A libP2P Implementation of the Bitcoin Block Exchange Protocol

BARBARA GUIDI, ANDREA MICHIENZI, and LAURA RICCI, Department of Computer Science, Università di Pisa, Italy

The progressive decentralization of web services brought a need for new instruments to support the development of P2P protocols and applications. The only tools available are network or event simulators, which help the development in artificial or extremely controlled environments. They usually provide low-level support with completely unstructured networks. LibP2P is a library that tries to solve all such problems, letting the users easily deploy nodes on the Internet. It also comes with a set of functionalities, to solve the most important problems of P2P networks, such as NAT traversal, peer and content discovery and routing, and much more. In this paper, we employ LibP2P to implement min-bitcoin, a minimal version of the protocol used by the Bitcoin network to exchange the blocks. We evaluate our implementation to assess the applicability of LibP2P in a real-life blockchain scenario. The evaluation uncovers that setting up communication channels is time-consuming, but data transfers are fast. Additionally, LibP2P efficiently manages cases in which nodes are behind a NAT, under VPN, or in geographically distant places.

CCS Concepts: • **Networks** → **Peer-to-peer protocols**; **Application layer protocols**; • **Computing methodologies** → *Distributed algorithms.*

Additional Key Words and Phrases: LibP2P, Blockchain, Bitcoin

ACM Reference Format:
Barbara Guidi, Andrea Michienzi, and Laura Ricci. 2021. A libP2P Implementation of the Bitcoin Block Exchange Protocol. In *International Workshop on Distributed Infrastructure for Common Good (DICG '21), December 6–10, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3493426.3493822

1 INTRODUCTION

Nowadays, decentralization through Peer to Peer (P2P) networks or wireless sensor networks is considered more important, and it represents the future of several applications. In a P2P network, all participant nodes are considered equal, peers, and in principle there are no statically defined roles: each peer can provide a service and access services used by other peers. P2P networks were mainly used for file sharing, and the killer application is Bittorent. Then, with the introduction of the blockchain technology, P2P networks have been utilized for several contexts, like cryptocurrencies, supply chains, and decentralized finance. However, implementing a stable, fully functional P2P network is extremely challenging, while simulation requires less cost and less computational resources. Therefore, scientists and developers tried multiple times to develop simulators [2] supporting the developer in prototyping the P2P network. A simulator lets a developer focus only on the service implemented, while disregarding all the problems related to the creation and management of the P2P network. However, simulators have limitations, because they are not always capable of handling a large number of nodes, and cannot accurately simulate all the problems, such as content routing, peer discovery, and so on. In this direction, the InterPlanetary File System (IPFS), and in particular libP2P represents a revolution. IPFS is a peer-to-peer content-addressable distributed file system that seeks to connect all computing devices with the same system of files. It is an open-source community-driven project, with a global community of millions of users. libP2P was developed specifically for the IPFS as an open-source project, and it became very popular for many other projects as well. The library offers a set of predefined functionalities and protocols to support the developer, such as a peer can very easily connect to a publish-subscribe P2P network. Additional support covers the problems of peer

and content addressing, routing, and discovery, Network Address Translation (NAT) traversal, and multiplexed private communication channels. It is very versatile, and can be used in different scenarios, such as decentralized file systems [6, 10], or social media applications [4, 5, 7].

The aim of this paper is to investigate the usability of this library in a blockchain context. In more detail, we implement `min-bitcoin`, a minimal version of the protocol used by the Bitcoin nodes to exchange blocks. The implementation uses the Go version of libP2P, and is based on a Publish-Subscribe network to advertise the blocks owned by nodes, and direct connections to exchange the blocks. We evaluate our implementation, focusing on some critical steps automatically managed by libP2P, such as the setup of connections and the exchange of data in various scenarios. The evaluation highlights that libP2P is an extremely helpful tool in the scenario of blockchain, because the most burdensome delays are only caused during the setup of communication, but otherwise the data is transferred very quickly.

The paper is structured as follows. In Section 2 we briefly overview some relevant projects concerning P2P network simulators. Section 3 provides the most important features of libP2P used in the implementation of `min-bitcoin`. In Section 4 we present the implementation of `min-bitcoin`, while in Section 5 we evaluate the implementation under different circumstances. Section 6 concludes the paper, pointing out possible future works.

