Master's Thesis

Scaling Bitcoin

Chair of Economic Theory
University of Basel

Supervised by:
Professor Dr. Aleksander Berentsen

Author:
Remo Nyffenegger
2014-053-904
remo.nyffenegger@unibas.ch
Submission Date: 09.08.2018

Abstract

In our paper, we aim to give a detailed insight into the Bitcoin scaling debate and discuss some of the most relevant proposals. If Bitcoin shall become a widespread method of payment, the basic protocol introduced by Satoshi Nakamoto in 2008 has to be enhanced with new technical concepts. We focus on a block size increase, Segwit, payment channels and the corresponding payment channel networks. These topics are by far not the only concepts that aim to improve Bitcoin's scalability. To our knowledge, however, they are the most advanced ones which have played a dominant role in the scaling debate.

Plagiatserklärung

Ich bezeuge mit meiner Unterschrift, dass meine Angaben über die bei der Abfassung meiner Arbeit benützten Hilfsmittel sowie über die mir zuteil gewordene Hilfe in jeder Hinsicht der Wahrheit entsprechen und vollständig sind. Ich habe das Merkblatt zu Plagiat und Betrug vom 23.11.05 gelesen und bin mir der Konsequenzen eines solchen Handelns bewusst.

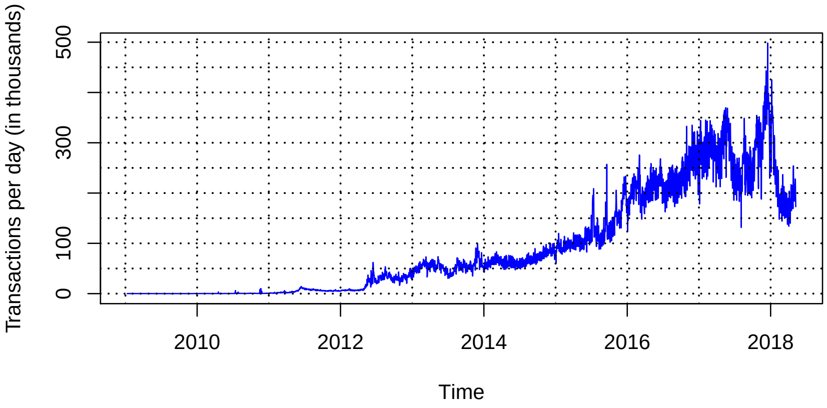
Basel, August 06, 2018

Remo Nyffenegger

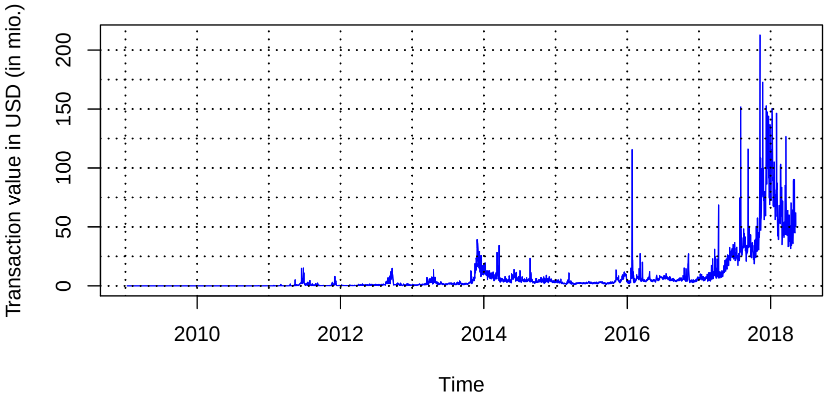
Contents

1 Introduction

With the increasing public interest in the blockchain technology and Bitcoin specifically, the use of the Bitcoin network itself has risen. Figure 1 depicts the increase in transactions per day in (a) and the development of the estimated US-Dollar transaction value in (b). The upper graph clearly shows a growing number of transactions that has been sent over the network per day, peaking at the end of 2017. There was a drop of almost 50% at the beginning of 2018. For the past few months, however, the number of transactions has been relatively constant at around 200'000 transactions per day. The estimated US-Dollar transaction value



(a) Average number of confirmed transactions per day (in thousands).



(b) Estimated transaction value in US-Dollar per day (in mio.).

Figure 1: Transactions in the Bitcoin network (Source: `https://charts.bitcoin.com/`).

per day developed more or less similarly. The extremely high transaction volumes in late 2017 may have been driven by the skyrocketing prizes in December.[1]

If more people regularly use Bitcoin, it is crucial that the network is able to scale to a high transaction throughput such that it can cope with an increasing traffic. This is, however, limited by several factors such as bandwidth, storage capacity for full nodes or transaction validation. Additionally, a hard-coded restriction in the Bitcoin code exists that limits the size of a block in the Bitcoin blockchain to one megabyte. This line of code was added only in July 2010[2] and not included in the original implementation of Bitcoin. According to a *bitcointalk.org* user called *Cryddit*, Hal Finney[3] was the first developer to propose a block size limit in a conversation with Satoshi Nakamoto and *Cryddit*.[44] The main motivation to add a limit were concerns about denial-of-service attacks or spam transactions abusing the capacity of the blockchain which could be prevented by artificially limiting the number of transactions that can be included in a block. However, they were aware that at some point in time scaling issues may emerge. *Cryddit* claimed that they agreed on implementing the limit only temporarily. This is in line with statements by Satoshi Nakamoto on *bitcointalk.org* where he[4] suggested how the block size may be raised at a later point in time.[95] The limit, however, has never been changed since 2010 which is the starting point of the Bitcoin scaling debate.

To get an idea about the implied limitations of the Bitcoin network, we set it in comparison to other payment networks. As illustrated in figure 1, Bitcoin processes around 100'000 to 400'000 transaction per day, which comes down to one to five transactions per second on average. The maximum transaction capacity per second can be approximately calculated using the one megabyte limit and the average size of a transaction, which is plotted in figure 2. We see that the average

[1] Griffin and Shams (2018) claim that the price surge was strongly driven by manipulating Tether purchases.[59]

[2] `https://github.com/bitcoin/bitcoin/commit/a30b56ebe76ffff9f9cc8a6667186179413c6349#diff-23cfe05393c8433e384d2c385f06ab93R18`

[3] Hal Finney was a computer scientist, software developer, one of the first Bitcoin users and the recipient of the first Bitcoin transaction.[9]

[4] We use the pronoun "he" due to the masculine pseudonym. It is, however, unknown, who (gender/group) is behind Satoshi Nakamoto.

