lab10 - Research #23

Reseach Ethereum's SWARM and compare with IPFS

20.12.2016 15:13 - didi

Status:	In Progress	
Priority:	Normal	
Assignee:		

Description

Blog announcement of public alpha

There's a dedicated swarm binary which can run standalone. If connected to an Ethereum node (rpc via --ethapi), it also supports name resolution (ENS.

There are many **similarities to IPFS**.
Files are divided up into **chunks** (SWARM currently uses a default size of 4kB vs IPFS's 256k). A merkle tree of the chunk hashes results in a root hash for every file.
Swarm uses the ***bzz*** *wire protocol* which like Ethereum is based on devp2p and rlpx while IPFS is based on libp2p, another P2P network layer.

An advantage of having the same network layer like Ethereum is that Dapps using both Ethereum and SWARM can **share the network layer**, while for Ethereum + IPFS there's currently the overhead of being connected to 2 distinct P2P networks (which mainly means more low level (TCP, UDP) connections and more book-keeping).

Both use a **Kadmlia** DHT for routing.

Where Swarm and IPFS significantly differ (according to my current understanding) is how uploaded content is distributed.
IPFS seems to so far have no built-in mechanism for automatically distributing content. It's probably more similar to BitTorrent in this respect. Nodes need to explicitly **pin** content they want to keep replicated.
As long as content uploaded to an IPFS node isn't explicitly requested by another node and then cached or pinned, the content just sits on the initial node.
As explained here by the founder, mechanisms for automated replication can and should be built on top of IPFS, an example for this is ipfs-cluster.

Swarm here takes a more "upload and forget" like approach. New content is synced to the network. In order to achieve a **random, uniform distribution**, Swarm nodes get assigned a random address which is in the same address space as the hashes of content chunks. Using a **distance function**, every content chunk is pushed to the *least distant* node(s). That allows for initial distribution.

Both IPFS and Swarm achieve **auto-scaling** by having nodes cache relayed chunks. A simple caching policy (LRU) is expected to have nice properties like good performance and DDOS resistance.
In the tradition of Ethereum, the Swarm design includes economic incentives to encourage nodes to behave this way. The Swarm Accounting Protocol (SWAP) uses Ethereum smart contracts to provide a means for paying for storage and bandwidth.
While Swarm allows for and encourages peering agreements between nodes (exchange of resources instead of payment), this makes Swarm suitable for more use cases. For example somebody wanting to store a highly replicated backup may decide to just pay for it. Or somebody without nodes but with highly popular content may boost availability of that content by just paying a smart contract.

However the current swarm client has SWAP disabled by default, because it's not yet ready. The current implementation reflects PoC 0.2 of the Roadmap.

IPFS also plans to add an incentive layer for ensuring content availability. Initially **Filecoin** (whitepaper) was intended to be built on the Bitcoin blockchain. On Devcon2, Juan Benet explained why Filecoin was late and that the plan was now to implement it on Ethereum (video).

Another difference between IPFS and Swarm is that Swarm is more tailored for hosting web content.
When uploading something, by default a **manifest** file is auto-generated. The hash returned by the upload process points to that manifest file.
Example:

```
$ swarm up /tmp/BCSC_highlights_2.mp4
I1220 14:20:04.062573 upload.go:171] uploading file /tmp/BCSC_highlights_2.mp4 (47566494 bytes)
I1220 14:20:05.792035 upload.go:180] uploading manifest
e1e7ac801bd7bedfc1da07158ece6a74d8c5f0aeb02679f2c8f3e45ebe38b7e5
```

e1e7ac801... is the hash of the generated manifest file. Its content is:

```
$ curl http://localhost:8500/bzzr:/e1e7ac801bd7bedfc1da07158ece6a74d8c5f0aeb02679f2c8f3e45ebe38b7e5
{"entries":[{"hash":"ca2eca8928f0ef957a8327d9de1cef6366b104fbc01d81e13357d94c61dd92fb","contentType":"video/mp4"}]}
```

Thus the root hash of the video actually uploaded is ca2eca8928....

If using the bzz **url scheme** instead of bzzr as above where the 'r' stands for *raw*, the e1e7ac801 hash directly leads to the video, see link.
The manifest concept was introduced in order to reflect the behaviour of web servers in regard to picking a file for the generic URL (for example Apache defaults to index.html.
More on Swarm url schemes and manifests can be found here.

While IPFS is intended to replace **http**, Swarm embraces it for its main API.
Using IPFS requires running an IPFS node (even if in a browser).
Swarm, on the other hand, doesn't consider its http api an intermediate workaround. As a consequence, it also allows and encourages using HTTP POST for uploading.
Usage of the RPC interface of the Swarm client is recommended for debugging purposes only (source.

IPFS supports upload via HTTP gateway, but defaults to an RPC API.

Here is a comparison from the viewpoint of Swarm developer Viktor Trón. Related Reddit thread.

Swarm has a dedicated **Gitter** channel which has become quite active since the alpha publication.

Related issues:

Related to lab10 - Research #13: Research IPFS	**In Progress**

History

#1 - 22.12.2016 20:47 - didi

- Related to Research #13: Research IPFS added

#2 - 30.12.2016 20:01 - didi

Another related project is Zeronet (self description: *Decentralized websites using Bitcoin crypto and BitTorrent network*).
Demo gateway here.
Index of some sites here.
Example site Cyphernomicon.

My understanding:
It's basically a Bittorrent overlay which makes it suitable for website hosting.
Similar auto-scaling like IPFS and Swarm, but no auto-distribution a la Swarm and no plans (?) for a built-in incentive system.

#3 - 01.01.2017 17:34 - didi

- Description updated