POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

Stratum V2: the next generation protocol for Bitcoin pooled mining

Politecnico di Torino

Supervisors
Prof. Antonio J. Di Scala

Candidate
Gabriele Vernetti

Company Supervisor
Alps Blockchain
Francesco Sannicolo'

Academic Year 2022-2023

Acknowledgements

This work would not have been possible without the trust and support that Professor Antonio Di Scala has given me for the past two years. Thanks to his enthusiasm, I had the first opportunities to explore the fascinating world of Bitcoin mining, from a pragmatic point of view. Thanks to him, my entire master's degree has been enriched by other academic experiences that have helped me grow tremendously.

I would like to extend a sincere thanks to Alps Blockchain, especially to Francesco, who gave me the opportunity to work on the practical part of this thesis. But more importantly, since the day I learned about their activities, they have introduced me to a world of possibilities where Bitcoin mining can be applied in different real-life contexts. These guys are doing an amazing job, and I am incredibly grateful for the chance to collaborate with them.

A special gratitude goes to Giorgio and Salvatore. Together, we have motivated and inspired each other to create one of the most remarkable experiences that has defined my entire time at Politecnico: BitPolito. Thank you for believing in the mission of our student team to an extent that I could have never imagined before.

Thanks to the entire Trust In Food team. Thank you for being with me during my first-ever startup experience. Without all of you and our shared journey, I wouldn't have developed certain thoughts that help me evaluate every step of my life.

Thanks to all the members of The Pub. Without all of you, I couldn't say that I have lived wonderful years of my life.

Thanks to Push, Jaco, and Gabri for making me rejoice in every day spent in Turin.

Thanks to my entire family, who has always understood me and supported every decision I have made in life.

Thanks to my Irene, who has revolutionized my life and for the past four years has been my driving force to become a constantly improving man.

Abstract

Bitcoin is a distributed, «peer-to-peer electronic cash system» [1].
Since the publication of its white paper, on 31st of October 2008, Bitcoin has experienced a very rapid evolution. Many aspects of the protocol has changed in the years, permitting it to keep up with the exponential growth in terms of users, services and companies involved into it.
The primary focus of the entire research centers on one of the fundamental pillars of the entire ecosystem: Bitcoin mining. More specifically, the thesis investigates the details of the approach that has characterized mining operations since Bitcoin's early history: pooled mining.
The introductory chapters of this thesis provide a comprehensive understanding of Bitcoin, its network composition, and the concept of Proof of Work. The subsequent chapters focus on mining, exploring both solo and pooled mining approaches, as well as the history and evolution of mining operations.
During its growth, many protocols have been developed to manage the pooled mining operations, such as Getwork, Getblocktemplate (GBT), and Stratum (V1).
However, the central focus of the thesis is Stratum V2: its inception, inner details, and the significant differences it brings compared to its predecessor, Stratum (V1). Concepts like protocol security, binary framing, and the power of transactions selection are deeply discussed, to provide all the detailed context necessary to understand the importance of this protocol update. The current implementations of Stratum V2 are also discussed to provide a practical perspective on its adoption.
In addition to this, the thesis introduces the Stratum Reference Implementation (SRI), a key tool for understanding and experimenting with Stratum V2. Detailed explanations are provided on how SRI works, how to get started with it, and potential future directions for research and development.
The research concludes with a reflection on the advancements made in Stratum V2 and their implications for the future of Bitcoin pooled mining. The concept of SV2 protocol benchmarking is introduced, along with the exploration of non-custodial pools as a promising pathway for future innovation. Overall, this thesis aims to dig into the profound significance of Stratum V2 as a transformative protocol in the domain of Bitcoin pooled mining, and its potential to shape the landscape of mining practices in the years to come.

Contents

List of Figures 4

1 Introduction 7

2 Bitcoin 9
- 2.1 What is Bitcoin . . . 9
- 2.2 How the network is composed . . . 13
  - 2.2.1 Bitcoin network nodes: types and roles . . . 13
  - 2.2.2 Extended Bitcoin network . . . 14
  - 2.2.3 Geographical distribution and statistics . . . 15
- 2.3 Proof of Work . . . 18

3 Mining 21
- 3.1 How mining works . . . 22
- 3.2 Mining history and evolution . . . 25
- 3.3 Solo mining and Pooled mining . . . 27

4 History of pooled mining protocols 29
- 4.1 Getwork . . . 29
  - 4.1.1 What is Getwork: why and when it was born . . . 29
  - 4.1.2 How Getwork works . . . 31
  - 4.1.3 Getwork usage into pooled mining and protocol extensions . . . 35
  - 4.1.4 Why Getwork usage ended up . . . 37
- 4.2 Getblocktemplate (GBT) . . . 38
  - 4.2.1 What is GBT: why and when it was born . . . 38
  - 4.2.2 How GBT works . . . 39
  - 4.2.3 Why GBT usage ended up . . . 44
- 4.3 Stratum (V1) . . . 45
  - 4.3.1 What is Stratum (V1): why and when it was born . . . 45
  - 4.3.2 How Stratum (V1) works . . . 46
  - 4.3.3 Stratum (V1) vulnerabilities and security issues . . . 53
  - 4.3.4 Why Stratum (V1) needs to be updated . . . 58

5 Stratum V2 61
5.1 What is SV2: why and when it was born 61
5.2 How SV2 works 64
5.3 Differences between SV1 and SV2 68
5.4 Current implementations 72

6 Stratum Reference Implementation (SRI) 75
6.1 How SRI works 75
6.2 Getting Started 79
6.2.1 Testing SRI configurations 79
6.3 Final thoughts and future ideas 87
6.3.1 SRI Pool fallback 87
6.3.2 Non-custodial pools 88
6.3.3 SRI benchmarking suite 88

7 Conclusion 91

Bibliography 93

List of Figures

2.1 Bitcoin supply and subsidy, *created by Coindesk Research* 11
2.2 Bitcoin prehistory, *created by @danheld* 11
2.3 Bitcoin Genesis Block, *in HEX and ASCII encoding* 12
2.4 Times headline, *Jan 3, 2009* 12
2.5 Functionalities that a node participating in the Bitcoin network can have. 14
2.6 Umbrel implementation of a Bitcoin full node 15
2.7 Antminer S19XP Hydro, one of the most powerful ASIC machines for Bitcoin mining 15
2.8 Map of the "Extended Bitcoin Network." 16
2.9 Screenshot of the live map on Bitnodes.io. 17
2.10 Price, hashrate and difficulty representation, *insights.braiins.com* . . 19

3.1 SHA-256 hash function in action 22
3.2 Bitcoin block and block header fields 23
3.3 Flow chart of mining activity 24
3.4 Bitcoin mining hardware evolution, in relation to the network difficulty 25
3.5 Antminer S19 model series, with relative efficiency (J/TH) 26
3.6 Mining pool share difficulty and network difficulty [2] 28

4.1 Satoshi *getwork* update announcement, Bitcoin Talk Forum 29
4.2 Bitcoin block header fields, *from [3]* 30
4.3 Example Bitcoin block header, in hexadecimal format 31
4.4 Getwork method details and parameters 31
4.5 Luke-Jr mining operations recommendations, on *Bitcoin Talk Forum* 37
4.6 Luke-Jr *getblocktemplate* announcement, Bitcoin Talk Forum 38
4.7 Description of the initial block template request parameters, using GBT 39
4.8 Mutations which can be asked from a miner, using GBT [4] 40
4.9 Description of the template fields sent by the mining pool, using GBT 41
4.10 Transaction fields, present in the template sent by pool server, using GBT 41

4.11 Slush *stratum* announcement, Bitcoin Talk Forum 45
4.12 Real communication between miner and pool server 52
4.13 Coinbase tx details 53
4.14 Preconditions of attack, *Blackhat Asia 2021* 54
4.15 Job injection based on set_extranonce, *Blackhat Asia 2021* 55
4.16 Time segment attack, *Blackhat Asia 2021* 56

5.1 Pavel Moravec, Jan Čapek and Matt Corallo, co-authors of SV2 specs 61
5.2 Typical Division of Downstream and Upstream Roles, *Galaxy Digital Research* [5] 66
5.3 SV2 protocol binary framing 67
5.4 Transaction selection decentralization brought by Stratum V2 [6] . . 68
5.5 First SV2 implementation released by Braiins team, in 2020 72
5.6 SRI homepage on *stratumprotocol.org* 73

6.1 SRI configuration A 77
6.2 SRI configuration B 77
6.3 SRI configuration C 78
6.4 SRI configuration D 78
6.5 SV2 Pool connected to the hosted Template Provider, *config C* . . . 80
6.6 Translator Proxy connected to the SV2 Pool, *config C* 81
6.7 SV1 CPU-miner running, using the Translation Proxy, *config C* . . 82
6.8 Pool settings of a Antminer S19J Pro 83
6.9 Translation Proxy logs successfully the ASIC miner SV1 requests . 83
6.10 SV2 Pool connected to the local Template Provider, *config D* 84
6.11 Translator Proxy which acts as a Job Negotiator, *config D* 85

Chapter 1

Introduction

The emergence of Bitcoin in 2008 introduced a unique and innovative concept of decentralized digital currency. Bitcoin, as described in its white paper published on October 31st, 2008, proposed a "peer-to-peer electronic cash system" [1]. Since then, Bitcoin has experienced remarkable growth, attracting a significant number of users, businesses, and services into its ecosystem. As the Bitcoin network expanded, mining became a crucial component for securing and maintaining the system's integrity.

To fully realize the importance of mining in the Bitcoin ecosystem, it's important to get a comprehensive understanding of Bitcoin itself. Chapter 2 provides an overview of Bitcoin, its underlying technology, and the composition of its network. The concept of Bitcoin network nodes, their types, and roles in supporting the decentralized nature of the system are explored. Bitcoin's security is based on a consensus mechanism known as Proof of Work (PoW). Chapter 2 dives into the concept of PoW and its role in the mining process. PoW ensures that miners invest computational power to solve a mathematical problem, validating transactions and adding new blocks to the blockchain. Understanding PoW is crucial for understanding the motivations behind mining and its evolution over time.

Chapter 3 focuses on the mining process, exploring its mechanics, historical development, and evolution. The workings of mining, the computational challenges involved, and the incentives for miners to participate in the network are explained. The transition from solo mining to pooled mining, which brought significant changes to the mining landscape, is highlighted. Pooled mining enabled miners to collaborate and share resources, improving their chances of receiving rewards in a more predictable way.

Chapter 4 provides a deep exploration of the history of pooled mining protocols. The initial protocol, Getwork, its functionalities, and limitations are analyzed. After that, the introduction of Getblocktemplate (GBT) and its improvements over Getwork protocol are discussed. However, the central focus of this chapter is on Stratum (V1), which revolutionized pooled mining by introducing a more efficient

and simpler protocol, becoming the standard "de facto" of the pooled mining activities. Then, the vulnerabilities and security issues associated with Stratum (V1) and the need for its subsequent update are discussed.
Chapter 5 dives into the core topic of the entire thesis, Stratum V2, a significant upgrade to the Stratum protocol. The motivations behind the development of Stratum V2 and its inner workings are explored. The differences between Stratum V1 and Stratum V2 are highlighted, particularly the enhanced security, transaction selection responsibility, and other improvements introduced by the new protocol. Moreover, the current implementations of Stratum V2 and its adoption path within the mining community are examined.
To facilitate a practical understanding and experimentation with Stratum V2, Chapter 6 introduces the Stratum Reference Implementation (SRI). The workings of SRI, guidance on getting started with it, and potential future directions for research and development are explained. Concepts such as SRI Pool fallback, non-custodial pools, and the importance of a benchmarking suite for Stratum V2 are explored.
The research presented in this thesis aims to shed light on the profound significance of Stratum V2 as a transformative protocol within the domain of Bitcoin pooled mining. By examining the history, evolution, and current implementations of mining protocols, a comprehensive understanding of the advancements made and their implications for the future is provided.

The aim of this exploration is to make a meaningful contribution to the continuous development and advancement of Bitcoin mining methods, consequently influencing the future direction of this essential aspect of the Bitcoin ecosystem.

Chapter 2

Bitcoin

2.1 What is Bitcoin

«The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust. Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve.» [7]

Beginning with the words of its creator, the pseudonym Satoshi Nakamoto, Bitcoin was born out of a desire to create a decentralized, secure, and accessible form of currency that does not rely on any form of trust in centralized institutions. Bitcoin provides an alternative to conventional currencies and aims to offer greater financial sovereignty to individuals while addressing some of the perceived weaknesses of the existing financial system. Besides that, Bitcoin is a digital currency ecosystem which incorporates many concepts and technologies discovered in the last 50 years of deep research and development around Cryptography, Internet protocol, and Peer-to-Peer networks. It operates through the use of units of currency called "bitcoin" (with a lowercase 'b'), which are used to store and transfer value among participants within the Bitcoin network.
Bitcoin protocol enables users to transfer bitcoin across the network, facilitating activities such as buying and selling goods, or sending money to individuals and organizations. Specialized currency exchanges allow for the purchase, sale, and exchange of bitcoin with other fiat currencies. Bitcoin's characteristics make it the perfect type of money for Internet as it offers speed, security, and borderless transactions. Since bitcoin are a form of virtual currencies, the coins are represented in transactions that transfer value from the sender to the recipient. Bitcoin users possess **private keys** that prove ownership of bitcoin within the Bitcoin network. These keys enable them to sign transactions, unlocking the value and transferring

it to a new owner. As already said, Bitcoin is as a **distributed** peer-to-peer system, who lives without the need for a central server or entity. The creation of new bitcoins is managed through a process called **mining**. Miners compete to solve mathematical problems while processing Bitcoin transactions. Approximately every 10 minutes, a miner successfully confirms the transactions received from its own node, mining them into a new block, and is rewarded with newly minted bitcoin, called block subsidy. Basically, when miners solve the **Proof-of-Work** algorithm on the block they are working on, they immediately communicate this new valid block of transactions to all the other Bitcoin nodes which are connected to. At this point, these latter verify the validity of the mined block, and simply add it to their local copy of the ledger shared by every node of the network, the so-called blockchain.
The Bitcoin protocol incorporates built-in algorithms that regulate the mining process across the network, such as the **difficulty adjustment**. The difficulty of the mining activity is dynamically adjusted to ensure that, on average, a miner succeeds every 10 minutes, regardless of the number of miners competing at any given moment. Additionally, the protocol halves the rate at which new bitcoin is created every 4 years (more precisely every 210.000 blocks), and there is a fixed maximum limit of just under **21 million** coins. Every coin is composed by 10^8 **satoshi**, which is the unit of account of the Bitcoin network. By protocol rules, there will be 32 **halvings**, one every 210.000 blocks. In 2009, the system rewarded successful miners with 50 bitcoin every 10 minutes. The entire emission of bitcoin units can be represented by the following formula:

$$\sum_{i=0}^{32} 210.000 \cdot \frac{50}{2^i}$$

Consequently, the number of bitcoins in circulation follows a predictable curve that approaches 21 million by the year 2140.

As greatly represented in Figure 2.2, «Bitcoin represents the culmination of decades of research in cryptography and distributed systems and includes four key innovations brought together in a unique and powerful combination. Bitcoin consists of:

- A decentralized peer-to-peer network (the Bitcoin protocol)
- A public transaction ledger (the blockchain)
- A set of rules for independent transaction validation and currency issuance (consensus rules)
- A mechanism for reaching global decentralized consensus on the valid blockchain (Proof-of-Work algorithm)» [8]

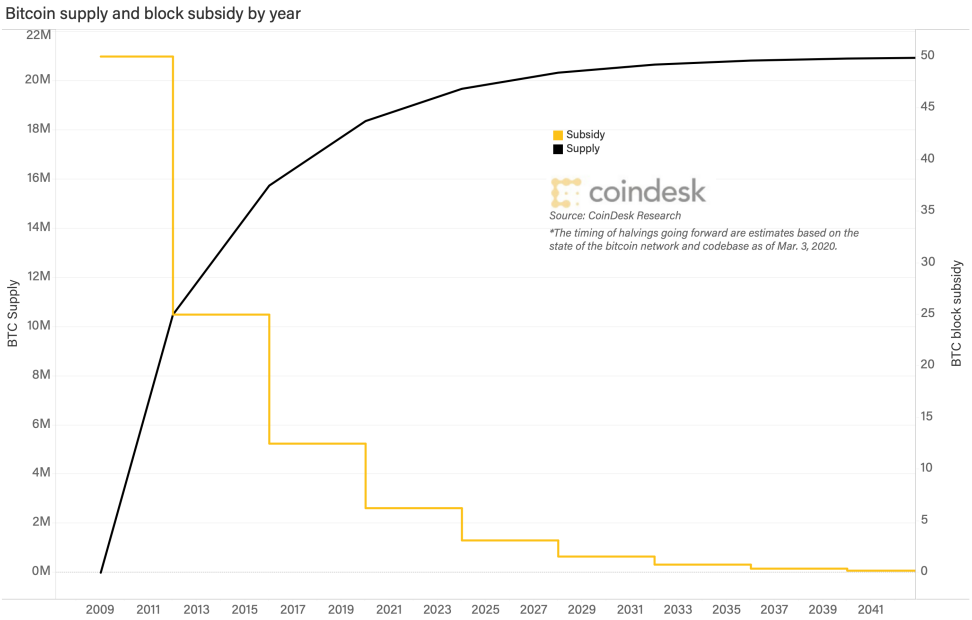


Figure 2.1: Bitcoin supply and subsidy, *created by Coindesk Research*

In fact, during the late 1980s, with the increasing understanding and availability of cryptography, researchers started to experiment on utilizing cryptography to construct digital currencies. [9] Valuable examples can be: Ecash (1982, David

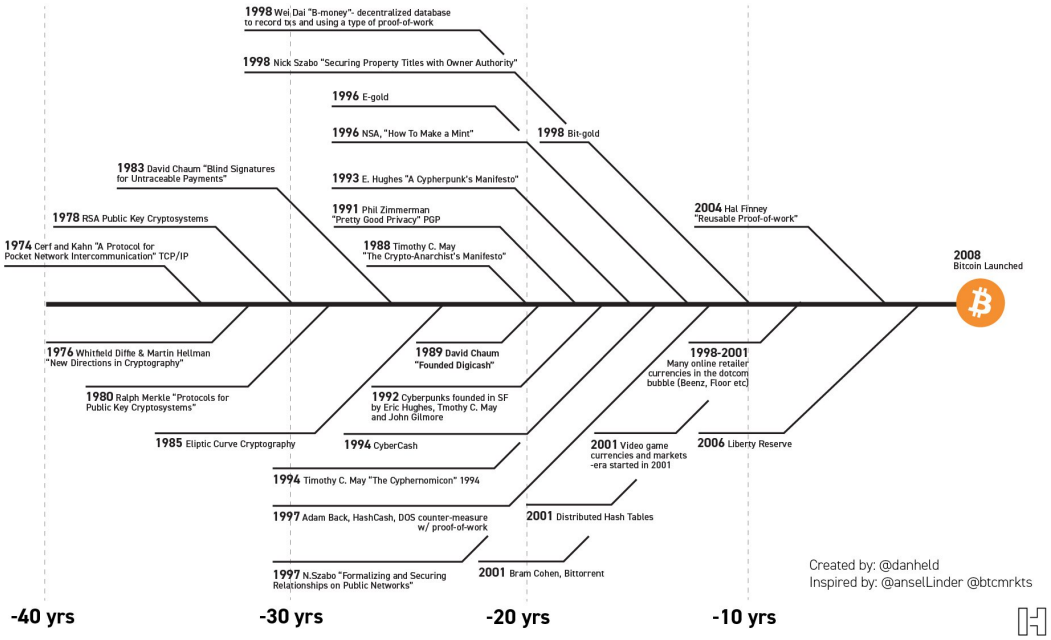


Figure 2.2: Bitcoin prehistory, *created by @danheld*

Chaum), E-gold (1996, Douglas Jackson and Barry Downey), Bit gold (1998, Nick Szabo), B-money (1998, Wei Dai), Reusable Proof of Work (2004, Hal Finney). Even if these initial digital currencies worked effectively, all of them possessed a centralized nature, and for this reason they were susceptible to attacks by governments and malicious hackers.

Finally, on **3rd of January 2009**, Satoshi Nakamoto created the first block in the Bitcoin blockchain. Since it's the official starting point of the entire Bitcoin history, it's called **Genesis Block**.
In this block, more precisely in the coinbase transaction data, Satoshi left a message which represents the Bitcoin's raison d'être:

```
"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"
```

This message is a reference to the Times headline of 3rd of January 2009, which proves that the Bitcoin Genesis Block could not have been created before that date. More importantly, the message is a clear statement of the entire Bitcoin movement. It declares the desire to fight the central bank policies, which are characterized by a culture of easy money. Bitcoin, on the other hand, aims to bring back individual responsibility through a monetary system based on sound money. Bitcoin aims to be money which can't be devalued or controlled to benefit a lucky few.

```
00000000  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 3B A3 ED FD  7A 7B 12 B2 7A C7 2C 3E  ....;£íýz{.²zÇ,>
00000030  67 76 8F 61 7F C8 1B C3  88 8A 51 32 3A 9F B8 AA  gv.a.È.Ã^ŠQ2:Ÿ¸ª
00000040  4B 1E 5E 4A 29 AB 5F 49  FF FF 00 1D 1D AC 2B 7C  K.^J)«_Iÿÿ...¬+|
00000050  01 01 00 00 00 01 00 00  00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 FF FF  FF FF 4D 04 FF FF 00 1D  ......ÿÿÿÿM.ÿÿ..
00000080  01 04 45 54 68 65 20 54  69 6D 65 73 20 30 33 2F  ..EThe Times 03/
00000090  4A 61 6E 2F 32 30 30 39  20 43 68 61 6E 63 65 6C  Jan/2009 Chancel
000000A0  6C 6F 72 20 6F 6E 20 62  72 69 6E 6B 20 6F 66 20  lor on brink of
000000B0  73 65 63 6F 6E 64 20 62  61 69 6C 6F 75 74 20 66  second bailout f
000000C0  6F 72 20 62 61 6E 6B 73  FF FF FF FF 01 00 F2 05  or banksÿÿÿÿ..ò.
000000D0  2A 01 00 00 00 43 41 04  67 8A FD B0 FE 55 48 27  *....CA.gŠý°þUH'
000000E0  19 67 F1 A6 71 30 B7 10  5C D6 A8 28 E0 39 09 A6  .gñ¦q0·.\Ö¨(à9.¦
000000F0  79 62 E0 EA 1F 61 DE B6  49 F6 BC 3F 4C EF 38 C4  ybàê.aÞ¶Iö¼?LÏ8Ä
00000100  F3 55 04 E5 1E C1 12 DE  5C 38 4D F7 BA 0B 8D 57  óU.å.Á.Þ\8M÷º..W
00000110  8A 4C 70 2B 6B F1 1D 5F  AC 00 00 00 00           ŠLp+kñ._¬....
```

Figure 2.3: Bitcoin Genesis Block, *in HEX and ASCII encoding*


THE TIMES

Saturday January 3 2009 timesonline.co.uk No 69523 £1.50

Eat Out from £5

More than 900 great restaurants, including four Gordon Ramsay favourites from £15

Chancellor on brink of second bailout for banks

Billions may be needed as lending squeeze tightens

Salman Rushdie I won't marry again

Giant killing? Guide to the FA Cup third round

99p

Figure 2.4: Times headline, *Jan 3, 2009*

2.2 How the network is composed

The architecture of the network that allows the existence of Bitcoin is of type *Peer-to-Peer* (P2P).
This means that each node participating in the network is equal to the others and collaborates with the others by providing the services and functionalities defined by the shared protocol, which is executed by all nodes in a similar manner. Being the antithesis of the most well-known and widely used architecture in the current context, defined as *Client/Server* (C/S), the Bitcoin network does not require centralized servers to provide any type of service. On the contrary, each node constitutes a point in the network, attributing decentralization and resilience to the protocol that only *Peer-to-Peer* networks can guarantee. Examples of P2P networks that have been very successful are those used for file-sharing, initially popularized by *Napster* in 1999, and later evolved into better protocols, such as *BitTorrent*.

Similarly to the aforementioned implementations, Bitcoin was designed to be a *Peer-to-Peer* system for the management and use of digital money, capable of functioning without the intervention of central authorities, with the specific goal of being an uncensorable and resilient tool against centralized control, distributed among the nodes that want to participate in the network. Due to these characteristics, the chosen design for Bitcoin could only be a P2P network.

2.2.1 Bitcoin network nodes: types and roles

The functionalities that nodes can perform to be part of the network are those defined by the Bitcoin protocol, and they can be identified in Figure 2.5. Based on the functionalities that each node decides to implement, several categories of nodes can be outlined:

- Full Node
- Simplified Payment Verification Node (SPV)
- Miner Node

A *full node* is a node that is responsible for maintaining a complete copy of the Bitcoin blockchain, keeping it up to date through constant communication with other nodes in the network. By having a local copy, it can verify every new transaction or block that arrives without depending on third-party nodes. At the time of writing, the technical requirements to implement a full node remain quite accessible, both in terms of hardware (2GB of RAM, 500GB of free disk space) and technical know-how, thanks to the abundance of online guides and "plug-and-play" style implementations (Figure 2.6) developed over the years. It is

worth emphasizing the importance, for true decentralization, of having the lowest possible technological barrier: the higher the number of full nodes, the more distributed the network is, resulting in increased overall robustness and resilience. A *Simplified Payment Verification (SPV) node*, on the other hand, contains a copy of all block headers but not the transactions and other data related to them. It verifies the validity of the headers of new blocks and is also able to verify transactions of interest with the help of a connected full node. For this reason, nodes of this type are also called lightweight nodes. The main advantage of this category of nodes is their lower use of hardware resources (disk space) compared to full nodes, making them more suitable for contexts with stricter resource limitations (e.g., smartphones). The main disadvantage of SPV nodes is their dependence on other full nodes when they require a specific series of blocks that they do not have locally. During this process, a *Sybil attack* could occur, compromising the privacy of an SPV client. To overcome and mitigate this risk, *bloom filters* have been introduced. [10] A *Miner node*, unlike the previous types, has the specific purpose of solving the *Proof-of-Work (POW)* algorithm in order to claim the reward for a new block. It uses increasingly specialized hardware, such as Application Specific Integrated Circuits (ASICs), which have been the norm for several years now (see Figure 2.7). Once a miner solves the block they are working on, they communicate the solution to the rest of the network through their full node or by transmitting the new block to a connected node. At that point, transactions contained in that mined block are considered confirmed, since they are included in a block and subsequent blocks will be added on top, forming a chain of confirmations. The number of confirmations increases the level of trust in the transactions.