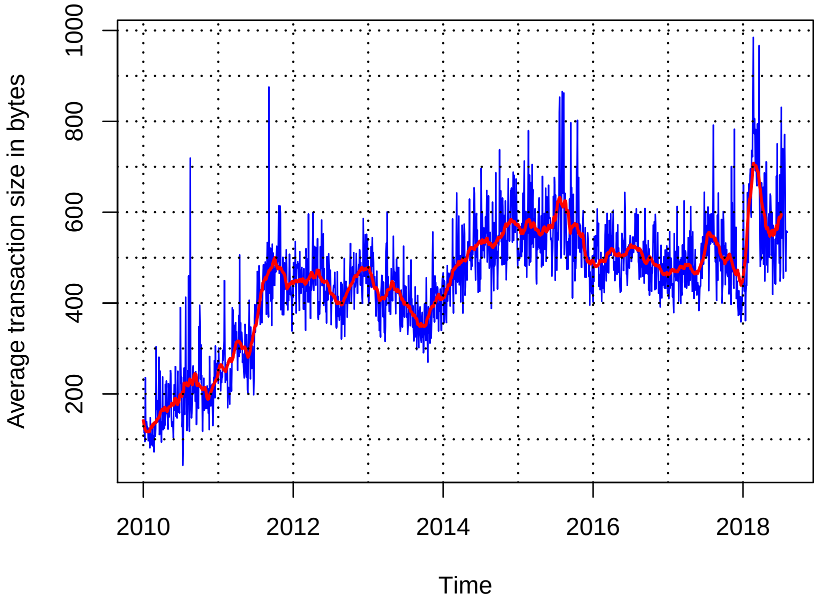


Figure 2: Average transaction size in bytes (Source: `https://blockchain.info`). Note: The red trend line is calculated by applying a two-sided simple moving average filter of order 15.

transaction size has been between 400 to 600 bytes over the last four years. Using these numbers results in around 1600 to 2500 transactions per block[5], i.e. about three to four transactions per second on average.[6]

Comparing these three to four transactions per second to other payment networks like PayPal or Visa clarifies the dimensions of these numbers. In 2016, PayPal processed 7.6 billion payment transactions, which comes down to about 240 transactions per second.[87] Visa scales even more as it transferred over 111.2 billion transactions over its network in 2017, which is about 3500 transactions per second.[113] According to a Visa report, the maximum capacity of their network is even 65'000 transactions per second.[114] To reach this number without any other improvement but raising the block size, around 20 gigabyte blocks would

[5] This rough calculation does not take Segwit into account. For further details, see section 4.

[6] The number of processed transactions can fluctuate in the short-term depending on the realized block interval. The difficulty to mine a block is adjusted every 2016 blocks (i.e. $\approx$ two weeks) such that it takes the miners ten minutes on average to find a valid block. However, if in a certain period the provided mining power strongly increases, blocks are mined more frequently than every ten minutes and the number of transactions would increase temporarily.

be required. Fixing such a big block size is complicated by physical restrictions like bandwidth, storage and transaction verification limitations.

To be precise, we have to highlight that a transaction in Bitcoin is not directly comparable to a transaction in conventional payment networks. In the Visa or PayPal network, each transaction sends exactly one single transfer of money from A to B. In Bitcoin, however, a transaction can have several inputs and outputs and thus can contain much more single money transfers (i.e. payments). As illustrated in figure 3, however, the number of payments is not much larger than the number of transactions. On average, there are about 1.6 times more payments than transactions in a block. In theory, a transaction could include much more payments which means that the potential is far form being exhausted.

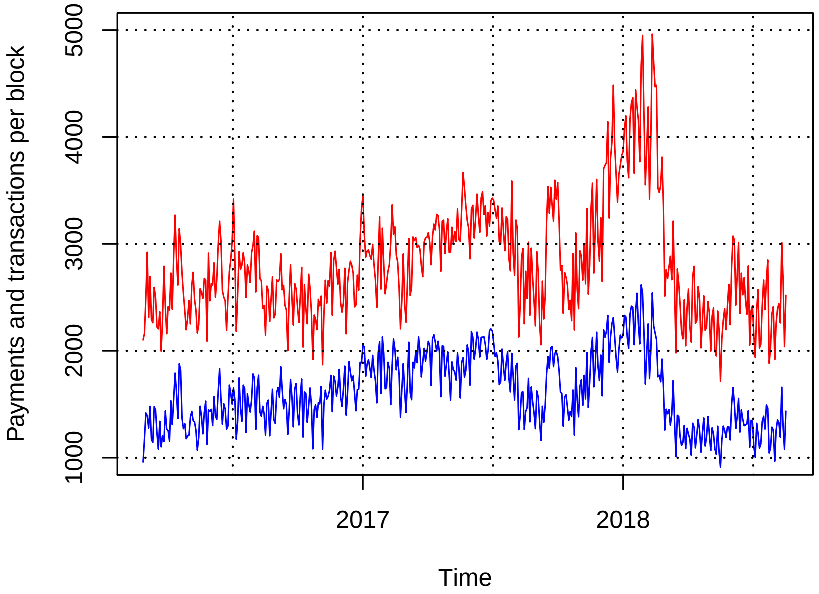


Figure 3: Payments (red) and transactions (blue) per block (Source: `https://transactionfee.info/`). Note: Payments is defined as the number of outputs in a transactions minus one, to control for the output that goes back to the sender. The series is smoothed with a moving average.

The comparisons presented above show the current dimensions and highlight the need for solutions to scale Bitcoin, if it aims to become a relevant alternative in the world-wide payment and transaction system. Our paper aims to give a deeper insight into scaling Bitcoin. We define scaling as every proposal, mechanism or concept that aims to increase transaction throughput or, in other words, enables to process more payments over the Bitcoin network. We especially look at a block size increase, Segwit, payment channels, the corresponding payment channel networks and discuss these concepts in more detail. To our point of view, these proposals are currently the most relevant ones since they have been heatedly discussed in the network and are mostly well developed.

To our knowledge, there is no extensive paper that provides a detailed description of the aforementioned concepts. A vast number of blogs, threads in forums, articles in online magazines, explanations in Github repositories and few papers (e.g. Croman et al., 2016) discuss the Bitcoin scaling debate, but the interested user has to arduously collect the information from many different sources. Hence, our main contribution is to provide a detailed overview of the Bitcoin scaling debate and discuss the different concepts. Our goal is to make the scaling debate comprehensible for the average Bitcoin user. We, therefore, try to provide a good mixture between technical details and intuition. Since the scaling debate is a very controversial topic, we abstain from subjectively assessing the applicability of the proposals. We only present arguments of different interest groups that are used in the context of the scaling debate.

