Sharing content in IPFS: monitoring the bitswap protocol

Gianmaria di Rito - 544013

18 aprile 2021

1 Bitswap Protocol monitoring

1.1 Code

The code implemented is divided into two basic modules:

- **script.sh**: a script written in bash that deals with monitoring the bitswap protocol through the use of the IPFS Command Line Interface. It actually starts a background process that runs the ipfs daemon to join my node to the public network. It then takes care of downloading an IPFS resource using the `<Ipfs get 'CID'>` command where `'CID'` is the identifier of a resource given as input when the script is run. Once the download has been made, it creates a *logBA.csv* where it will enter all the identifiers of the partners present in the bitswap agent list and a *logPeers.csv* where it enters the significant data of the peers who contributed to the download of the file. Finally terminate the ipfs daemon in the background.
- **plots.py**: this python module is responsible for creating the graphs useful for analyzing the data collected in the two log files created by the script.

1.2 How to run the code

The project was developed in a linux environment so some of the following commands may not be valid for other operating systems.

- First of all, to set the environment in which run the ipfs daemon, you will need to execute the command:`<Ipfs init>`
- To run the bash script correctly run the command: `<chmod +x sript.sh>`
- You can now run the bash script accompanied by one `'CID'` at choice: `<./script.sh QmdmQXB2mzChmMeKY47C43LxUdg1NDJ5MWcKMKxDu7RgQm>`
- Once the execution of the script is complete; we can pass to the commands for the execution of *plot.py* :
  - `<pip3 install pandas>`
  - `<pip3 install plotly-express>`
  - `<pip3 install functools>`
  - `<pip3 install ipstack>`
  - `<python3 plot.py>`

  (run *plot.py* in the same environment where the log files were created)

2 Results of the experiment

In order to analyze the data obtained from the script, I created 5 types of graphs built on the data collected from the various downloads made by me. I will report below only those relating to the 2 sets of data best differentiated from each other to try to bring a realistic representation of what happens when sharing content with IPFS. The cases examined are downloads of:

- XKCD Comics (from the web ui in the "Explore" tab)
  CID:QmdmQXB2mzChmMeKY47C43LxUdg1NDJ5MWcKMKxDu7RgQm
  Size: 107MB
- Old Internet files
  CID:QmbsZEvJE8EU51HCUHQg2aem9JNFmFHdva3tGVYutdCXHp
  Size: 210MB

2.1 Amount of bytes that peer sent to my IPFS client

The first graph relates to the bytes received by the partners during the downloads.

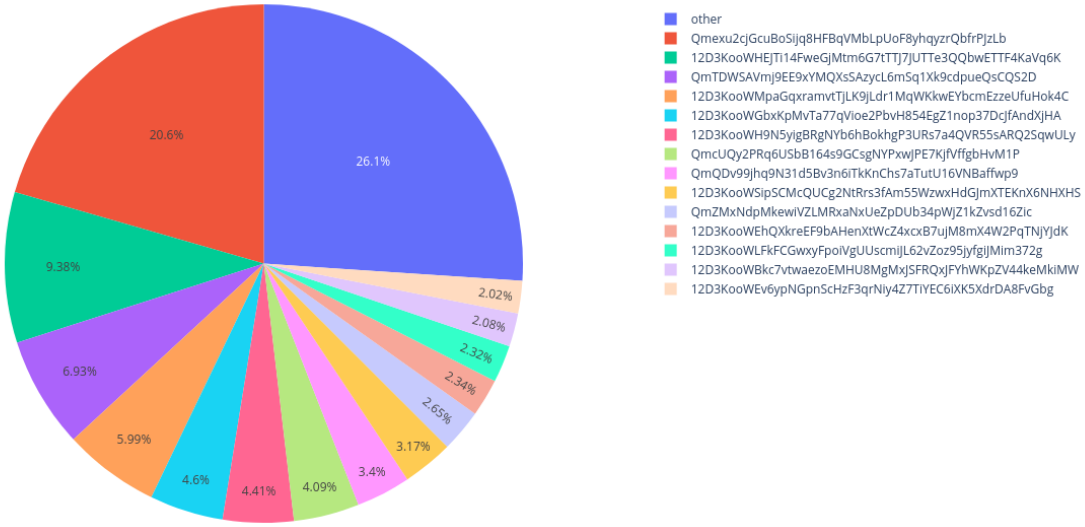

Figure 1: plot of the bytes received for the first downloaded file

As we can see from the graph above, many partners participated in the download of the file *"XKCD Comics"*, to be precise 72. To help reading this type of graph I decided to explicitly include in the count only the partners who participated with more than 2% of bytes sent to my client ; they are 14 and have participated in the download with about 75% of bytes received. The rest were included in the *"other"* label and cooperated for about a quarter of the download.

