing way a website is served.

Decentralized methods such as distributed hash tables allow for implementation of powerful client-side search engines.

Incentives

Peers who agree that the user’s data constitutes value will keep parts of the metadata and data stored, making it available to other peers. Peers might give other peers preferential treatment such as increased bandwidth and priority in communication if they are known to store and share data. This incentivizes sharing and disincentivizes leeching.

Clients are encouraged to reserve a portion of the user’s disk as encrypted blob, for example 20%, to store and provide data of other peers.

The process of agreeing what data constitutes value can be accomplished by the proposed stars (☆) and subscriptions (📄) functionality.

There is an incentive for peers to remain honest – as dishonest peers that violate the protocol will be isolated by other peers. Dishonesty can occur if a peer decides to keep data that was requested to be deleted by the key holder. However, such dishonesty can be detected by requesting such deleted data and if the peer responds, isolating it and at least disconnecting it from honest peers. Dishonest peers may also face legal liability for storing content that was marked for deletion.

Future incentives will include a virtual currency that allows users to pay or donate to specific content creators and allows users to pay for guaranteed availability of their published data.

Legal Considerations

Each user is the “king of his own castle” by holding the private key that allows the user to extend his or her own blockchain – including sending delete commands. Peernet is not a lawless land; the liability falls completely on the end-user (instead of the platform or service provider). Courts may still compel individual users to delete data and it is up to the private key holder to comply with such demands.

Network Health

Ensuring network health is paramount for a successful long-lasting network. Since there is no central authority to engage and punish dishonest actors, network health must be guaranteed by peers. Clients must implement algorithms to detect dishonest peers, fake peers, spam and hacking attacks.

Dishonest peers are peers who do not honor the protocol or actions. For example, a peer that does not delete data that was marked as to-delete. Fake peers are peers that do not exist. A bad actor might attempt to create fake peers to disrupt the network.

There are many known and potential unknown attacks in the future. Well-known attacks against P2P networks can be found by analyzing past disruption attempts – especially by law enforcement in the case of malicious P2P networks by malware such as Sality and ZeuS Gameover. Some of the attacks are:

- Isolation attacks: Isolating peers from each other to essentially disconnect the entire network.
- Replay attacks: Replaying network packets for a certain outcome. For example, replaying an outdated vulnerable update.
- Sybil attacks: Flood the network with nodes that you control for further attacks.
- Denial of service.
- Illegal content distribution.
- Exploiting vulnerabilities of the client or worse, the protocol.
- Spam transactions.

Clients are encouraged to use blacklisting techniques to punish bad peers. Blacklisting may be done based on the IP address (or CIDR) and the peers’ public key. Since IP addresses are not free, this can significantly increase the cost for an attacker and prevent attacks including Sybil.

Protocol

The functionality of the protocol is separated into:

1. Peer discovery
    a. Bootstrapping
    b. DHT for peer discovery using blake3 hashes for keys.
2. Metadata sharing
    a. Using the same DHT for blockchain discovery.
    b. DHT for file discovery based on file names and file tags.
3. File transfer

Peernet only uses UDP. The default port for exchanging packets is ‘p’ (112). UDT (UDP-based Data Transfer Protocol) is used for any other data transfer. If the default port is unavailable, a random port may be used. (The only reason for the default port is for initial discovery in isolated networks.)

Peers are identified via their ECDSA (secp256k1) public key (= peer ID). At the same time, it is the root of the peers’ blockchain. This allows anyone to verify the chain of trust of virtual folders created by that peer and verify signatures of messages. Blockchains are versioned, and if a newer version appears, the old one must be deleted. This allows to restructure a blockchain if it becomes too big and contains dead entries like deleted file descriptors.

The client software is identified via a User Agent which must include the software name and version number. Peers may have multiple IP addresses assigned at the same time. Public and private IPs are supported (although private IPs may not be shared publicly) and IPv4/IPv6 dual stack is supported.

Routing of peers follows the Kademlia protocol. Neighbors are discovered using the XOR distance between node IDs. The node ID is the blake3 hash of the public key compressed form. Blake3 is used for hashing the DHT keys. Copies of blockchains are supposed to be stored by neighbors. However, blockchains that are considered large might also be stored (and discovered) as files.

Each peer may store any file. It will announce that it stores the file in the DHT. Peers maintaining the file discovery DHT may randomly verify that the listed peers are alive and store the files.