The paper proceeds as follows. In section 2 we give a historical background of the whole scaling debate and how it developed. Section 3 discusses a block size increase and section 4 details Segwit. Payment channels are described in section 5 and the Lightning network in section 6. Section 7 presents other concepts that may help to scale Bitcoin, while section 8 concludes.

2 Historical Background

Before discussing the technical details, we want to provide a historical background of the scaling debate. Analogous to the following parts of the paper the focus will be on a block size increase, Segregated Witness, payment channels and the

Lightning network.

To ensure that a non-informed reader has an idea about the content of these proposals, we start by giving a quick and intuitive explanation. A block size increase deals with a simple raise of the hard-coded one megabyte limit which allows to include more transactions in a block. Segregated Witness (Segwit) aims to solve transaction malleability (see section 4.3) and comes with an indirect block size increase due to the way it was implemented. Upon spending some Bitcoin units, a user has to prove that he or she is the rightful owner of the respective Bitcoins. This is cryptographically proven by using a private and public key pair and a respective signature. These signatures need a lot of space but are only used at the time of validation. Thus, Segwit aims to disregard them whenever possible. The main point is that the signatures are removed from their original field in the transaction and appended at the end of it. Hence, the base transaction data is separated from the witness data. In doing so, signatures are not included in the one megabyte block size limit anymore.

Payment channels make use of 2-of-2 multisignature addresses[7] to raise the number of transactions in the network. A transaction processed in a payment channel adjusts the balances in the multisig address but is not added to the blockchain. Only two transactions enter the blockchain: the channel's opening transaction, which funds the multisig address and the closing transaction, which pays out the current balances to the two parties. Depending on the set up, an arbitrary number of transactions can be sent over the channel. The Lightning network makes use of these payment channels. A payment can be routed through the network from A to B without the requirement that these parties maintain a payment channel with each other. The burden on the blockchain is considerably reduced with payment channels or the Lightning network, because the transactions are moved away from the blockchain. This is why these proposals are said to be off-chain or second layer solutions.[8]

[7] Funds in a 2-of-2 multisignature address can only be spent if the corresponding transaction is signed by two out of two signatures of two pre-defined keys. Normally, this is achieved by using a Pay-to-Script-Hash (P2SH) transaction type. The public keys and signatures are provided in the redeemScript of the scriptSig/witness field.

[8] The first layer being the blockchain and the second layer being mechanisms that increase

Subsequently, we start with the historical background. The scaling debate goes back to 2010. Originally, users in the network mainly debated about scaling Bitcoin through a block size increase. When Satoshi Nakamoto left the Bitcoin network in late 2010, he handed over the position of *Bitcoin Core's*[9] lead developer to Gavin Andresen who worked closely together with Satoshi from the beginning. Andresen decided to give four other developers commit access to the *Bitcoin Core* source code, namely Jeff Garzik, Gregory Maxwell, Pieter Wuille and Wladimir van der Laan who became lead developer after Gavin Andresen stepped back in 2014.[126] Gavin Andresen and Jeff Garzik were in favour of a block size increase, the other three, however, were opposing it. This disagreement on the lead developer level combined with a huge discrepancy of opinions in the network made it nearly impossible in the early years to reach a consensus on how Bitcoin should be scaled.[10] Additionally, the one megabyte block size limit became only really relevant in 2016 when the average block size approached the one megabyte limit as shown in figure 4. Before, there was no urgent need for a quick solution and hence, finding a consensus on how to scale Bitcoin was even harder.

Nevertheless, there have been several different proposals since 2010. As aforementioned, the discussion was originally about a block size increase, but also payment channels were discussed on an early stage already. A very preliminary draft of a payment channel was included in the first software release *Bitcoin 0.1* by Satoshi Nakamoto.[120] The idea was developed further over the years. Originally, research was mostly conducted in terms of unidirectional payment channels.[11] The first unidirectional payment channel implemented was the Spillman channel in the BitcoinJ client in 2013.[18] In 2014, Alex Akselrod proposed for the first time a bidirectional payment channel.[27] Also other projects were developed which never really took off such as the concept of a hub-and-spoke system first proposed

the number of transactions in the network off-chain.

[9] By that time, the reference client was actually called *Bitcoin* but renamed to *Bitcoin Core* in 2014 to reduce confusion.[12]

[10] We want to highlight that Bitcoin is not directly dependent on the *Bitcoin Core* client which is the reference software. A large majority of the network could decide to follow different rules that the ones implemented by *Bitcoin Core*.

[11] See section 5 for an explanation of unidirectional and bidirectional payment channels.

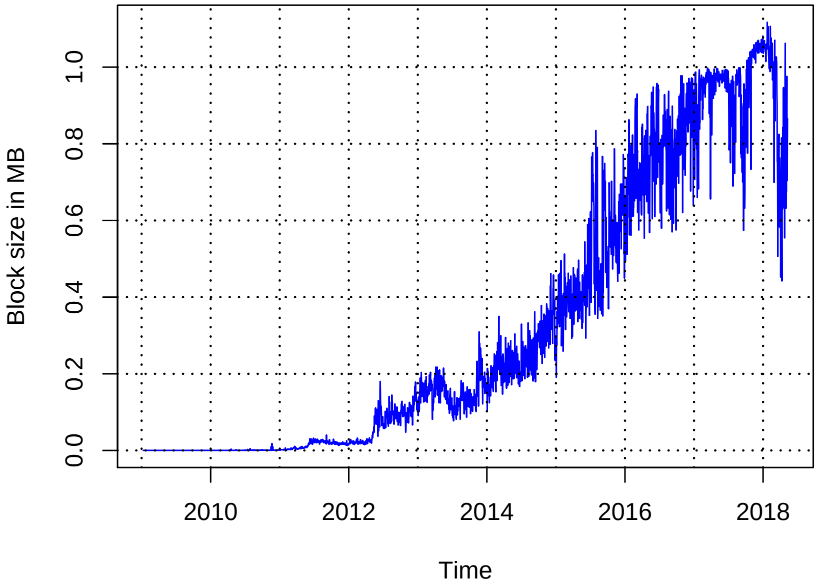


Figure 4: Average block size in megabytes (Source: `https://charts.bitcoin.com/`).

by Meni Rosenfeld in 2012,[106] inter-channel payments described by *BitPay* in January 2015[66] or *AmikoPay*, a Bitcoin-based Ripple system that would have implied far-reaching alterations to the Bitcoin protocol.[120]