2 BACKGROUND

During the years, numerous P2P simulators have been proposed. NS2 [3] is a general purpose simulator which can simulate many types of low level networks (LAN, WAN, wireless, satellite), but its documentation is lacking which makes it hardly usable because to configure the network one must extend the classes provided. It supports the development via the low level tools, such as TCP connections. NDP2PSim [8] partially extends this simulator by abstracting the concepts of network structure, socket, and more. PlanetSim [11] is a P2P network simulator written in Java, which supports both structured (Chord, Symphony) and unstructured (Gnutella) overlays. It offers the possibility to customize the bandwidth and the latency of the overlays simulated, and supports the simulation of multi-overlay networks, where specific nodes work as bridges between the overlays. OMNeT++ [12] is a general purpose discrete event simulator, which can be used for modelling numerous scenarios, such as multiprocessors, communication networks, and other parallel or distributed systems. Oversim [1] is similar to OMNeT++, but includes more structured overlay protocols such as Kademlia, and introduces important optimizations, which makes possible to simulate up to 100,000 nodes. Lastly, PeerSim [9] is a simulator written in Java, which proposes two different simulation engines: a cycle based simulation, where each node can perform a set of actions periodically for a predetermined set of times, and an event based simulation, where nodes send asynchronous messages and the simulation goes until no more messages are sent or the simulation is interrupted forcefully. The two modes can be combined to better simulate a real world scenario.

All the presented proposals have two shortcomings: firstly, they are not tools with which one can build actual P2P applications, and cannot accurately reproduce node churn, communication latency and other problems. Secondly, they usually do not provide high level functionalities, such as multiplexed connections or publish-subscribe networks, and solutions to real world scenarios, such as easy and automatic solutions to NAT traversal, peer and content discovery, and so on.

3 LIBP2P: HOW IT WORKS

LibP2P is a library, highly modular, where each module addresses a specific problem (see Figure 1).

LibP2P uses *multiaddress* to specify peers and protocols in a string format. In libP2P a connection between two peers consists of the possibility to establish a bidirectional and secure communication channel, called *stream*. More than

one stream can be active between the same pair of peers, and each stream is logically independent of one another. Streams support *backpressure*, so that readers are not flooded by malicious writers, and *half-close*, that is a peer can autonomously decide to use its end of the stream in read-/write-only. Each stream can use different protocols, according to a negotiation phase.

When a new connection or stream is opened, the two peers can negotiate which protocols should be used, and when a peer proposes a protocol to use, the other can accept it or send special message saying that it does not support it. The name of the protocols is specified using multiaddress, and when a consensus is reached, the peers will start using the agreed protocol.

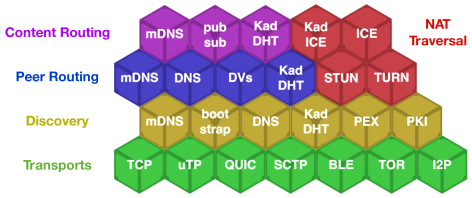

Fig. 1. Modules and protocols of libP2P[1]

LibP2P offers three protocols for content routing: *Multicast DNS* (mDNS), *Kademlia DHT* (KAD), and *Publish-Subscribe*. mDNS consist of broadcasting a query, asking the peer with a specific content to identify itself, while KAD leverages the DHT to find a peer with a specific content. Both are general purpose and can be used also for peer routing and discovery. *Publish-Subscribe* protocol has two implementations: *FloodSub* (based on network flooding) and *GossipSub*. In GossipSub peers organise in a 2-layer logical network: a sparse layer, called full-message, where the published messages are exchanged, and a dense layer, called metadata-only, mainly used to maintain the other layer.

NAT traversal protocols are among the most important protocols offered by libP2P. If the router supports `UPnP` (*Universal Plug and Play*) or `nat-pmp` (*NAT Port Mapping Protocol*), libP2P automatically tries to configure the router to enable inbound traffic. Other techniques used by libP2P include trying to listen to incoming connection on the same public port of the router as the port associated by the router to the peer connection (*Hole Punching*), or on other ports as observed by the other peers (*AutoNAT*). If none of the other techniques work, a *Circuit Relay* is needed, which consist of letting a peer relaying all the traffic among two peers.

To discover new peers in the network, libP2P offers the already cited mDNS, and the possibility to leverage the DHT, such as by making random queries and analyse the replies. In case the peer was online for some time, it can observe the P2P network to understand which are the most reliable peers, which were found online for long periods of time, by storing their ids in a bootstrap list. Another practical techniques is the usage of *Rendezvous* points. A node can connect to a rendezvous point and store a record which can be later retrieved by other nodes.

4 THE IMPLEMENTATION OF MIN-BITCOIN

To assess the functionality of the library, we implemented, a minimal version of the Bitcoin block exchange protocol. This choice was driven by the need of implementing a real life scenario protocol, which is currently adopted at a large scale. The core of the protocol is made of 3 types of messages:

- *have:* this message is used by peers to advertise the list of blocks owned by the peer to other peers.
- *want:* when a peer receives a message of type *have*, it analyses the message and checks if the other peer is in possession of some blocks that it does not have. If this is the case, the peer can reply with a *want* message to inform the receiving peer that it wants one or more blocks.
- *data:* a peer that receives a *want* message, can reply with a *data* message, containing the requested blocks.