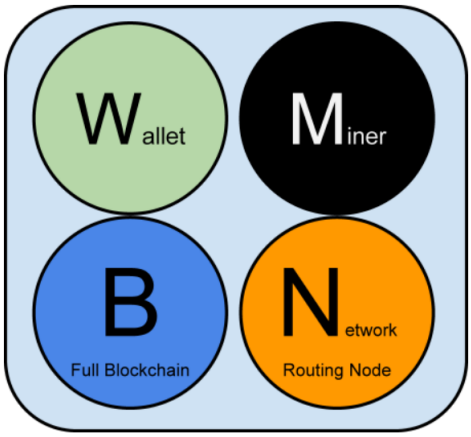


Figure 2.5: Functionalities that a node participating in the Bitcoin network can have.

2.2.2 Extended Bitcoin network

The composition of the network described above represents the basic functioning of the Bitcoin P2P protocol. Actually, Bitcoin network is further enriched by other nodes that execute specific protocols (e.g. Stratum, FIBRE). An overview of the so-called *Extended Bitcoin Network* can be better understood through Figure 2.8. Analyzing the map, three protocols (highlighted in different colors) can be observed,

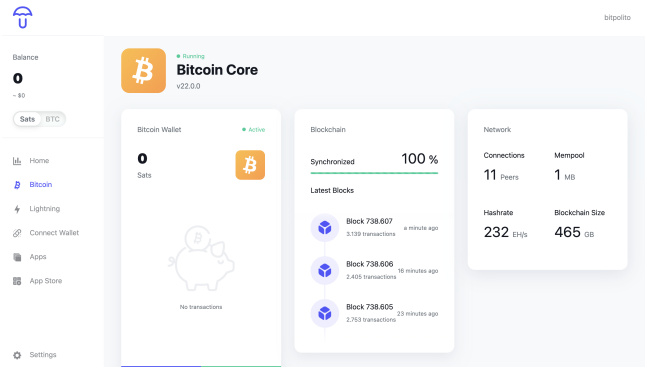


Figure 2.6: Umbrel implementation of a Bitcoin full node

Figure 2.7: Antminer S19XP Hydro, one of the most powerful ASIC machines for Bitcoin mining

which are fully compatible with each other and allow various interactions between different types of nodes, depending on the functionalities each node decides to implement and make available to the extended network. For example, a portion of the network is strongly connected to a *Pool Server*, which consists of a series of nodes responsible for mining operations by aggregating their hash power in a mining pool. This is done to reduce the inherent variance associated with Proof-of-Work. The pool server manages communication and interaction through its own protocol specifically developed for these mining operations. The opposite of pooled mining is represented by nodes called *Solo Miners*. They do not rely on any mining pool for their mining operations and require a local copy of the blockchain (indicated by the blue dot within the nodes) to independently verify new transactions before starting the search for the *nonce*. All the differences between Solo Mining and Pooled Mining will be deeply discussed and analyzed in Chapter 3. The protocol represented by the red connections is called the *Stratum Protocol*. It was developed around 2012 as an alternative to the previous pooled mining model. The goal of Stratum is to define a kind of standard for managing communications between mining pool servers and miners. This pooled mining protocol will be deeply discussed and analyzed in section 4.3.

2.2.3 Geographical distribution and statistics

Using certain functions defined within the Bitcoin protocol, implemented and made available by most nodes, it is possible to perform a recursive search of the connections present between nodes and obtain some statistics related to the global geographical distribution of the Bitcoin network. It is important to note that many popular implementations for creating a personal full node in a more user-friendly

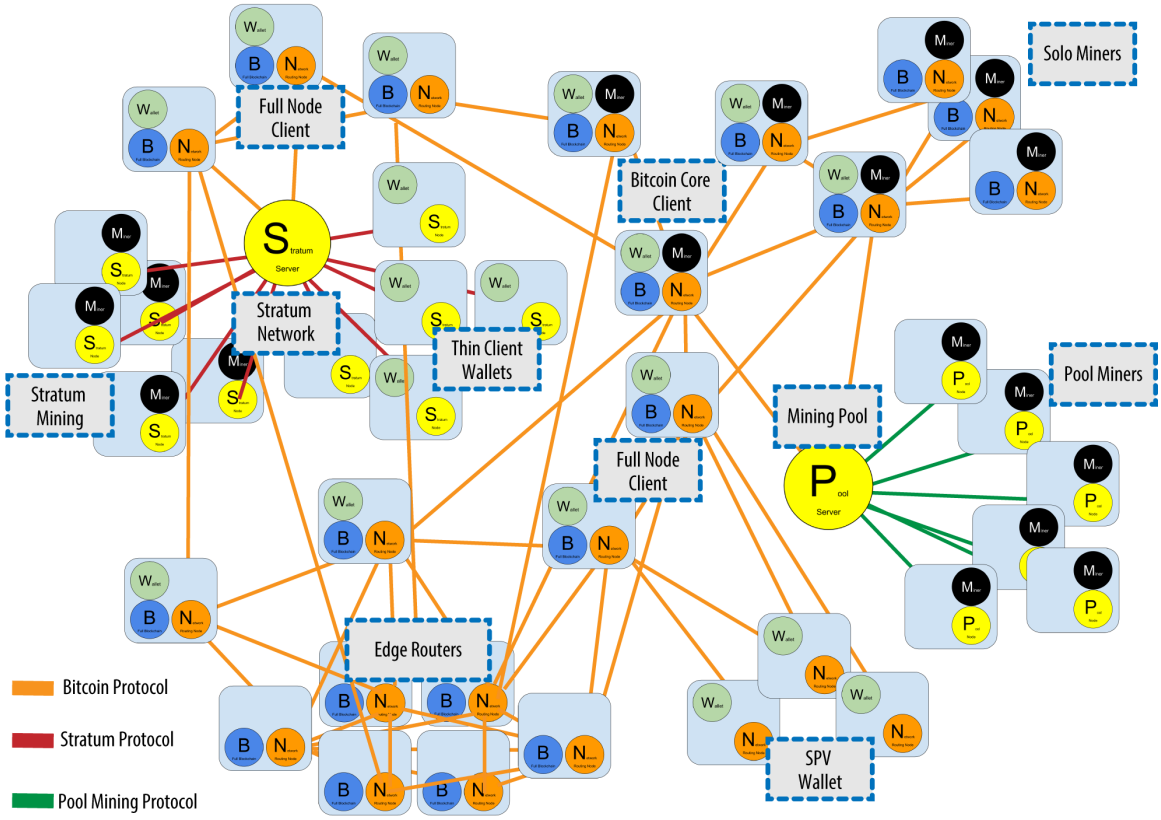


Figure 2.8: Map of the "Extended Bitcoin Network."

way (such as Umbrel, RaspiBlitz, MyNode, or RoninDojo) operate through the Tor network. In these cases, the IP address detected by Bitnodes does not geolocate the actual physical location of the node but rather the Tor exit node to which it is connected.

Based on the data collected from Bitnodes.io, at the time of writing, the entire Bitcoin network consists of approximately 17,000 reachable nodes. The term "reachable" refers to nodes that accept new incoming connections, providing access to new nodes and allowing them to download the entire blockchain starting from the genesis block. On the same website, it is possible to find the estimate of the total number of nodes (both reachable and unreachable), which currently exceeds 44,000 nodes. [11]

In addition to statistics on countries and cities worldwide, another interesting data point is the percentage of nodes that are updated to the latest version of the Bitcoin

protocol (25.0.0), which is approximately 23%. Regarding the data on Autonomous System Numbers (ASNs) identified by Bitnodes.io's crawlers, it can be observed that around 60% of nodes communicate through the Tor network, while only 2.3% are hosted on AWS servers.

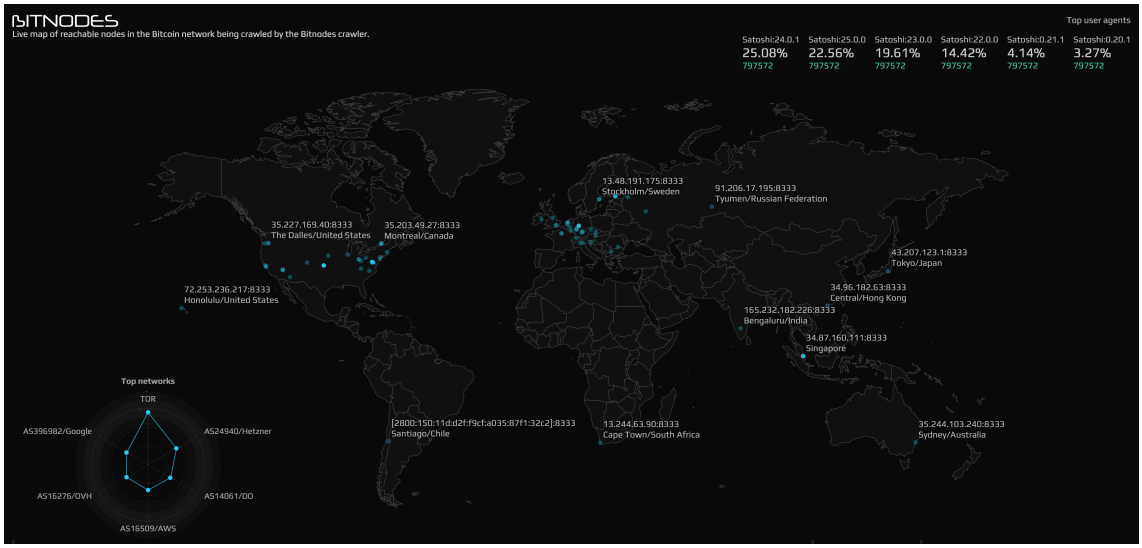


Figure 2.9: Screenshot of the live map on Bitnodes.io.

2.3 Proof of Work

As mentioned in the previous introductory section, Proof-of-Work (PoW) is defined as the **consensus algorithm** of the entire Bitcoin network.

However, the term "Proof of Work" was born even before Bitcoin began, since it was introduced by the cryptographer Adam Back, in his work on the so-called **Hashcash** system: «Hashcash was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers in May 1997.» [12] The basic idea behind Hashcash was to require a computational effort from the sender of a message, making it costly and time-consuming to send a large number of messages or launch coordinated attacks. In order to send an email to a Hashcash user, the sender would have to find a hash of the email that fell within a certain range. Since a hash is a large, unpredictable number, producing such a hash takes many guesses. Once the valid hash was found, the sender could include it in the message header. In this way, the recipient was able to immediately verify the validity of the result contained in the message header. In that case, the Proof-of-Work system was intended as a method to combat email spam and denial-of-service attacks. Satoshi Nakamoto, while designing the Bitcoin protocol, most likely took inspiration from it: for this reason Hashcash is also considered a precursor to the Bitcoin consensus algorithm.
Moreover, with his invention, the creator of Bitcoin was able to solve with a practical solution a very known problem in distributed computing, defined as the **Byzantine Generals' Problem**. This concept can be understood by reading the first lines of the original paper which described and gave birth to the term itself: «Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement.» [13]
Bitcoin found definitely a solution to this, by introducing the concept of Proof of Work as the foundational problem to solve during mining operations. In this way it also found a way to ensure that users could agree on a single state of the ledger, at the same time avoiding any **double spending** attempts or other invalid transactions. To summarize, PoW in Bitcoin definitely enabled consensus without relying on a central trusted authority.

The other innovative concept introduced in Bitcoin, is the above-mentioned mechanism called **difficulty adjustment**. It's a fundamental piece of the entire protocol, since it allows the network to automatically regulate the level of difficulty associated

with mining a block. The difficulty adjustment is based on the mining speed of participants in the network. By dynamically adjusting the mining difficulty, increasing or decreasing the range of valid hash results, Bitcoin ensures that new coins are produced at a predetermined rate, one block every 10 minutes (on average), regardless of the total computing power dedicated to mining. This automatic adjustment mechanism is done every 2016 blocks mined (about 2 weeks), and it contributes to the network's resilience and scalability, preventing issues like hyperinflation or compromised security. From an economic perspective, the difficulty adjustment is a unique mechanism which provides a never discovered property for something who can aspire to be used as money. As perfectly stated by the economist Saifedean Ammous, in his great work "The Bitcoin Standard": «For anything to function as a good store of value, it has to beat this trap: it has to appreciate when people demand it as a store of value, but its producers have to be constrained from inflating the supply significantly enough to bring the price down.» [14]
This is a very unique discovery, since every form of money before Bitcoin has always followed the following rule: the more the good used as money appreciated, because of a demand growth for it, the more it has been produced. In this case, very uniquely, the demand for the good used as money (bitcoin) will never be able to modify (by increasing or decreasing) the offer of it. This property guarantees that Bitcoin supply in almost perfectly inelastic, since the production rate on newly minted bitcoin is totally determined by the difficulty adjustment mechanism.

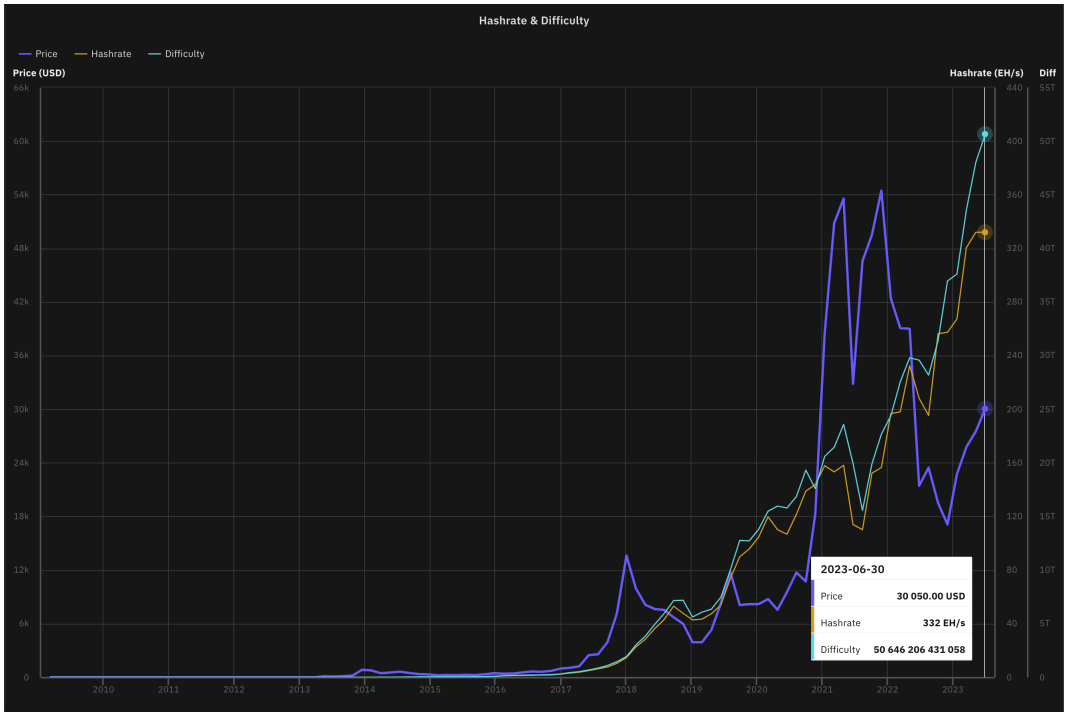


Figure 2.10: Price, hashrate and difficulty representation, *insights.braiins.com*

Furthermore, Bitcoin introduced the notion of blockchain **immutability**, ensuring the integrity of past transactions. Each new block is added to the end of the existing blockchain, forming an unbreakable chain of transactions. Modifying a past transaction would require rewriting the entire block containing the transaction and all subsequent blocks. Additionally, the attacker would need to generate new blocks at a faster rate than the entire Bitcoin network combined in order to catch up and create a longer chain, which is practically infeasible.

In summary, the combination of PoW, difficulty adjustment, which regulates the rate of coin emission, and the immutability of the blockchain, which fortifies past transactions, contribute to the decentralized and secure nature of the Bitcoin network. These features allow Bitcoin to operate autonomously and efficiently without relying on a central authority.

Chapter 3

Mining

In the previous chapter was introduced the concept of Proof of Work, with a particular focus on the previous systems which gave inspiration to Satoshi Nakamoto's invention. In addiction, it was analyzed the problems that Proof-of-Work solves, such as double spending and consensus in an asynchronous distributed network. The scope of this chapter is to dive more deeply into the Bitcoin mining theme, exploring how it actually works, the evolution in term of hardware used for the operations, and the differences between Solo Mining and Pooled Mining.

To better understand the basics of Bitcoin mining, since it's totally based on solving the above-described PoW (2.3), a little reminder about hash functions is needed. A **hash function** is a mathematical function that takes an input (of any size) and produces a fixed-size string of characters, which is typically a sequence of alphanumeric characters. The output generated by a hash function is called "digest," or simply "hash". Hash functions are used in computer science and cryptography for many purposes, since the characteristic features of a hash function are:

- Deterministic: for the same input, a hash function will always produce the same output. This property allows for consistent, verifiable and repeatable hashing.
- Fixed-size output: a hash function generates a hash value of a fixed length, regardless of the size of the input. For example, a hash function might always produce a 256-bit hash value.
- One-way (pre-image resistance): given a hash value, it is computationally infeasible to determine the original input. Basically, it should be extremely difficult to reverse-engineer the input given its hash value.
- Collision resistance: it should be highly improbable that two different inputs produce the same hash value. While collisions are theoretically possible a good hash function minimizes the chances of collisions.

Figure 3.1: SHA-256 hash function in action

In the case of Bitcoin, the hash function used is the SHA-256, as illustrated in Figure 3.1.

3.1 How mining works

Bitcoin mining, as already anticipated, is a mechanism which constitutes a foundational piece of the entire protocol. To put it simply, «mining is a lottery to create new blocks in the Bitcoin blockchain. There are two main purposes for mining:

1. To Permanently add transactions to the blockchain without the permission of any entity.

2. To fairly distribute the 21 million bitcoin supply by rewarding new coins to miners who spend real world resources (electricity) to secure the network.» [2]

In order to mine a valid block, miners need to find the pre-image which corresponds to a *hash* (or *digest*) which satisfies the current Bitcoin network difficulty. As described earlier, the difficulty adjustment mechanism regulates the mining difficulty for the entire Bitcoin network, every 2016 blocks. From a practical point of view, what it does is to simply increase or decrease a value which is present into Bitcoin block headers, which is called ***target*** (or *nBits*). The *target* represents the current maximum value that a hash found by miners has to be, in order to be considered valid for the entire network. From another point of view, it represents the size of the set of valid solutions for the current PoW algorithm.
Of course, the object which miners utilize as input for the SHA-256 hash function, is a block fulfilled with transactions received from their Bitcoin full-nodes. More specifically, the input is constituted by the **block header** which is derived from the block template built by miners, along with a random value called **nonce**. The *nonce* is constantly changed until a valid hash value is found.
As represented in 3.2, miners build the so-called *candidate block*, which contains all the best transactions received from their full-nodes (which typically are the transactions that pays more fees to be firstly included in a new block). Once they

got the transactions, they need to complete the *candidate block* by building the corresponding block header, which includes the following fields:

- **Version**: a 4 bytes field which indicates the version of the Bitcoin protocol being used. It allows for future upgrades and compatibility with different protocol enhancements.
- **Previous Block Hash**: a 32 bytes field which contains the hash value of the previous block in the blockchain. It links the current block to the previous one, forming a chain of blocks.
- **Merkle root**: a 32 bytes hash computed by combining the transaction hashes of all the transactions included in a particular block. These transaction hashes are then paired and hashed together until a single hash is obtained.
- **Difficulty target**: a 4 bytes field that represents the current network difficulty level, which, as already explained, indicates the maximum allowed value for the block hash to be considered valid.
- **Timestamp**: a 4 bytes field that records the approximate time when the block was mined.
- **Nonce**: a 32-bit value that miners adjust in their attempts to find a valid block hash. Miners repeatedly change the nonce, double hash the block header, and check if the resulting hash meets the difficulty requirement.

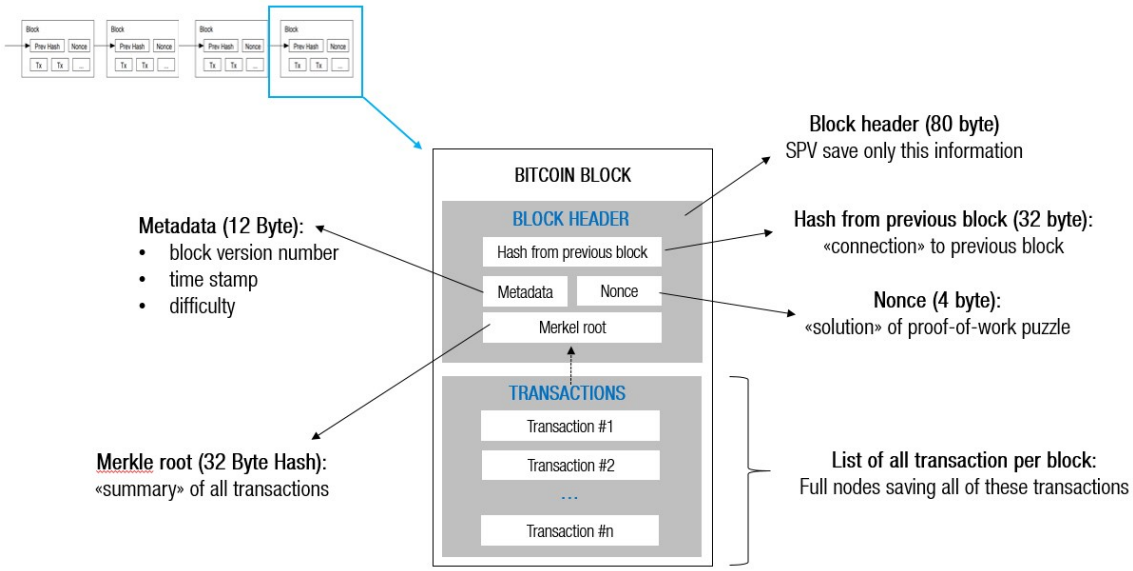


Figure 3.2: Bitcoin block and block header fields

Basically, during mining activity, the miner tries to solve the Proof-of-Work by constantly changing the nonce value contained in the block header which is working on. Every time the miner applies a **double SHA-256** to the block header, and if the output is greater than the current Bitcoin difficulty *target*, the miner needs to increment the nonce value and retry the double hash again. When the output of the double SHA-256 corresponds to a number which is less than the Bitcoin current target, it means the miner found a valid solution to the Proof-of-Work, and so he adds the new mined block to its local copy of the blockchain and he immediately broadcasts the new block to all the other peers which he is connected to. He needs to be as fast as possible during the so-called *block-propagation*, since it's crucial to let him claim the ***block reward***. To better resume the mining activity, the flow chart present in Figure 3.3 can be very helpful.

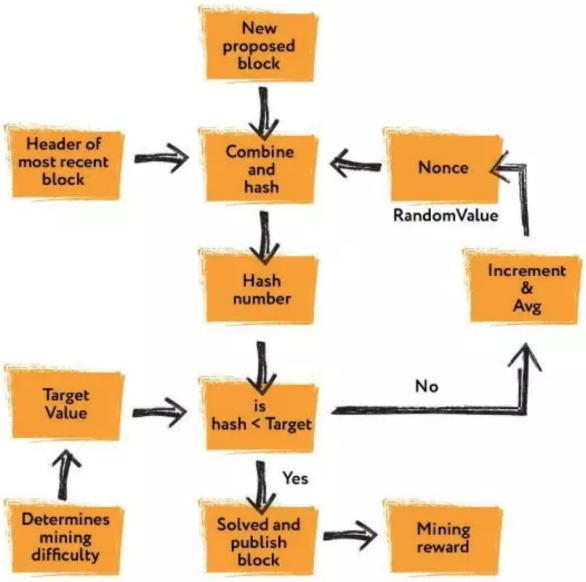


Figure 3.3: Flow chart of mining activity

In the Bitcoin ecosystem, miners play a crucial role by utilizing electricity to solve the above-described Proof-of-Work. When a miner successfully completes the task and validates all the transactions according to the consensus rules, they become eligible to receive a reward. This reward is typically in the form of new minted Bitcoin and transaction fees. Importantly, the reward is granted only when the miner demonstrates their ability to accurately validate the transactions without the need for a central authority. This delicate balance between mining, validation, and reward ensures the security of Bitcoin's network.

As already anticipated, the reward is constituted by the sum of the block reward and transaction fees from all the transactions included in the block. This quantity is contained and encoded in the so-called ***coinbase transaction***. Differently from regular transactions in Bitcoin, the coinbase transaction does not use Unspent Transaction Outputs (UTXOs) as inputs. Instead, it contains only one input known as the coinbase, which essentially generates new Bitcoin out of nothing. The coinbase transaction has a single output, which is designed to be paid to the miner's own Bitcoin address.