In 2014, Mike Hearn published a Bitcoin Improvement Proposal (BIP), called BIP64, which became the basis of his work on an alternative client called *Bitcoin XT*. He was not satisfied with the missing progress in the scaling debate and an advocate of an increased block size. In mid 2015, several BIPs were formulated that dealt with a block size increase. For example, there was Jeff Garzik's BIP100 which would abolish the static block size limit and introduce a dynamic maximum block size dependent on miner vote or Gavin Andresen's BIP101 which intended to introduce a predictable growth rate of the block size. When it became clear in August 2015 that BIP101 would not be implemented in the *Bitcoin Core* client, Mike Hearn and others decided to implement it in *Bitcoin XT*. There was a notable disagreement between different players of the network. Some large Chinese mining pools like *F2Pool*, *BTCChina Pool* and *Huobi Pool* opposed a switch to *Bitcoin XT*[12] [116] while other large Bitcoin companies like *BitPay* or *Blockchain.info* approved it[85]. The user numbers in *Bitcoin XT* were eventually very small, though. On January 11, 2016, which was the first possible date on which Bitcoin miners could convert to the *Bitcoin XT* protocol, only 10% did adopt it.[86] Over the following months, the number of nodes declined to a neglectable number[80]. Consequently, the *Bitcoin XT* project could be regarded as failed.[13]

Perhaps due to increased disagreement in the network about how to scale Bitcoin, two Scaling Bitcoin Workshops were organized in 2015 and repeated in the following years.[14]

In the same year, the Lightning network with the corresponding Poon-Dryja payment channels and Segwit were presented for the first time. At the San Francisco Bitcoin Developer Seminar in February, Joseph Poon and Thaddeus Dryja presented a first draft of the Lightning network, which was supposed to be a powerful solution for solving the scalability problem in Bitcoin.[75] *Blockstream* was the first company to place a large focus on the Lightning network. They started developing a Lightning implementation called *c-lightning.* Soon other companies began working on a Lightning network implementation, such as *ACINQ* or *Lightning Labs.* Moreover, Christian Decker and Roger Wattenhofer proposed an alternative payment network with duplex payment channels in August 2015.[46] However, their proposal has not been as successful as the Lightning network so far.

The idea of separating the base transaction data from the witness data in a block was first publicly proposed in June 2015 as a feature of the *Blockstream* sidechain

[12] At the same time, they published a statement together with *AntPool* and *BW Mining* to support a block size increase to eight megabytes in general. However, they opposed an even larger raise to 20 megabytes mostly because of the *Great Firewall of China* and a not sufficiently developed internet infrastructure.[117]

[13] Mike Hearn stated that *Bitcoin XT* was not given a fair chance.[65] For example, some users claimed that discussion about *Bitcoin XT* was censored in the /r/Bitcoin subreddit[105] and that Distributed Denial of Service (DDoS) attacks were executed which led to a collapse of computers running an *XT* node.[40]

[14] The workshops took place in Montreal (12th-13th September 2015), Hong Kong (6th-7th December 2015), Milan (8th-9th October 2016), Stanford (4th-5th November 2017) and Tokyo (6th-7th October 2018).

project *elements.* Pieter Wuille was the main researcher of the proposal which was called Segregated Witness (Segwit).[50] The original intention of Segwit was not to raise the block size and to increase the number of transactions but to find a solution for transaction malleability.[15] It was not clear whether Segwit could be implemented as a soft fork[16] until Peter Wuille presented *Segregated Witness - as a soft fork* at the *Scaling Bitcoin - Hong Kong* workshop.[129] Many users, however, criticized Segwit to be a non-sufficient short-term scaling solution[57] or "ugly and awkward" if implemented as a soft fork.[92] The proposal was formalized in December 2015 as BIP141 by Eric Lombrozo, Johnson Lau and Pieter Wuille.

Following the *Scaling Bitcoin - Hong Kong* workshop, *Bitcoin Core* published a roadmap describing which projects they aimed to implement to scale Bitcoin.[11] Especially the importance of Segregated Witness, second layer solutions like the Lightning network or a more efficient block relay were highlighted. Missing was a remark about a block size increase.

Nevertheless, advocates of a block size increase developed new proposals. Gavin Andresen created BIP109 in January 2016 which should double the block size to two megabytes. This proposal was rejected. In early 2016, there was a further attempt to build a client with an increased block size called *Bitcoin Classic.*[17] At the beginning, the *Bitcoin Classic* developers promoted an increase of the block size to two megabytes. In November 2016, however, they decided to adopt a solution in which miners and nodes determine a dynamic block size limit. *Bitcoin Classic* was never really successful and ceased operation when *Bitcoin Cash* forked out of Bitcoin in August 2017.

In February 2016, there was another meeting in Hong Kong, where *Bitcoin Core* contributors, mining pool operators and other Bitcoin industry members discussed the scaling debate. They agreed on working on a block size increase pro-

[15] Transaction malleability means that it is possible to modify the signature in a transaction which is not included in the blockchain yet without invalidating it. As a result, the transaction ID changes which can break the link between two transactions. This is especially a problem for second layer protocols such as the Lightning network.

[16] There have never been specific intentions to implement Segwit as a hard fork due to fears of a network split.

[17] Furthermore, *Bitcoin Classic* suggested to use *Flexible transactions* (Flextrans) instead of Segwit to deal with transaction malleability.[41]

posal and a concrete Segwit release. It became known as the *Bitcoin Roundtable Consensus* or *Hong Kong Agreement*.[17]

Shortly after *Scaling Bitcoin - Milan* took place in October 2016, contributors to the Lightning network[18] came together for the first *Lightning network summit*. They worked out a basic implementation called *BOLT* (Basis of Lightning Technology) that was interoperable for all the implementations developed so far. *BOLT* is the basis of the Lightning network as it is known today. In January 2017, the first alpha version of the Lightning network was released called *lnd*. [120]

Also in October 2016, Segwit was officially introduced in *Bitcoin Core* version 0.13.1. Because Segwit was implemented using BIP9[19], 95% of the miners had to signal support for Segwit in order for it to be activated on the network. Despite the *Hong Kong Agreement*, several big mining pools were either opposing Segwit (e.g. Antpool[108]) or only approving it under the condition that it was followed by a block size increase (e.g. ViaBTC[125]). Consequently, Segwit was not successfully activated at that point in time.