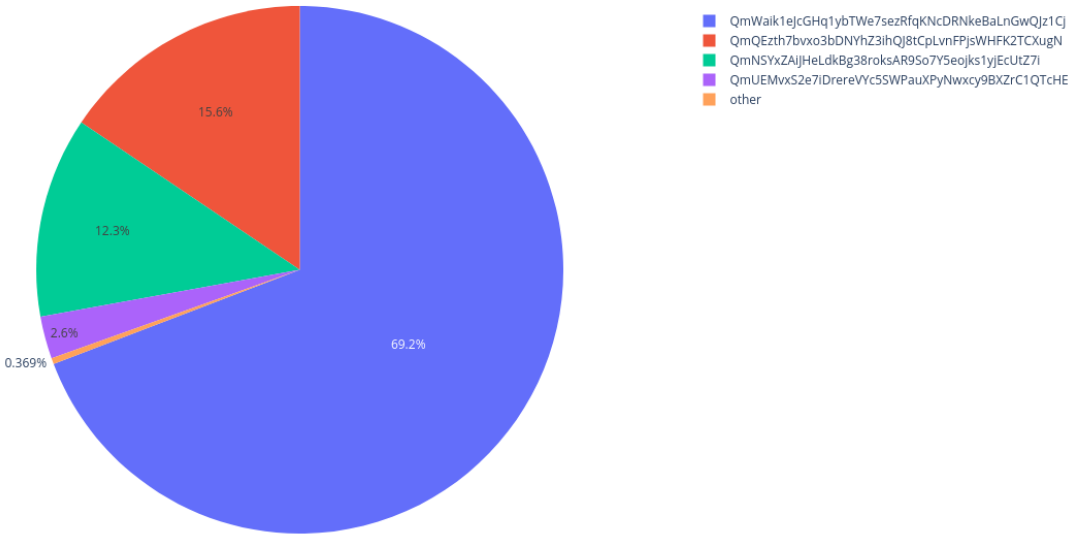

Figure 2: plot of the bytes received for the second downloaded file

While, for the second case taken into consideration, that is the one relating to the second download, we can see that in this case far fewer partners have contributed to sending bytes to my client node. In fact, the entry *"other"* is practically null and only 4 peers stand out who have provided the full amount of bytes; including the one with
$CID = QmWaik1eJcGHq1ybTWe7sezRfqKNcDRNkeBaLnGwQJz1Cj$
which basically sent almost 3/4 of the bytes received. It can be deduced from the two previous cases that the amount of partners who participated in the download by sending bytes to my client node was very variable. In this regard, below you can see the two graphs relating to the percentage of peers present in the list of the bitswap agent who participated in the download of the two contents.

