1. Introduction

Twitter

> *Microblogging service whereby users may send and receive messages of up to 140 characters, which can be loosely categorised using tags for content discovery.*

Features

The fundamental principle of the platform, which is the compaction of ideas into 140 characters, has sparked a revolution in how we communicate.

Users could spark a conversation on topics (tags) with strangers, discussing anything and everything in real-time.

These informal exchanges, combined with the real-time nature of the platform, allowed for information to be relayed quickly and with ease.

Finally, tags allowed for these messages to be loosely categorised based on topic, and the trending topics feature facilitated the discovery of topics and ideas that would otherwise be concealed.

Flaws

To begin, *Twitter* is not heavily suited to threaded conversations. It would be valuable to have both a mechanism for individual threads within a topic, and individual replies within those threads.

Aside from this there is also the system itself, which is proprietary and centralised, which contributes to multiple issues:

- Lack of proper security audit due to it being closed-source.
- Mandatory tracking of users for advertising (privacy issues).
- Lack of openness means lack of user control over data. The public API is also restricted which hinders external development and innovation.
- The service is completely centralised, which raises concerns about the stability and reliability of its services and its vulnerability to censorship.

2. Alternatives

FETHR	This project stimulated my research and subsequent creation of *BitWeav*. *FETHR* is a design for an open decentralised micropublishing protocol that runs atop *HTTP*. It improves on Twitter's semantics by including fields for threading, and achieves decentralisation by having each user maintain and serve their own profile. The concept seemed sound but it lacked any tags or discovery mechanisms for content, and instead relied on an *RSS*-like manual approach to sourcing content.
Tent	Tent is like *FETHR* in that it runs atop *HTTP*, but is built more as an all-encompassing decentralised social protocol. It improves on FETHR through additional types for all sorts of content, a more strictly defined publish-subscribe protocol and HTTP header discovery of hosts (more autonomy). Tent is promising, but still lacks this autonomous content discovery mechanism.

BitWeav

While both FETHR and Tent have provided the mechanisms for decentralised social content storage, their fatal flaw is that they do not facilitate autonomous discovery of content — both rely on the user to manually enter their subscriptions, and have no method of categorising content with tags (as *Twitter* does) nor discovering content using a mechanism such as 'trending topics'.

With this in mind, I set out to build just this — a design for a decentralised and peer-to-peer micropublishing network that would function as the missing piece to decentralised social protocols on the Internet. It would be a peer-to-peer overlay orientated around a set of topics, wherein nodes could publish short messages (akin to 'tweets') and subscribe to such topics.

The idea behind these short messages is that *BitWeav* would act as a generic discovery/distribution mechanism for both messages published to the network (in the micropublishing sense) and *URLs* of resources from decentralised social protocols like Tent.

3. Design

The ID

The identifier/ID serves to provide a simple and fixed-size data type for identifying contacts, messages, hashtags and threads. To do this I use two hash functions computed over various aspects of each item (e.g. for a message I hash the content but for a contact I hash their public key). Firstly SHA-256 is used to get a unique identifier, and then it is securely truncated by hashing it again with RIPDEMD-160.

Contacts and Messages

Contacts are people who access the *BitWeav* network via nodes. They maintain:

- A public/private keypair for authenticating data.
- One or more nodes for accessing the network.
- A set of interests (a list of IDs), including subscriptions to:
  - Contact profiles
  - Hashtags
  - Threads

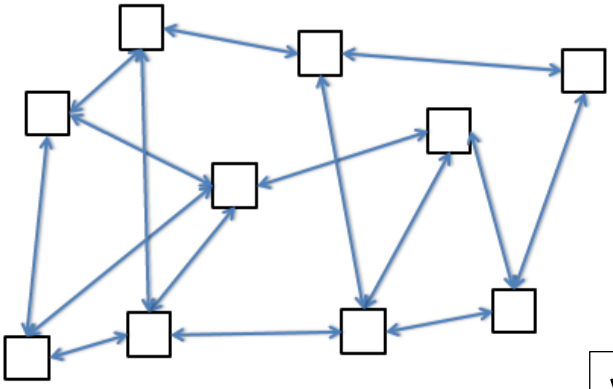
As interests may vary over time, contacts securely timestamp them so updates may be distributed easily.