In early 2017, almost all blocks were very close to the one megabyte limit, as illustrated in figure 4. This increased pressure to find a consensus on scaling and let the memory pool (mempool) of unconfirmed transactions rise considerably, as shown in figure 5.[20] Moreover, another interesting development could be observed during this time. Figure 6 compares the fraction of Bitcoin's market capitalization compared to ten of the most relevant alternative cryptocurrencies.[21] The share of Bitcoin's market capitalization was relatively stable until early 2017 but then decreased at about the same time the scaling issue became acute. Perhaps, this was a main driver for the loss in market share. However, there may be other

[18] *ACINQ*, *Amiko Pay*, *BitFury*, *Blockstream*, *Lightning Labs* and *Purse*.[115]

[19] BIP9 is a mechanism that defines how soft forks and especially several parallel soft forks can be implemented. The *nVersion* field in the block header is interpreted as a bit vector, where each bit can be used for signalling purposes.

[20] A transaction that is propagated to the network is first verified by different nodes. If the verification is successful, the transaction will be stored in a node's memory pool until a miner includes it in a block whereby it is confirmed. If the amount of newly generated transactions is higher than the maximum amount that can be included in a block, the number of unconfirmed transactions and thereby the mempool size will increase.

[21] Namely: Dash, Digibyte, Dogecoin, Ethereum, Litecoin, Vertcoin, Stellar, Monero, Ripple, Verge.

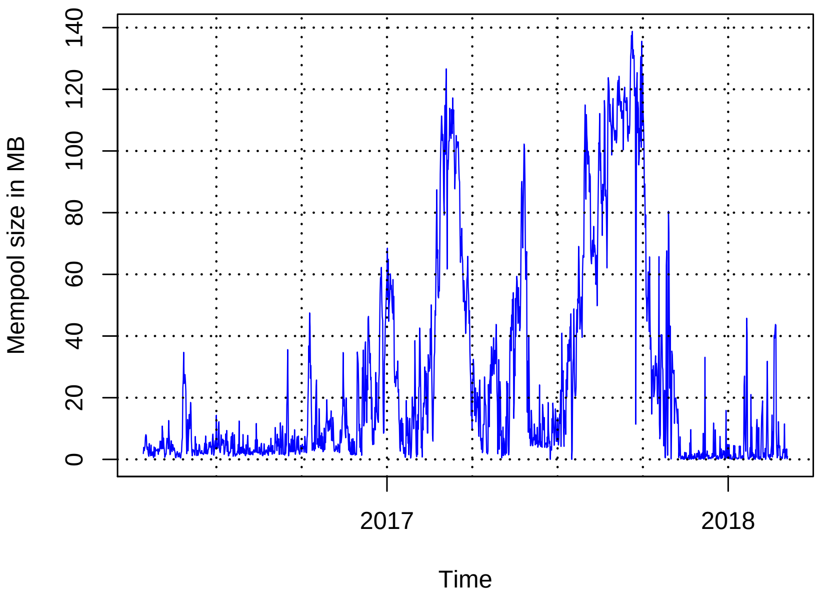


Figure 5: Mempool size in megabytes (Source: `https://blockchain.info`).

factors as well, such as a general increasing interest in cryptocurrencies, a move to smart-contract compatible currencies or diversification of investors.

In February 2017, a pseudonymous developer called *Shaolinfry* suggested a *user activated soft fork* mechanism (UASF) to implement Segwit which he (or she) motivated by a list of shortcomings of the usually applied hash-rate based soft fork activation mechanism under BIP9.[96] In a UASF, all users in the Bitcoin community running a full node enforce the new rules of the soft fork, whereas under BIP9 it is activated by the miners. At a pre-defined date, the so-called *flag day activation*, the soft fork is implemented and all upgraded full nodes would start to reject blocks that are mined under the old rules. Miners are incentivised to follow the new rules as well, since they may loose their reward from mining the block if it is not considered valid by a majority of nodes, which would be the case if they followed the old rules. The proposal attracted wide support in the community. *Shaolinfry* worked out a proper proposal by March 17 (BIP148). The *flag day activation* was chosen for August 01, 2017. The question arose, however, how to ensure that a large majority of users upgraded to the new rules. If some nodes had followed the old and some the new rules,

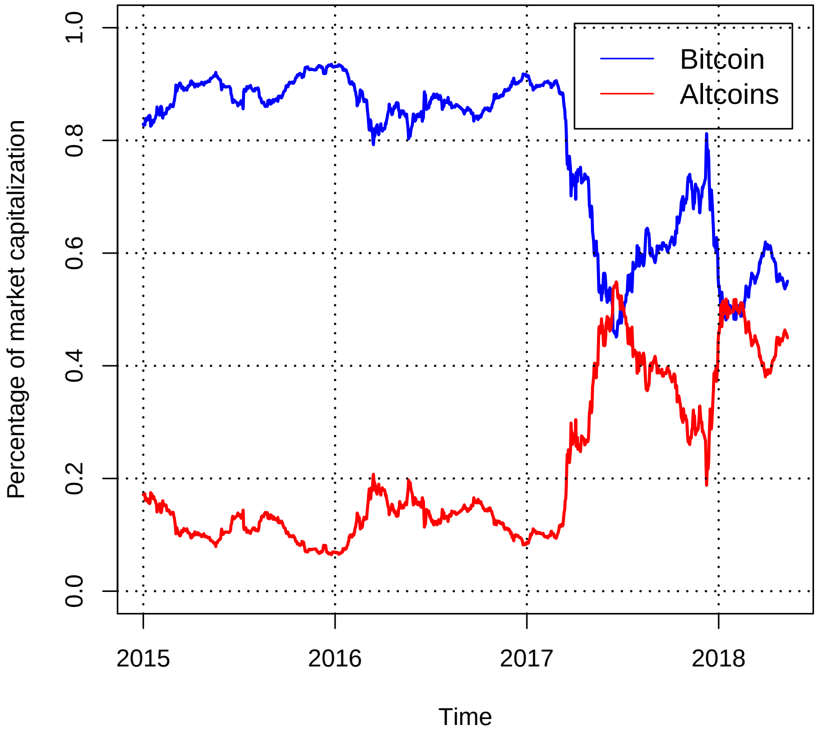


Figure 6: Percentage of Bitcoin's (blue) and selected Altcoins' (red) added market capitalization (Source: `https://coinmetrics.io/`). Note: Altcoins: Dash, Digibyte, Dogecoin, Ethereum, Litecoin, Vertcoin, Stellar, Monero, Ripple, Verge.

the network would have split. Serious criticism emerged[79] which is why other implementation mechanisms were developed.