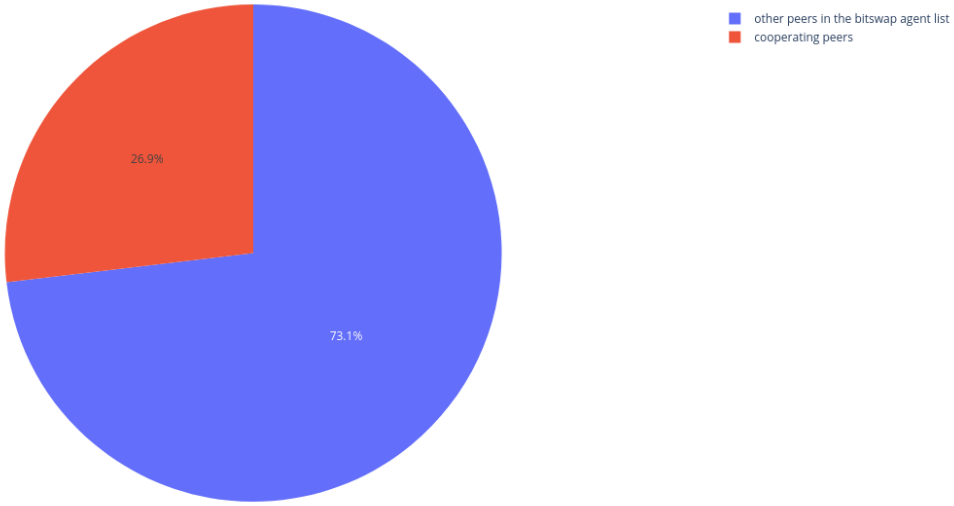

Figure 3: plot on the percentage of participation of the partners relative to the first downloaded content

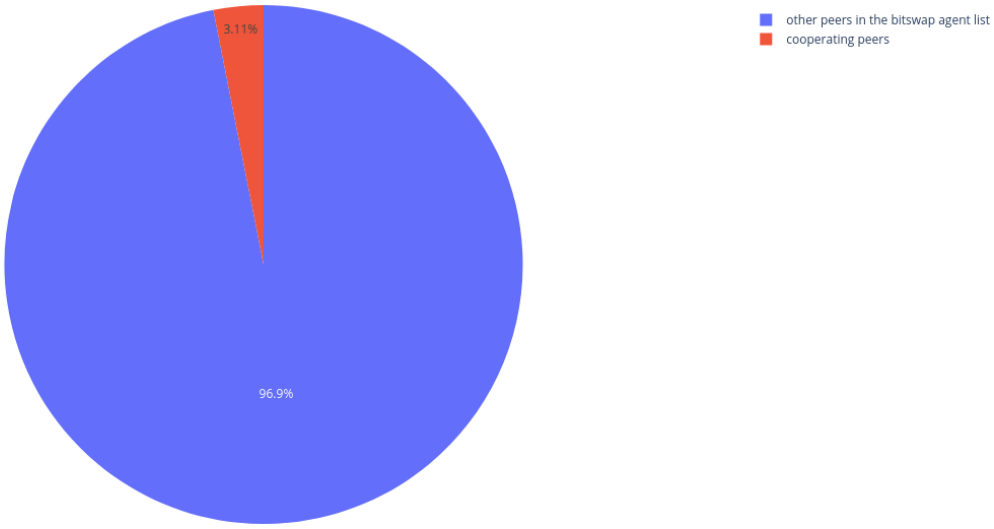

Figure 4: plot on the percentage of participation of the partners relative to the second downloaded content

2.2 Classification of peers

In order to classify the partners present in the bitswap agent list, I wanted to distinguish them based on the version of the cid that identified them. However, this was not possible since by doing the cid inspection of the peers detected during the experiment they were all of type *"base58btc - cidv0"*. The only noticeable differentiation was that relating to the multihash used. In fact, the detected peers are divided between those with the prefix *"Qm"*

that use multihash *"sha2-256"* and those with the prefix *"12D3KooW"* which use multihash *"identity "*. The two graphs that show the diversification of the multihash used by the peers detected for the first and second downloaded content are shown below.

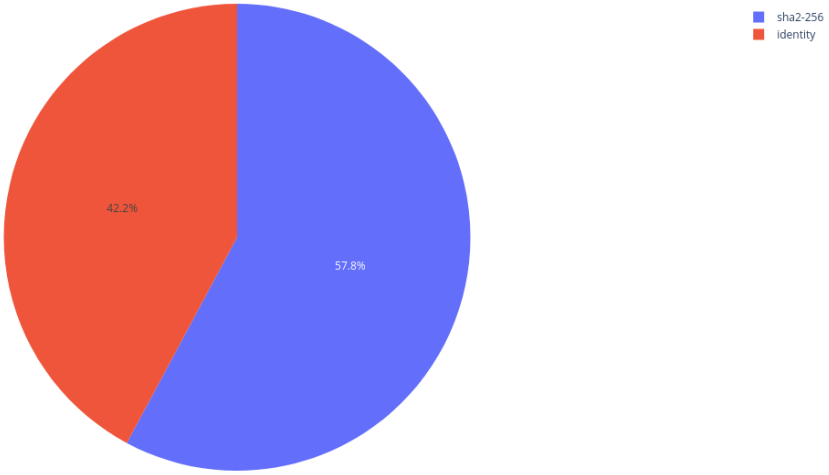

Figure 5: plot on the classification based on the multihash used by the peers detected during the first examined download

