SAND: A P2P Protocol for Anonymous Distribution of Large Files

David Harabagiu
harabagiudavid@gmail.com
www.github.com/davidharabagiu/sand

February 2021

Abstract

A protocol which focuses on protecting the anonymity of parties which distribute or download files, while being feasible for large file transfers, would simultaneously solve the main problems of centralized systems - BitTorrent and Tor. The idea is for the two parties (the downloader and uploader) to not have any way of discovering each other's identity, in case one of them is an adversary - the state authority for example. This is achieved through a peer-to-peer network of volunteers which can, at any time, assume the role of proxies carrying control messages and file transfers on other nodes' behalf. A vital component of the protocol is the file searching algorithm: by propagating a search request recursively through the network between nodes in random parts of the world will conceal the identity of the search initiator and make the complete network path unknowable to any individual party. File transfers are performed through a two-layer proxy, which conceals the identities of both uploader and downloader. The network participants are volunteers which sometimes passively donate bandwidth for the benefit of the community.

1 Introduction

In the age of hypercapitalism, wealth inequality has become a growing and prominent issue. The rise of the internet provided people the means of constructing a new reality in which they can be free of social division and the constructs fabricated by the establishment while freely expressing themselves. In this new reality each individual is equal and has access to the same limitless resources. This did not last long however, as more businesses discovered the opportunity of extracting wealth from this new reality. This demonstrated advantage of significantly increasing the amount of content and services available. However, the provided software and media is hidden behind pay walls and region locking mechanisms. The application of the free market - with all its

disadvantages (like being prone to abuse) - in the world of the internet is even less justified because we cannot speak about the concept of scarcity in this context, one being able to replicate information indefinitely. Sharing copyrighted content became the norm for a while, in this way bridging the gap between the low and high income earners [1], but more aggressive policies crept in the world of the internet trying to regulate and punish the free flow of digital information.

Internet *pirates*[1] discovered ways to circumvent sanctions from the authority by using different means, each with its own disadvantages. Private torrent trackers are unaccessible to any legitimate pirate, while VPN services are not free and one should be skeptical about the providers of these services. The Tor anonymity protocol is also great, but has the downside of having high latency and being prone to network congestion [3]. Furthermore, piracy is considered unethical over the Tor network because it might slow down more important communication between individuals living in autocratic regimes with high degree of censorship, which rely on anonymous communication on a daily basis.

The SAND protocol (recursive acronym for *SAND Anonymous Distribution*) leverages the power of a worldwide P2P network, in which each node can act as proxy for another node to provide anonymity in communication. The primary advantage over the BitTorrent protocol is that the identities of the two parties (uploader and downloader) are unknown to each other. This is achieved by the means of a file searching algorithm that recursively propagates a request through the network while probing each node until the file is found. The chosen solution tries to attain the optimal balance between network overhead and security. Each node is a volunteer in the network, as a part of the bandwidth will have to be donated for the benefit of the collective while the client application is running.

2 Peer Discovery

In large-scale P2P networks, such as the one proposed in this work, peer discovery presents a difficult challenge. To better preserve anonymity, nodes will have limited knowledge of the network topology and so, the peer discovery algorithm was chosen accordingly. Because of the large scale of the network, anonymity benefits, robustness, fault tolerance and scalability, the *Gossip Algorithm for Peer Discovery based upon Local In-Degree (GPDL)* was found to be the right tool for the job. This algorithm uses *Gossip Pulling* and *Gossip Pulling* messages to exchange information about neighbors between nodes, while maintaining a *Neighbor Table* data structure, which stores local topology information [4].

It is vital to consider the method by which a new node which joins the networks forms the initial list of neighbors. Those will have to be selected from various regions or countries to prevent attacks by local adversaries (for example those which have access to network information from ISPs). A straightforward solution is to have special and publicly known nodes, which maintain a list of

[1] Person who downloads and distributes copyrighted content digitally without permission, such as music or software [2]