Further controversial discussions and the introduction of another software client called *Bitcoin Unlimited*, aiming to raise the block size limit per hard fork, triggered the need to hold another conference in New York. Several important companies, people and miners attended it. However, no *Bitcoin Core* contributor was participating.[121] Based on a proposal of Sergio Demian Lerner[74], they agreed to support a combined upgrade of a Segwit soft fork and a following hard fork to increase the block size to two megabytes. The proposals was called Segwit2Mb or Segwit2x[49] and the agreement became known as the *New York Agreement*.

Three ways to activate Segwit existed at that point in time: The *Bitcoin Core* miner-sided activation under BIP9, the UASF user-sided activation and Segwit2x. The UASF, however, was not compatible with the *New York Agreement* which again was not compatible with the way *Bitcoin Core* wanted to implement Segwit. In consequence, James Hilliard created BIP91, also called *segsignal* (see section 4.5 for more details). The Bitcoin community started to signal BIP91 which locked in on July 20, 2017, whereby Segwit locked in on August 9, 2017. After another two weeks grace period, Segwit was officially activated within the Bitcoin network.[121] In February 2018, *Bitcoin Core* introduced default wallet support for Segregated Witness in version 0.16.0.

At about the same time, another important process was ongoing that had started in mid-June 2017 with a blog post by *Bitmain.*[25] They criticized that no *Bitcoin Core* representatives were attending the conference in New York and rejected the UASF since it did not include a block size increase as agreed on in the *New York Agreement.* Instead, they proposed a *user activated hard fork* (UAHF) as a "contingency plan" for BIP148. They advocated a block size increase to eight megabyte, twelve hours and 20 minutes after the Segwit UASF activation. A first software implementation called *Bitcoin ABC*[22] was designed by Amaury Sechét who presented it at the *Future of Bitcoin* conference in Arnhem, Netherlands, at the end of June 2017.[119] In July 2017, the Chinese mining pool *ViaBTC* stated in a blog post that they supported the project described by *Bitcoin ABC* and gave it the name *Bitcoin Cash.*[112] After further development of the project, it was announced on July 22, 2017 that *Bitcoin Cash* would fork from Bitcoin on August 01. Many participants in the Bitcoin community thought that the split would only occur if BIP148 was triggered. However, the disagreement between the users supporting on-chain to the ones approving off-chain scaling solutions was so large that the hard fork eventually took place and the Bitcoin blockchain split into two chains. As a consequence, many long-term advocates of a block size increase switched to *Bitcoin Cash* (e.g. Gavin Andresen[60]).

Despite the implementation of Segwit, the average transaction fees continued to increase on the Bitcoin network. Historically, fees have been very small but

[22] *Bitcoin ABC* is still providing a full node implementation of *Bitcoin Cash.*

started to rise in mid-2017 to a few US-Dollars per transaction. In December 2017, transaction fees skyrocketed up to even 50 US-Dollars.[24] This was highly related to the fact that block space became scarce since most of the blocks were almost one megabyte in size.[23] Also the transaction confirmation time increased considerably.[35] Average transaction fees declined again in 2018 and fluctuate around one US-Dollar per transaction at the moment (July 2018).[24]

The activation of Segwit in August 2017 was crucial for the further development of the Lightning network. In December 2017, the Lightning developers announced that the first Lightning payments were processed in the *ACINQ*, *Blockstream* and *Lightning Labs* implementations.[76] Also other users started to use the alpha implementation and a growing number of Lightning channels were opened. In mid-March 2018, *Lightning Labs* released *lnd* in beta on the mainnet and only shortly after, also *ACINQ's eclair* was released in beta. By August 04, 2018, there are about 2000 nodes which opened up over 11'000 payment channels[1].[120]

3 Block Size Increase

We start our technical analysis with the discussion of a block size increase. The idea of a simple adjustment of the hard-coded limit is technically the simplest improvement, since it requires only minimal code alterations. However, it is also a very controversial idea which triggered a heated debate between advocates and opponents lasting for years. This section first shortly describes what a block in the Bitcoin blockchain is actually composed of, then details several different proposals of how to increase the block size limit and lastly discusses advantages and disadvantages of such a block size increase.

[23] We find a correlation between transaction fees and block size of 0.67.

3.1 Block Structure

We visualize the structure of a block in the Bitcoin blockchain in figure 7.[24] A block does not only consist of transactions but is composed of different components which we call fields. As mentioned above, the hard-coded maximum block size limit is one megabyte of which the largest part is available for transactions. We discuss all components of a Bitcoin block below.[9,15,19,20,111]

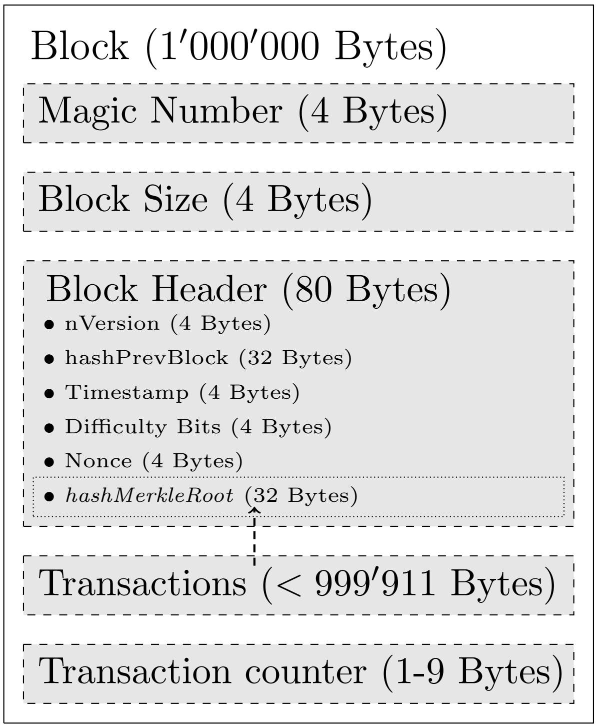


Figure 7: Structure of a block.[9] Note: The numbers in brackets show the size of the respective fields.

Magic Number: Identifies the blockchain network and the start of the block. A Bitcoin block has always the value `0xD9B4BEF9`.

Block Size: Size of the block.

Block Header: Crucial for the consensus of the system since it links a block to its predecessor.

[24] The order of the different fields of the block is not exactly correct. We slightly adapted the order for the sake of simplicity.

nVersion: The protocol version that was in place when the block was mined. Relevant to know the set of block validation rules that were followed. Sometimes used for voting purposes on BIPs. It is a number consisting of 32 bits.