[1]https://ipfs.io/ipfs/QmdzerM4fsnNGVf5jnogxu6QpXQa5rHDK87oHStC5xGCS4/

Description	Time
Connection to bootstrap servers	2-4 s
Connection to GossipSub network	2.6 s
Direct connection to another peer of the GossipSub network	68s (localhost) 72s (lan) 81s (relay)
Send an *have* massage on the GossipSub network	1.85 ms
From receiving an *have* massage to receiving a block requested	3.4 ms (localhost) 73 ms (lan) 473 ms (relay)

Table 1. Delays observed of the key steps of the implemented protocol.

Nodes of the simulation create fake blocks at random time intervals. Each time a new block is created, the peer that created the block broadcasts this fact to the whole network using an *have* message. The other peers react accordingly, asking for the new block through a *want* message, if they are not already in possession, with the id of the requested block. When the block creator receives the *want* message, it replies with the block itself contained in a *data* message. To take into account problems that may be caused by network delays, dropped messages, or network churn, all nodes periodically broadcast a list of the most recent blocks they obtained through an *have* message. The size of this list and the delay between two of these periodic messages can be customized, according to needs and the machine running the protocol.

The implementation of `min-bitcoin` makes use of a Publish-Subscribe network to simplify some of its aspects. Indeed, we are not interested in dealing with the problems related to peer discovery, message routing, and network management. Therefore, we let libP2P manage these problems natively through a GossipSub network. Peers subscribe to the topic representing the implemented protocol. The GossipSub network is only used to exchange messages of type *have*, namely the ones used to advertise owned or new blocks. We assume that each message of type *have* is broadcasted to all subscribers. On the other hand, the other messages are used in private, direct connections, through low level streams. Indeed, when a peer receives a *have* message containing the id of a block it does not own, it tries to directly connect to the peer that advertised the needed block. Once the private connection is established, the block can be exchanged using the other message types. A single *want* or *data* message can contain up to 16 blockchain blocks. The maintenance of the GossipSub network, the protocols used for implementing a secure and multiplexed connection, the DHT and the NAT traversal scheme, is automatically and seamlessly managed by libP2P. For the multiplexing, `yamux` and `mplex` are supported, while security is enforced through `noise`. NAT traversal is addressed via `circuit-relay`, with automatically detected nodes. The messages use an `UTF-8` encoding. The code of the `go` implementation is publicly available online[2].

5 EVALUATION

We propose an evaluation of `min-bitcoin`. In particular, we decided to observe several aspects related both to the library libP2P and the details of our implementation. All results are obtained from an average of multiple executions.

[2]Link removed for double-blind review process.

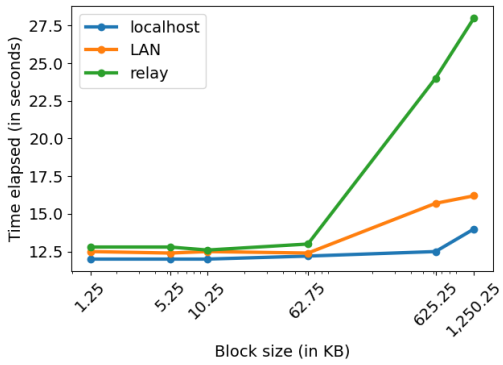

Fig. 2. Data transfer times when the two nodes are running on the same machine (localhost), two machines in the same local network (lan), two machines communicating via a relay node (relay).

Location	1.2 MB	2.5 MB
No VPN	320ms	580ms
New York, USA	400ms	710ms
Madrid, Spain	350ms	600ms
Sidney, Australia	390ms	615ms
Johannesburg, South Africa	365ms	610ms
Moscow, Russia	340ms	590ms
São Paulo, Brazil	355ms	610ms

Table 2. Transfer times with one peer behind VPN.

Table 1 reports the delays introduced by libP2P during several steps of the protocol. The largest portion of time is taken by the establishment of direct connections, which can take up to one and a half minute. As expected, this is a complex and very expensive step in the protocol, because it requires the two peers to find a suitable way of communicating with each other, including solving the NAT traversal, and negotiate the protocols to use, as explained in Section 3. Interestingly, establishing connections is very expensive even if the two peers are launched from the same machine, a clear hint that finding peers in a P2P network is quite challenging.