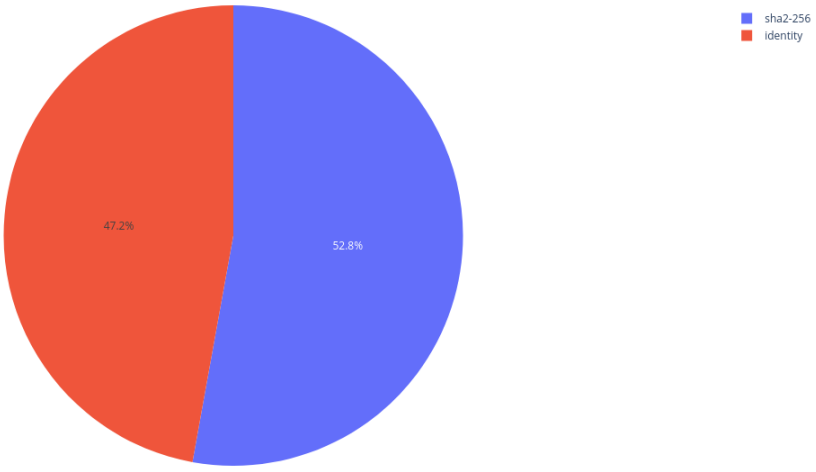

Figure 6: plot on the classification based on the multihash used by the peers detected during the second examined download

From the two graphs it is clear that the distribution of the two types of multihash used is mostly similar in fact in both cases taken in exam it is around 50 % for both types of multihash with a slight prevalence of peers with CID with multihash of type *Identity*.

2.3 Geolocation of peers

Finally, I geolocated the peers who sent bytes to my client.