Hash Previous Block: This field links the new block to the old block and builds the foundation of the chain structure of the blockchain. It is the hash value (double SHA-256) of all fields of the previous block's block header. It ensures that content in old blocks cannot be altered without changing this value and consequently the hash of this block header.

Timestamp: The *timestamp* adds information about the point in time a block is mined. The consensus rules in Bitcoin do not mandate strict ordering of block timestamps. For example, block B which is mined after block A can still have an earlier timestamp than block A. This follows from the issue that without a central authority it is not possible to know the exact current network time. Thus, the protocol accepts blocks with a timestamp not later than two hours after current network time and not earlier than the median timestamp of the previous eleven mined blocks.[25] Miners can use this flexibility to introduce additional variation for solving the cryptographic puzzle.

Difficulty Bits: The *difficulty bits* describe the maximum threshold the hash value of the block header must be less or equal to in order to be accepted by the network.

Nonce: The *nonce* is a 32-bit number of arbitrary data. Miner may change the data in the nonce to eventually produce a hash value of the block header which is smaller or equal to the difficulty target. For instance, a miner can start at a pre-defined number and incrementally try different numbers until the target is undercut.

Merkle Root Hash: The *Merkle Root hash* is a compact 256-bit hash

[25] An example are blocks 180966 and 180967. Both were mined on May 20, 2012, but the first one's timestamp is 23:02:53 and the second one's 23:02:13. Even though the first one was mined before the second one, it has a later timestamp.

value of all transactions included in the block. It is calculated by using a Merkle tree, in which transactions are always arranged in pairs and then hashed together until only one number is left, i.e. the Merkle root.

Transactions: Lists all transactions included in a block. The first transaction listed is always the *coinbase transaction* which has to be part of a block and creates new Bitcoin units. Theoretically, miners can send the *coinbase transaction* to whomever they want. In practise, however, they send the newly created Bitcoin units to themselves as a remuneration for mining the block.

Transaction counter: Counts the number of transactions included in the block.

3.2 Proposals to Increase the Block Size

Over the years, there have been many different proposals how the block size could be increased. Already in 2010, Satoshi Nakamoto suggested that a larger block size could be phased in over a certain time period as soon as the network is closer to needing it.[95] Eight BIPs between BIP100 and BIP109 and numerous other proposals in forums, mailing lists, conferences etc. deal with a block size increase. Discussing all of them would be beyond the scope of this paper, which is why we focus on the most relevant ones in our view.

Each proposal suggests either a one-time increase or a dynamic limit, where the limit changes over time depending on some pre-definable factors. All proposals discussed below would have required a hard fork to be implemented. None of them, however, was included in the *Bitcoin Core* source code.

BIP100[58]

Jeff Garzik, Tom Harding and Dagur Valberg created BIP100 on June 11, 2015. The proposal suggests that the one megabyte block size limit should be replaced with a variable limit set by coinbase vote. A 75% supermajority can trigger a change of the maximum block size. The in- or decrease must not exceed 5% of the previous block size limit.

Miners can vote about altering the block size during a 2016 blocks retargeting period which equals about two weeks. The vote is encoded within the scriptSig of the coinbase transaction where a megabyte value is proposed using the BIP100 pattern. For instance, `/BIP100/B8/` is a vote for a eight megabyte block size limit.

Votes are ordered at the end of a retargeting period. If there is no vote in a coinbase transaction, it is counted as a vote for keeping the current block size limit. Subsequently, the first and the third quartile are analysed. We define the new block size limit prevailing at the first quartile as the `raise value` and the one at the third quartile as the `lower value`. At the first quartile (i.e. the 1512th highest block), 75% of the votes are in favour of the `raise value` or a larger one. Thus, if the `raise value` is larger than the current value but smaller than the current value plus 5%, the `raise value` is set as the new block size limit for the subsequent 2016 blocks. If the `raise value` exceeds the current value plus 5%, the new value is set at `raise value` plus 5%. The process of decreasing the block size is analogous to raising the block size. If the `lower value` is less than the current value, but greater than the current value minus 5%, the block size is reduced to the `lower value` for the next period. If it is even lower than the current value minus 5%, the new block size limit is set to `lower value` minus 5%. If the `raise value` is lower and the `lower value` greater than the current value, there is no change of the block size limit.

For the sake of understanding the proposal, we give a quick example. Assume for now that the current block size is five megabytes and that there are six coinbase votes. One of them voted for a block size of four megabytes, another one for six megabytes, three for seven megabytes and one for eight megabytes (i.e. 4,6,7,7,7,8). The `raise value` equals the first quartile which is six megabytes and the `lower value` equals the third quartile which is seven megabytes. On the one hand, the `lower value` is not smaller than the current value which is why the block size is not decreased. On the other hand, the `raise value` is larger than the current block size which means that more than 75% of the votes are in favour of an increased block size and the limit is raised. However, the `raise` `value` exceeds the current block size plus 5% (i.e. 5.25 megabytes). Hence, the new limit for the subsequent retargeting period is set to 5.25 megabytes.

A point of criticism of this proposal is that it gives too much power to the miners, since only they are able to vote about the block size limit while users cannot directly influence it.

BIP101[2]

On June 22, 2015, Gavin Andresen proposed BIP101. The idea was to introduce a predictable growth rate of the block size limit.

BIP101 would be deployed if miners signal their support in at least 750 of 1000 consecutive blocks. They do so by setting the first, second, third and thirtieth bit in the version number of the block header equal to `0x20000007` in hex. If at least 750 blocks are in favour, BIP101 would be activated after a two weeks grace period.

When activated, the block size increases directly from one megabyte to eight megabytes. Afterwards, it doubles every two years for ten periods. After 20 years, a maximum block size limit of 8.192 gigabytes would be reached. In between the doubling periods, the block size limit increases linearly based on the timestamp in the block header.

According to Andresen, the initial jump to eight megabyte was mainly chosen in regards to miners on bandwidth-constrained networks. The two years doubling interval was chosen based on long-term growth trends for CPU power, storage and internet bandwidth.

Elastic Block Cap with Rollover Penalties[93]

Also in June 2015, Meni Rosenfeld proposed an elastic limit with a rollover fee pool and a penalty for large blocks. Miners are free to chose an arbitrarily large block size. However, there is a penalty for producing larger blocks where the penalty is a convex function of the block size. The penalty is deducted from the transaction fees a miner receives upon mining a valid block and is paid into a rollover fee pool.