3.2 Mining history and evolution

This section aims to dive into the fascinating history and evolution of Bitcoin mining equipment, tracing its origins from the early days of Bitcoin to its present-days. The focus will be especially onto the technologies that have transformed, year after year, Bitcoin mining into a globally recognized industry.
As already said, the first block was mined by Satoshi Nakamoto on January 3, 2009. At the beginning of Bitcoin, the network difficulty was 1. Since there was very little people who was mining in the first days, the difficulty didn't increased, and so it was possible to mine Bitcoin blocks using an average personal computer. In fact, it was the unique time in which only a Central Processing Unit (**CPU**) was enough to mine bitcoin. As the potential reward for mining received more media attention, especially after the historical first-ever real-world transaction using bitcoin where the programmer named Laszlo Hanyecz spent 10000 bitcoin on two Papa John's pizzas (May 18, 2010 [15]), the mining difficulty started to rise.
In **October 2010**, the first mining device based on Graphics Processing Units (**GPU**s) was developed. As clearly represented in 3.4, thanks to GPU's excellence at computing simple mathematical operations in parallel, GPUs hugely increased the global hashrate, leading to a network difficulty increase.

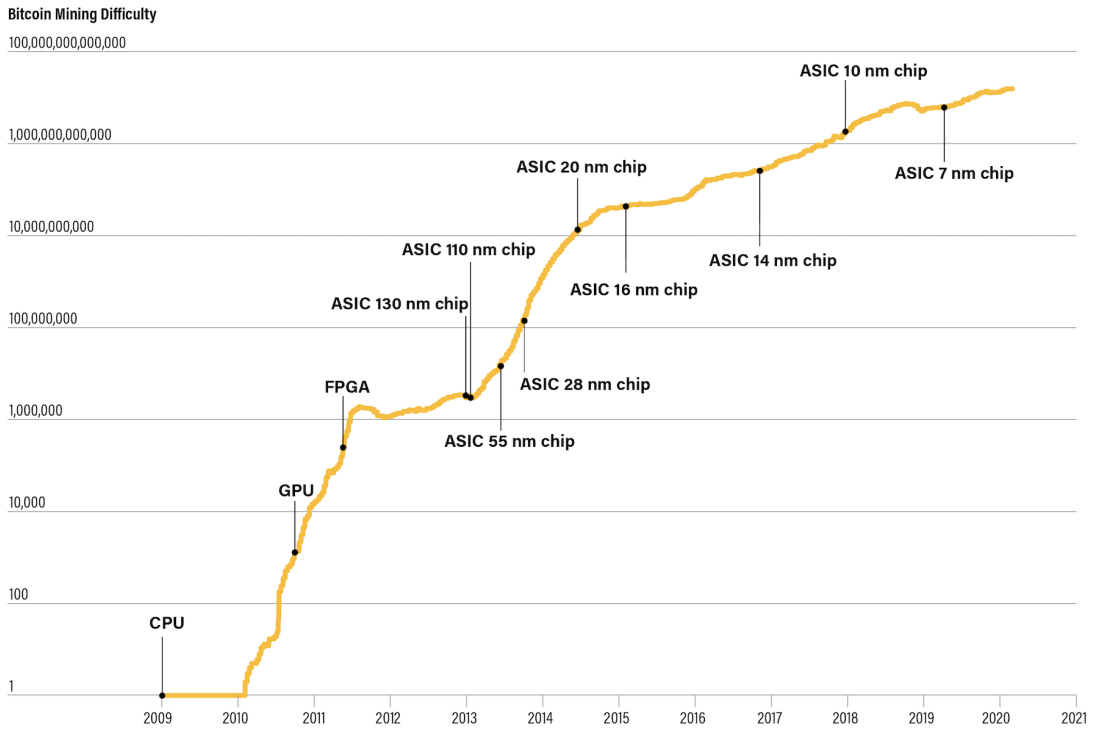


Figure 3.4: Bitcoin mining hardware evolution, in relation to the network difficulty

During the following year, **2011**, the Field Programmable Gate Arrays (**FPGA**) came into the mining game. They were even faster than GPUs in term of hashing power, contributing to the ever-increasing network hashrate, and difficulty.
The third major advancement in Bitcoin mining required an extensive allocation of resources, time, and development efforts. The focus during those years was on creating a completely novel machine solely specialized to Bitcoin mining. The hard work in research & development got the first results in **2013**, when the Chinese company called Canaan Creative, introduced the first set of Application-Specific Integrated Circuits (**ASIC**s) designed exclusively for Bitcoin mining.
In contrast to CPUs, GPUs, and FPGAs, these ASIC devices were specifically built with the intention of being used exclusively for Bitcoin mining. This included pre-designing and optimizing all hardware and software components of these ASIC devices to efficiently compute the precise calculations required for generating new Bitcoin blocks. While Canaan Creative emerged as the inaugural Bitcoin ASIC manufacturer, other players like Bitmain and MicroBT also joined the game, introducing their own models of ASIC mining devices with increasingly advanced hardware. A significant evolution in ASIC mining technology since 2013 has been the consistent reduction in chip size. Beginning at a size of 130nm in 2013, the ASIC chips have undergone remarkable shrinkage, with the latest hardware models featuring diminutive sizes as small as 5nm. The transistor size reduction led to an increasing efficiency in ASIC machines. Nowadays, an ASIC bitcoin mining device is estimated to be 100 billion times more efficient than the average CPU back in 2009.

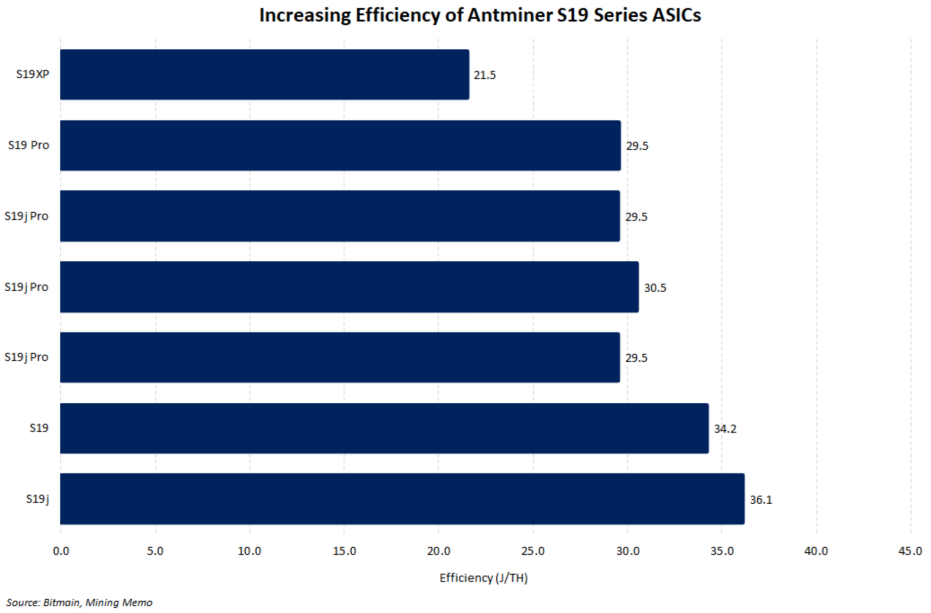


Figure 3.5: Antminer S19 model series, with relative efficiency (J/TH)

3.3 Solo mining and Pooled mining

Given the huge transformation that the entire Bitcoin mining activity faced, as analyzed in the previous section, a clarification about two opposite approaches that characterized mining operations since its beginning is needed. In this section the aim is to clarify the operational differences between the so called **Solo mining** and **Pooled mining**, focusing on their relative pros and cons.

Solo mining

In Solo mining, the miner relies solely on its own computational power to compete against the entire network in the race to find a valid block. In this process the hashes are calculated individually, in order to find a valid block whose reward will be **paid entirely to the miner** in ownership of the hashing power. This is obtained by explicitly put the miner Bitcoin address into the coinbase output script, when preparing the block template to mine on. In this case, the miner needs to run a local Bitcoin full-node, to get transactions to validate from, in addiction to the other fields needed to build the block header.
However, considering a certain network difficulty defined as D, «Block finding when mining solo with a constant hashrate h is a Poisson process with $\frac{h}{2^{32}.D}$ as the rate parameter. We said that mining for time t results in $\frac{ht}{2^{32}.D}$ blocks on average.» [16]
As described in the last section, Bitcoin mining became a very competitive field, and since its first years (2011-2013) it definitely started to transform more and more into an industrial activity. Since those years, miners had to start considering many factors during their business activity, such as the intrinsic variance of valid blocks finding during Solo mining. As the Bitcoin network has grown, the mining difficulty has increased significantly. This means that the chances of an individual miner successfully mining a block and earning rewards have diminished considerably. For this reason, since the first years of Bitcoin mining activity, the concept of Pooled mining started to become increasingly popular.

Pooled mining

In the current highly competitive mining landscape, solo miners operating alone, face significant challenges. The probability of solo miners successfully finding a block to pay their electricity and hardware expenses has become so low that it constitutes a very risky behaviour. To counter these odds, miners have resulted to collaborating and forming mining pools, systems used to combine their hashing power and sharing the resulting rewards among a large number of participants.
Mining pools coordinate thousands of miners, efficiently splitting the nonce search-space and assigning them to individual miners. The individual miners configure their mining equipment to connect to a pool server, and communicate a Bitcoin address to the pool, which is used to receive their share of the rewards. Their

mining hardware remains connected to the pool server while mining, synchronizing their efforts with the other miners. In this case, candidate block are built to pay the reward to a pool Bitcoin address, differently from the Solo mining approach. At regular intervals, the pool server initiates payments to the Bitcoin addresses of participating miners once they have accumulated a specific threshold of rewards. The main business model of the pool operators is typically a percentage fee which is cut off from the rewards collected by miners. To understand how the pool collects the work done by individual miners, the concept of ***share*** has to be explained. Similarly to how a solo miner has to satisfy the Bitcoin network current difficulty, in order to submit a valid block to the entire network, a miner who works in a mining pool has to find an output to the double SHA-256 function which is lower than the so called ***share difficulty target***. As represented in Figure 3.6, the mining pool sets a higher target (lower difficulty) for earning a share, typically more than 1,000 times easier than the Bitcoin network's target. When someone in the pool successfully mines a block, the reward is earned by the pool and then shared with all miners in proportion to the number of shares they contributed to the effort.

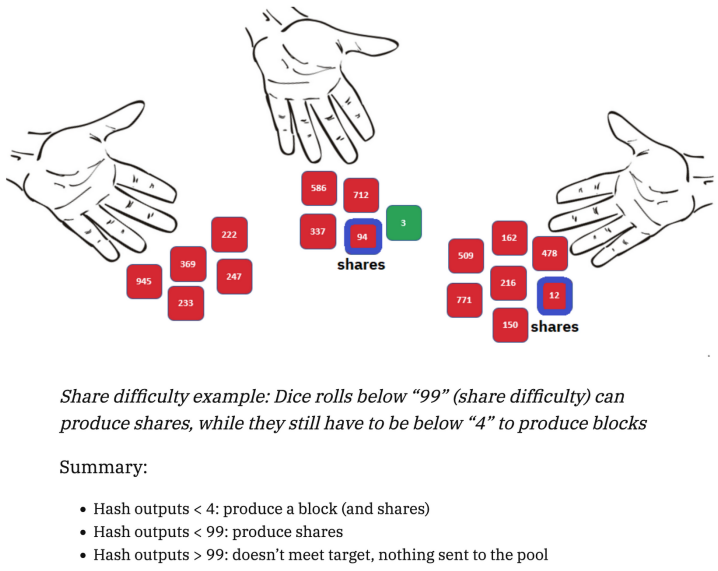


Figure 3.6: Mining pool share difficulty and network difficulty [2]

The first mining pool ever created is the so called *slushpool*, born in 2010. Since those year, in fact, the need for mining pool operations were increasingly faced, to reduce the above-mentioned reward variance associated with the solo mining approach. However, in order to manage all the communications between individual miners and mining pool servers, some kind of specialized pooled mining protocol had to be developed.
The scope of the entire next chapter is about the history and the evolution of the pooled mining protocols, which began with the so called *Getwork* protocol. It contains a deep exploration of these protocols from an operational point of view, underlining the differences between them, with their relative pros and cons.

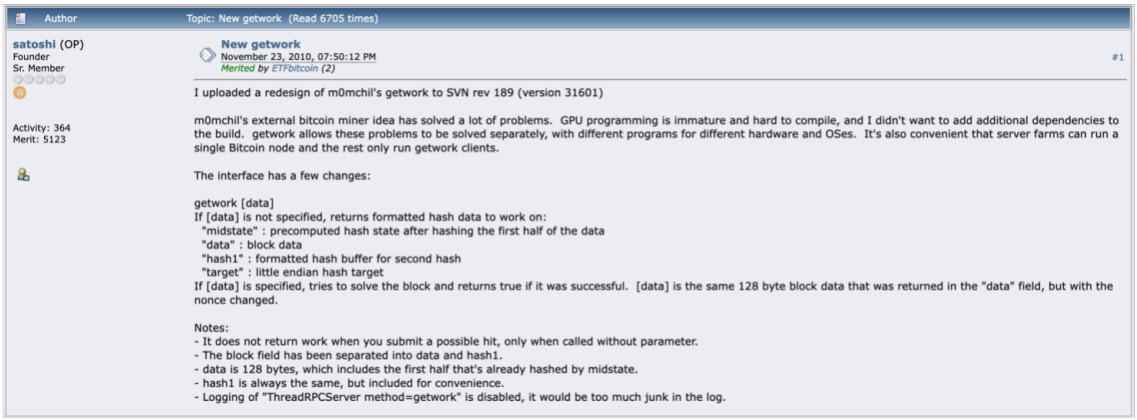
Chapter 4

History of pooled mining protocols

4.1 Getwork

4.1.1 What is Getwork: why and when it was born

Getwork is a RPC method used by miners for retrieving block headers to find a solution for, or sending valid proofs of work; it was designed and introduced by the user called m0mchil in 2010. [17] Getwork was implemented in Bitcoin Core on 23rd November 2010: it's possible to read the official announcement regarding the Bitcoin Core update by Satoshi on BitcoinTalk Forum. [18]


Author | Topic: New getwork (Read 6705 times)

satoshi (OP)
Founder
Sr. Member
Activity: 364
Merit: 5123

New getwork
November 23, 2010, 07:50:12 PM
Merited by ETFbitcoin (2)
#1

I uploaded a redesign of m0mchil's getwork to SVN rev 189 (version 31601)

m0mchil's external bitcoin miner idea has solved a lot of problems. GPU programming is immature and hard to compile, and I didn't want to add additional dependencies to the build. getwork allows these problems to be solved separately, with different programs for different hardware and OSes. It's also convenient that server farms can run a single Bitcoin node and the rest only run getwork clients.

The interface has a few changes:

getwork [data]
If [data] is not specified, returns formatted hash data to work on:
"midstate" : precomputed hash state after hashing the first half of the data
"data" : block data
"hash1" : formatted hash buffer for second hash
"target" : little endian hash target
If [data] is specified, tries to solve the block and returns true if it was successful. [data] is the same 128 byte block data that was returned in the "data" field, but with the nonce changed.

Notes:
- It does not return work when you submit a possible hit, only when called without parameter.
- The block field has been separated into data and hash1.
- data is 128 bytes, which includes the first half that's already hashed by midstate.
- hash1 is always the same, but included for convenience.
- Logging of "ThreadRPCServer method=getwork" is disabled, it would be too much junk in the log.

Figure 4.1: Satoshi *getwork* update announcement, Bitcoin Talk Forum

The idea behind the getwork method was born by m0mchil's project called POCLbm (PyOpenCL bitcoin miner), the first open-source GPU miner software ever developed. [19]

Before entering the details of the getwork method, a reminder of the block header structure is needed, as represented in Figure 4.2.

Bytes	Name	Type	Description
4	version	int32_t	The block version number indicates which set of block validation rules to follow. See the list of block versions below.
32	previous block header hash	char[32]	A SHA256(SHA256()) hash in internal byte order of the previous block's header. This ensures no previous block can be changed without also changing this block's header.
32	merkle root hash	char[32]	A SHA256(SHA256()) hash in internal byte order. The merkle root is derived from the hashes of all transactions included in this block, ensuring that none of those transactions can be modified without modifying the header. See the merkle trees section below.
4	time	uint32_t	The block time is a Unix epoch time when the miner started hashing the header (according to the miner). Must be strictly greater than the median time of the previous 11 blocks. Full nodes will not accept blocks with headers more than two hours in the future according to their clock.
4	nBits	uint32_t	An encoded version of the target threshold this block's header hash must be less than or equal to. See the nBits format described below.
4	nonce	uint32_t	An arbitrary number miners change to modify the header hash in order to produce a hash less than or equal to the target threshold. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the merkle root updated.

Figure 4.2: Bitcoin block header fields, *from [3]*

NOTE: Data present in the block header are all represented in little-endian order, except for the hashes (Hash of previous block's header and Merkle root) which are in internal-byte order (the original order of the SHA256 output, which can be considered little-endian as well).

Considering the example in Figure 4.3, hash of previous block's header is represented in its internal byte order, but if you want to look for the corresponding block using a blockchain explorer, you must transform it to the RPC Byte Order (big-endian), obtaining this: **00000000000000000cca48eb4b330d91e8d946d344ca302a86a280161b0bffb6**. The block corresponding to this hash is the one at height 328733. [20]

For any further clarifications, on the BTC Information website can be found a complete documentation about the differences between internal-byte order and RPC Byte Order. [21]

```
02000000 ............................ Block version: 2

b6ff0b1b1680a2862a30ca44d346d9e8
910d334beb48ca0c0000000000000000 ... Hash of previous block's header
9d10aa52ee949386ca9385695f04ede2
70dda20810decd12bc9b048aaab31471 ... Merkle root

24d95a54 ............................ [Unix time][unix epoch time]: 1415239972
30c31b18 ............................ Target: 0x1bc330 * 256**(0x18-3)
fe9f0864 ............................ Nonce
```

Figure 4.3: Example Bitcoin block header, in hexadecimal format

4.1.2 How Getwork works

Getwork is a JSON-RPC method sent over a HTTP transport. If it's called without any argument, it provides a pre-processed block header to work on, contained in the data key field. During this process, little endian byte order is used; in addiction to this, due to the getwork implementation in Bitcoin Core, every 4 bytes chunks are byte swapped. To check the code which swaps the chunks, it's possible to have a look at lines 196-198 in "src/rpcmining.cpp" contained in commit *1f3bfa329f96b0e4564c410b539765909601ad1d* of Bitcoin Core.

getwork job			
Key	**Required**	**Type**	**Description**
data	Yes	String	Pre-processed SHA-2 input chunks, in little-endian order, as a hexadecimal-encoded string
target	Yes	String	Proof-of-work hash target as a hexadecimal-encoded string
algorithm	No	String	Brief specification of proof-of-work algorithm. Not provided by bitcoind or most poolservers.

Figure 4.4: Getwork method details and parameters

For this reason, before starting to hash the header received, miner must do some bytes manipulation on data, which will be illustrated in the following examples.

Example N.1: data content present in the response of a getwork request

00000002 0597ba1f0cd423b2a3abb0259a54ee5f783077a4ad45fb620000021800000000
8348d1339e6797e2b15e9a3f2fb7da08768e99f02727e4227e02903e43a42b31 51155310
1a051f3c 00000000 0000008000000000000000000000000000000000000000000000000000
0000000000000000000000000000000080020000

- **Version** –> 4 bytes –> 0x00000002

- **Prev_block_hash** –> (need to transform it) –> 0597ba1f0cd423b2a3abb0259a54ee5f783077a4ad45fb620000021800000000 –> from little endian to big endian (every 4 bytes chunks) –> 1fba9705b223d40c25b0aba35fee549aa477307862fb45ad1802000000000000 –> reverse (every byte) –> **0000000000000218ad45fb62783077a49a54ee5fa3abb0250cd423b20597ba1f**

  This is the hash of previous block's header of block at height 220249, which is the hash of block 220248.

- **Merkle root** –> transform it as the Prev_block_hash –> 32 bytes
- **Timestamp** –> 4 bytes –> 0x51155310 (hex) –> 1360352016 (dec) –> **8 February 2013 19:33:36**
- **nBits** –> 4 bytes –> **0x1a051f3c**
- **Nonce** –> 4 bytes –> **0** (initial value)

At this point, to mine the work received in the response, the miner need first byte swap each 32-bit chunk from little-endian to big-endian:

02000000 1fba9705b223d40c25b0aba35fee549aa477307862fb45ad1802000000000000 33d14883e297679e3f9a5eb108dab72ff0998e7622e427273e90027e312ba443 10531551 3c1f051a 00000000 80000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000280

Take the first 80 bytes (block template header):

02000000 1fba9705b223d40c25b0aba35fee549aa477307862fb45ad1802000000000000 33d14883e297679e3f9a5eb108dab72ff0998e7622e427273e90027e312ba443 10531551 3c1f051a 00000000

Hash them (SHA-256) twice, changing the nonce value, until a value which is lower than the difficulty target is reached.

Example N.2: data content present in the response of a getwork request

00000002 b15704f4ecae05d077e54f6ec36da7f20189ef73b77603225ae56d2b00000000 bcf59695a4e35a2f7535e1a86b306a3b08c212bf0b833764018fe39f01919381 510c2811 1c0e8a37 00000000 00000080000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000080020000

- **Version** –> 4 bytes –> 0x00000002

- **Prev_block_hash** –> (need to transform it) –> b15704f4ecae05d077e54f6ec36da7f20189ef73b77603225ae56d2b00000000 –> from little endian to big endian (every 4 bytes chunks) –> f40457b1d005aeec6e4fe577f2a76dc373ef8901220376b72b6de55a00000000 –> reverse (every byte) –> **000000005ae56d2bb77603220189ef73c36da7f277e54f6eecae05d0b15704f4**

  This is the hash of previous block's header of block at height 49133 of the bitcoin testnet, which is the hash of block 49132.

- **Merkle root** –> transform it as the Prev_block_hash –> 32 bytes
- **Timestamp** –> 4 bytes –> 0x510c2811 (hex) –> 1359751185 (dec) –> **1 February 2013 20:39:45**
- **nBits** –> 4 bytes –> **0x1c0e8a37**
- **Nonce** –> 4 bytes –> **0** (initial value)

Now, to mine the work received by the response, same operations on data field than before has to be done (bytes swapping, hashing twice SHA-256).
After that, a **HTTP POST request** must be sent to pool server to submit the valid block.

```
POST / HTTP/1.1
Authorization: Basic Y2RlY2tlcjphYmMxMjM=
Host: localhost:18332
Content-type: application/json
X-Mining-Extensions: longpoll midstate rollntime submitold
Content-Length: 305
User-Agent: cgminer 2.8.1
{
"method": "getwork",
"params": [ "00000002b15704f4ecae05d077e54f6ec36da7f20189ef
73b77603225ae56d2b00000000b052cbbdeed2489ccb13a526b77fadcee
f4caf7d3bb82a9eb0b69ebb90f9f5a7510c27fd1c0e8a37fa5313380000
0080000000000000000000000000000000000000000000000000000000
0000000000000000000000080020000"],
"id":1
}
```

Example N.3: getwork complete protocol flow

1. **getwork *request*:**

```
{"method":"getwork","params": [],"id":1}
```

2. **getwork *response*:**

```
{"id": "1",
"result": {
    "hash1":"00000000000000000000000000000000000000000000
    00000000000000000000000000000080000000000000000000000000
    00000000000000000000000000010000",
    "data":"00000001c570c4764aadb3f09895619f549000b8b51a
    789e7f58ea75000709700000000103ca064f8c76c390683f820
    3043e91466a7fcc40e6ebc428fbcc2d89b574a864db8345b1b00
    b5ac000000000000080000000000000000000000000000000000
    00000000000000000000000000000000000000000000008002
    0000",
    "midstate":"e772fc6964e7b06d8f855a6166353e48b2562de4
    ad037abc889294cea8ed1070",
    "target":"ffffffffffffffffffffffffffffffffffffffffffff
    ffffffffffff00000000"
},
"error": null}
```

3. **Same operations on data field than before** (bytes swapping, hashing twice SHA-256)

4. **getwork request *sending a valid proof of work*:**

```
{"method": "getwork",
"params":
    ["0000000141a0e898cf6554fd344a37b2917a6c7a6561c20733b09
    c8000009eef00000000d559e21882efc6f76bbfad4cd13639f4067c
    d904fe4ecc3351dc9cc5358f1cd54db84e7a1b00b5acba97b604000
    00080000000000000000000000000000000000000000000000000000
    00000000000000000000000000080020000"],
"id":1
}
```

4.1.3 Getwork usage into pooled mining and protocol extensions

The getwork RPC method was developed and written into Bitcoin Core as all the other RPCs, in this way anyone who was running a full node was able to use it to mine new blocks.
By the way, even if Bitcoin was just born (3rd January 2009), the first mining pools started to appear (*slushpool* was born in 2010 as well). Due to the nature of the RPC method, it had been used also into a pooled mining context: it could be sent over HTTP transport layer, providing an elegant way for the pool operators to get and distribute new jobs to miners who were joining the pool.
In the context of pooled mining, where every single miner had to communicate with the pool server, the exploitation of some protocol extensions seemed to be particularly useful in that scenario. [22]
More specifically, when getting new work, miners could send a **X-Mining-Extensions** header with a space-delimited list of supported extensions:

- **Hostlist**

  The server may include a X-Host-List header with a list of available servers formatted in JSON as an array of objects with server details. The field "host" specifies the server's hostname or IP address, "port" specifies a TCP port, and "ttr" is "time to return". If you use server with non-zero ttr you should try to return to the server with 0 ttr after this number of minutes.

  Example:

  ```
  X-Host-List: [{"host":"server.tld","port":8332,"ttr":0},
  {"host":"backup.tld","port":8332,"ttr":20}]
  ```

  This string says that "server.tld" is the main server. When you detect connection problems, you need to try the next server - "backup.tld" for 20 minutes and then try to switch back to "server.tld". If the main server is still offline you should continue to use "backup.tld" for another 20 minutes.