Packet Structure

Each UDP command packet has the following structure (see table below). Since the message is Salsa20 encrypted using the receiver’s peer ID and is signed using the sender’s public key, it is secured against simple impersonation attacks (the receiver cannot use the same packet to impersonate the sender, since different receivers will fail to decrypt it). Packets contain a random amount of garbage appended to the payload to prevent fingerprinting based on packet size (side channel attack). Encryption of the packet provides a basic protection against fingerprinting of the traffic based on content (ISPs and corporate firewalls are known to block or degrade P2P traffic).

		Offset	Size	Info
ECDSA Signature applied		0	4	Nonce
	Salsa20 Encrypted Key = Receiver ID	4	1	Protocol version = 0
		5	1	Command
		6	4	Sequence
		10	2	Size of payload data
		12	?	Payload
			?	Randomized garbage. Protects against fingerprinting of the traffic based on size of packet.

Salsa20 Encrypted	65	Signature ECDSA. Sender's public key can be extracted which is the senders peer ID.

The nonce must be pseudorandom generated (may be generated from a message counter), but it must not be an incremented number for each packet, as it might otherwise provide a vector for fingerprinting. The 4-byte nonce is calculated into an 8-byte nonce for Salsa20 by using the same 4 bytes twice. Messages that are responses to a request must use the same sequence number. It allows to filter out unsolicited responses and protect against replay and poisoning attacks. Messages like Response that potentially return multiple replies have a flag SEQUENCE_LAST that indicates the last reply in the sequence. Single reply messages like Pong do not need such a flag.

Note above that the encryption and signing scheme does not protect against deep packet inspection attacks where the attacker knows the receivers peer ID (because it is the encryption key). However, it provides a reasonable encryption and entropy against current real-world firewall detections.

The target packet size shall, where possible, not exceed 508 bytes to prevent fragmentation and increase delivery probability. The next limit that should not be exceeded is 1472 bytes, which is right below the Ethernet MTU (maximum transmission unit) limit.[3] The hard cap is 65507 bytes.[4]

The following commands exist. Responses may only be sent after receiving a corresponding request.

	#	Name	Info
Peer & DHT Management	0	Announcement	Message to a new peer to inform about self and to request information about other peers.
	1	Response	Response to the announcement.
	2	Ping	Keep-alive message (no payload).
	3	Pong	Response to ping (no payload).
	4	Local Discovery	Local discovery.
	5	Traverse	Help establish a connection between 2 remote peers.
File Transfer	8	Get Block	Request blocks for specified blockchain.
	9	File Transfer	Start file transfer.
Discovery	16	File Search	Search for files. Response contains blocks or reference to blocks.

Announcement

The announcement (command 0) is used for multiple purposes: initial connectivity (bootstrapping), (unsolicited) announcement of self, sending of neighbor peers, requesting closest peers for a peer ID or file hash, and informing about data stored. It is up to the receiver whether to add the sender into its peer table, and whether to respond to some or all the requested actions.

Offset	Size	Info
0	4 bits	Protocol version support. Might be a higher number than used for sending this packet.

[3] https://stackoverflow.com/questions/14993000/the-most-reliable-and-efficient-udp-packet-size

[4] https://stackoverflow.com/questions/1098897/what-is-the-largest-safe-udp-packet-size-on-the-internet

0.5	4 bits	For future use.		
1	1	Feature bit array. Informs about what the sender supports.		
		Bit	Feature	Info
		0	IPv4_LISTEN	Sender listens on IPv4
		1	IPv6_LISTEN	Sender listens on IPv6
2	1	Action bit array. The receiver is asked to provide the following.		
		Bit	Action	Info
		0	FIND_SELF	Request closest neighbors to self
		1	FIND_PEER	Request closest neighbors to peer
		2	FIND_VALUE	Request data or closest peers
		3	INFO_STORE	Sender indicates storing provided data
3	4	Sender's blockchain height.		
7	8	Sender's blockchain version. Any stored previous versions must be deleted, and the blocks must be requested again.		
15	2	Sender's internal listening port		
17	2	Sender's external listening port, if known		
19	1	Size of User Agent in bytes. Max 255.		
20	?	User Agent in the format "Software/Version number". Required in the initial announcement/bootstrap. Example: "Test Client/1.0". UTF-8 encoded. Must not contain null bytes. Max length is 255 bytes.		
?	?	Additional data depending on the actions requested in consecutive order.		