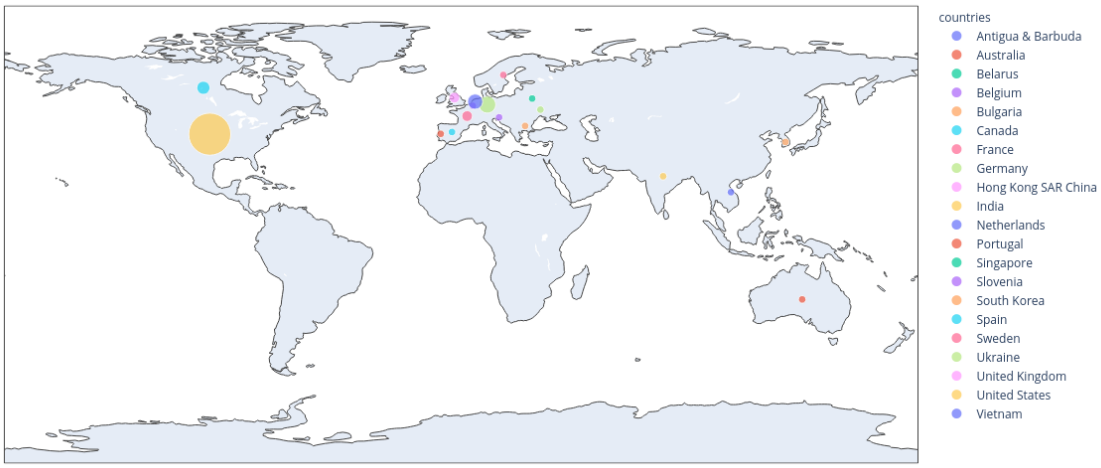

Figure 7: World map with peers per country for the first downloaded content

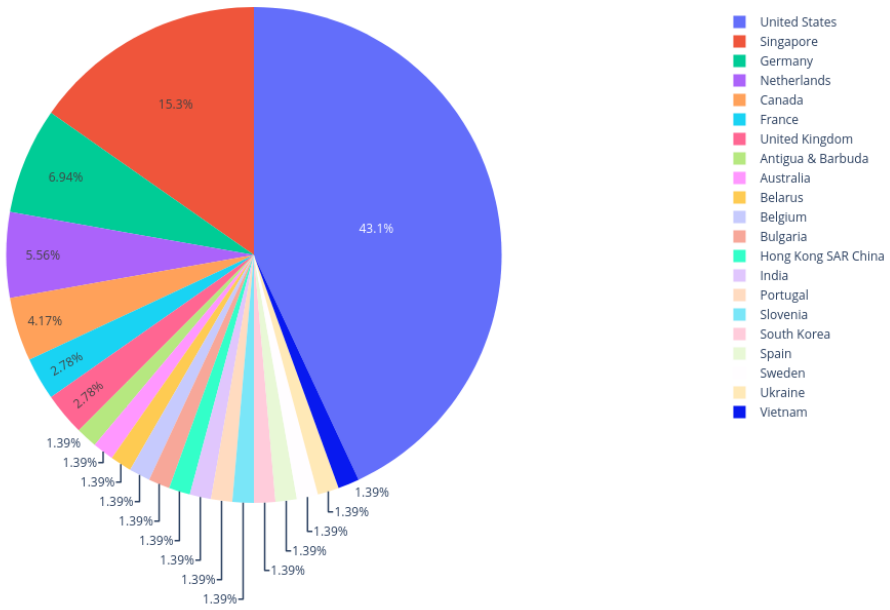

Figure 8: Graph with percentage of peers' countries relating to the first content

As you can see from the graphs relating to the first monitoring, the peers who participated in the download by sending bytes to my client node are scattered all over the world with peers present for example in Vietnam, Germany and Australia. Note that the percentage of peers grouped in the United

States which reaches 43% of the total and Singapore with 15%. The graphs relating to the second monitoring follow.

Figure 9: World map with peers per country for the second downloaded content

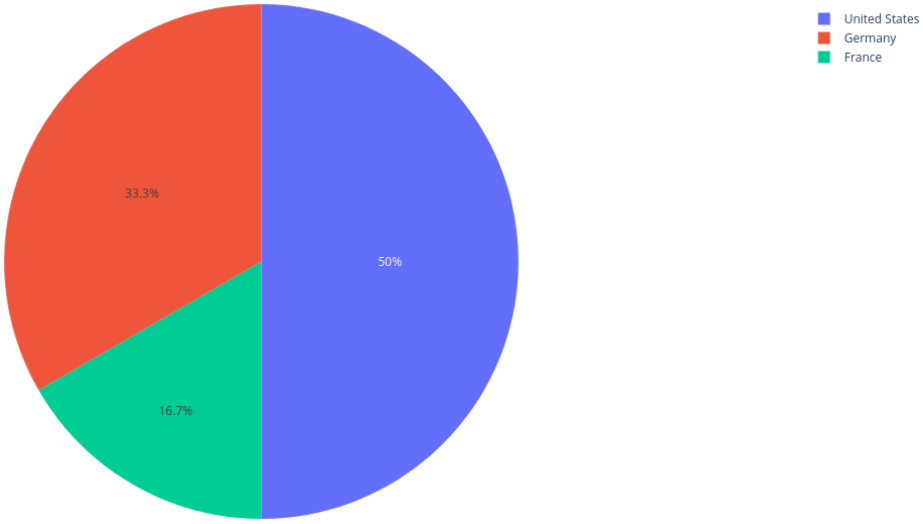

Figure 10: Graph with percentage of peers' countries for the second downloaded content

In this case the situation is different because the peers who sent bytes to my node are concentrated in only three countries, namely France, Germany and United States. We see that, as in the first case, the greatest concentration is present in the United States followed by Germany and France.

3 Answer the following questions

- Describe at least two differences existing between the classical Kademlia protocol and the version of the protocol used by IPFS.

  IPFS uses a version of the Kademlia protocol called S / Kademlia. It is a secure key-based routing protocol based on the classical Kademlia protocol. It offers high resilience against common attacks through a number of security improvements over the classical Kademlia. These improvements lead to differences from the classical Kademlia which consist of:

  1. limitation of free node identifier generation.
  2. parallel look-ups over multiple disjoint paths
  3. extension of Kademlia routing table by a sibling list

  1. Limitation of free node identifier generation
  This is implemented through the use of Proof of Works (PoW) through cryptopuzzles and public key cryptography. Proof of Works is a mechanism that allows one party to prove to another party that it has used a considerable amount of compute resources that cannot be bypassed to calculate Pow. This is often implemented through cryptographic puzzles so that it is easily verifiable but difficult to compute. And it is used to discourage for example DOS attacks and e-mails SPAM.