- **Longpoll**

  If mining pool does supports Long Polling, it should include a X-Long-Polling header with a relative or absolute URI. The absolute URI may specify a different port than the original connection. Miner must start a request to long polling URI with GET method and same basic authorization as on main connection. This request is not answered by server until it wishes to expire current block data, and new data is ready. The answer is the same as getwork on the main connection. Upon receiving this answer, miner should drop current calculation in progress, discard its result, and start working on

received data and make a new request to a long polling URI. There is a 60 second limit before new work should be requested (the normal way) regardless of response from longpoll (though this may be overridden by the rollntime extension below).

- **Noncerange**

  In addition to X-Mining-Extensions, the miner should also send X-Mining-Hashrate, with an integer value of expected hashrate measured in full hashes per second. The server may then add an additional field to the JSON response, "noncerange", which contains two 32-bit integers specifying the starting and final nonce the miner is allowed to scan. While both values are given in big endian, miners should iterate over the range in their native 32-bit integer type (SHA256 works with 32-bit integers, not character data). The miner should implement rollntime by starting from the first nonce in the range every second.

  Example:

  ```
  "noncerange": "000000001fffffff"
  ```

  Response:

  ```
  solution: "...dddddd1f..."
  ```

  A very interesting discussion about the noncerange extension can be found on Bitcoin Talk Forum [23].

- **RollNtime** If the getwork response includes a "X-Roll-Ntime" header with any value other than "N" or the null string, the miner may (within reason) change the ntime field in addition to the nonce. The server may send a value of "expire=<N>", where <N> is an integer number of seconds it is willing to accept the other headers for. Note that if the "X-Roll-NTime" header is NOT present in a work response, that work may NOT be rolled, even if earlier work from the same server allowed it. Also note that the headers of a share submission should not influence the behavior of work. More specifically, if a share submit does not have the header, it should not disable rollntime for the current work (which did).

  To investigate more deeply, an official discussion about the RollNtime extension can be found on Bitcoin Talk Forum [24].

Figure 4.5: Luke-Jr mining operations recommendations, on *Bitcoin Talk Forum*

4.1.4 Why Getwork usage ended up

During those years (2010-2012), the entire mining ecosystem was growing, and so many things started to change: as seen in Chapter 3, network hashrate was increasing, solo mining was slowly disappearing, and mining equipment was becoming more powerful year after year.
Getwork method was the only mining protocol that was used at that time, but, as we've seen before, it permitted miners to change only the 32 bits of original nonce field in the header received from bitcoind. In addition to those 4 bytes of nonce space, miners could exploit the nTime field of the header (thanks to the RollNtime extension that was supported by most mining pools at that epoch). Anyway, one getwork job was enough for a 4.2GHash/s (2^{32} bits) mining rig and (thanks to nTime rolling) this job was usable for one minute or until a new Bitcoin block arrived (depending on what happened first). Beyond that, a block created from massively modified nTime could/can be rejected by the Bitcoin network.
That maximum rate was extremely too low for the mining performances of the newest equipment of that period, so an alternative needed to be found. Fortunately, a group of Bitcoin Core developers specialized in mining operations, worked very hard during 2012, to show up a valid alternative to the standard getwork method: they developed the so called getblocktemplate method or GBT.
The enhancements brought by GBT will be deeply explained and described in the next chapter, which will be followed by the chapter focused on Stratum: the pooled mining protocol which became the standard "de facto" nowadays. It was developed at the same time of getblocktemplate, from the big effort of the developer called *slush*, the founder of the first mining pool ever created, *slushpool*.

The official removal of Getwork method from Bitcoin Core had been proposed on 16th August 2013, through the Pull Request #2905 [25]. To dig into more details of the removal, a very interesting discussion can be found on Bitcoin Talk Forum [26].

4.2 Getblocktemplate (GBT)

4.2.1 What is GBT: why and when it was born

Getblocktemplate, or GBT, was born over the **mid 2012**. It was the first alternative to the *getwork* method, openly developed by some of the most skilled Bitcoin Core developers of that time, promoted and directed firstly by the Bitcoin Core developer Luke Dashjr. The official announcement of GBT was made by Luke Dashjr himself on 12th September 2012, on Bitcoin Talk Forum [27].

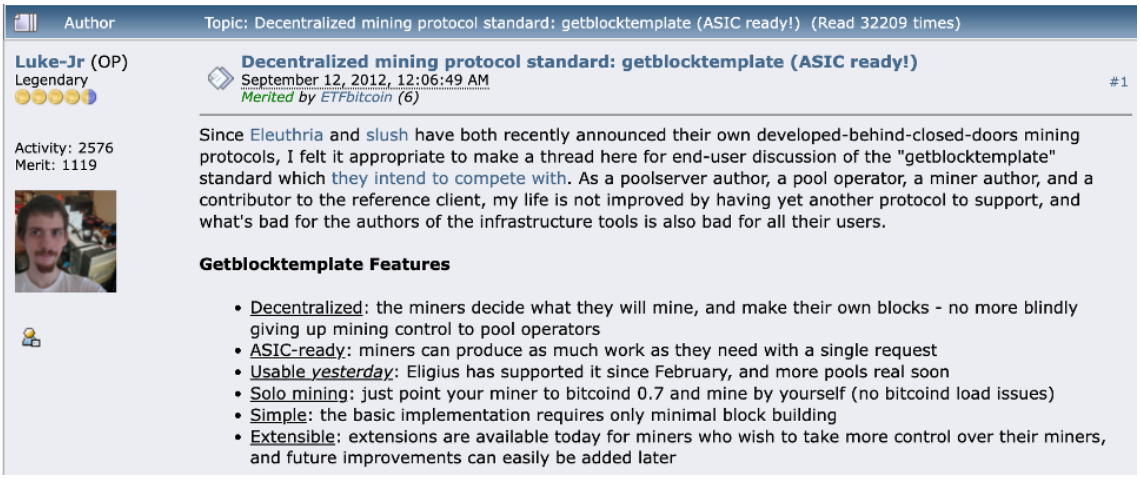


Figure 4.6: Luke-Jr *getblocktemplate* announcement, Bitcoin Talk Forum

GBT was defined as a new mining protocol standard, developed to accommodate both solo mining and pooled one, thanks to its usage flexibility. The main reason why it was developed was the need to find a solution for the issue described at the end of the previous chapter: miners were becoming more and more powerful, and the nonce research space of the previous getwork method was not enough for the maximum hashrate of the newest equipment.
Getblocktemplate introduced different new features to the mining operations of that time, but the most important point which mainly drove the entire development (apart from the solution at the nonce research space issue) was the attempt to decentralize the power obtained from the mining pool operators. At that time, as said in the previous chapter, most of the miners were using the getwork method in a pooled mining context: in this way the only entities responsible for building the blocks, selecting the transactions from the mempool, were the pools themselves. Most miners were just mining onto the block headers received from the pool server, never building their own block templates: this was a huge security risk from the point of view of the GBT developers and supporters, that could have led to a high single-point failure and censorship risk.

The group of developers involved into GBT followed the standard procedure for proposing and implementing new features into Bitcoin Core: they published two Bitcoin Improvement Proposal(s) [28], specifically BIP22 and BIP23.
Getblocktemplate was officially implemented in Bitcoin Core release v.0.7.0, on 17th September 2012 [29].

4.2.2 How GBT works

As explained in the above section, getblocktemplate was coded into Bitcoin Core as any other RPCs, so it was (and it still is) perfectly usable by anyone who has got a Bitcoin full node. However, the main reasons which took to its development, derived from the ever-growing context of pooled mining.
Because of this, the focus will be about how GBT works and how it was used as a communication mining protocol between single miners and mining pools.

First, the miner must connect to the pool server, asking for an initial block template. There can be only one JSON Object in the request, containing request parameters (divided in "capabilities" and "mode"):

```
{
    "id": 0,
    "method": "getblocktemplate",
    "params":
    [{
      "mode": "template", (or "proposal")
      "capabilities": ["coinbasetxn", "workid", "coinbase/append"],
    }]
}
```

template request			
Key	**Required**	**Type**	**Description**
capabilities	No	Array of Strings	SHOULD contain a list of the following, to indicate client-side support: "longpoll", "coinbasetxn", "coinbasevalue", "proposal", "serverlist", "workid", and any of the mutations
mode	No	String	MUST be "template" or omitted

Figure 4.7: Description of the initial block template request parameters, using GBT

In the template request, the miner can indicate to pool server the features supported client-side, such as "longpoll", specifying the desired mutations in the "capabilities" field.

mutations	
Value	**Significance**
coinbase/append	append the provided coinbase scriptSig
coinbase	provide their own coinbase; if one is provided, it may be replaced or modified (implied if "coinbasetxn" omitted)
generation	add or remove outputs from the coinbase/generation transaction (implied if "coinbasetxn" omitted)
time/increment	change the time header to a value after "time" (implied if "maxtime" or "maxtimeoff" are provided)
time/decrement	change the time header to a value before "time" (implied if "mintime" is provided)
time	modify the time header of the block
transactions/add (or "transactions")	add other valid transactions to the block (implied if "transactions" omitted from result)
prevblock	use the work with other previous-blocks; this implicitly allows removing transactions that are no longer valid (but clients SHOULD attempt to propose removal of any required transactions); it also implies adjusting "height" as necessary
version/force	encode the provide block version, even if the miner doesn't understand it
version/reduce	use an older block version than the one provided (for example, if the client does not support the version provided)

Figure 4.8: Mutations which can be asked from a miner, using GBT [4]

At this point, pool server must return a JSON Object containing all the details needed to begin mining:


```
{"result": {
    "coinbasetxn": {
     "data": "010000000100000000000000000000000000000000000000000000000000
     0000000000000000000ffffffff1302955d0f00456c667697573005047dc66
     085fffffffff02fff1052a010000001976a9144ebeb1cd26d6227635828d60d
     3e0ed7d0da248fb88ac01000000000000001976a9147c866aee1fa2f3b3d5ef
     fad576df3dbf1f07475588ac00000000"
    },
    "previousblockhash": "000000004d424dec1c660a68456b8271d09628a80c
    c62583e5904f5894a2483c",
    "transactions": [],
    "expires": 120,
    "target": "00000000ffffffffffffffffffffffffffffffffffffffffffffffff
    ffffffffff",
    "longpollid": "some gibberish",
    "height": 23957,
    "version": 2,
    "curtime": 1346886758,
    "mutable": ["coinbase/append"],
    "bits": "ffff001d"
},"id": 0}
```

template			
Key	**Required**	**Type**	**Description**
bits	Yes	String	the compressed difficulty in hexadecimal
curtime	Yes	Number	the current time as seen by the server (recommended for block time) - note this is not necessarily the system clock, and must fall within the mintime/maxtime rules
height	Yes	Number	the height of the block we are looking for
previousblockhash	Yes	String	the hash of the previous block, in big-endian hexadecimal
sigoplimit	No	Number	number of sigops allowed in blocks
sizelimit	No	Number	number of bytes allowed in blocks
transactions	Should	Array of Objects	Objects containing information for Bitcoin transactions (excluding coinbase)
version	Yes	Number	always 1 or 2 (at least for bitcoin) - clients MUST understand the implications of the version they use (eg, comply with BIP 0034 for version 2)
coinbaseaux	No	Object	data that SHOULD be included in the coinbase's scriptSig content. Only the values (hexadecimal byte-for-byte) in this Object should be included, not the keys. This does not include the block height, which is required to be included in the scriptSig by BIP 0034. It is advisable to encode values inside "PUSH" opcodes, so as to not inadvertently expend SIGOPs (which are counted toward limits, despite not being executed).
coinbasetxn	this or ↓	Object	information for coinbase transaction
coinbasevalue	this or ↑	Number	total funds available for the coinbase (in Satoshis)
workid	No	String	if provided, this value must be returned with results (see Block Submission)

Figure 4.9: Description of the template fields sent by the mining pool, using GBT

template "transactions" element		
Key	**Type**	**Description**
data	String	transaction data encoded in hexadecimal (byte-for-byte)
depends	Array of Numbers	other transactions before this one (by 1-based index in "transactions" list) that must be present in the final block if this one is; if key is not present, dependencies are unknown and clients MUST NOT assume there aren't any
fee	Number	difference in value between transaction inputs and outputs (in Satoshis); for coinbase transactions, this is a negative Number of the total collected block fees (ie, not including the block subsidy); if key is not present, fee is unknown and clients MUST NOT assume there isn't one
hash	String	hash/id encoded in little-endian hexadecimal
required	Boolean	if provided and true, this transaction must be in the final block
sigops	Number	total number of SigOps, as counted for purposes of block limits; if key is not present, sigop count is unknown and clients MUST NOT assume there aren't any

Figure 4.10: Transaction fields, present in the template sent by pool server, using GBT

In the template sent by pool server, the "mutable" key can be used to specify which modifications the miner is allowed to make, on that specific template. In this specific case, the pool has agreed upon the miner request to edit the coinbase transaction, with can be used as an extranonce, to enlarge the nonce research space. This is specified by both miner template request and pool template response, in the "mutable" key field, as ["coinbase/append"] value.

So, as soon as the template is received, the miner needs to customize the coinbase transaction, with the only limitation about not exceeding the 100 bytes data limit. Coinbase data (which are the ones customizable) begins after 42 bytes, in the coinbase transaction, and the 42nd byte represents the data length. So, miner must insert the custom data right after the "original" data already present in the coinbase provided by pool server. At the end, it needs to change the 42nd byte, inserting the new data length. This is an example Python script which customizes the coinbase transaction data, inserting the 'my block' string:

```
import binascii
coinbase = binascii.a2b_hex(template['coinbasetxn']['data'])
extradata = b'my block'
origLen = ord(coinbase[41:42])
newLen = origLen + len(extradata)
coinbase = coinbase[0:41] + chr(newLen).encode('ascii') +
coinbase[42:42 + origLen] + extradata + coinbase[42+origLen:]
```

Since the coinbase transaction data has been modified, the new merkle root needs to be re-built, before starting hashing on the new customized block header. First, miner must put the modified coinbase transaction at the first place in the transactions list received from the pool server. At this point he needs to apply double SHA-256 to every transaction in the list; after that, he needs to redo the double SHA-256 to every couple of transactions (concatenated), as long as it remains just one single hash string, which is the final merkle root. To do this, the following Python script can be used as an example:

```
import hashlib
def dblsha(data):
  return hashlib.sha256(hashlib.sha256(data).digest())
    .digest()

txnlist = [coinbase] + [binascii.a2b_hex(a['data'])
for a in template['transactions']]
merklehashes = [dblsha(t) for t in txnlist]
while len(merklehashes) > 1:
   if len(merklehashes) % 2:
      merklehashes.append(merklehashes[-1])
  merklehashes = [dblsha(merklehashes[i] + merklehashes[i + 1])
  for i in range(0, len(merklehashes), 2)]
merkleroot = merklehashes[0]
```

Once the new merkle root has been computed, the new block header has to be assembled. To do this, miner can take the data already present in the template sent by pool server, substitute the merkle root with the one just computed, and start hashing on the customized block header.

```
import struct
blkheader = struct.pack('<I', template['version']) + \
            binascii.a2b_hex(template['previousblockhash']) +
            merkleroot + \
            struct.pack('<I', template['curtime']) + \
            binascii.a2b_hex(template['bits']) + \
            b'NONC'
```

Whenever the miner finds a share (or block) which is valid, he needs to send it immediately to the pool server, to be rewarded for it. To do this, a submitblock method is exploited, which requires just one parameter: the serialized block data. To build a serialized block, miner needs to concatenate the block header, the number of transactions, and the transactions in the block (placed in the same order of the merkle tree used to compute the merkle root) [30].

```
{"id": 0,
    "method": "submitblock",
    "params": ["020000003c48a294584f90e58325c60ca82896d071826b45680a6
    61cec4d424d00000000de6433d46c0c7f50d84a05aec77be0199176cdd47f77e3
    44b6f50c84380fddba66dc47501d00ffff000010001010000000100000000000
    0000000000000000000000000000000000000000000000000ffffffff1302
    955d0f00456c6967697573005047dc66085ffffffff02fff1052a01000000197
    6a9144ebeb1cd26d6227635828d60d3e0ed7d0da248fb88ac0100000000000000
    1976a9147c866aee1fa2f3b3d5effad576df3dbf1f07475588ac00000000"]
}
```

While the miner is hashing on the block header, changing nonce and time fields, he needs to be updated from the pool whenever a new block is found, to not wasting time and energy. To achieve this, an optional long polling extension was designed into getblocktemplate protocol. To use it, miner needs to explicitly indicate it in the template request message, like this:

```
{"id": 0,
"method": "getblocktemplate",
"params": [{
    "capabilities": ["coinbasetxn", "workid", "coinbase/append"],
    "longpollid": "some gibberish",
}]}
```

In this way, as soon as a new block is found on the Bitcoin network, pool server can immediately notify the miner which is putting hash power into the pool.

In addition to the described "standard" behavior, getblocktemplate protocol permits many possible execution flows, depending on the mutations and pool extension used by the miner. To investigate them more deeply, in the description of BIP23 there's a dedicated section for it: moreover, there are some interesting answers to basic questions and doubts which came out during the weeks which followed the publication of the protocol itself.

4.2.3 Why GBT usage ended up

As mentioned in the last words of the previous section, *getblocktemplate* wasn't the only mining protocol which was developed and proposed during 2012, since the discussed issues related to the getwork protocol touched lots of different people and businesses in the mining sector.
During the same year, the developer called *slush* proposed his alternative to the *getwork* protocol, which is called **Stratum**. Marek "slush" Palatinus was the founder of one of the first Bitcoin mining pools, called *slushpool*, and he was worried about the performance issues related to the mining protocol used at that time. With his experience as a mining pool operator, he developed the above-mentioned protocol alternative, publishing the design details right after the announcement of *getblocktemplate*.
For this reason, under the announcement of GBT on Bitcoin Talk Forum can be found a lot of discussions related to the comparison between GBT and Stratum, adorned by not a few mutual criticisms. During the weeks which followed the first discussions, mining pools operators started to test both protocols, trying to establish the one to implement in their own businesses.

At the end, performances of the *Stratum* protocol were better than *getblocktemplate*'s, and the design was a way cleaner and easier to be understood and implemented by the mining pools operators of that time. Thanks to its efficiency improvements, and for other specific features which will be deeply discussed in the next chapter, *Stratum* became the standard "de facto" of the pooled mining protocols.

4.3 Stratum (V1)

4.3.1 What is Stratum (V1): why and when it was born

As said in the conclusion of the previous sub-chapter, Stratum protocol was born during the same period of getblocktemplate, in the second half of 2012. The main developer and proposer of Stratum was Marek Palatinus (aka "slush"), who was the founder of one of the first Bitcoin mining pools, called *slushpool*. He announced this new protocol on **11th September 2012**, in a thread on Bitcoin Talk Forum [31].

Figure 4.11: Slush *stratum* announcement, Bitcoin Talk Forum

The reason which took *slush* to develop Stratum, was the same as *getblocktemplate*'s: the mining protocol used at that time (*getwork*) was not efficient anymore, due to the newest more powerful mining equipment, which was coming, in addition to the ever-growing pooled mining activities. As discussed in the announcement thread, *getwork* permitted just 32 bits manipulation for the nonce research (in most cases they could also change the timestamp field, thanks to the nTime-rolling extension supported by most of the pools), so frequent requests from miners to mining pool server were needed to get new jobs to work on.
The other bad aspect of *getwork* was related to the necessity of HTTP protocol to transport the JSON-RPC methods: since HTTP is a protocol which is most suitable for websites navigation, it was not ideal for Bitcoin mining operations. With the growing global hashrate of that time, it was revealing very inefficient to manage the frequent requests coming from miners, especially for what regards the load on the pool servers and the bandwidth needed. As we have seen in 4.1.3, a mitigation that was implemented in the *getwork* protocol was the exploitation of Long Polling extension. However, its usage led to another issue: the packet storms which were received by the pool server from the miners who were trying to reconnect to the server after long polling broadcasts: most of the times it was hard to distinguish long polling re-connections from possible DDoS attacks.
For these reasons, *getwork* was totally unable to scale pooled mining operations: that's why Stratum was designed very differently from it.

To investigate more deeply upon the architectural choices of Stratum protocol,

it can be useful to look at the draft protocol specifications present in a publicly shared Google document [32].

4.3.2 How Stratum (V1) works

As explained in the above introduction, Stratum aimed to solve two main inefficiencies of the previous getwork protocol:

1. HTTP used as a transport protocol.

2. Excessive number of job requests made by miners to the pool server due to the absence of an extranonce field to modify during mining activity.

To solve the first issue, Stratum was developed as «a line-based protocol using plain TCP socket, with payload encoded as JSON-RPC messages. Client simply opens TCP socket and writes requests to the server in the form of JSON messages finished by the newline character. Every line received by the client is again a valid JSON-RPC fragment containing the response. There's no HTTP overhead involved and there are no hacks like mining extension flags encoded in HTTP headers anymore. But the biggest improvement from HTTP-based getwork is the fact, that server can drive the load by itself, it can send broadcast messages to miners at any time without any long-polling workarounds, load balancing issues and packet storms.» [33]

To go into deeper details, Stratum solved the inefficiencies introduced by using HTTP as the transport protocol through the following key solutions:

- **Line-Based Protocol**: Stratum introduced a line-based protocol over plain TCP sockets. Instead of relying on the complexities of HTTP, the communication in Stratum is simplified by sending and receiving messages as lines of text. Each line represents a valid JSON-RPC fragment containing requests or responses.

- **JSON-RPC Encoding**: Stratum encodes the payload of the communication as JSON-RPC messages: JSON provides a lightweight and structured data format that is easy to parse and generate. By utilizing JSON-RPC, Stratum achieves efficient and compact encoding of data, reducing the overall size of the transmitted messages.

- **Direct Socket Connection**: Stratum utilizes direct socket connections between the mining pool server and the miners. This direct communication enables a more efficient data exchange, eliminating the need for the additional overhead and complexities associated with HTTP.

- **Real-time Updates and Push Mechanism**: Stratum introduced a push mechanism for real-time updates. Unlike the previous getwork protocol, where miners had to explicitly request new mining jobs, Stratum allows mining pool servers to proactively push mining jobs to subscribed miners. This eliminates the delay and latency caused by frequent client requests, ensuring that miners are always provided with the correct mining work.

- **Load Management and Broadcast Messages**: Stratum enables mining pool servers to manage the load and send broadcast messages to miners as needed. This eliminates the need for workarounds such as Long Polling and mitigates the issues of load balancing and packet storms that were present in the getwork protocol. The mining server can efficiently control and distribute the workload across the connected miners.

By implementing these solutions, Stratum solves the inefficiencies introduced by using HTTP as the transport protocol. The line-based protocol, combined with JSON-RPC encoding and direct socket connections, reduced the communication overhead and improved the efficiency of data transfer. Real-time updates and the push mechanism allowed for immediate job distribution, eliminating the delays caused by explicit client requests.

To overcome the second getwork inefficiency, Stratum protocol introduced the concept of an **extranonce** field. The extranonce field is a mutable portion of the coinbase transaction in the block template that miners can modify during the mining process. By allowing miners to modify the extranonce, Stratum expanded the search space for a valid block nonce without requiring frequent job requests to the pool server.
With the extranonce field, miners could vary its value while keeping the rest of the block template unchanged. This effectively increased the available nonce research space, allowing miners to continue their operations for a longer period without needing to request entirely new jobs from the server. Miners could exhaust the nonce space of the original job and then modify the extranonce to start a new search without interrupting the mining process.
By reducing the number of job requests, Stratum greatly improved the efficiency and scalability of pooled mining operations. Miners could engage in uninterrupted mining activity for longer durations, reducing the pressure on the pool server and optimizing network resources. In summary, the introduction of the extranonce field in the Stratum protocol provided miners with a more efficient solution to search for valid block nonces. It minimized the need for frequent job requests to the pool server, enhancing the overall mining experience by improving efficiency and scalability.

Stratum typical messages exchange [34]

In the Stratum protocol, the communication between the mining pool server and the miners involves a typical message exchange that follows a specific pattern.

1. **Connection Setup**: the miner establishes a TCP socket connection with the mining pool server. This connection is typically initiated on a specific port designated for Stratum communication, which typically is 3333.

2. **Subscription message**: upon successful connection, the miner sends a subscription request to the server. This request is sent as a JSON-RPC message and tells the server that the miner wants to subscribe to mining work.

   ```
   {"id": 1, "method": "mining.subscribe", "params": ["user agent/
   version", "extranonce1"]}
   ```

   The optional second parameter specifies a mining.notify subscription id the client wishes to resume working with (possibly due to a dropped connection). If provided, a server may issue the connection the same extranonce1.
   The server responds to the subscription request with a subscription response, providing the miner with necessary details such as the mining job, extranonce, and other relevant information.

   ```
   {"id": 1, "result": [[["mining.set_difficulty","subscription id
   1"],["mining.notify","subscription id 2"]], "extranonce1", "ext
   ranonce2_size"], error: null}
   ```

   The result contains three items:

   - **Subscriptions details**: 2-tuple with name of subscribed notification and subscription ID.
   - **Extranonce1**: Hex-encoded, **per-connection unique** string which will be used for coinbase serialization later.
   - **Extranonce2_size**: Represents the length of extranonce2 which will be generated by the miner.

3. **Authorization message**: after receiving the subscription response, the miner may need to provide authorization details, such as a username and password, to the server.

   ```
   {"id": 2, "method": "mining.authorize", "params": ["username",
   "password"]}
   ```

The server validates the authorization details and responds with an authorization result:

```
{"id": 2, "result": true, "error": null}
```

The result indicates whether the authorization was successful or not.