We evaluate some tests concerning the receiving of data during the execution of numerous instances of the min-bitcoin protocol. In particular, we simulated a peer that creates 100 blocks, and a second peer, started when the first finished creating all the blocks, that downloads the blocks. The results of our experiments are shown in Figure 2, where we report the time elapsed for the blockchain synchronization only, depending on the size of a single block. The Figure shows that when the two peers are in the same local network, the elapsed time does not increase linearly with the size of the block. In the case of a relayed connection we have similar results when the block size is small (up to few tens KB). However, for the largest block sizes tested the time needed to transfer all the data increases, although at a much slower pace. For instance, for a block size of 62.75KB the transfer time is almost 12.8 seconds, and when the block is ten times bigger, the transfer time is less than double. Since the size of the data does not seem to impact heavily the transfer time, we can ascribe this result to the difficulty of routing in general purpose P2P networks.

We decided to investigate the possible impact of using libP2P under a Virtual Private Network (VPN) and in different geographic locations. When a peer is inside a VPN, the only way for it to communicate with other peers is using a relay peer, therefore all the analyses in this Section are implicitly performed with relayed connections. The VPN networks are created using a service available online[3]. Table 2 shows the time elapsed in our experiments varying the location of a node and the amount of data transferred. Table 2 shows that using a VPN introduces only a small delay, which is mostly due to the geographical location of the VPN server. The results hint that libP2P can be used in scenarios in which the access to Internet is difficult or partially blocked.

[3]https://surfshark.com/

6 CONCLUSIONS AND FUTURE WORKS

In this paper we implemented `min-bitcoin`, a minimal version of the Bitcoin block exchange protocol. The implementation make use of the libP2P library, an innovative library to write P2P protocols and applications. The library is already used in some real world scenarios, such as the H2020 European project HELIOS[4] or the IPFS distributed file system. In this paper we tested the applicability of this library in the scenario of blockchain. Our implementation shows that libP2P make the development of P2P applications very easy, managing almost automatically most of the problems inherent to P2P networks, such as peer discovery, content routing, and NAT traversal. Although some setup steps, such as peer discovery, are quite time consuming, the library manages to obtain very high data transfer rates, even when one peer is in a VPN, or the connection is relayed. As future works, we plan to provide a complete implementation of Bitcoin over libP2P, and we plan to evaluate other scenarios, such as video games, text/audio/video messaging applications and so on. Additionally, we will investigate whether the library can be used under specific constraints, such as in remote locations where the strength of the connection signal is low and there are very strict battery consumption constraints.

ACKNOWLEDGMENTS

This work was partially funded by the European Commission under contract number H2020-825585 HELIOS.

REFERENCES

[1] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. 2007. OverSim: A flexible overlay network simulation framework. In *2007 IEEE global internet symposium*. IEEE, 79–84.

[2] Mansoor Ebrahim, Shujaat Khan, and Syed Sheraz Ul Hasan Mohani. 2014. Peer-to-peer network simulators: an analytical review. *CoRR* abs/1405.0400 (2014).

[3] Kevin Fall, Kannan Varadhan, et al. 2005. The ns Manual (formerly ns Notes and Documentation). *The VINT project* 47 (2005), 19–231.

[4] Barbara Guidi, Kristina G Kapanova, Kevin Koidl, Andrea Michienzi, and Laura Ricci. 2020. The contextual ego network p2p overlay for the next generation social networks. *Mobile Networks and Applications* 25, 3 (2020), 1062–1074.

[5] Barbara Guidi, Andrea Michienzi, Kevin Koidl, and Kristina Kapanova. 2019. A multilayer social overlay for new generation DOSNs. In *Proceedings of the 5th EAI international conference on smart objects and technologies for social good*. ACM, New York, NY, USA, 114–119.

[6] Barbara Guidi, Andrea Michienzi, and Laura Ricci. 2021. Data Persistence in Decentralized Social Applications: The IPFS approach. In *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 1–4.

[7] Barbara Guidi, Laura Ricci, Rolf Nyffenegger, and Robin Ribback. 2021. HELIOS CJ App: The decentralization of the Citizen Journalism. In *Proceedings of the Conference on Information Technology for Social Good*. ACM, New York, NY, USA, 31–36.

[8] Wu Kun, Dai Han, Yang Zhang, Sanglu Lu, Daoxu Chen, and Li Xie. 2005. Ndp2psim: A ns2-based platform for peer-to-peer network simulations. In *International Symposium on Parallel and Distributed Processing and Applications*. Springer, 520–529.

[9] Alberto Montresor and Márk Jelasity. 2009. PeerSim: A scalable P2P simulator. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE, 99–100.

[10] Yiannis Psaras and David Dias. 2020. The InterPlanetary File System and the Filecoin Network. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. 80–80.

[11] Jordi Pujol-Ahullo and Pedro Garcia-Lopez. 2009. Planetsim: An extensible simulation tool for peer-to-peer networks and services. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. 85–86.

[12] Andras Varga. 2010. OMNeT++. In *Modeling and tools for network simulation*. 35–59.

[4] https://helios-social.com/