Actions FIND_PEER and FIND_VALUE send an array of hashes to look up in the hash table:

Offset	Size	Info
0	2	Count of keys that follow
2 + 32 * n	32	Key = blake3 hash of peer ID (compressed format) or data

Action INFO_STORE allows to inform other peers (usually the closest ones to the blake3 hash) that the sender stores certain data by sending the below structure. Receivers may choose to maintain that information in their hash table to inform others when requested (via FIND_VALUE). The protocol intentionally does NOT provide a push mechanism for data stored by 3rd party peers to prevent flooding and spam. The receiver may verify if the sender stores the data by requesting it. This can be done randomly to prevent unnecessary overhead. INFO_STORE does not trigger a response.

Offset	Size	Info
0	2	Count of records that follow
Each record:		
0	32	Key = blake3 hash of the data
32	8	Size of the data
40	1	Type of the file: 0 = File, 1 = Header file containing list of parts for splitting up large files and allowing transfer in pieces.

Response

The response has the same basic format as the announcement, except that the actions field has a different meaning. Bit 0 of Action is the SEQUENCE_LAST flag. Multiple response messages may be sent for a single announcement. This structure follows the User Agent:

Offset	Size	Info
? + 0	2	Count of peer responses
? + 2	2	Count of embedded files
? + 4	2	Count of hashes not found

FIND_SELF, FIND_PEER, and FIND_VALUE use each below structure to report back peers. The announcement message uses the feature field (IPv4_LISTEN and IPv6_LISTEN) to indicate whether the response shall contain IPv4 and IPv6 addresses.

Offset	Size	Info
0	32	Requested Key = blake3 hash.
32	2	Count of peer records that follow (15 bits). Bit 15 indicates the last result.
Each peer record:		
Offset	Size	Info
0	33	Peer ID compressed form
33	4	IPv4 Address
37	2	IPv4 Port
39	2	IPv4 Internal port as reported by that peer
41	2	IPv4 External port as reported by that peer
43	16	IPv6 Address
59	2	IPv6 Port (actual one used for connection)
61	2	IPv6 Internal port as reported by that peer. This can be used to identify whether the peer is potentially behind a NAT.
63	2	IPv6 External port as reported by that peer. This is used in case of port forwarding (manual or automated). 0 if not known.
65	4	Last contact with peer inbound in seconds
69	1	Reason: 0 = Peer is close to requested hash, 1 = Peer is known to store the data for requested hash

A response to FIND_VALUE may provide the requested data as embedded file. This is only possible if the entire file fits into the UDP packet, otherwise it must be transferred using a file transfer request. Each embedded file has the following header:

Offset	Size	Info
0	32	Key = blake3 hash
32	2	Size of data
34	?	Data

FIND_SELF, FIND_PEER and FIND_VALUE requests may result into not-found records. The hashes that were not found are stored at the end of the packet (as array of 32-byte hashes).

Local Discovery

The local discovery message is used to discover peers over the local network. It may only be sent as part of a local IPv4 broadcast, IPv6 multicast or brute-force. It may not be sent via the internet. Since the broadcast/multicast targets unknown recipients, it cannot use the receiver's peer ID for encryption. Instead, it uses hardcoded private keys for encryption and decryption. Note that the local discovery feature with hard-coded keys is a tradeoff with deep packet inspection security. The structure is the same as the announcement message (the Action field is not used). The local discovery message has no response.

Traverse

The traverse message helps 2 remote peers to establish a connection to each other. If Peer M sends Peer A the information of Peer B, and A identifies that B is behind a NAT, it will send M the traverse message which forwards it to B. When B receives the message, it will verify the signature of A and then processes the embedded message as if it were sent from A directly.

In this case M acts as a proxy. If A sends both the Announcement message directly to B and via the Traverse message through M, B is likely to receive it. When B sends back the Response message it should automatically open the port on B's NAT for further communication.

Note that this UDP hole punching mechanism is likely to fail if both peers are behind symmetric NATs. Note that full proxying of both the Announcement and Response messages is currently not considered in this version.

Offset	Size	Info
0	33	End receiver peer ID
33	33	Peer ID that is authorized to relay this message.
66	8	Expiration time when this forwarded message becomes invalid.
74	2	Size of embedded packet.
76	?	Embedded packet.
76 + ?	65	Signature by original sender. Must match the embedded packet.
?	4	IPv4 address of the original sender. Set by authorized relay.
?	2	IPv4 port used for connection.
?	2	IPv4 external port as reported by the original sender.
?	16	IPv6 address of the original sender. Set by authorized relay.
?	2	IPv6 port used for connection.
?	2	IPv6 external port as reported by the original sender.