4. **Notify message**: once the miner is subscribed and authorized, it can start requesting mining jobs from the server.

```
{"id": null, "method": "mining.notify", "params": ["job_id", "pre
vhash", "coinb1", "coinb2", "merkle_branches", "version", "nbits",
"ntime", "clean_jobs"]}
```

Description of the notification field in the order:

- **job_id** - ID of the job. Use this ID while submitting share generated from this job.
- **prevhash** - Hash of previous block.
- **coinb1** - Initial part of coinbase transaction
- **coinb2** - Final part of coinbase transaction.
- **merkle_branch** - List of hashes, will be used for calculation of merkle root. This is not a list of all transactions, it only contains prepared hashes of steps of merkle tree algorithm.
- **version** - Bitcoin block version.
- **nbits** - Encoded current network difficulty
- **ntime** - Current ntime
- **clean_jobs** - When true, server indicates that submitting shares from previous jobs don't have a sense and such shares will be rejected. When this flag is set, miner should also drop all previous jobs, so job_ids can be eventually rotated.

With the mining job details obtained from the job notification, the miner performs the mining calculations using its hashing power to search for a valid block nonce. The miner **modifies the extranonce2** field within the coinbase transaction to expand the search space and increase the chances of finding a valid nonce.

Details about how to build the coinbase transaction with data received from pool server, and how to assemble the block header to start mining on, will be explained in the next paragraph called "How to build Coinbase Transaction and Block Header".

5. **Set difficulty message**: the mining pool server can adjust the difficulty required for miner shares with the "mining.set_difficulty" method.

```
{"id": null, "method": "mining.set_difficulty", "params": [2]}
```

This means that difficulty 2 will be applied to every next job received from the pool server.

6. **Submit Share message**: If the miner successfully finds a valid share nonce, it submits a share to the server for verification and potential inclusion in the blockchain. The miner sends a share submission request to the server.

```
{"id": 4, "method": "mining.submit", "params":["username", "job_
id","extranonce2", "ntime", "nonce"]}
```

Values in particular order: worker_name (previously authorized), job_id, extranonce2, ntime, nonce.

The server processes the share submission and responds with a share submission result:

```
{"id": 4, "result": true, "error": null}
```

The result indicates whether the submitted share was accepted or rejected by the server.

How to build Coinbase Transaction and Block Header

Once the miner has received all the necessary data to serialize the coinbase transaction, including Coinb1, Extranonce1, Extranonce2_size, and Coinb2, the process of constructing the coinbase transaction can begin.

The following steps outline the procedure:

1. **Generate Extranonce2**: The miner needs to generate Extranonce2, which should be unique for each job_id. The Extranonce2_size parameter specifies the expected length of the binary structure. It is crucial to ensure that the Extranonce2 generator always produces an Extranonce2 with the correct length. For example, if the Extranonce2_size is set to 4, a valid Extranonce2 in hexadecimal format would be: 00000000.

2. **Concatenate Components**: To build the coinbase transaction, the miner concatenates the following components together in the specified order: Coinb1 + Extranonce1 + Extranonce2 + Coinb2. This concatenation creates a cohesive coinbase transaction structure.

With the necessary components at hand, the final step is to construct the block header to mine on.

The following process describes the steps involved:

1. **Concatenate Components**: Combine the following components in the specified order to build the block header for hashing: version + prevhash + merkle_root + ntime + nbits + ’00000000’ + ’000000800000000000000000000000000000000000000000000000000000000000000000000000000000000080020000’.

   The initial zeroes represent the blank nonce, followed by padding to uint512, and the latter part remains constant for all block headers.

2. **Reversed Byte Order**: Ensure that the merkle_root component is in reversed byte order.

Stratum (V1) protocol real interaction

This section contains a real log of the communication between miner and pool server which solved the testnet block with hash equal to: 000000002076870fe65a2b6eeed84 fa892c0db924f1482243a6247d931dcab32. (https://blockstream.info/testnet/block/ 000000002076870fe65a2b6eeed84fa892c0db924f1482243a6247d931dcab32)

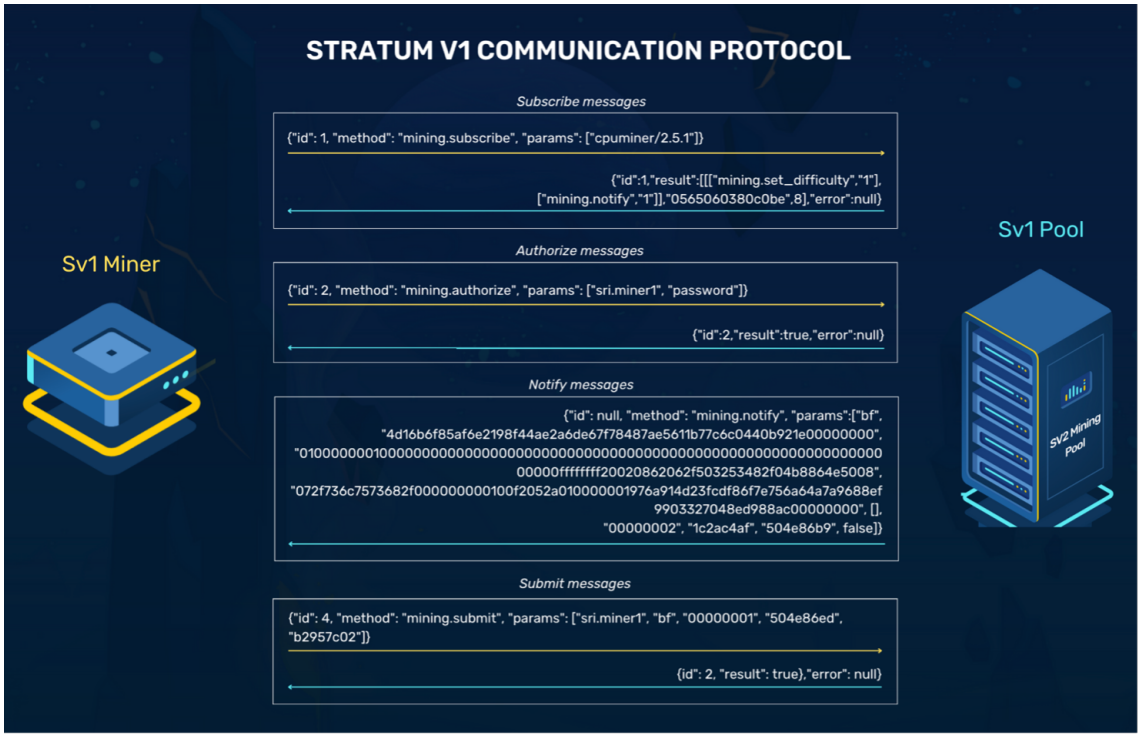


Figure 4.12: Real communication between miner and pool server

4.3.3 Stratum (V1) vulnerabilities and security issues

Stratum protocol, as we've seen, dramatically increased the performances of pooled mining operations, being effectively an efficient, robust, and scalable communication protocol. It introduced a very different approach to pooled mining, giving the responsibility to drive the load and to distribute jobs for the miners to the mining pool operators.
By the way, it was developed in 2012: at that time the Bitcoin network difficulty was around 3.000.000 (global hashrate was 13 TH/s, at the time of writing it's 342 EH/s). As already said, the main purpose who took to the development of Stratum protocol, was to find a valid alternative to the previous getwork, given the fact that newest mining ASIC equipment was arriving at that time.
However, it's important to note that the security aspects of the Stratum protocol were not the primary focus during its development. At that time, the overall hashrate was relatively low compared to the present scenario, and considerations regarding encryption or secure communication were not extensively taken into account. As a result, the Stratum protocol was built with plaintext transmission for all protocol messages, without any encryption mechanisms to protect against potential security threats.

Blackhat Asia 2021 - hashrate stealing attacks

In 2021, during the Blackhat Asia event, a group of researchers (Xin Liu, Rui Chong, Yuanyuan Huang, Yingli Zhang, Qingguo Zhou) demonstrated how they succeeded in stealing some hashrate secretly, exploiting the plaintext communications present in the Stratum protocol [35].

Before entering in the details of the two attacks which they discovered, a brief reminder of the **coinbase transaction** data, the **set.extranonce** message, and **client.reconnect** message is needed.
The coinbase transaction is not sent entirely by the mining pool server to the miner, but it's constructed by the miner once every needed information is received. After the **subscription request** sent by the miner, the mining pool server answers with a message containing the **extranonce1** (which is different for every connection) and the **extranonce2 size**. Using the **notify message**, instead, pool server sends to the miner the other components needed to build the entire coinbase transaction: the

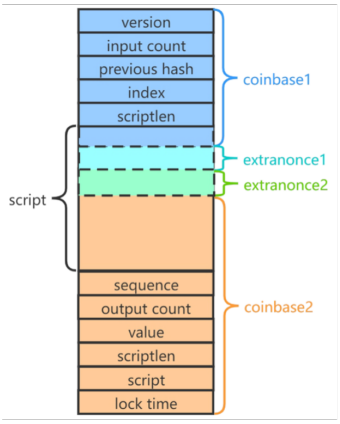


Figure 4.13: Coinbase tx details

so called **coinbase1**, and **coinbase2** data.
The **set.extranonce** message, instead, it's used from the mining pool server to replace the initial subscription values beginning with the next **mining.notify** job.

```
mining.set_extranonce("extranonce1", extranonce2_size)
```

The **client.reconnect** message, instead, can be used by mining pool server to ask for a re-connection to the miner, and the syntax is:

```
client.reconnect("hostname", port, waittime)
```

The attacks which will follow, exploit specifically this set.extranonce message, since it can be used to redirect hashrate of the miner to another "malicious" mining pool used by the attacker. The precondition of the two attacks described by the researchers, is using MITM strategies to hijack the communication between the miner and the mining pool connected. At the same time, the adversary opens a TCP connection to another "malicious" mining pool, which will be used to redirect hashrate to in the next steps.

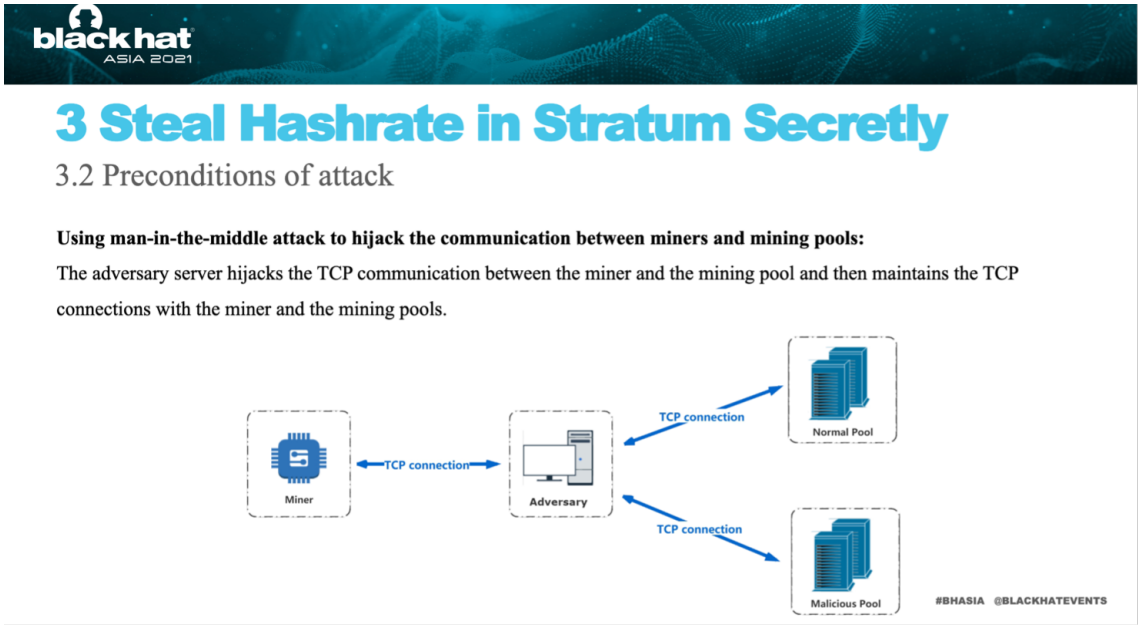


Figure 4.14: Preconditions of attack, *Blackhat Asia 2021*

Given this precondition, the research group studied and analyzed two different possible attacks:

- **Job injection based on set_extranonce**
- **Time segment**

Job injection based on set_extranonce

In the first attack scenario, basically the adversary firstly collects the subscription response of the two mining pool servers, saving locally the two couple of (entranonce1, extranonce2_size).
At this point, the attacker sends to the miner the correct pool data, and he transfers all the future messages without changing them.
In the moment in which the adversary wants to steal the miner hashrate, he sends a set.extranonce message in which is put the "malicious" pool data (entranonce1, extranonce2_size). Doing this, the miner will start working for the mining pool chosen by the attacker, without noticing it.

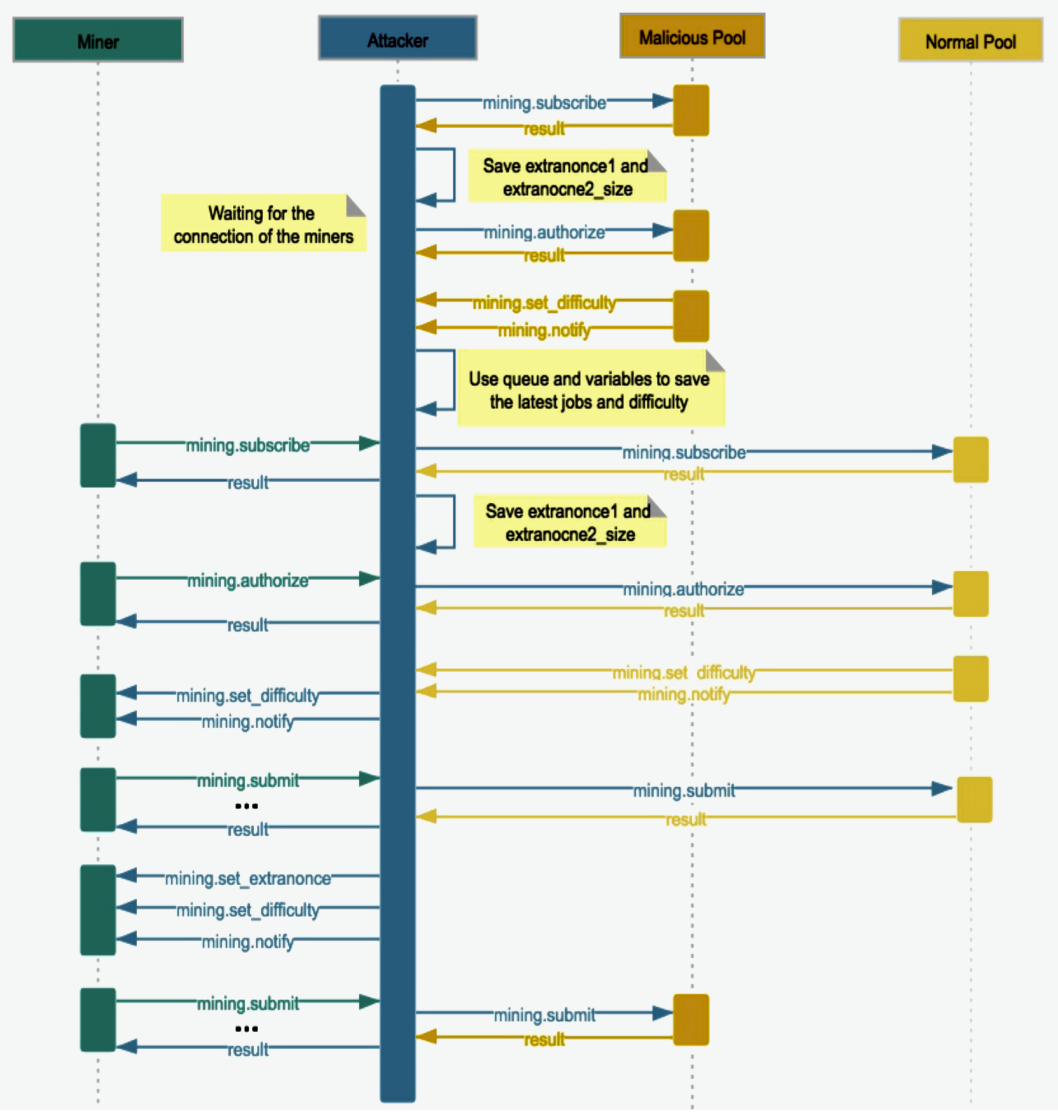


Figure 4.15: Job injection based on set_extranonce, *Blackhat Asia 2021*

Time segment

Regarding the second attack documented by the research group, it has some similarities to the previous one, but this time the client.reconnect message is used to ask the miner a re-connection to the "malicious" pool server, after a specific time segment.

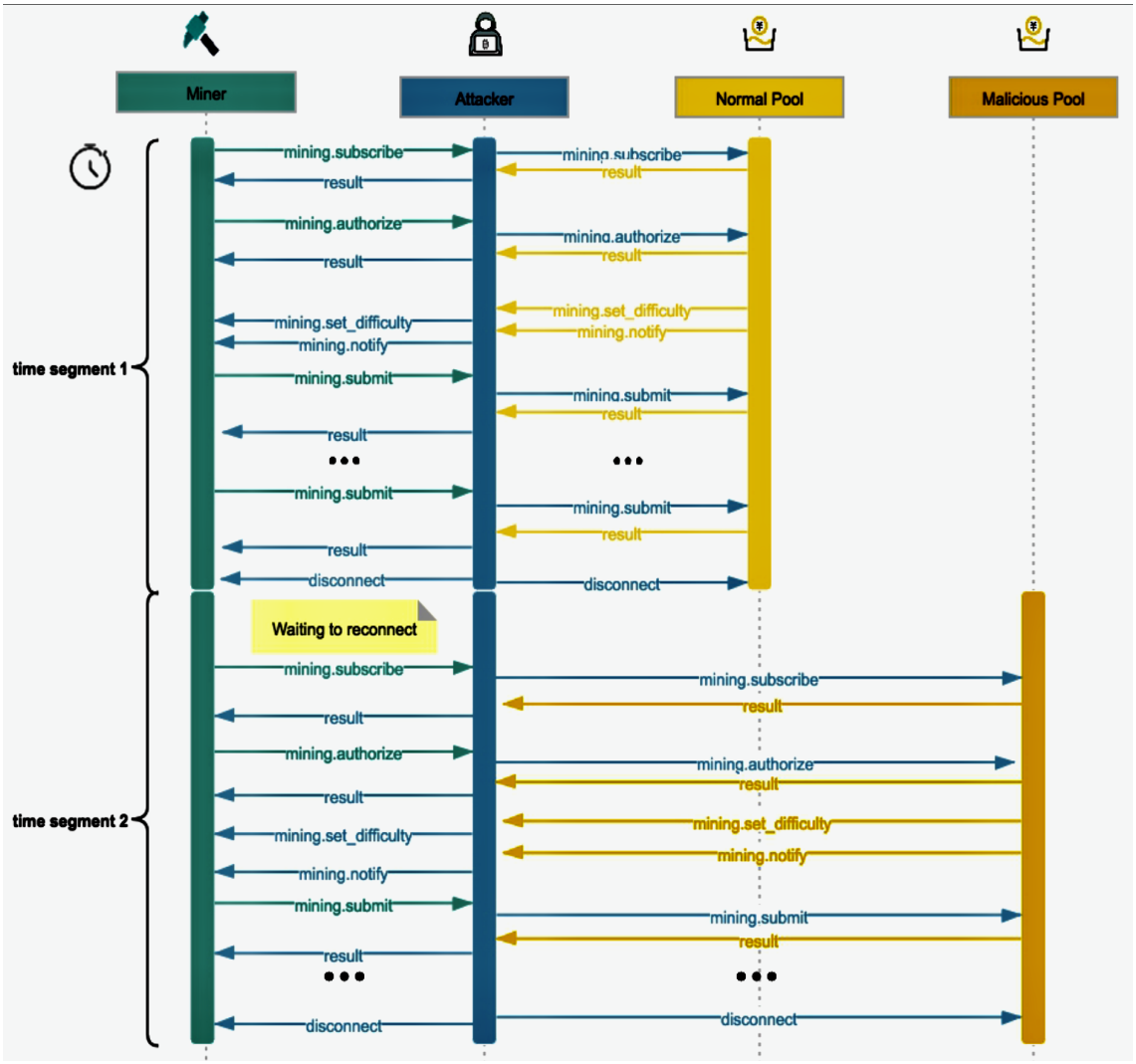


Figure 4.16: Time segment attack, *Blackhat Asia 2021*

The job injection based on the set_extranonce attack model provides better hiding aspect due to its ability to insert a small number of "malicious" mining pool jobs into the miner's job flow at a low frequency. This makes it difficult for the mining pool operator and the miner to detect the presence of the attack.
In the second attack model, the connection between the legitimate pool and the

"malicious" pool is switched within specific time segments. In this way, the mining pool administrator may observe fluctuations in the overall computing power.

Both of the attack schemes described above are designed with the intention of illicitly stealing part of the miner hashrate, and both of them perfectly work. To investigate more about the Proof of Concept done by the above-mentioned research group, some live-demonstration videos are available on Youtube. [36]

4.3.4 Why Stratum (V1) needs to be updated

In conclusion, Stratum (V1) protocol, introduced in 2012, significantly improved the performances of pooled mining operations by efficiently distributing jobs to miners and managing in a brilliant way the load on the mining pool servers. However, as described in the previous sub-chapters, its development was primarily focused on performance over security. To better resume the overall evaluation of the protocol which became the standard "de facto" in the pooled mining context, it's better to analyze its pros and cons.

Pros:

1. **Efficiency**: Stratum (V1) has demonstrated high efficiency and scalability in managing pooled mining operations. It effectively distributes jobs to miners, optimizing the overall mining process.

2. **Easy implementation**: the simplicity of the Stratum (V1) protocol made it relatively easy to be implemented and integrated into mining software, firmware, and hardware. This has contributed to its widespread adoption and compatibility across different mining setups.

3. **Wide adoption**: Stratum (V1) has been widely adopted in the Bitcoin mining industry. Its widespread usage has led it to be the "de-facto" standardized communication protocol, allowing miners to easily connect with various mining pools.

Cons:

1. **Security vulnerabilities**: Stratum (V1) lacks crucial security features. The protocol relies on plaintext communication, making it sensible to attacks such as hashrate stealing.

2. **Privacy concerns**: the absence of encryption in Stratum V1 exposes sensitive information, including plaintext mining pool subscriptions and job data. This compromises miners' privacy and makes their activities easily traceable.

3. **Limited authentication**: Stratum (V1) lacks robust authentication mechanisms, making it vulnerable to man-in-the-middle attacks. This increases the risk of miners connecting to malicious or untrusted pools.

4. **Data bandwidth**: the payload of Stratum (V1) messages is encoded JSON-RPC, so it can be more efficient, saving precious bandwidth during mining operations.

5. **Centralization risks**: since the transactions selection is delegated to the mining pool servers, Stratum (V1) contributes to the centralization of mining power as mining pool operators hold significant control and responsibility in job distribution. This concentration of power raises concerns regarding network resilience, decentralization, and censorship-resistance of the entire network.

To address these security and efficiency concerns, and provide a more optimized protocol, the development of Stratum V2 became necessary. Stratum V2 aims to fix the vulnerabilities of its predecessor by introducing encryption and other security mechanisms. It focuses on enhancing the security, privacy, and efficiency of pooled mining operations. By incorporating new sub-protocols like job negotiation, encrypted communication channels, and binary framing, Stratum V2 aims to provide a more secure, efficient, and decentralized framework, for miners and mining pool operators.

Chapter 5

Stratum V2

5.1 What is SV2: why and when it was born

Stratum V2 was initially proposed in the year of 2019. It was introduced by Pavel Moravec and Jan Čapek (the two founders of the company called Braiins), in collaboration with Matt Corallo and other experts in the mining field.
Stratum V2 was proposed with a specific purpose in mind: to address the limitations and shortcomings (analyzed in the previous chapter) of its predecessor, Stratum V1. The introduction of Stratum V2 aimed to overcome the inefficiencies, lack of security measures, and inadequate performance associated with the JSON-based Stratum V1 protocol.

Figure 5.1: Pavel Moravec, Jan Čapek and Matt Corallo, co-authors of SV2 specs

As the Bitcoin mining industry continued to mature and expand, there was a growing need for a more efficient and secure solution. Stratum V2 was proposed to meet these evolving demands, offering a precise and well-defined protocol for pooled mining operations. By incorporating authentication, optimizing data transfers, and enhancing security against potential attacks, Stratum V2 aims to provide a more streamlined, robust, and reliable framework for miners, proxies, and pool operators. To encapsulate the principal aspects of its predecessor, Stratum V1, which SV2 aims to address, they can be classified into four categories:

- **Security concerns**
  As described in the previous chapter, no security measures against MITM attacks are takled, by protocol. In addiction to this, no strong authentication mechanism are considered in the Stratum (V1) protocol.

- **Data encoding inefficiencies**
  Messages payload in Stratum (V1) is JSON-encoded: it has been revelead a very winning technique at the time of its announcement (2012), due to its simplicity for debugging and implementation purposes. However, JSON is not the most optimal method for encoding specialized data, compared to more compact binary protocols.

- **Transaction selection centralization**
  In Stratum (V1) protocol, the mining pool server is responsible for selecting which transactions are included in the block that miners are trying to solve. This means that the mining pool operators have control over which transactions are prioritized and included in the block's transaction list. This can undermine the censorship resistance of the network.

- **Non-standardization**
  The Stratum (V1) protocol was announced by Marek "Slush" Palatinus in 2012, and it received criticism for its lack of community-centeredness. Additionally, it suffers from inadequate documentation and a lack of standardized shared specifications. For example, biggest mining farm typically use some custom proxies which are helpful in the connections aggregation, but they are not standardized someway by the protocol.