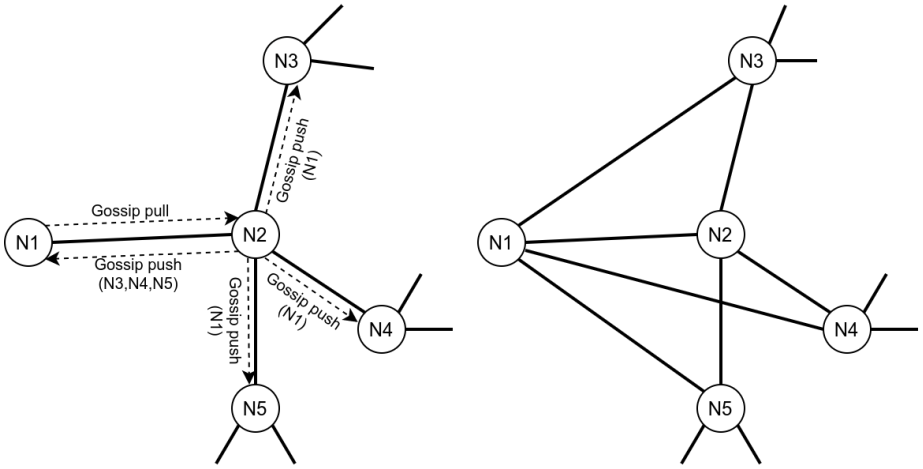

Figure 1: Gossip pull and push messages visualization (left). Altered network topology after message exchanges (right).

all the active nodes in the network, called *Distributed Node List (DNL)*. Each DNL host will have its own full copy of the list of addresses. When a new node joins the network, it will request a list of node addresses from one of the DNL hosts, which will arbitrarily select them from the list. Besides fulfilling a role for initial node setup, DNL hosts can also serve as a background in case all the neighbors of a node go offline. When a node leaves the network by closing the client application, it will notify one of the DNL hosts, which will them remove the address from the list. If a node leaves the network by force, for example if the client crashes, its address will still be in the DNL. This is not a severe problem because the DNL host can simply ping the addresses before sending them to the node which requested them. If one of the addresses belongs to a dead node, then it will merely be removed from the list and replaced by another randomly chosen address. It is important to note that the DNL hosts are not fully fledged nodes in the network, as they will not participate in the usual node discovery and file searching algorithms. Plus, they do not have any knowledge about the network topology; they just know who are the network participants.

Full consistency of DNL copies between the DNL hosts is not a requirement and differences are completely acceptable. SAND takes advantage of this by minimizing the number of synchronization messages between the DNL hosts. Instead of sharing every update immediately, changes can be appended to a larger message, which is sent less frequently when enough changes are accumulated.

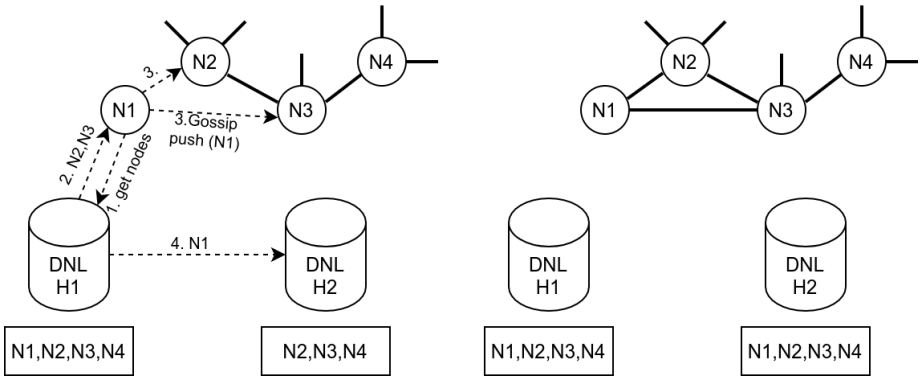

Figure 2: Initial setup of a node N1 joining the network. Left: Message exchanges. Right: New topology.

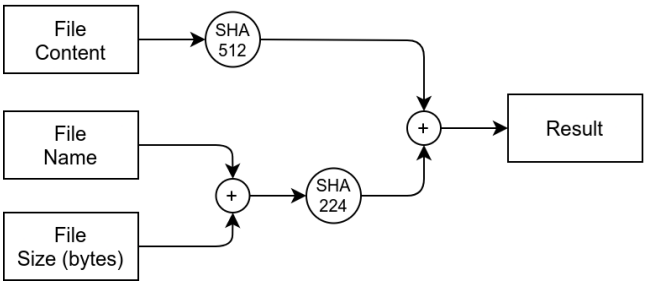

Figure 3: Obtaining the augmented hash (AHash)

3 File Identity

Another important aspect of the system is the way files are identified. Identification simply by file name is out of the picture. We can take as example two installers of the same applications, but each for a different version. This would cause a lot of collision possibilities, which is not desirable the file to identifier mapping has to be "as injective as possible." Another possibility would be to use a hashing function on the file contents. This is a good approach but, in theory, specials files can be crafted in such a way as its hash to match the hash of another file. To greatly increase the difficulty of such an endeavor, a combination of the file content hash and file size can be used. To further reduce the probability of eventual accidental collisions, the file name can be used as well. The combinations of these three values is called the *augmented hash* of the file (or *AHash* for short).

