DecentChat: A Peer-To-Peer Chat Network

Ishaan Raja
Purdue University
decentchat.ishaanraja.com

November 14, 2020

Abstract

DecentChat is a peer-to-peer chat room with a proof of work system that solves two of the problems that plague traditional chat platforms: server cost and moderation. Being peer-to-peer reduces centralization and allows peers to come and go as they please. Public and private keypairs form the basis of a verifiable identity matching system. A dynamic proof-of-work system based on RSA signatures that adjusts based on message volume prevents flooding and Sybil attacks. As long as nodes remain on the network and are reachable, DecentChat will continue to exist far into the future.

1. Introduction

There are many ways to chat with others on the internet. However, nearly all of them require two things: a central server and a moderating body. Both requirements inherently fix the lifetime of a chat room or chat service to the lifetime of the owner or moderating body's interest. Eventually, the owner of any chat room will stop paying for the server(s) or lose interest in moderating it. These two requirements are and will be responsible for the downfall of virtually every centralized chat room or service to come.

I seek to create a chat application and by extension a chat network that solves these problems by moderating itself and not making use of a central server. By not requiring a central server, the application can live on for as long as users continue to use it. Self-moderation is another component to a long lasting chat network. Nobody will choose to make use of a chat room that is constantly filled with spam, malware, or otherwise undesirable content.

2. Peer-To-Peer

To avoid the use of a central server, this application must be peer-to-peer. Every node in the network must pass messages directly between each other. The key to a peer-to-peer network is ensuring that all nodes can find each other and that any new node can integrate into the network quickly.

When a node first connects to the network, it must connect to an initial "seed" peer to find other nodes to connect with. Seed peers are found via DNS resolution. One or more A records on a given set of domains will point to seed nodes. The node will connect and ask the seed nodes for a list of their peers and slowly construct a list of many peers on the network. Once the network attains a certain size, DNS resolution may no longer be necessary.

DecentChat will make use of a file to store the IP addresses of known peers. Peers inside this file will be the first to be contacted after re-establishing connection to the network.

DNS resolution will not happen every time, but rather only when the client has no working peers to fall back on.

When a client receives a message of type "chat" (see section 4), it verifies it (see section 7), and if it is valid, forwards it to all of its peers to propagate it across the network. This would naturally result in clients receiving messages multiple times. Duplicate messages are ignored and not forwarded.

3. Identity

When chatting with another person, it is important to know their identity. Even when chatting semi-anonymously over the internet, it is necessary to be sure that one is not conversing with an impersonator.

To provide a robust identity verification system, PKCS#8 standard 2048 bit private/public keypairs will be used. Each client will generate an RSA keypair and sign their messages in accordance with the SHA256withRSA standard (see section 5) with their private key. Every message transmitted will contain the sender's public key. This way, a cohesive, verifiable identity can be associated with any given public key on the network. Any attempts at impersonation will be easily noticed as the message signature and public key will not match.

Of course, an RSA public key is not human readable. That is why every chat message will include a "human readable" username and the sender's public key. Both the username and a string derived from the public key (the "identifier") will be displayed on the screen, so that the user can mentally form associations between usernames and identifiers and simultaneously be able to spot impersonation attempts.

The human readable username is set by the user and must be between 1 and 16 characters.

The identifier is the first 10 characters of the SHA-256 hash of the incoming message's public key. Using the public key's hash helps to increase the workload required to create a keypair with a similar identifier to someone else on the network. This added cost will assist in preventing impersonation.

4. Message Architecture

In order for succinct communication between nodes, this network must use a compact message format that structures data in an organized way. The best choice for this is JavaScript Object Notation (JSON). JSON is human-readable, serializable, and has support for multiple data types. Most importantly, it has widespread support amongst multiple languages and Java in particular.
Every message sent to the network will have the following JSON elements:

- Type - Can be "ping", "pong", "peers", "peerAsk", "history", "historyAsk", or "chat". Support for more types and further functionality can be added later.

- Version - Different DecentChat versions may have differences in protocol. The version element allows future releases to properly account for that.
- Timestamp - POSIX Standard Epoch Timestamp representing the number of seconds from January 1, 1970 UTC. Messages not within 2 minutes (the "timestamp tolerance") of the current UTC epoch time at the time of receipt will be ignored.