- **Lack of flexibility**
  Stratum (V1) is considered to be a relatively simple and basic protocol. However, it lacks built-in mechanisms for easy upgrades or extensions. This can make it challenging to introduce new features, improve security, or address emerging issues.

In the next section, there will be deeply explanations about the differences between Stratum V2 and Stratum (V1), focusing on the enhancements brought by Stratum V2, to specifically solve the above-mentioned issues relative to Stratum (V1).
To dig more into this first overview of the main Stratum (V1) issues and how Stratum V2 aims to solve them, it's recommended to listen the interview to the co-authors of the protocol available on Youtube [37].

5.2 How SV2 works

In the previous section have been analyzed the goals of Stratum V2, especially in regards to the inefficiencies related to its predecessor.
Before entering into the detailed differences between SV1 and SV2, it's necessary to provide a concise explanation about the new **sub-protocols**, **roles**, and **channel types**, introduced and standardized by the Stratum V2 protocol specifications.

Roles

The roles involved in data flow can be classified as either downstream or upstream in relation to each other. Here are the roles and their respective classifications:

- **Mining Device**
  The mining device is the physical hardware that carries out the hashing process. It is considered the most downstream role.

- **Pool Service**
  This role belongs to the entity where the actual hashrate produced by the mining devices is consumed. It is considered the most upstream role.

- **Mining Proxy**
  This role represents the proxy server situated between the mining device and the pool service. It handles message coordination and aggregation. In relation to the mining device, it is considered upstream, while in relation to the pool service, it is considered downstream.

- **Job Negotiator**
  The job negotiator receives transactions from the Template Provider (which is essentially the Bitcoin client) and constructs custom block templates. It also negotiates the use of these templates with the pool.

- **Template Provider**
  This role is fulfilled by a Bitcoin client responsible for generating custom block templates. These templates are then sent to the Job Negotiator for mining purposes.

Sub-protocols

To fulfill the goals related to pooled mining operations efficiency, decentralization of the transaction selection process, and the other aspects previously declared,

Stratum V2 had to introduce some new sub-protocols. Regarding the main mining protocol, new types of communication channels were standardized.

- **Mining Protocol**
  The mining protocol is the primary protocol used for mining and serves as the direct successor to Stratum (V1). It enables communication between a mining device and its upstream node, pool, or proxy. This protocol is essential and must be implemented in all mining scenarios. In cases where a miner or pool does not support transaction selection, the mining protocol is the only protocol used.
  The protocol defines three types of communication channels:
  - **Standard channels**: they don't manipulate the Merkle path / coinbase transaction, greatly simplifying the communication required between them and upstream nodes.
  - **Extended channels**: they are given extensive control over the search space so that they can implement advanced use cases (e.g. translation between V1 and V2, difficulty aggregation, custom search space splitting, etc.).
  - **Group channels**: they are simply collections of standard channels that are opened within a particular connection so that they are addressable through a common communication channel.
- **Job Negotiation Protocol**
  The job negotiation protocol is utilized by a miner, typically a mining farm, to negotiate a block template with a pool. The results of this negotiation can be reused for all mining connections to the pool, reducing computational intensity. In other words, a single negotiation can be applied to an entire mining farm or even multiple farms with a large number of devices, leading to greater efficiency. This protocol is separate to allow pools to terminate these connections on separate infrastructure from mining protocol connections.
- **Template Distribution Protocol**
  The template distribution protocol shares a similar structure to facilitate obtaining information about the next block from Bitcoin Core. It is designed to replace *getblocktemplate* with a more efficient and easier-to-implement solution for those incorporating other aspects of Stratum V2 into their systems.

To better understand the list of differences between SV1 and SV2 which will follow, a preface with some more deep aspects which regard the SV2 protocol choices in term of protocol security, binary framing, and censorship resistance enhancements is needed.

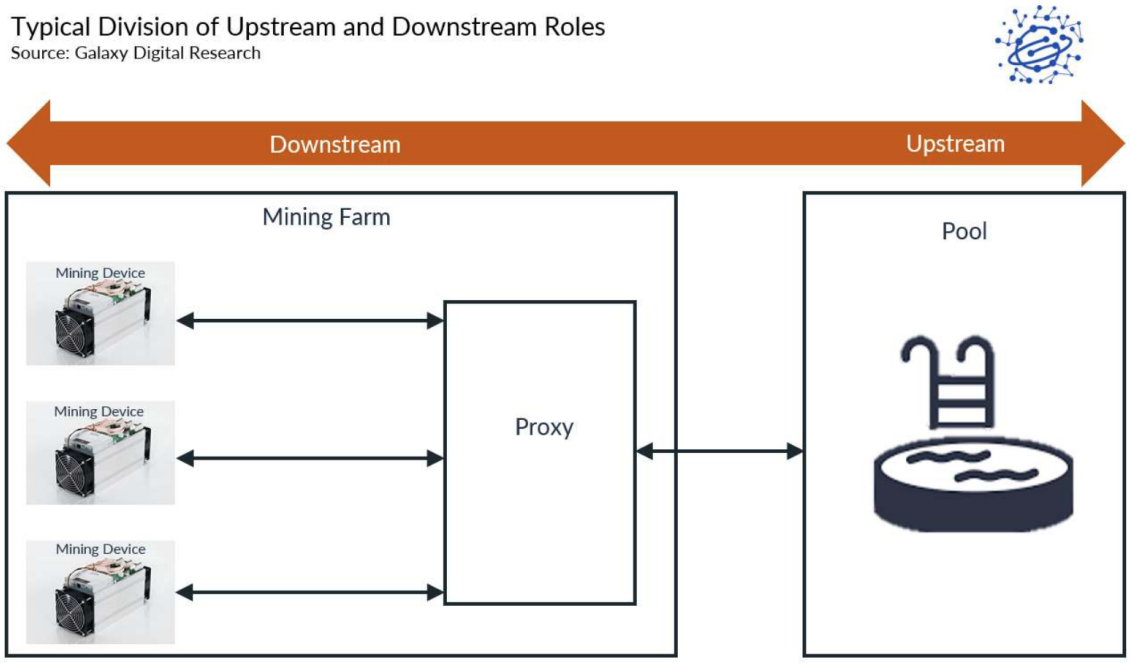


Figure 5.2: Typical Division of Downstream and Upstream Roles, *Galaxy Digital Research* [5]

SV2 protocol security

«Stratum V2 employs a type of encryption scheme called AEAD (authenticated encryption with associated data) to address the security aspects of all communication that occurs between clients and servers. This provides both confidentiality and integrity for the ciphertexts (i.e. encrypted data) being transferred, as well as providing integrity for associated data which is not encrypted. Prior to opening any Stratum V2 channels for mining, clients MUST first initiate the cryptographic session state that is used to encrypt all messages sent between themselves and servers. Thus, the cryptographic session state is independent of V2 messaging conventions. At the same time, the SV2 protocol specification proposes optional use of a particular handshake protocol based on the Noise Protocol framework [38]. The client and server establish secure communication using Diffie-Hellman (DH) key agreement, as described in greater detail in the Authenticated Key Agreement Handshake section of the specifications document.
Using the handshake protocol to establish secured communication is optional on the local network (e.g. local mining devices talking to a local mining proxy). However, it is mandatory for remote access to the upstream nodes, whether they be pool mining services, job negotiating services or template distributors.» [39]

SV2 binary framing

The Stratum V2 protocol is binary, with fixed message framing. Each message begins with the extension type, message type, and message length (six bytes in total), followed by a variable length message. Figure 5.3 describes the message framing used by the protocol.

Field name	Byte Length	Description
extension_type	U16	Unique identifier of the extension describing this protocol message. Most significant bit (i.e.bit 15, 0-indexed, aka channel_msg) indicates a message which is specific to a channel, whereas if the most significant bit is unset, the message is to be interpreted by the immediate receiving device. Note that the channel_msg bit is ignored in the extension lookup, i.e.an extension_type of 0x8ABC is for the same "extension" as 0x0ABC. If the channel_msg bit is set, the first four bytes of the payload field is a U32 representing the channel_id this message is destined for (these bytes are repeated in the message framing descriptions below). Note that for the Job Negotiation and Template Distribution Protocols the channel_msg bit is always unset.
message_type	U8	Unique identifier of the extension describing this protocol message
message_length	U24	Length of the protocol message, not including this header
payload	BYTES	Message-specific payload of length message_length. If the MSB in extension_type (the channel_msg bit) is set the first four bytes are defined as a U32 "channel_id", though this definition is repeated in the message definitions below and these 4 bytes are included in message_length.

Figure 5.3: SV2 protocol binary framing

SV2 transaction selection

In section 4.3, related to Stratum (V1), it's well explained how the current pooled mining operations work. Today, the selection of the transactions to be inserted in the block templates, which are distributed in the form of jobs to the miners, is a pool responsibility. The only entities who took valid Bitcoin transactions from the mempool and decide the ones who will be mined in the next block are the pool operators. Since pool are public entities, they can be attacked from governments, and this is not so ideal for the censorship-resistance property of the Bitcoin network as a whole.
With Stratum V2, miners now have the ability to choose their own work (i.e. choose their own transaction set), making mining process more decentralized. This is implemented separately from the main mining protocol, as described previously, and it is optional for pools and miners. In Figure 5.4, it's very clear the benefits that will be brought by having miners (single miners, mining farms, etc.) selecting transactions from their local bitcoin node's mempool, and negotiating their own block template with the pool, instead relaying on the pool responsibility.

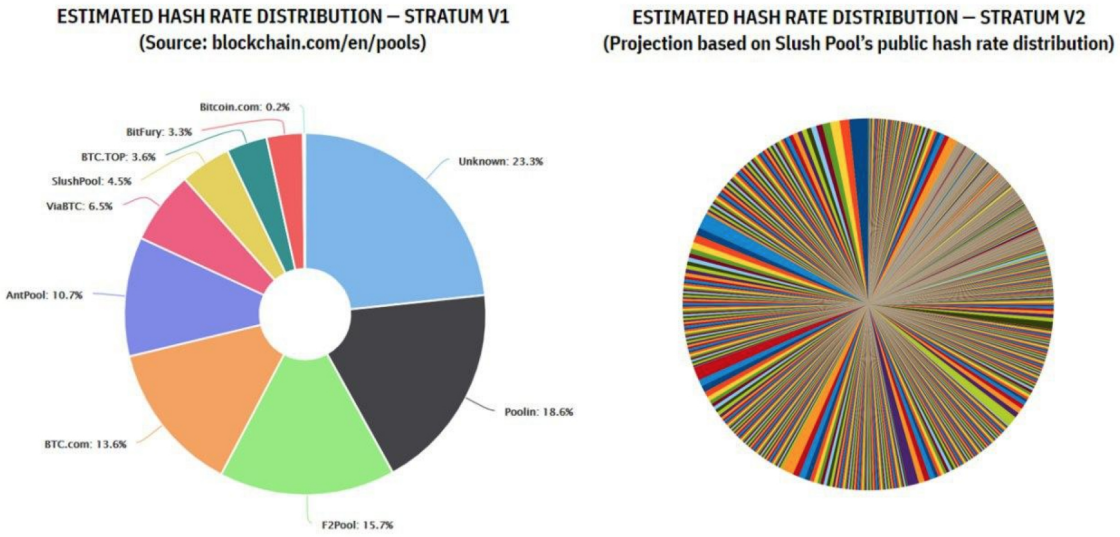


Figure 5.4: Transaction selection decentralization brought by Stratum V2 [6]

5.3 Differences between SV1 and SV2

This section aims to explore and analyze the key differences between Stratum V1 and Stratum V2, highlighting the advancements and improvements introduced by the latter. It will follow the public SV2 documentation from Braiins, summarizing the most interesting SV2 enhancements [40].

Bandwidth consumption

The use of a binary protocol instead of a text-based one significantly reduces bandwidth usage. In Stratum V1, making messages readable by humans resulted in some messages being unnecessarily large, approximately 2-3 times bigger than needed. However, in V2, these messages have been minimized to their essential size. Furthermore, V1 includes certain unnecessary messages like mining.subscribe, which have been eliminated in V2.

Stratum V2	Stratum V1
V2 typical share submission message is 32 bytes without encryption and 48 with it.	V1 typical share submission message is approximately 100 bytes

Server CPU load

Thanks to the introduction of standard and group channels, Stratum V2 achieves

a reduction in server CPU load by implementing header-only mining for end devices (it will be explained later). This implies that the Merkle root, which was previously handled by end devices, is now always provided by an upstream node. Consequently, end devices are lighter since there's no need to perform any coinbase modifications. This simplifies the computational tasks for miners and, at the same time, significantly lightens the workload required for work validation (i.e. CPU load) on the server side.

Header-only mining
As anticipated in section 5.2, Stratum V2 introduces the possibility for miners to open standard mining channels that don't permits coinbase transaction manipulation. In other words, end mining devices don't do any extranonce or Merkle path handling. This process if called **header-only mining**. The size of the search space for a device doing header-only mining for a particular value in the nTime field is $2^{NONCE\_BITS+VERSION\_ROLLING\_BITS}$ = 280Th, where NONCE_BITS = 32 and VERSION_ROLLING_BITS = 16. This is a guaranteed search space before nTime rolling. The client that opens a particular standard channel owns the entire assigned search space and can split it further (e.g. between multiple hashing boards or individual chips) if necessary.

Binary vs non-binary
As described in 5.2, SV2 has fixed message framing and it is precisely defined, which means that there isn't room for different interpretations of Stratum V2 like there was with V1.
Instead, Stratum V1 protocol relies on JSON, which has a sub-optimal ratio between the size of the message payload and the actual information transmitted. By transforming Stratum V2 into a binary protocol, the data efficiency significantly improves. This enhanced efficiency allows for the saved bandwidth to be allocated towards more frequent job submissions, thereby reducing hashrate variance.

Job distribution latency
In Stratum V1, mining pool servers send jobs to miners containing both the prevhash and Merkle root of the transaction lists to be included in the next block (this is done using the SV1 mining.notify message, as discussed in 4.3.2). So, these two pieces of data aren't separable, so there is a heavy (slow) data transfer necessary to distribute new jobs as soon as a new block (with a new

Stratum V2	Stratum V1
V2 separates the messages, making it possible for the miners to start working on new (full) blocks more quickly after a previous block has been found.	V1 prevhash and future job are part of the same message, so pools send empty blocks.

prevhash) has been found and propagated.
In Stratum V2, it's possible to separate the prevhash from the rest of the predefined block data, which allows for the block data to be sent before a new prevhash is available. As a result, the new prevhash message can be sent on its own as soon as a valid block is found, and this transmission can occur much faster because the message doesn't include heavier data. This enables miners to begin working on new jobs more quickly than they could with Stratum V1.

Man-in-the-middle attack prevention
As analyzed at 5.2, Stratum V2 introduces a type of encryption scheme called AEAD (authenticated encryption with associated data) to address the security aspects of all communication that occurs between miners and pool servers. This provides both confidentiality and integrity for the ciphertexts (i.e. encrypted data) being transferred, as well as providing integrity for associated data which is not encrypted. Stratum V2 uses authenticated encryption with associated data (AEAD) so that possible adversaries will be unable to use share submission data to identify particular miners, thus maintaining the privacy of miners and protecting them against hashrate hijacking.

Empty block mining elimination
Very similarly to the 5.3 point, the elimination of the incentive for empty block mining comes down to the separation of the prevhash message from other block header data. With Stratum V1, there is an incentive for pools to send empty blocks containing the new prevhash as soon as possible, as these messages will arrive faster than a message containing a full block. By separating these two messages in Stratum V2, it's now possible for pools to send full blocks to miners before the new prevhash message. In other words, the miners can be prepared to start working on a new (full) block before the previous block has been found, and then all they need is the new prevhash message to begin working on that next block. Since this prevhash message is the same size (i.e. takes the same amount of time to arrive) regardless of whether or not the pool has sent an empty block or a full block, there is no longer an incentive to mine on empty blocks.

Job selection
Job selection by end miners has been included as an optional component of Stratum V2, separate from the main mining protocol. The name Job 'Negotiation' Protocol is telling, as job selection is indeed a negotiation process between a miner and a pool. The miner proposes a block template, and it is up to a pool to accept or reject it. Once a negotiated template has been accepted, the results can be used by any number of mining devices, even hundreds of thousands of them. The reason this is separate from the main mining protocol is to allow pools to terminate connections on separate infrastructure from the main mining protocol, that way there is no

impact on the efficiency of actual share submissions.

Multiplexing
In SV2, there can theoretically be as many as 2^{32} (around 4.3 billion) open channels within a single physical connection (e.g. TCP) to an upstream stratum node. These channels are independent and have unique channel IDs, meaning that many devices can simultaneously receive different job assignments using the same connection, saving on infrastructure costs. At the same time, the channels may all share some information for greater efficiency, such as when a new prevhash is broadcasted.

Native version rolling
Each Bitcoin block header contains a version field whose bits can be freely used to extend the search space for a miner. Rolling the version bits can greatly reduce the frequency with which new jobs need to be distributed, and it's already a common technology (BIP320, https://en.bitcoin.it/wiki/BIP_0320). With SV2, version rolling becomes a native part of the mining protocol.

The features and improvements mentioned earlier are the main additions brought by the Stratum V2 protocol. They are taken from the protocol specifications released in 2019, as mentioned before.
In the next section, it will be explained the current implementations of the SV2 protocol specifications, including the birth of an independent development group, which is focused on standardizing a totally open source and community-based implementation, called SRI (Stratum Reference Implementation).

5.4 Current implementations

BraiinsOS

As explained in the previous sections, Stratum V2 protocol specifications were designed and published in the late 2019. However, the first SV2 implementation appeared in the first half of 2020, announced from the Braiins team, again. They developed a first basic Stratum V2 implementation, which was directly incorporated in their own Braiins OS firmware.
«That's finally ready to change. Today, we are launching a new product that includes a working implementation of Stratum V2 as well as additional autotuning functionality that has strong user demand. The product is an ASIC firmware called Braiins OS, which was the first mining firmware to implement overt AsicBoost back in 2018 and is also the first fully-open source firmware in the industry.» [41].

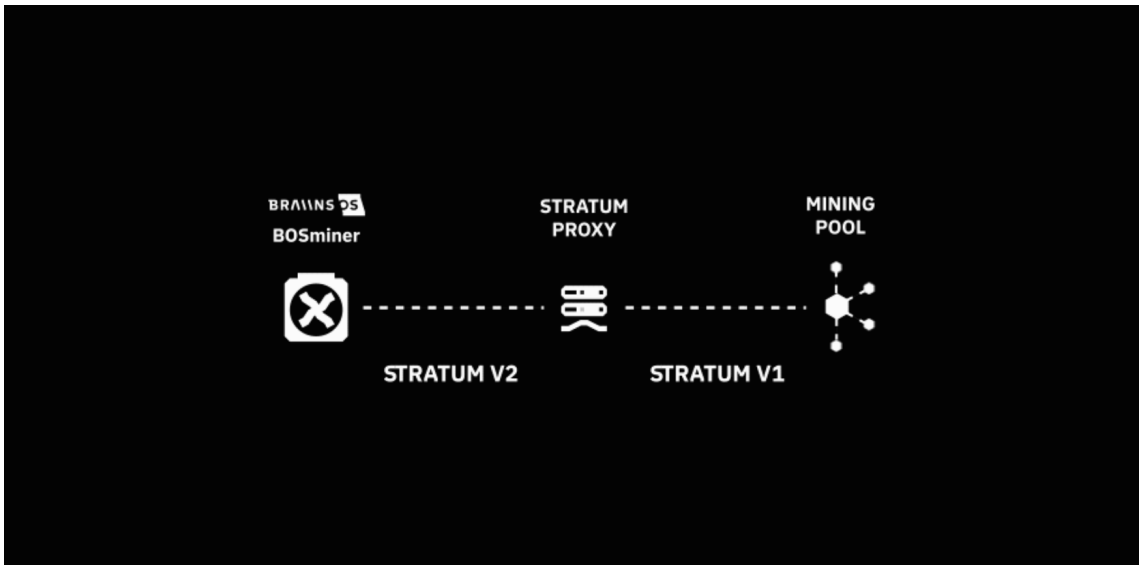


Figure 5.5: First SV2 implementation released by Braiins team, in 2020

Initially, the Braiins team focused on implementing the ASIC protocol logic in the ASIC's firmware, suggesting to use a translation proxy which was in charge of translating the SV2 messages received from the SV2 firmware-miner. As described in 5.5, in that manner the proxy would let the miner communicate correctly with any mining pool which was supporting only Stratum (V1).
In that way, any miner could take advantage of the security and data efficiency brought by Stratum V2 protocol, using an encrypted communication channel and the binary framing defined in the SV2 specs.

However, after some months, the Braiins co-founders (co-authors of the SV2 specifications) decided to pass the future development of the protocol to someone who was

external to their business. Doing this, they declared that the Stratum V2 protocol implementation should have been done by an independent community, without having the responsibility to be linked in some way to the Braiins company itself.

Stratum Reference Implementation (SRI)

For the above-mentioned reason, during 2020 a new group of people composed by independent developers started to work on a fully open-source implementation of Stratum V2, called SRI (Stratum Reference Implementation).
The purpose of SRI group is to build, beginning from the SV2 specifications discussed in the previous section, a community-based implementation, with the aim to discuss and open the development with as many people of the Bitcoin community as possible.

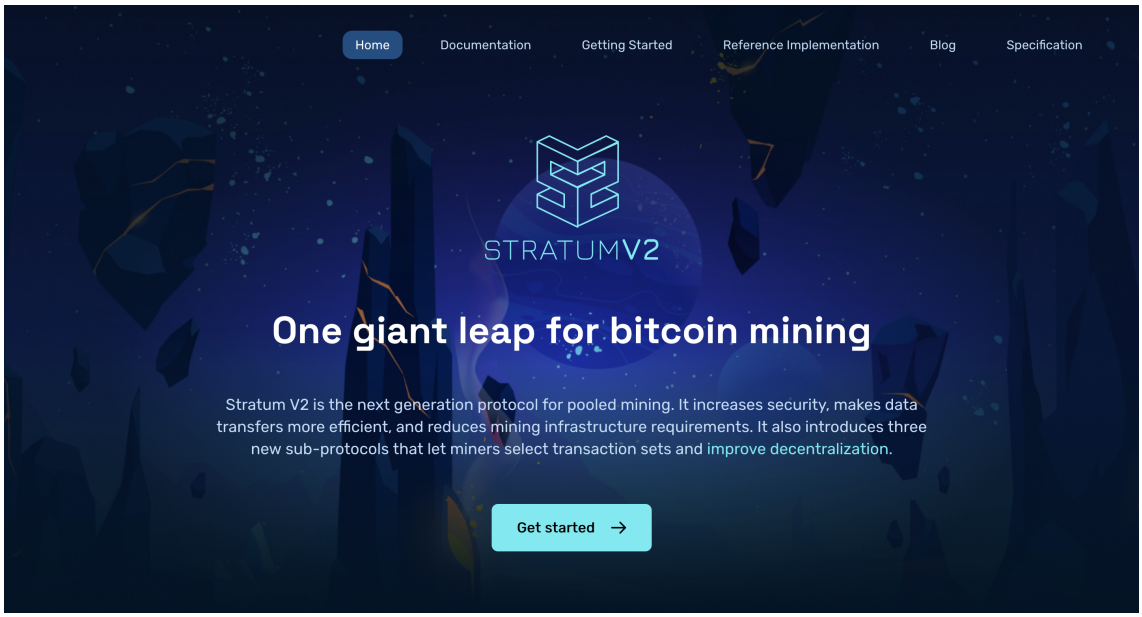


Figure 5.6: SRI homepage on *stratumprotocol.org*

In the last year, SRI group did a great work in expanding the SV2 features already covered by the Braiins implementation.
For example, they shipped the translator proxy, capable of translating SV1 messages coming from a "SV1" miner to the SV2 logic. Some months ago, SRI group developed and announced the so called **job-negotiator**, which permits the transaction selection by miners, further decentralizing the power which is currently in the hands of the mining pools operators: «the new update is "a major milestone in democratizing transaction selections in pooled mining and decentralizing bitcoin," as it allows miners to select transactions via a new sub-protocol and their node.» [42]

In this chapter have been discussed all the major differences between Stratum (V1) and Stratum V2, the enhancements brought by the latter, the new sub-protocols and roles, and the current SV2 implementations.

In the next chapter, the focus will be about the most important adoption path, which is the Stratum Reference Implementation (SRI). There will be a very deep analysis of the four different configuration which it permits, with a practical part in which two of them will be tested on both cpuminers and real ASIC machines.

Chapter 6

Stratum Reference Implementation (SRI)

6.1 How SRI works

As explained in the previous chapter, the Stratum Reference Implementation (SRI), is a full open-source, community based implementation of the Stratum V2 protocol specifications. The team who is building it started some years ago, and it's composed by independent developers majorly funded by individual grants. The project is supported by many companies involved into mining operations, such as Braiins, Foundry, Galaxy Digital. In addiction to them, there are engaged also entities like Bitmex, Human Rights Foundation, Spiral and the Summer of Bitcoin.
Nowadays, most of the implementation work has been done, but there are still some open discussions related to the protocol specifications, such as roles structure, noise encryption, job negotiation/declaration protocol.
However, as previously anticipated, Stratum V2 is a very flexible protocol, especially if compared to the current one used for pooled mining operations (Stratum V1). To fill the lack of flexibility of its predecessor, Stratum V2 introduced some news sub-protocols and roles.

SRI roles

The Stratum Reference Implementation, in particular, provides a well defined set of these new roles, which are contained in the "roles" folder of its Rust codebase [43].