Figure 3 illustrates the process of obtaining the AHash of a file. SHA-3-512 is used for hashing the file content. Because files have virtually no limit in size, the algorithm which reduces collisions the most is needed. File names and the string representation of the size are relatively short and thus SHA-3-224 is

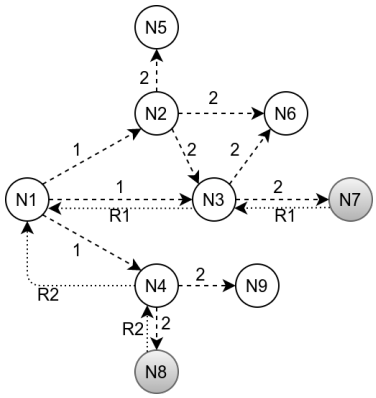

Figure 4: Run of the file searching algorithm with a maximum depth of two

enough. The "+" sign means string concatenation. The final string will be 184 characters long.

The resulted AHash can be distributed on the internet, and other people may utilize it to search through the network for the distributed file.

4 Searching

The system is based upon the fact that no single node knows where can a file be found. With this in mind, a distributed searching procedure or algorithm is needed every time a file needs to be found somewhere in the network of peers.

Basic idea: When a node A asks a node B if it can provide file F, and then B replies with "yes", node A can't know if B has F or it just simply forwards the reply of another node C, which has F, but it's unknown to A. In fact, the "yes" in this case really means that B is a proxy located in the path between the requester (A) and the provider (C), and thus can facilitate communication between the two parties which are unknown to each other. This procedure can be applied recursively, which means that such proxy paths of arbitrary length (or depth) can be obtained. To avoid cycles, each request will contain a unique randomly generated identifier. If a node encounters a request identifier for the second time, it won't forward it again. Figure 4 illustrates a search with a maximum depth of two, initiated by node N1. The nodes N7 and N8 have the requested file.

4.1 Request Table

In order for a node to be part of a search request path, it needs to remember where the request came from. When a reply which contains the same request identifier arrives, the node will know where to direct it based on the association between the request identifier and the local request origin address. The list

of these associations is called *Request Table*. When a reply has been satisfied locally, the entry should not yet be removed, because another reply for the same request might be issued by another node.

The key reason why multiple responses to the same request are collected is that if, for some reason, the file provider disconnects, the requesting node may have backups for the transfer. Besides that, a future version of the protocol might support retrieving a file from multiple providers at the same time (similar to BitTorrent). An entry from the Request Table might have an expiration time, after which it can be deleted.

4.2 Reply Table

The requesting node will have to confirm the transfer with one of the file providers by sending a confirmation through the same path as the request. In order to allow such a scheme for confirmations, replies coming back through a path will have to be saved along with each hop. Let each reply have a unique identifier. When a node is on a reply path, it remembers the reply identifier and its origin in a *Reply Table* (similar to the Request Table). A transfer confirmation will then have to contain the reply identifier corresponding with the route to the file provider. After a reply identifier is used within a confirmation message to forward it, the corresponding entry in the Reply Table can be removed. The other replies associated with the request from which they originated will still have to be kept in the Reply Table for the same reasons stated in the previous subsection.

4.3 Encryption

The reply contains sensitive information, like the *Transfer Key* and *Rendezvous Addresses*, which are used for the actual file transfer and will be discussed in the following section. To avoid snooping by potential adversaries located in the search path, the sensitive information has to be encrypted using a public key scheme. A public key is appended to the request message, which is used by the file provider to encrypt the sensitive information from the reply. The request sender can obtain that information by decrypting it using its private key (figure 5).

The key pair should be regenerated for every search query to greatly reduce the risk of identifying the search initiator.