PeerAsk type messages are meant for new nodes to integrate into the network. New nodes ask existing nodes for their peer lists with this type of message.
PeerAsk type messages will have the following elements:

- Type: "peerAsk"
- Version - see above
- Timestamp - see above

Peers type messages will have the following elements:

- Type: "peers"
- Peers - An array of IP addresses of this node's peers.
- Version - see above
- Timestamp - see above

All clients will respond with a "peers" type message to a "peerAsk" message.
Ping type messages are to verify that a given host is up and running. Once a host receives a "ping" message, it should respond with a "pong" message to let the sender know it is functioning.
Pong type messages will have the following elements:

- Type: "pong"
- Difficulty - Integer representing the difficulty that this client has determined (see section 6).
- Version - see above
- Timestamp - see above

Unsolicited pong messages are ignored to prevent a malicious client from attempting to rig the client's difficulty (see section 6).

HistoryAsk type messages will have the following elements:

- Type: "historyAsk"
- NumMessages - The requested number of historical messages
- Version - see above
- Timestamp - see above

History type messages will have the following elements:

- Type: “history”
- MessageHistory - An array of previous chat messages
- Version - see above
- Timestamp - see above

All clients will respond with a “history” type message to a “historyAsk” message.

A chat message will have the following elements:

- Type: “chat”
- Username - Human readable username between 1 and 16 characters (see section 3)
- Public Key (abbreviated as “pubKey”) - Full Base 64 encoded RSA public key of the sender
- Message - Chat message between 1 and 256 characters
- Signature - Base 64 encoded signature of message digest (see below) that meets proof of work difficulty (see section 5)
- Nonce - signed 32 bit integer that provides the proof of work (see section 5)
- Version - see above
- Timestamp - see above

The chat message digest is equal to the type, message, timestamp, and nonce concatenated with each other in that order.

The timestamp is included as an additional value that can be varied in the case that no nonce meets the network difficulty. It also allows the user to send the same message twice at different times and not have the second message marked as a duplicate.

The client operates on a first-in-first-out system. Chat messages will be displayed to the user in the order that they are received. Although some messages may be received out of order, this should not impede conversation in any meaningful way.

Any incoming message must have a timestamp that is within 2 minutes of the current UTC epoch time. This cap is in place to prevent anyone from trying to remove their messages from difficulty calculations (see section 6) which rely upon counting messages time stamped from the past hour.

5. Proof of Work

Nobody wishes to use a chat room constantly filled with spam, malware, or otherwise undesirable content. To prevent any one user from being able to fill the chatroom with messages, DecentChat makes use of a proof-of-work system.

To do this, all users must sign their chat message digests (see section 4) to create a signature that has a certain number of zero bits at the front. The number of leading

zero bits required by the network is called the "difficulty" and will periodically change depending on the message volume (see section 6).

In order to find a signature that meets this difficulty, the sender must try combinations of the nonce in order to change the digest to yield a signature that meets or exceeds the difficulty. If no possible nonce yields a signature that meets the difficulty, the timestamp can be changed and a new nonce can be found. This is quite similar to Hashcash's proof of work [1] system, except this system uses SHA256withRSA signatures instead of SHA-1 hashes.

Using message signatures instead of hashes prevents replay attacks and deters Sybil attacks; even if someone creates many different public key identities, they still must calculate a distinct proof of work for every single identity they send from.

With this system, even if a user creates many different identities on the network, they must find a different nonce for each message and each identity they intend to send from. This will inevitably take up a lot of computational power, preventing them from flooding in the first place.

When a client receives a message, it can easily verify this proof of work by reconstructing the digest (see section 4), checking that the signature meets the difficulty, and using the public key to verify that the sender signed the digest.

6. Difficulty Modification

Difficulty is a value that measures how difficult a proof of work is to compute. This value specifies the required number of leading zero bits in the message signature to be accepted by the network. Since a message signature is 256 bytes in length with the current RSA implementation, the theoretical maximum difficulty is 2048 and the theoretical minimum difficulty is 0.

The inherent goal of the difficulty is for it to provide a steady flow of 1024 messages per hour (or ≈17 messages per minute). The client's default difficulty is set at 10, which was found to be calculable by the average computer in about 5 seconds.

At the start of every hour ("difficulty determination time"), all clients that have been connected to the network for at least an hour count up the number of valid (see section 7) messages that they have received in the past hour.
The new difficulty is calculated as follows:

$$f(d, n) = \frac{d - 9}{1024} n + 9$$

The variable d represents the current network difficulty and the variable n represents the number of messages a client has received in the past hour. Since difficulty must be an integer, the above output is fed into the Math.ceil() function.