Blockchain

The blockchain height is 32-bits and the version number 64-bits, which is the absolute limitation on the user's modifications of his blockchain. Adding files, removing them, renaming them, changing other metadata tags etc. produces additional blocks (and increasing the height). However, when there are many deletions or many small blocks, it makes sense to instead increase the version number and start the height at 0. This is beneficial to keep the blockchain small when many files are deleted, while at the same time increasing privacy. There is no hard cap of block sizes, but clients may choose not to download large blocks. For efficiency, blocks slightly smaller than 64 KB (minus packet overhead) are strongly desirable as they can be shared in a single UDP packet.

Block header:

Offset	Size	Info
0	65	Signature of entire block
65	32	Hash (blake3) of last block. 0 for first one.
97	8	Blockchain version number
105	4	Block number
109	4	Size of entire block including this header

113	2	Count of records that follow

Each block record that follows the block header has the below format. The record type identifies the structure of the data field and allows new record types to be defined in the future. If records are unique in their nature (like username), then the most recent one is valid, and all previous ones shall be ignored. Existing records in the blockchain can be marked as deleted using record type 5. Clients must honor this delete record. To completely remove a record, the blockchain should be refactored and the version number increased.

Offset	Size	Info	
0	1	Record Type:	
		0	Username
		1	Directory. Only valid in the context of the current block.
		2	File metadata
		3	Content rating (positive).
		4	Content report (negative). This includes spam and abuse reports.
		5	Delete existing specified record.
1	4	Size of data	
5	?	Data	

Directory structure (record type 1):

Offset	Size	Info
0	2	Directory ID
2	?	Directory name

File structure (record type 2):

Offset	Size	Info
0	32	Hash blake3 of the file content
32	1	File type (low-level)
33	2	File format (high-level)
35	8	File size
43	2	Directory ID
45	2	Size of file name
47	2	Count of tags
49	?	File name
?	?	Tags (TBD)

Design Considerations

The design considerations for this protocol were:

1. First packet must establish connection. This includes sending metadata and actions.
2. Secure against eavesdropping.
3. Secure against message forgery.
4. Secure against tampering and by extension bitflip.
5. Small as possible to increase deliverability.

6. TCP-like acknowledgment and resending are not desired due to the decentralized nature of the network and natural concurrency. The exception to this is file transfer.

This protocol was developed by keeping real world use cases including Bitcoin, Ethereum, Sality, ZeuS Gameover, and ZeroAccess in mind.

Peer Discovery

Initial peer discovery is done via bootstrapping. There are various ways of how peers may find other peers over the internet and locally. The official Peernet core supports local discovery (in the same LAN) via IPv4 Broadcast and IPv6 Multicast. Multicast DNS was intentionally not supported as it is very unreliable. For initial connectivity it uses as list of root peers to connect. Other ways for peer discovery in the future could include DNS seeds and potentially brute-force port scanning for peers.

Port scanning, while obviously very noisy (and such traffic potentially getting incorrectly flagged as malicious), would allow the initial discovery process to be truly decentralized. This ultimately fixes the no connectivity problem that exists with most file sharing software that is a couple of years old.

Ongoing peer discovery is accomplished by exchanging peer lists. Clients shall prioritize exchanging high quality peers. High quality can be determined by uptime, reach (for example, is the port forwarded), and other metrics such as connection speed and ping time. Having an up-to-date list of neighbors is important for speedy data discovery and file transfer.

For privacy reasons, initial bootstrapping via hard-coded root peers is preferred, as other methods are likely noisier (for example, DNS creating logs entries in typical corporate networks and some ISPs) and should only be used as fallback options.

File Transfer

TODO

File Discovery

Files may be discovered in a decentralized way using the file metadata DHT. Peers shall index files and associated metadata tags in a sanitized way and make it searchable via DHT.

TODO

Conclusion

The first version of the Peernet Protocol and the Peernet Browser must meet the minimum viable product test. There will be many challenges to solve down the road including related to privacy, security, and spam. Peernet is an open network which means that anyone can see other peer's IP addresses and files they share. Although this is by design, there might be future proposals to reduce potential privacy risks.

The current limitations of the protocol are that only static content is possible; dynamic websites that rely on a backend with a database cannot run on Peernet. This is not necessarily a bad thing; it hopefully enables content creators to spend more time working on their content like in the early days of the internet.

The initial protocol was designed to be simple and powerful. Future proposals may extend the functionality of Peernet. One proposal in the works is adding a virtual currency to support content creators and creating an additional incentive to sharing resources.