4.4 Path Caching

Popular files could be requested multiple times and thus, to improve performance and to decrease network congestion, a route caching mechanism is proposed. When a reply passes through a node, indicating the requested file was found on the current path, this information is saved in a local *Path Cache*. An entry in this cache contains the file AHash and the addresses which represent the continuation of the paths to the file provider. When a request for the same

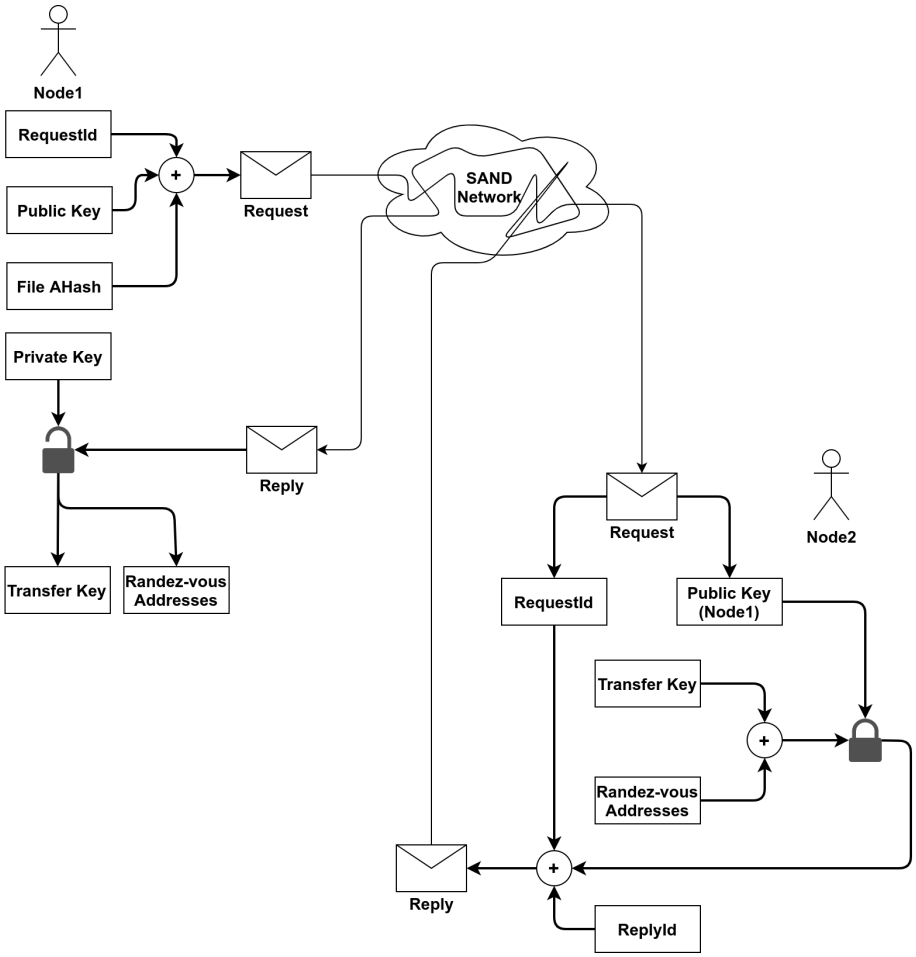

Figure 5: Search request and reply

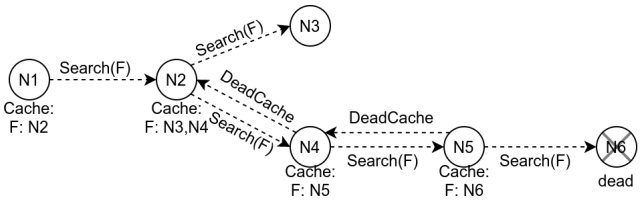

Figure 6: Dead node in a cached path

file is received the cache is consulted then, the request is forwarded only to the nodes indicated in the cache entry.

There are two primary problems with this approach:

- A node in the cached path may go offline and thus invalidate the cache. The solution, in this case, would be to send a special message back along the path which indicates the cache is invalidated.
- New nodes may appear in the network which could create new potential paths. The simplest solution here is to implement a cache expiration mechanism.

Figure 6 illustrates a scenario in which a node from a cached path goes offline. *DeadCache* messages are sent back along the path. N2 does not forward the DeadCache message because the cached path going through N3 is still valid, so N2 remains part of a cached path. Nodes N4 and N5 will remove their entries for file F, while N2 will merely remove node N4 from the entry.