The current repository contains 4 different **roles**:

- **SV2 Pool**
  This role represents a Stratum V2 Pool server. It can open any kind of

communication channels (as defined in 5.2) with downstream roles (proxies or mining devices).

- **SV2 Mining Proxy**
  The SV2 Mining Proxy acts as an intermediary between the mining devices and the SV2 Pool. It receives mining requests from multiple devices, aggregates them, and forwards them to the SV2 pool. It can open group/extended channels with upstream (the SV2 pool) and standard channels with downstream (SV2 Mining Devices).

- **SV2 Mining Device**
  This role represents a conceptual Mining Device written in Rust that is compatible with SRI stack. It can connect to an SV2 Pool or Mining Proxy and performs the mining operations.

- **SV1-SV2 Translator Proxy (+ Job Negotiator)**
  The SV1-SV2 Translator Proxy is responsible for translating the communication between SV1 actual Mining Devices and an SV2 Pool or Mining Proxy. It enables SV1 devices to interact with SV2-based mining infrastructure, bridging the gap between the older SV1 protocol and SV2. It can open extended channels with upstream (the SV2 pool or Mining Proxy)
  If correctly configured, it can act as a Job Negotiator, so it can enable the **transaction selection** feature for the miners which are connected to it.

For what regards the so called **Template Provider** [44], as already said, it enables the extraction of transactions from the Bitcoin nodes which are miner-side. In this way, miners are now able to create custom block templates and negotiate their use with the Job Negotiator via the Job Negotiation Protocol. On June 11, 2023 a first official proposal to add a SV2 template provider natively in Bitcoin Core, has been opened and discussed on the Bitcoin Core repository [45].

SRI configurations

As described in the homepage of the SRI website [46], thanks to all these different roles and sub-protocols, Stratum V2 can be used in many different mining contexts. The SRI working group defined 4 main possible configurations which can be the most probable real use-cases, and they are defined as the following listed.

- **Configuration A**
  As already said, before Stratum V2, transaction sets to be mined in the next blocks were selected by pools. With this SV2 configuration they're selected by individual miners, making the network more censorship-resistant. In this case, miners run SV2 compatible firmware, connecting to the SV2 Mining Proxy. Using the Job Negotiator role, individual miners are able to pick up their transactions locally, extracting them from their local Template Provider, and declare them to an SV2 Pool.

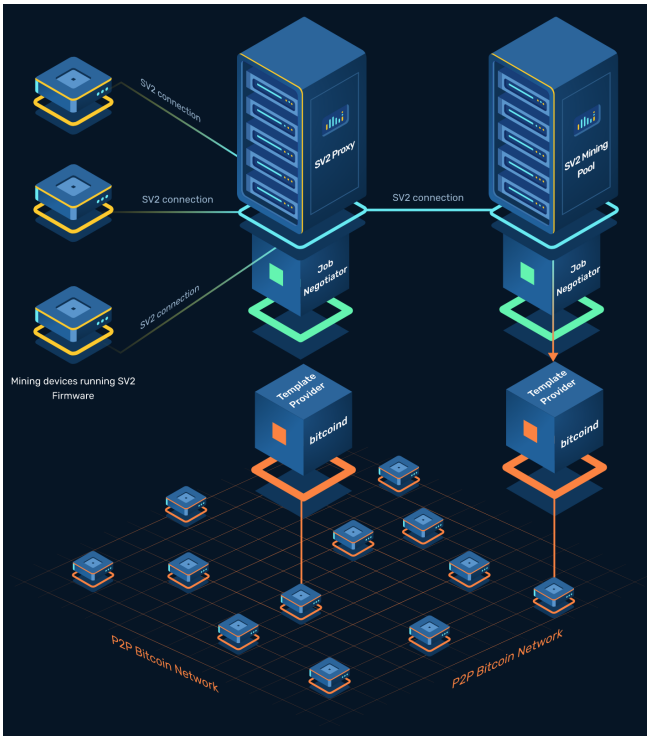


Figure 6.1: SRI configuration A

- **Configuration B**
  Mining Devices run SV2 firmware, so they are able to connect to a SV2 Mining Proxy (typically through a standard channel). The proxy aggregates all the standard channels opened into just one open channel with the SV2 Pool (group channel or an extended channel). In this configuration, the Proxy doesn't have the Job Declarator setup, so it's unable to select transactions from its local Template Provider. Transactions selection is done by the SV2 Pool, as it was done in Stratum V1, but now it can benefit from all the security and performance features brought by SV2.

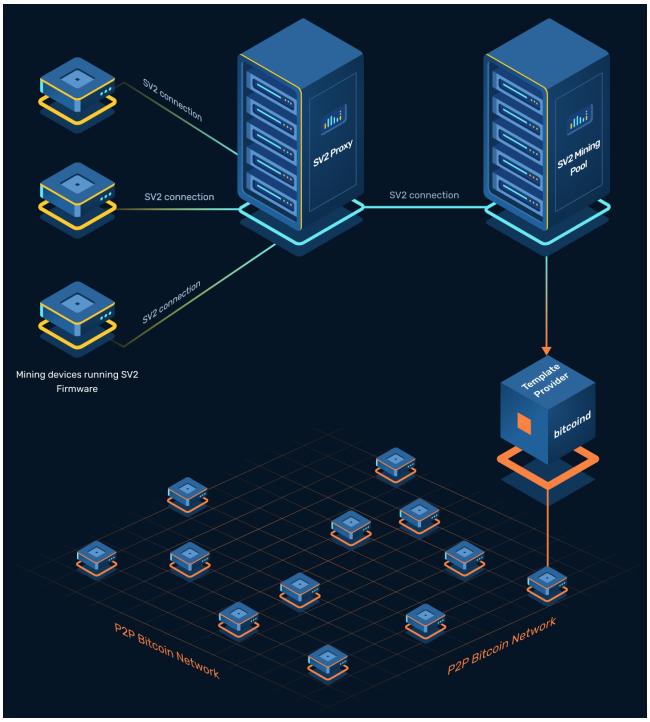


Figure 6.2: SRI configuration B

- **Configuration C**
  With this setup, Mining Devices don't need to run a SV2 compatible firmware. The Proxy which is used to let for efficiency, is also able to translate the SV1 messages that come from the Mining Device into SV2 messages for the SV2 Pool. In this case, the Translator Proxy is not configured to talk to a local Template Provider, so transactions selection is done by the pool. However, this configuration permits to test and use the SV2 protocol features without installing any other SV2 firmware on the machines.

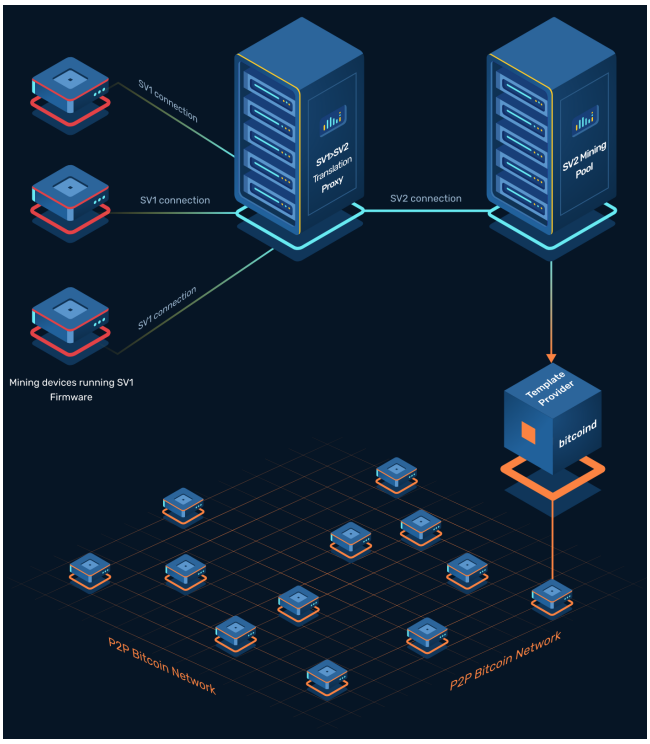


Figure 6.3: SRI configuration C

- **Configuration D**
  This configuration is very similar to the previous (config C), but it's able to add the transactions selection feature to it. As represented in Figure 6.4, the Translator Proxy is joined by a Job Negotiator and a Template Provider: it's able, in this way, to build its own block templates and declare them to the SV2 Pool, through an extended channel.

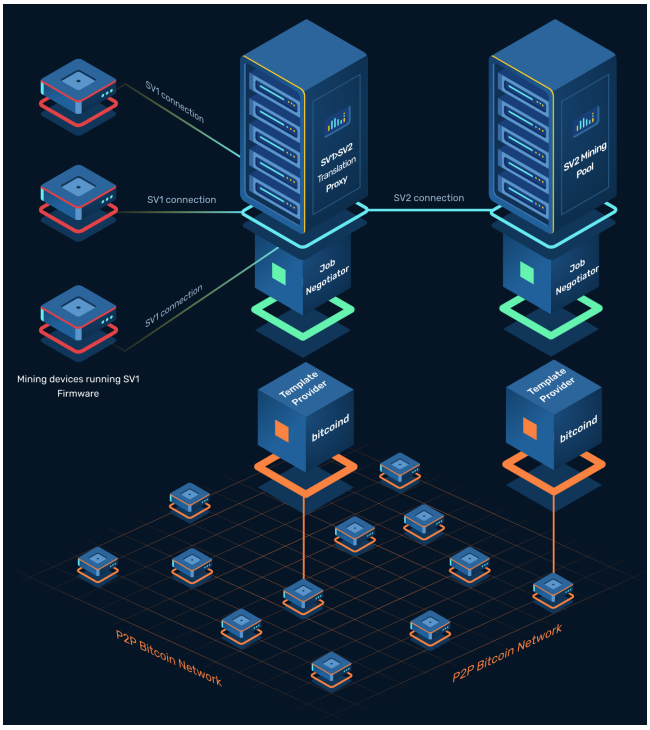


Figure 6.4: SRI configuration D

6.2 Getting Started

In this section, the goal is to explore in details the SRI configurations which are available for testing purposes. Tests will be done on both CPU-miner and real ASIC machine (Antminer S19J Pro), following the official *Getting Started* guide which can be found at [46].

6.2.1 Testing SRI configurations

SRI working group currently developed and shipped all the roles which are needed for every of the 4 configurations introduced in the previous section. However, the ones which will be tested here, are the **Config C and D**, since they are the most ready and documented for testing on real ASIC machines.
To start testing also Config A and B, of course, it's possible to run every individual role, following the indications reported in the README of them (e.g. for the SV2 Mining Proxy, follow the guidelines on the official SRI Github repository).

Prerequisites

Before entering the Configurations details, there are some first-steps that needs to be checked:

- Rust installed: if not, install it by running this command in the terminal:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- Clone the SRI repository locally:

```
git clone https://github.com/stratum-mining/stratum.git
```

At this point everything is correctly setup and ready for the testing phase.

Config C

This configuration, as seen in 6.1, allows Mining Devices running SV1 firmware to connect to a SV2 Pool through a Translation Proxy (tProxy). The proxy is designed to sit in between a SV1 downstream role (most typically Mining Device(s) running SV1 firmware) and a SV2 upstream role (most typically a SV2 Pool Server).
In this case, since there is not intended to be a Template Provider miner-side, transactions selections is delegated to the SV2 Pool server, which is running its local Template Provider, as explained in Figure 6.3.

First of all, once the SRI project repository just cloned is accessed, the **SV2 Pool server has to be run**:

```
cd stratum/roles/v2/pool/
cp pool-config-example.toml ./conf/pool-config.toml
cd conf/
```

Now the example config file called *pool-config.toml* is copied into the */conf* directory. This file contains all the parameters needed for the pool to be correctly configured. It's possible to enter the file and customize it for the most desired behaviour, following the guidelines available in the README file of the pool role. For simplicity, it's possible to use the **testnet** Template Provider which is hosted by the SRI working group: in this way it's not needed to be run a local one. To do that, it's needed to comment the line corresponding to *tp_address = "127.0.0.1:8442"*, and uncomment the one which corresponds to *tp_address = "89.116.25.191:8442"*.

After that, to finally run the SV2 Pool:

```
cargo run -p pool_sv2
```

As described in the command output, the Pool is now connected to the hosted testnet Template Provider, and it will get the transactions to be put in the next block template from it.
Now the Pool is ready to get connection requests, listening on the port 342254.

```
2023-06-26T19:40:50.851514Z  INFO pool_sv2: Pool INITIALIZING with config: "pool-config.toml"
2023-06-26T19:40:50.928044Z  INFO pool_sv2::lib::template_receiver: Connected to template distribution server at 89.116.25.191:8442
2023-06-26T19:40:51.087193Z  INFO pool_sv2::lib::template_receiver::setup_connection: Setup template provider connection success!
2023-06-26T19:40:51.087321Z  INFO pool_sv2::lib::job_negotiator: JN INITIALIZED
2023-06-26T19:40:51.087775Z  INFO pool_sv2::lib::mining_pool: PUB KEY: [TxOut { value: 5000000000, script_pubkey: Script(OP_DUP OP_
HASH160 OP_PUSHBYTES_20 55a2e914aeb9729b4cd265248cb67a865eae95fd OP_EQUALVERIFY OP_CHECKSIG) }]
2023-06-26T19:40:51.087815Z  INFO pool_sv2::lib::mining_pool: Starting up pool listener
2023-06-26T19:40:51.087896Z  INFO pool_sv2::lib::mining_pool: Listening for encrypted connection on: 0.0.0.0:34254
```

Figure 6.5: SV2 Pool connected to the hosted Template Provider, *config C*

At this point, since will be used SV1 Mining Devices, **the Translator Proxy is needed**. In a new terminal:

```
cd stratum/roles/translator/
cp proxy-config-example.toml ./conf/proxy-config.toml
cd conf/
```

Since the configuration file provided is already prepared for the Config C, the Translator Proxy is ready to be run:

```
cargo run -p translator_sv2
```

The Translator Proxy is now running, and as can be seen in the command output, it has requested a connection to the SV2 Pool, asking to open an *extended channel* [47], and it received an *extended mining job*, and a *prev hash*. So it's now ready

to customize the templates that it will distribute to the Mining Devices which will be connected to it!

Note: to learn more about the messages exchanged during this phase, look at:

- *Setup connection* message
- *OpenExtendedMiningChannel* message
- *NewMiningJob* message
- *SetNewPrevHash* message

  All the messages details are well explained in the SV2 mining protocol specifications repository.

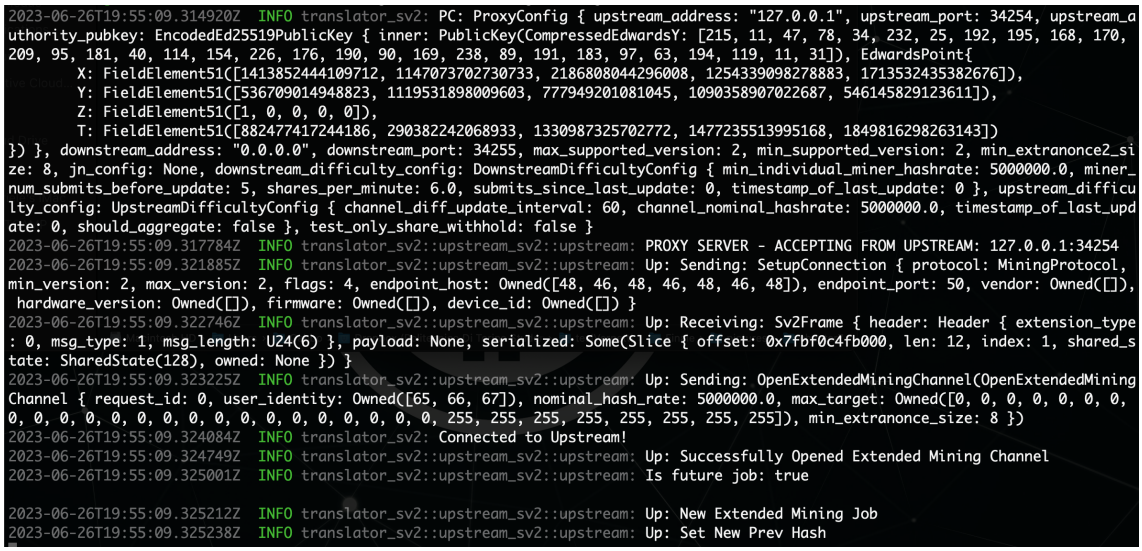


Figure 6.6: Translator Proxy connected to the SV2 Pool, *config C*

Now, the only role which is missing is the **SV1 Mining Device**. There are two alternatives to run it:

- **CPU-miner**
  An open-source SHA-256, multi-threaded CPU-miner for Bitcoin which works following the Stratum (V1) protocol. Once downloaded, in a new terminal:

```
cd Downloads/
./minerd -a sha256d -o stratum+tcp://localhost:34255 -q -D -P
```

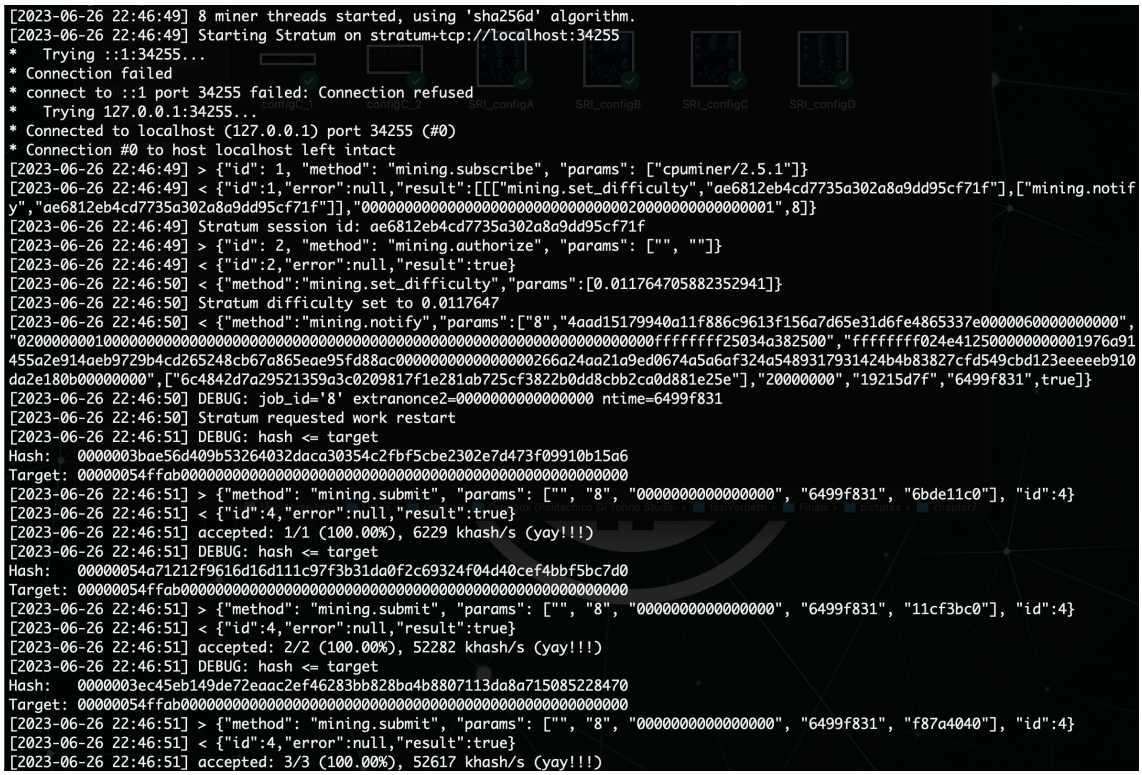


Figure 6.7: SV1 CPU-miner running, using the Translation Proxy, *config C*

As described in the Figure 6.7, the SV1 CPU-miner is running correctly: it connected to the Translator Proxy on port 34255, doing the subscription and receiving a new job with the *mining.notify* SV1 message. Then, it started working and sending shares to the tProxy, with the *mining.submit* SV1 message.

- **ASIC miner**
  With a real ASIC machine, it's very easy to configure it to point to the Translator Proxy. In the miner pool settings, the following string has to be added to the current endpoints:

  ```
  stratum+tcp://<tProxy ip>:34255
  ```

  Once configured, the ASIC miner will restart automatically and it will point its hashrate to the Translator Proxy IP previously set.
  As captioned in Figure 6.9, the Translator Proxy logs correctly the connection request coming from the machine, parsing the SV1 messages (subscribe, authorize, etc.).

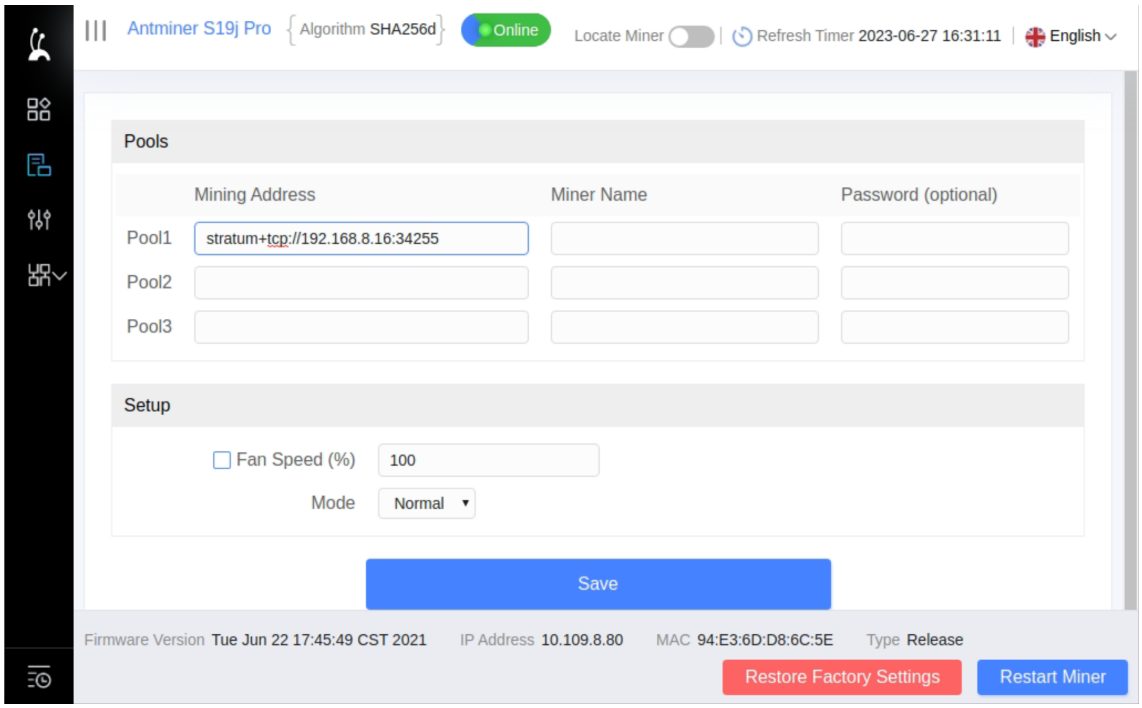


Figure 6.8: Pool settings of a Antminer S19J Pro

```
2023-06-27T14:37:29.096292Z  INFO translator_sv2: PC: ProxyConfig { upstream_address: "127.0.0.1", upstream_port: 34254, upst
ream_authority_pubkey: EncodedEd25519PublicKey { inner: PublicKey(CompressedEdwardsY: [215, 11, 47, 78, 34, 232, 25, 192, 195
, 168, 170, 209, 95, 181, 40, 114, 154, 226, 176, 190, 90, 169, 238, 89, 191, 183, 97, 63, 194, 119, 11, 31]), EdwardsPoint{
        X: FieldElement51([1413852444109712, 1147073702730733, 2186808044296008, 1254339098278883, 1713532435382676]),
        Y: FieldElement51([536709014948823, 1119531898009603, 777949201081045, 1090358907022687, 546145829123611]),
        Z: FieldElement51([1, 0, 0, 0, 0]),
        T: FieldElement51([882477417244186, 290382242068933, 1330987325702772, 1477235513995168, 1849816298263143])
}) }, downstream_address: "0.0.0.0", downstream_port: 34255, max_supported_version: 2, min_supported_version: 2, min_extranon
ce2_size: 8, jn_config: None, downstream_difficulty_config: DownstreamDifficultyConfig { min_individual_miner_hashrate: 10000
0000000.0, miner_num_submits_before_update: 5, shares_per_minute: 6.0, submits_since_last_update: 0, timestamp_of_last_update
: 0 }, upstream_difficulty_config: UpstreamDifficultyConfig { channel_diff_update_interval: 60, channel_nominal_hashrate: 500
0000.0, timestamp_of_last_update: 0, should_aggregate: false }, test_only_share_withhold: false }
2023-06-27T14:37:29.098309Z  INFO translator_sv2::upstream_sv2::upstream: PROXY SERVER - ACCEPTING FROM UPSTREAM: 127.0.0.1:3
4254
2023-06-27T14:37:29.114491Z  INFO translator_sv2::upstream_sv2::upstream: Up: Sending: SetupConnection { protocol: MiningProt
ocol, min_version: 2, max_version: 2, flags: 4, endpoint_host: Owned([48, 46, 48, 46, 48, 46, 48]), endpoint_port: 50, vendor
: Owned([]), hardware_version: Owned([]), firmware: Owned([]), device_id: Owned([]) }
2023-06-27T14:37:29.117702Z  INFO translator_sv2::upstream_sv2::upstream: Up: Receiving: Sv2Frame { header: Header { extensio
n_type: 0, msg_type: 1, msg_length: U24(6) }, payload: None, serialized: Some(Slice { offset: 0xb64ae008, len: 12, index: 1,
shared_state: SharedState(128), owned: None }) }
2023-06-27T14:37:29.117976Z  INFO translator_sv2::upstream_sv2::upstream: Up: Sending: OpenExtendedMiningChannel(OpenExtended
MiningChannel { request_id: 0, user_identity: Owned([65, 66, 67]), nominal_hash_rate: 5000000.0, max_target: Owned([0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255]), min_extrananonce_size:
 8 })
2023-06-27T14:37:29.118113Z  INFO translator_sv2: Connected to Upstream!
2023-06-27T14:37:29.118987Z  INFO translator_sv2::upstream_sv2::upstream: Up: Successfully Opened Extended Mining Channel
2023-06-27T14:37:29.170422Z  INFO translator_sv2::upstream_sv2::upstream: Is future job: true

2023-06-27T14:37:29.170580Z  INFO translator_sv2::upstream_sv2::upstream: Up: New Extended Mining Job
2023-06-27T14:37:29.170741Z  INFO translator_sv2::upstream_sv2::upstream: Up: Set New Prev Hash
2023-06-27T14:37:30.908013Z  INFO translator_sv2::downstream_sv1::downstream: PROXY SERVER - ACCEPTING FROM DOWNSTREAM: 10.10
9.8.80:35542
2023-06-27T14:37:30.908441Z  INFO translator_sv2::downstream_sv1::downstream: Down: Configuring
2023-06-27T14:37:30.910892Z  INFO translator_sv2::downstream_sv1::downstream: Down: Subscribing
2023-06-27T14:37:30.912638Z  INFO translator_sv2::downstream_sv1::downstream: Down: Authorizing
```

Figure 6.9: Translation Proxy logs successfully the ASIC miner SV1 requests

Config D

As described in 6.1, this configuration allows Mining Devices running SV1 firmware to connect to a SV2 Pool through a Translation Proxy (tProxy). In this case the

tProxy is designed also to implement the Job Negotiation (JN) sub-protocol: allowing miners to select transactions locally and send them to the Pool-side JN. In the following guide a Template Provider (TP) is installed locally on the same machine, to provide block templates to the JN.
Since Config D is very similar to the configuration previously tested, the focus here wants to be on the **Template Provider** and **Job Negotiator** roles.