The function will equal 0 if d is equal to 9. A special case is made just for this, if d is equal to 9 and n is greater than 0, then the difficulty is set to 10. Otherwise, the client will calculate the new difficulty with the above function.

The minimum value of the function is 9, and the theoretical maximum value is infinite, however, clients will not accept a difficulty greater than 2048. A difficulty of 2048 would be impossible to calculate since all bits of the signature would need to be 0.

If a client has not been connected to the network for an hour at difficulty determination time it asks all nodes it is connected to for their difficulties and sets its difficulty to the

one with the highest occurring frequency among its peers. These clients also share a difficulty of 1 with other clients, indicating that they have not been on the network long enough to produce a proper difficulty calculation.

At minute 5 of every hour, all clients change their difficulty to the new difficulty they have calculated themselves or found from other peers.

If at least 51% of any client's peers are genuine clients, then 51% of them will share the same difficulty value, and thus that difficulty value will have the highest frequency. This prevents one malicious client from being able to cripple the network by sharing an inflated difficulty.

7. Message Verification

Before processing any received messages of type "chat" (other types are exempt from this), the client must verify the message meets requirements in the following order:

1. Duplicate - The client checks that the message's signature does not match that of a previously received, valid message.
2. Timestamp - The client verifies that the timestamp is within 2 minutes of the current UTC epoch time.
3. Bounds - The client verifies whether the message is between 1 and 256 characters and that the human readable username is between 1 and 16 characters.
4. Proof of Work - The client verifies that the message signature meets the current difficulty by having the proper number of zero bits at the front.
5. Signature - The client assembles the message's digest and using the message's public key, verifies that message is signed properly.

If the message fails at any point in the verification process, it is ignored and not propagated. If the message passes all validation checks, then it is propagated onwards to the client's peers, excluding the peer it was received from.

The message is displayed to the user if the message sender's identifier is not ignored (see section 8).

For messages of a type other than "chat", the client will only check that the timestamp is within two minutes of the current UTC epoch time.

8. Ignore

Even with a proof of work system, there will inevitably be users that are disliked by others. To remedy this, DecentChat provides an ignoring system. Users can ignore a specific identifier (see section 3) and the client will cease to show any messages from that identifier. However, the client will continue to propagate messages from ignored identifiers to ensure that all peers on the network can produce the same difficulty calculation.

The ignoring functionality is very similar to the blocking features provided on nearly every forum and chat room today.

9. Commands

In order for every user to easily be able to use DecentChat's various features, the client provides a commands system. To use a command, the user will type a "/" at the beginning of their message to indicate that they want to use a command rather than broadcast a message.
Available commands are as follows:

- "/addpeer [address]" - Allows a user to manually add a peer by specifying an IP Address
- "/changeusername [new_username]" - Changes the client's human readable username for all following messages (see section 4).
- "/difficultyinfo" - Shows the client's current difficulty
- "/help" - Provides a list of available commands
- "/ignore [identifier]" - Ignores a specified identifier
- "/ignorelist" - Shows the list of currently ignored identifiers
- "/info" - Shows information about the current client
- "/peerinfo" - Shows a list of connected peers
- "/unignore [identifier]" - Removes a specified identifier from the ignore list

More commands may be added in the future to facilitate additional features or functionality.

10. History

When a client first connects to the network, it has no record of previous messages sent. Loading the recent chat history on the network is integral to allowing smooth conversation through DecentChat.

When the client is first started it asks all of its peers for a certain number of historical chat messages (currently set at 32). After a certain period of time, the client then finds the history message with the highest proof of work value. This value is determined by summing up the number of leading zeros on all chat messages contained inside a specific history message.

The history message with the highest proof of work value is the one that gets loaded in and shown to the user. This proof of work system is intended to ensure that the client receives valid chat history and so that no malicious clients can easily send fake history (unless they specifically expend a lot of resources to do so).

11. Conclusion

DecentChat's primary purpose is to be a self-sustaining chat network. Without requiring a central server or moderation body, it can accomplish this purpose. RSA public/private keypairs form the basis of cohesive identity matching system and the proof of work requirement ensures that no one user can feasibly flood the chat room without having enormous amounts of computational power. As more users join the network, the difficulty function will modify the proof of work to keep the number of messages at 1024 per hour. DecentChat will continue to exist as long as the software exists and there are clients running it. With a large enough user base, DecentChat will survive far into the future.

References

1. A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.