5 Transfer

Transferring files should also preserve the anonymity of the two participating parties. The original idea for achieving this consisted of transferring the files along the same paths used for searching. This is not completely feasible as the search paths can have an arbitrary length and the resulted network congestion would be quite significant (similar to Tor).

The current idea involves the file provider randomly choosing a number of *Rendezvous nodes* (or *Drop Points*), where it will transfer chunks of the file. The addresses of these Drop Points are communicated to the file downloader by the means of the search reply message (figure 5). The chunks of the file are encrypted using the Transfer Key sent using the search reply, because of the possibility of one of the Drop Points being controlled by an adversary. The number of Drop Points is chosen based on the size of the file and the configuration of the Drop Points. The value of 100 MiB was selected intuitively for the default maximum size of a chunk. The Drop Points are chosen from the pool of neighbors. If the file provider does not have enough neighbors, it may request more addresses from one of the DNL hosts. These new addresses will be used only for the current transfer and not added as neighbors.

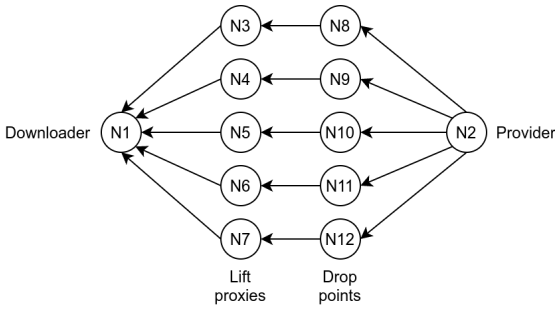

Figure 7: File transfer

At this point, the node which requested the file could just ask the Drop Points for the file chunks, but there is a Man-in-the-middle attack possibility to consider. If an adversary replied to a search request and offers to transfer the file, it could provide node controlled by him as Drop Points. If the victim accesses those Drop Points directly then its identity would be known to the attacker. The downloader does not access the Drop Points directly, but instead delegates the "lifting" of the file chunks to a number of randomly chosen nodes, called *Lift Proxies*. The process of choosing Lift Proxies is almost identical to the process of choosing Drop Points. A Lift Proxy node can be configured to lift chunk of a maximum size. Figure 7 illustrates a file transfer process.

6 Conclusion

The proposed system is a practical way of distributing large files anonymously among peers on the internet. It is easy for everyone to join the network: it does not come with any associated cost, and transfers are not a considerable inconvenience for the 3rd party nodes.

The only downside is the fact that the nodes in the network which are not actively transferring will occasionally have to donate a part of their bandwidth. The users could configure a bandwidth limit amount in such a way as to not negatively impact their internet usage. Other data that is not big, but should be mentioned, is the frequent background traffic required for node discovery messages and forwarding search requests and replies. An upcoming version of this paper will be released, in which this background traffic and passive transfer assistance are measured in a simulation.

Another downside is the fact that a file cannot be downloaded from multiple sources at a time, similar to BitTorrent, to reduce traffic for individual file distributors and increase transfer speeds. A future version of the protocol might include this functionality.

SAND could work really well is low to mid income countries where piracy and cheap high-speed internet broadband are common, like in Romania for example. As a side note, in Romania, these circumstances allowed for many citizens to

acquire technical skills and be lifted out of poverty while the country is now one of the leadings in technology workers per capita [5].

Clandestine software and specialty literature usage without legal consequences may effectively aid people in other underdeveloped countries, like those in Africa. We could consider this as a means of the western world to pay back a bit for the continuing neocolonialist practices.

References

[1] S. Asongu, "Software piracy, inequality and the poor: Evidence from africa," *SSRN Electronic Journal*, 01 2012.

[2] D. Y. Choi and A. Perez, "Online piracy, innovation, and legitimate business models," *Technovation*, vol. 27, no. 4, pp. 168–178, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166497206001040

[3] K. Girry, S. Ohzahata, C. Wu, and T. Kato, "Reducing congestion in the tor network with circuit switching," *Journal of Information Processing*, vol. 23, pp. 589–602, 09 2015.

[4] T. Lu, B. Fang, X. Cheng, and Y. Sun, "Peer discovery in peer-to-peer anonymity networks," 01 2006, pp. 131 – 136.

[5] A. Fiscutean, "The mix of poverty and piracy that turned romania into europe's software development powerhouse," *ZDNet*, 08 2014. [Online]. Available: https://tinyurl.com/3necxbef