Messages are the basis for which contacts operate; they are the goods of the network. A message contains fields for:

- **Content** – the *content* field is the bulk of the message, and contains a maximum of 200 bytes to be interpreted as a UTF-8 string. UTF-8 was chosen as it supports various alphabets and is prominent on the web.
  To further aid in international content, the *language* field contains a string of maximum 13 characters in length to identify with all present languages and scripts, and future ones too. It uses the IETF language code format.
- **Threading** – the *reply* field maps to the ID of the message one is in reply to, and the *root* field maps to the ID of the original stimulus message for that thread (or the ID of a contact in the circumstances when they are publishing to their profile).
- **Authentication** – the *from* field, the public key of the publishing contact, works in conjunction with the *signature* field, a digital signature computed over the remainder of the message, to authenticate the message.

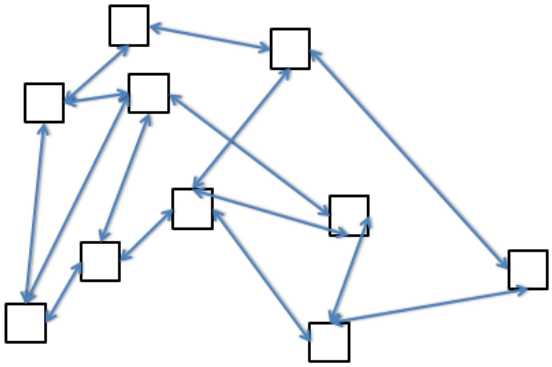
4. Network Overview

BitWeav uses a P2P publish-subscribe network design called *PolderCast*, which consists of three layered modules: *Cyclon*, *Vicinity* and *Rings*. In the *PolderCast* overlay design, nodes organise themselves around topics, to which they can subscribe to and publish on. At a conceptual level, each topic is modelled as a ring that connects all corresponding subscribers of it and only them. These rings are then augmented by random links across topics which produce a single, connected and navigable overlay.



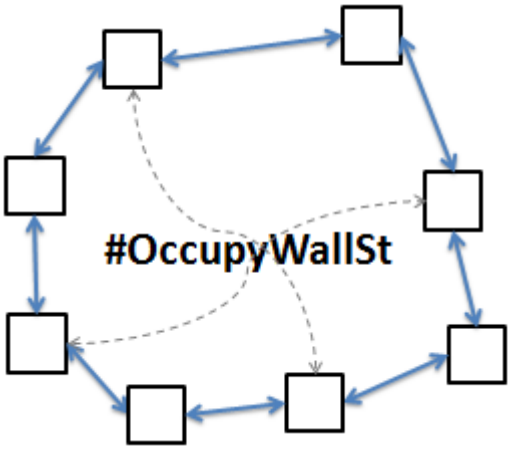
Cyclon

Cyclon sits just above the transport layer. It forms an overlay by maintaining random links between nodes using an epidemic algorithm. Each node knows a small, continuously changing set of neighbours and occasionally contacts one to exchange some of their neighbours.



Vicinity

Vicinity maintains random links between nodes that share interest in one or more topics, by discovering others with mutual interests by contacting nodes discovered with *Cyclon*. Such links serve as input to the *Rings* module.



Rings

Finally, *Rings* discovers a node's successor and predecessor for each topic in its subscription. Nodes are placed into rings in order of their IDs, and quickly adapt to new successors and predecessors in dynamic networks.

The process of publishing a message involves firstly sending it to your successor, who will forward it on along the ring. This provides a linear dissemination speed, so to make things faster, a node will additionally forward the *publish* message to $F - 1$ arbitrary subscribers (sourced from *Vicinity*), which achieves an exponential dissemination speed. F is a dissemination fanout and is typically $F = 2$.