As already explained, here the big difference comes from the addition of the transactions selection feature: to let the miner locally do it, a **local Template Provider** is needed.
First of all, in a new terminal window:

```
git clone https://github.com/stratum-mining/bitcoin.git
git checkout last-tested-tp
cd bitcoin/
./autogen.sh && ./configure --enable-template-provider
make check
```

Once the local Template Provider (which is a version of Bitcoin Core full node with the ability to act as a SV2 TP) is installed:

```
./src/bitcoind -testnet
```

After checking that the TP is correctly running, the **SV2 Pool** server has to be run:

```
cd stratum/roles/v2/pool/
cp pool-config-example.toml ./conf/pool-config.toml
cd conf/
```

Since the example config file is already configured to let the SV2 Pool connect to a local instance of Template Provider, nothing has to be changed in the configuration parameters. So, now run the SV2 Pool:

```
cargo run -p pool_sv2
```

As described in the command output, the Pool is now connected to the local Template Provider (127.0.0.1), and it will get the transactions to be put in the next block template from it.

```
2023-06-26T21:59:48.317771Z  INFO pool_sv2: Pool INITIALIZING with config: "pool-config.toml"
2023-06-26T21:59:48.318375Z  INFO pool_sv2::lib::template_receiver: Connected to template distribution server at 127.0.0.1:8442
2023-06-26T21:59:48.381959Z  INFO pool_sv2::lib::template_receiver::setup_connection: Setup template provider connection success!
2023-06-26T21:59:48.382185Z  INFO pool_sv2::lib::mining_pool: PUB KEY: [TxOut { value: 5000000000, script_pubkey: Script(OP_DUP O
P_HASH160 OP_PUSHBYTES_20 55a2e914aeb9729b4cd265248cb67a865eae95fd OP_EQUALVERIFY OP_CHECKSIG) }]
2023-06-26T21:59:48.382348Z  INFO pool_sv2::lib::mining_pool: Starting up pool listener
2023-06-26T21:59:48.382357Z  INFO pool_sv2::lib::job_negotiator: JN INITIALIZED
2023-06-26T21:59:48.382663Z  INFO pool_sv2::lib::mining_pool: Listening for encrypted connection on: 0.0.0.0:34254
```

Figure 6.10: SV2 Pool connected to the local Template Provider, *config D*

At this point, like in the previous configuration tested, **the Translator Proxy** is needed. In a new terminal:

```
cd stratum/roles/translator/
cp proxy-config-example.toml ./conf/proxy-config.toml
cd conf/
```

This time, since the Translator Proxy needs to act as a **Job Negotiator**, selecting the transaction from its own local Template Provider, some changes in its configuration file has to be done.

- Uncomment the line 27 of the *proxy-config.toml*, enabling the JNP.
- Ensuring that line 31 (*tp_address = "127.0.0.1:8442"*) is uncommented.

Now the Translator Proxy is ready to be run:

```
cargo run -p translator_sv2
```

```
2023-06-26T22:10:31.078400Z  INFO translator_sv2: PC: ProxyConfig { upstream_address: "127.0.0.1", upstream_port: 34254, upstream
_authority_pubkey: EncodedEd25519PublicKey { inner: PublicKey(CompressedEdwardsY: [215, 11, 47, 78, 34, 232, 25, 192, 195, 168, 1
70, 209, 95, 181, 40, 114, 154, 226, 176, 190, 90, 169, 238, 89, 191, 183, 97, 63, 194, 119, 11, 31]), EdwardsPoint{
        X: FieldElement51([1413852444109712, 1147073702730733, 2186808044296008, 1254339098278883, 1713532435382676]),
        Y: FieldElement51([536709014948823, 1119531898009603, 777949201081045, 1090358907022687, 546145829123611]),
        Z: FieldElement51([1, 0, 0, 0, 0]),
        T: FieldElement51([882477417244186, 290382242068933, 1330987325702772, 1477235513995168, 1849816298263143])
}) }, downstream_address: "0.0.0.0", downstream_port: 34255, max_supported_version: 2, min_supported_version: 2, min_extranonce2_
size: 8, jn_config: Some(JnConfig { jn_address: "127.0.0.1:34264", tp_address: "127.0.0.1:8442" }), downstream_difficulty_config:
 DownstreamDifficultyConfig { min_individual_miner_hashrate: 5000000.0, miner_num_submits_before_update: 5, shares_per_minute: 6.
0, submits_since_last_update: 0, timestamp_of_last_update: 0 }, upstream_difficulty_config: UpstreamDifficultyConfig { channel_di
ff_update_interval: 60, channel_nominal_hashrate: 5000000.0, timestamp_of_last_update: 0, should_aggregate: false }, test_only_sh
are_withhold: false }
2023-06-26T22:10:31.079930Z  INFO translator_sv2::upstream_sv2::upstream: PROXY SERVER - ACCEPTING FROM UPSTREAM: 127.0.0.1:34254
2023-06-26T22:10:31.082473Z  INFO translator_sv2::upstream_sv2::upstream: Up: Sending: SetupConnection { protocol: MiningProtocol
, min_version: 2, max_version: 2, flags: 6, endpoint_host: Owned([48, 46, 48, 46, 48, 46, 48]), endpoint_port: 50, vendor: Owned(
[]), hardware_version: Owned([]), firmware: Owned([]), device_id: Owned([]) }
2023-06-26T22:10:31.083197Z  INFO translator_sv2::upstream_sv2::upstream: Up: Receiving: Sv2Frame { header: Header { extension_ty
pe: 0, msg_type: 1, msg_length: U24(6) }, payload: None, serialized: Some(Slice { offset: 0x7f0f5cf86010, len: 12, index: 1, shar
ed_state: SharedState(128), owned: None }) }
2023-06-26T22:10:31.083267Z  INFO translator_sv2::upstream_sv2::upstream: Up: Sending: OpenExtendedMiningChannel(OpenExtendedMini
ngChannel { request_id: 0, user_identity: Owned([65, 66, 67]), nominal_hash_rate: 5000000.0, max_target: Owned([0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 255]), min_extranonce_size: 8 })
2023-06-26T22:10:31.083356Z  INFO translator_sv2: Connected to Upstream!
2023-06-26T22:10:31.084396Z  INFO translator_sv2::template_receiver: Template Receiver try to set up connection
2023-06-26T22:10:31.086854Z  INFO translator_sv2::job_negotiator: JN proxy: setupconnection Proxy address: 0.0.0.0:34255
2023-06-26T22:10:31.087368Z  INFO translator_sv2::job_negotiator: JN CONNECTED
2023-06-26T22:10:31.160399Z  INFO translator_sv2::template_receiver: Template Receiver connection set up
2023-06-26T22:10:31.160852Z  INFO translator_sv2::upstream_sv2::upstream: Up: Successfully Opened Extended Mining Channel
2023-06-26T22:10:31.161249Z  INFO translator_sv2::upstream_sv2::upstream: Is future job: true

2023-06-26T22:10:31.161280Z  INFO translator_sv2::upstream_sv2::upstream: Up: Set New Prev Hash
2023-06-26T22:10:31.213323Z  INFO translator_sv2::template_receiver: Received SetNewPrevHash, waiting for IS_NEW_TEMPLATE_HANDLED
2023-06-26T22:10:31.213509Z  INFO translator_sv2::template_receiver: IS_NEW_TEMPLATE_HANDLED ok
2023-06-26T22:10:31.213603Z  INFO translator_sv2::upstream_sv2::upstream: Send custom job to upstream
```

Figure 6.11: Translator Proxy which acts as a Job Negotiator, *config D*

As before, the Translator Proxy is now correctly running. This time, the big difference is inside the last log message of the command output:

```
INFO translator_sv2::upstream_sv2::upstream: Send custom job to upstream
```

In this moment, as described in the logs of 6.11 the Translator Proxy was able to create a custom job (block template to work on) using its own local Template

Provider, and it communicated it to the Job Negotiator which is Pool-side!

However, since the details about the messages involved into the **Job Negotiator Protocol** are still material of discussions, the analysis about it won't go deeper than this. To be updated upon the final version and implementation details about this sub-protocol, it's suggested to have a look at https://github.com/stratum-mining/sv2-spec/blob/main/06-Job-Negotiation-Protocol.md.

Now, the only role which is missing is the **SV1 Mining Device**. And, as in the previous configuration, it will work with both CPU-miner and the real ASIC machine, in the same way described above.

6.3 Final thoughts and future ideas

As described and analyzed in the previous section, Stratum Reference Implementation (SRI) has achieved a substantial progress during last year, delivering crucial features and roles like the **Translator Proxy** and the **Job Negotiation**.
While the first permits to use the SV2 protocol without changing the ASIC machine's firmware, the second is the critical feature that allows a real decentralization of the transactions selection power (which is now entirely in the hands of mining pools operators).
However, as already anticipated, some details about the messages involved into the Job Negotiation Protocol are material of discussion: firstly it will be renamed into **Job Declaration Protocol**, and the reasons for that will be explained in the next subsection about *SRI Pool fallback*.
Besides of that, SRI developers group defined many future further enhancements of the SV2 protocol, which are already been studied, such as the implementation of some specific payment pool, necessary to build a non-custodial pool, and the development of a protocol benchmarking suite.

6.3.1 SRI Pool fallback

The SRI Pool fallback is a feature which is already in the SRI roadmap, and it will be a very crucial piece of the protocol.
Basically, once the last little changes about the **Job Declarator Protocol** will be done, a miner who aims to work with a setup like the previously analyzed Config D (6.1), or even better Config A (6.1), will be able to build its own block templates, extracting the most profitable Bitcoin transactions from its local Template Provider. At this point, the miner will have a Job Declarator Client who is in charge of declaring this own block template to the Job Declarator Server (JDS) which will be Pool-side.
At this stage, the Pool-side JDS can still refuse the block template proposed by the miner (for any reason, could also be for censorship imposed by States or governmental agencies), and if this will be the case, the Job Declarator Client will **automatically declare the same block template** (containing the same transactions set) to another JDS of another mining Pool, choosing from a customized pre-configured backup list.
In the very extreme case in which all the JDS of the backup Pools are refusing the block template proposed by the miner, it will automatically start to do **solo mining**, without the need of any manual intervention.
By doing in this way, any possible future attacks to the censorship-resistance of the entire network will be extremely disincentivized and ineffective.

6.3.2 Non-custodial pools

Another subject of research of the SRI group is related to the current centralized and trusted payout mechanism used by the actual mining pools. As described in section 3.3, nowadays the addresses inserted into the coinbase output to get the block reward is the ones belonging to the mining pools operators. Then, accordingly to the shares submitted from every miner who joins the pool, this reward is split and sent to the miners, through normal asynchronous Bitcoin transactions. The concentration of the entire funds in a central entity exposes pooled mining operations to a significant risk. As the payout process is based on a trusted centralized third-party pool service, miners must place complete trust in the fairness of their payouts, without the ability to independently verify whether the pool is withholding a portion of their rewards, a practice known as **pool skimming** [48]. The most valuable solution to address this issue is based on implementing a payout scheme where miners directly collect the coinbase reward, without the need for a centralized pool to control their funds: in this way, it would be possible to operate a fully non-custodial pool.
In the past, some possible solutions emerged from the market, but the most promising one was called **P2Pool**, who was announced in this way: «P2Pool is a decentralized pool that works by creating a P2P network of miner nodes. These nodes work on a chain of shares similar to Bitcoin's blockchain. Each node works on a block that includes payouts to the previous shares' owners and the node itself. There is no central point of failure, making it DoS resistant.» [49]. However, its payout scheme was based on locking funds to miners' individual addresses within the coinbase transaction outputs, leading to a significant increase in the size of the coinbase: for this reason it revealed to be a very inefficient solution.
Three developers from the SRI team, published a RFC containing their own new payout scheme for a non-custodial mining pool on the bitcoin dev list [50]. As stated into the document, «Our scheme is introduced through the concept of a payment pool, where the participants are the miners of the mining pool. The presented payment pool scheme uses ANYPREVOUT [51], does not rely on any off-chain technology and it is trustless, in the sense that a participant does not have to trust in collaboration of all other participants: a non-collaborating participant is automatically ejected from the payment pool and it is not a threat for accessibility of funds. Our study assumes the pool to be centralised, but it can be generalised to decentralised pools. Our payment pool scheme is meant to be a future extension of Stratum V2 mining protocol.» [52]

6.3.3 SRI benchmarking suite

As already told in 5.4, SRI development started some years ago. However, a major mining protocol update like the one proposed by the SRI is very sensitive, due to

the ever growing importance of the mining operations of nowadays.
During last year the SRI work started to get some real encouraging feedbacks from the Bitcoin community, thanks to the last major updates and to the communication efforts done in the last months.

By the way, to encourage Stratum V2 wide adoption, the SRI developers group think that a complete evaluation and precise measurements of the enhancements brought by SV2 is needed. A benchmarking suite which is able to easily test and benchmark protocol performances in different mining scenarios, capable of comparing the current version of Stratum (V1) with SV2 is necessary. In this way, mining industry professionals and the broader market will be able to easily understand every possible configuration permitted by SRI, evaluating and measuring themselves the potential benefits in terms of efficiency and consequently, profitability. The main purpose of benchmarking is to demonstrate, with precise measurements, all the performance improvements brought by SV2, pushing at this point its natural adoption by both miners and mining pools.

Chapter 7

Conclusion

In conclusion, this thesis has provided a comprehensive exploration of the evolution of mining protocols in the Bitcoin ecosystem, with a specific focus on the transformative potential of Stratum V2. By examining the history, mechanics, and limitations of previous protocols such as Getwork and Getblocktemplate, as well as the current dominant protocol Stratum (V1), the need for a more efficient and secure protocol became apparent. The emergence of Bitcoin as a decentralized digital currency highlighted the importance of mining in maintaining the system's integrity and security. The concept of Proof of Work (PoW) was introduced as a consensus mechanism, ensuring that miners invest computational power to validate transactions and add new blocks to the blockchain. However, the transition from solo mining to pooled mining brought significant changes, enabling miners to collaborate and increase their chances of receiving rewards in a more predictable manner. Unfortunately, because of the way Stratum (V1) protocol works, this benefit is gained at the expense of a centralization about the selection of the transactions to be included in blocks.

The main argument of this thesis centers around the significance of Stratum V2 as a transformative protocol for Bitcoin pooled mining. Stratum V2 addresses the centralization concerns associated with Stratum (V1) and introduces enhanced security, operational efficiency, transaction selection decentralization, and other improvements. By decentralizing power and giving more control to individual miners, Stratum V2 aims to maintain the decentralized and uncensorable nature of the Bitcoin network.
While Stratum V2 presents a promising solution, its adoption in the mining community is still in progress. Real-world data and concrete evidence showcasing the efficiency improvements brought by the protocol update will be essential in encouraging individual miners to embrace Stratum V2. Continued research and development, as well as the creation of benchmarking suites and practical implementations like the Stratum Reference Implementation (SRI), will play a vital role in further

advancing the protocol and its adoption. It is crucial to emphasize the dangers associated with centralization in the hands of a few mining operators. Stratum V2 offers a pathway to mitigate these risks and maintain the decentralized nature of the Bitcoin network. By fostering a collaborative and secure mining environment, Stratum V2 has the potential to shape the future direction of Bitcoin mining methods and ensure the continued development of the Bitcoin ecosystem.

In conclusion, this thesis contributes to the understanding of the advancements made in mining protocols and highlights the significance of Stratum V2 as a groundbreaking protocol within the Bitcoin ecosystem. By addressing centralization concerns and introducing improvements in security and efficiency, Stratum V2 builds the way for a more decentralized and resilient Bitcoin network. As further research and development take place, and as the benefits of Stratum V2 become more evident through real-world implementations and data, it is expected that the mining community will increasingly adopt this protocol, ultimately enhancing the overall strength and uncensorability of the entire Bitcoin network.

Bibliography

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Oct 2008.

[2] Daniel Frumkin. *Bitcoin Mining Handbook*. Braiins Publishing, 1st edition, 2022.

[3] Block Chain - Bitcoin — developer.bitcoin.org. `https://developer.bitcoin.org/reference/block_chain.html#block-headers`.

[4] BIP 0023 - Bitcoin Wiki — en.bitcoin.it. `https://en.bitcoin.it/wiki/BIP_0023#Mutations`.

[5] Rachel Rybarczyk. The Future of Bitcoin Mining Protocols: Making Every Watt Count — galaxy.com. `https://www.galaxy.com/research/insights/future-of-bitcoin-mining-protocols/`, 2022.

[6] Bitcoin's Decentralization with Stratum V2 | Braiins — braiins.com. `https://braiins.com/blog/stratum-v2-bitcoin-decentralization`, 2020.

[7] P. Champagne. *The Book Of Satoshi*. Wren Investment Group, LLC, first edition, 2014.

[8] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st edition, 2014.

[9] Bitcoin Legacy Project. The Bitcoin Legacy Project — thebitcoinlegacyproject.org. `https://www.thebitcoinlegacyproject.org/`.

[10] Bitcoin Optech. Transaction bloom filtering — bitcoinops.org. `https://bitcoinops.org/en/topics/transaction-bloom-filtering/`.

[11] Addy Yeow. Global Bitcoin Nodes - Bitnodes — bitnodes.io. `https://bitnodes.io/nodes/all/`.

[12] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[13] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[14] Saifedean Ammous. *The Bitcoin Standard: The Decentralized Alternative to Central Banking*. Wiley Publishing, 1st edition, 2018.

[15] laszlo. Pizza for bitcoins? — bitcointalk.org. `https://bitcointalk.org/index.php?topic=137.0`, May 2010.

[16] Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.

[17] m0mchil. GitHub - m0mchil/bitcoin-getwork: bitcoin getwork patch —

github.com. https://github.com/m0mchil/bitcoin-getwork, 2010.
[18] satoshi. New getwork — bitcointalk.org. https://bitcointalk.org/index.php?topic=1901, November 2010.
[19] m0mchil. GitHub - m0mchil/poclbm: PyOpenCL bitcoin miner — github.com. https://github.com/m0mchil/poclbm.
[20] mempool - Bitcoin Explorer — mempool.space. https://mempool.space/block/00000000000000000cca48eb4b330d91e8d946d344ca302a86a280161b\0bffb6?showDetails=true#details.
[21] RPC Byte Order - Bitcoin Glossary — btcinformation.org. https://btcinformation.org/en/glossary/rpc-byte-order.
[22] Getwork - Bitcoin Wiki — en.bitcoin.it. https://en.bitcoin.it/wiki/Getwork.
[23] Luke-Jr. Mining protocol extension: noncerange — bitcointalk.org. https://bitcointalk.org/index.php?topic=24336.0.
[24] Luke-Jr. X-Roll-Ntime extension — bitcointalk.org. https://bitcointalk.org/index.php?topic=22561.0.
[25] jgarzik. RPC: Remove 'getwork' deprecated mining protocol by jgarzik Â · Pull Request #2905 - bitcoin/bitcoin — github.com. https://github.com/bitcoin/bitcoin/pull/2905.
[26] jgarzik. [RFC] removal of "getwork"; RPC mining protocol — bitcointalk.org. https://bitcointalk.org/index.php?topic=277631.0, August 2013.
[27] Luke-Jr. Decentralized mining protocol standard: getblocktemplate (ASIC ready!) — bitcointalk.org. https://bitcointalk.org/index.php?topic=108854.0, September 2012.
[28] Emi Lacapra. What are Bitcoin improvement proposals (BIPs), and how do they work? — cointelegraph.com. https://cointelegraph.com/explained/what-are-bitcoin-improvement-proposals-bips-and-how-do-they-work.
[29] Bitcoin-Qt version 0.7.0 released — bitcoin.org. https://bitcoin.org/en/release/v0.7.0#bitcoin-improvement-proposals-implemented, 2012.
[30] Block Chain - Bitcoin — developer.bitcoin.org. https://developer.bitcoin.org/reference/block_chain.html#serialized-blocks.
[31] Marek "Slush" Palatinus. [ANN] Stratum mining protocol - ASIC ready — bitcointalk.org. https://bitcointalk.org/index.php?topic=108533.0, September 2012.
[32] Marek "Slush" Palatinus. Stratum - network protocol spec — docs.google.com. https://docs.google.com/document/d/17zHy1SUlhgtCMbypO8cHgpWH73V5iUQKk_0rWvMqSNs/edit?hl=en_US. [Accessed 06-Jul-2023].
[33] Marek "Slush" Palatinus. Stratum V1 Docs | Mining Protocol — braiins.com. https://braiins.com/stratum-v1/docs, 2012.
[34] Stratum mining protocol - Bitcoin Wiki — en.bitcoin.it. https://en.bitcoin.it/wiki/Stratum_mining_protocol.

[35] X. Liu, R. Chong, Y. Huang, Y. Zhang, and Q. Zhou. Disappeared coins: Steal hashrate in stratum secretly. 2021. Blackhat Asia.

[36] Eddy Zhang. Job injection based on set_extranonce. https://www.youtube.com/watch?v=ZvpdOj6U0vM, 2021.

[37] Braiins. Bitcoin mining stratum v2 - behind the scenes with slush pool's ceos & square crypto's matt corallo. https://www.youtube.com/watch?v=sp6QEFzkAyI, 2020.

[38] Trevor Perrin (noise@trevp.net). The Noise Protocol Framework — noiseprotocol.org. https://noiseprotocol.org/noise.html.

[39] Stratum V2 - protocol security specifications - stratum-mining/sv2-spec — github.com. https://github.com/stratum-mining/sv2-spec/blob/main/04-Protocol-Security.md.

[40] Stratum V2 | The next generation protocol for pooled mining — braiins.com. https://braiins.com/stratum-v2.

[41] Driving Stratum V2 Adoption with Braiins OS+ Autotuning Firmware | Braiins — braiins.com. https://braiins.com/blog/driving-stratum-v2-adoption-with-braiins-os-autotuning-firmware, 2020.

[42] btccasey. Developers announce stratum v2 update for decentralized bitcoin mining — bitcoinmagazine.com. https://bitcoinmagazine.com/technical/developers-announce-stratum-v2-update-for-decentralized-bitcoin\-mining, 2023.

[43] stratum-mining — github.com. https://github.com/stratum-mining/.

[44] Template provider - stratum-mining/bitcoin at last-tested-tp — github.com. https://github.com/stratum-mining/bitcoin/tree/last-tested-tp.

[45] ccdle12. [WIP] add a stratum v2 template provider by ccdle12 Â · Pull Request #27854 - bitcoin/bitcoin — github.com. https://github.com/bitcoin/bitcoin/pull/27854.

[46] StratumV2 — stratumprotocol.org. https://stratumprotocol.org/.

[47] Stratum V2 - mining protocol specifications - stratum-mining/sv2-spec — github.com. https://github.com/stratum-mining/sv2-spec/blob/main/05-Mining-Protocol.md#512-extended-channels.

[48] chasdabigone. Mining Pool Skimming — bitcointalk.org. https://bitcointalk.org/index.php?topic=1750050.0.

[49] forrestv. p2pool: Decentralized, DoS-resistant, Hop-Proof pool — bitcointalk.org. https://bitcointalk.org/index.php?topic=18313.0.

[50] Fi3 and Lorban. [bitcoin-dev] A payout scheme for a non custodial mining pool — lists.linuxfoundation.org. https://www.mail-archive.com/bitcoin-dev@lists.linuxfoundation.org/msg12616.html. [Accessed 06-Jul-2023].

[51] Decker Christian and Towns Anthony. BIP118 - SIGHASH_ANYPREVOUT for Taproot Scripts — bips.xyz. `https://bips.xyz/118`, 2017.

[52] Fi3, Lorban, and Rachel Rybarczyk. RFC-payment_pools-0.3 — docs.google.com. `https://docs.google.com/document/d/1qiOOSOT7epX658_nhjz-jj0DlnSRvytemOv_u_OtMcc/edit#heading=h.z5edqe4exehb`.