  - **Dos attack**: solvable by allowing users to access a particular service only if they solve a computationally expensive problem. This will certainly decrease the number of requests and will allow the service provider the possibility to check if the requester has carried out the requested work in a simple way since the pow through cryptographic puzzle is easily verifiable. Therefore, only after this verification the service is provided.
  - **e-mail SPAM**: a proof of work protocol can be associated with an email message. If someone sending a small number of messages the pow will be performed only a very limited number of times and will not constitute an excessive amount of work for the user;for a spammer, however, who could send hundreds of thousands of messages, it could be prohibitive to use so much computational power for each message sent.

  In S/Kademlia the pow via cryptopuzzles is used for ID generation. This generation solves the problem of nodeID generation in Kademlia which was a critical issue since it exposed the protocol to Eclipse attack and Sybil attack.The generation of IDs in S/Kademlia takes place

via a given crypto hash function H(SHA1, SHA256 ....) and $\bigoplus$. First a key K is chosen according to the rules of the static cryptopuzzle ,therefore, it is chosen so that the first c_1 bits of H(H(K))=0. We then choose nodeID=H(K) so that the choice of nodeID is not free. We then move on to apply the dynamic cryptopuzzle by choosing X such that the first c_2 bits of H(key $\bigoplus$ X)=0. You can then increase c_2 over time to keep the generation of nodeIDs expensive. Like all cryptopuzzles, the verification will cost less (O(1)) than the creation that cost much more ($O(2^{c_1} + 2^{c_2})$).

2. Parallel look-ups over multiple disjoint paths
S/Kademlia solves the problem of routing attacks that could occur in Kademlia where attackers reroutes packets into own subset. It does this by using a parallel look-ups over disjoint paths. Lookup procedure:

- lookup k closest nodes in own routing table
- distribute them over d path lookups
- do parallel path lookups in which check if a node already visited/result is used.

In this way, even if a path is controlled by an adversal, the packet will still be able to reach the desired destination since several parallel paths are used.

- Describe the main advantages of the distance metric used by Kademlia with respect to other distances, like the ring distance of Chord.

  The distance metric used by Kademlia is the *XOR* metric. It defines the distance between two objects by doing a bitwise $\bigoplus$ (XOR) on their identifiers. The XOR metric respects the following properties:

  - d(x,x) = 0
  - d(x,y) > 0, if x! = y
  - $\forall$ x,y: d(x,y) = d(y,x) *symmetry*
  - d(x,y) $\bigoplus$ d(y,z) = d(x,z) *transitivity*
  - d(x,y) $\bigoplus$ d(y,z) $\geq$ (x,z) *triangular inequality*
  - given x and a distance Δ, exists a single y such that d(x,y) = Δ *unidirectionality*

  The *XOR* metric was used since, the larger the prefix common to two nodes, the smaller their distance computed by $\bigoplus$ are. In addition, as we have already seen, this metric is:

- Symmetric: allows Kademlia to learn contacts from ordinary queries it receives and helps in building the routing tables. This is one of the major differences that the XOR metric distance has with other non-symmetrical metrics such as Chord fingers in which if a node x receives a query from y, y has a reference to x in its finger table, but x may not be a finger of y so the information included in a received query cannot, in general, be exploited to enrich the finger table.
- Unidirectional: there is a single node at minimal distance with the key, then, the lookups for same key converge to the same path.

Chord distance is not symmetric because takes into consideration that the identifier space is "wrapped". This leads to the fact that in chord: *distance(A, B)*=

B - A (for B $\geq$ A)

B - A + 2^N (for B $<$ A)

=> *distance(A, B)* = (B - A + 2^N) mod 2^N

for instance:

distance(0, 13) = (13 - 0 + 16) mod 16 = (29) mod 16 = 13

distance(13, 0) = (0 - 13 + 16) mod 16 = (3) mod 16 = 3

This asymmetry, therefore, leads to different distances between two nodes which are virtually always at the same distance depending on who calculates it.