A miner's earnings are composed of the block reward, the transaction fees and a share of the rollover fee pool minus the possible penalty, i.e.

$$Earnings = \underbrace{block\,reward + fees + fraction\,of\,fee\,pool}_{Income} - penalty \geq 0$$

The block is invalid, if the penalty is larger than the income. Thus, a miner faces a trade-off between larger blocks with the disadvantage of a penalty and smaller blocks with the disadvantage of fewer transaction fees.

BIP103[127]

On July 21, 2015, Pieter Wuille proposed BIP103 which intends to connect block size growth to technological growth. Wuille stated that a large block size is only problematic if it increases faster than technological progress. If BIP103 had been accepted, the first step would have occurred only in January 2017 to give the network enough time to prepare for the hard fork. Afterwards, the block size limit would increase by 4.4% every 2^{23} seconds (around 97 days). This results in an annual growth rate of 17.7% which is consistent with the average growth rate of bandwidth over the years before 2015. The last step would occur in July 2063, when the block size limit would reach about 1.87 GB. If the growth rate of technological progress would shrink at some point in the future, the growth rate of the block size limit could be reduced with a soft fork.

BIP 102 & 109[5,56]

Jeff Garzik in BIP102 and Gavin Andresen in BIP109 both intended to change the maximum block size limit in a one-time increase to two megabytes.

Bitcoin Cash

As discussed in section 2, Bitcoin split into Bitcoin and *Bitcoin Cash* on August 01, 2017. *Bitcoin Cash* is identical to Bitcoin up to block 478'558 and maintains a separate blockchain from block 478'559 on.[90] Every user who owned Bitcoin units at that point in time received the same amount of *Bitcoin Cash* units. Moreover, *Bitcoin Cash* had mainly three new features. Firstly, it exhibited a block size of eight megabytes.[26] Secondly, the transaction signature hashing algorithm was slightly adapted to verify its distinction from Bitcoin. Lastly,

[26] *Bitcoin Cash* increased the block size to 32 megabytes in May 2018.

if the hash power in the network is low, the difficulty may adjust faster than the regular 2016 blocks, which was called the *Emergency Difficulty Adjustment* (EDA).[99] The *EDA* was mainly implemented to deal with a dramatically lower hash rate after the split. Because it failed to stabilize the block interval to ten minutes afterwards, *Bitcoin Cash* had to perform another hard fork in November 2017 to adjust the algorithm.

3.3 Discussion

A simple increase of the block size limit is probably the most controversial proposal in the Bitcoin scaling debate, which triggered a heated debate between advocates and opponents lasting for years. Below, we present some important arguments of both sides.

An increase of the block size limit is technically very easy to implement and achieves the objective of rising the maximal number of transactions that can be processed in the network. On the other hand, opponents of an increase state that a simple increase is not enough, if Bitcoin wants to be on a par with payment networks like Visa or PayPal. If blocks become very large, other physical limitations like bandwidth, storage capacities for a bigger blockchain or the time used to verify the transactions limit the number of possible transactions on the network. Therefore, the block size cannot be increased arbitrarily and other solutions besides simple block size increases are needed to scale Bitcoin successfully.[23,43,46] Regarding these increased costs for storing a bigger blockchain, the concept of blockchain pruning is introduced. With pruning, a full node only stores the latest blocks - e.g. the last 550 - but still maintains the UTXO set[27] on a local storage space to validate transactions. Obviously, it is critical for network security reasons that there are still enough nodes storing the whole blockchain.[38]

Furthermore, sceptics of a block size increase claim that the increased costs of running a full node will reduce the number of nodes in the network and thus is

[27] Unspent transaction outputs (UTXO) are outputs of transactions that have not been spent in another transaction yet.

detrimental to decentralization in the Bitcoin network. On the contrary, advocates state that running a full node is not as expensive as often claimed. Even with an increased block size up to 20 megabytes, running a full node should not cost more than five to ten US-Dollars per month based on some calculations by Andresen (2015)[6] and Bevand (2017)[10].

An often used counter-argument is that a block size increase had to be implemented through a hard fork. The Bitcoin network has no experience in an organized hard fork and many users fear another split. Proponents answer that hard forks have occurred in other networks without any negative consequences. Monero, for instance, successfully performed hard forks about every six months. Only recently, a hard fork about a change in the CryptoNight Proof-of-Work algorithm, which makes it impossible to mine with ASICs, led to a network split in April 2018.[118] Also *Bitcoin Cash* managed to increase the block size from eight to 32 megabytes through a hard fork in May 2018.

Moreover, larger blocks may ease the creation of malicious blocks which take longer to verify and can be abused in Denial of Service (DoS) attacks. Furthermore, more spam or useless transactions in general might be added to the blockchain, since the cost of including them is lower with bigger blocks. This would inflate the blockchain unnecessarily. In contrast, proponents argue that there is no critical risk of DoS attacks in the current Bitcoin network, because the costs of mining have increased tremendously over the last years. In early stages, the reward for mining a block was only a few US-Dollars and the costs were negligible. Hence, there was a serious risk of DoS attacks on the network. Nowadays, however, with a Bitcoin price of a few thousand US-Dollars and the current coinbase reward of 12.5 Bitcoin units, a miner gains several tens of thousands of US-Dollars for mining a block. Consequently, the costs of mining have increased as well which makes creating a poisonous block very expensive.[4] Moreover, Sergio Lerner showed already in 2013 that even with a one megabyte block size limit attackers could produce blocks which are very expensive to verify.[73] The fact that nobody has ever attacked Bitcoin in such a way shows that DoS attacks are probably not worthwhile and that it may be more profitable to just invest the resources needed for such an attack into mining itself.

Another notable point of discussion are transaction fees. If the block size is not increased and block space becomes a scarce resource, transaction fees are going to rise which makes the network less attractive as a payment network. On the contrary, opponents of a block size increase highlight the relevance of a healthy fee market, such that miners are still incentivised to provide computing power as soon as the block reward approaches zero. This fee market could only emerge, if the block space is limited artificially.

Lastly, larger blocks are relayed slower over the network due to natural physical limitations. This would be disadvantageous for some miners. For instance, if Chinese mining pools hold the majority of the network's mining power and thus create most of the blocks, US and European miners would be disadvantaged, since there exists some latency until the block is broadcast over the network.[28]

4 Segwit

In this section, we are going to discuss Segregated Witness (Segwit). Segwit was first presented at the *Scaling Bitcoin - Hong Kong* workshop. Its original purpose was not to allow for more transactions on the network but to solve the transaction malleability issue. Subsequently, we first provide some technical background to understand Segwit. The structure of a transaction and the most relevant transaction types are discussed and transaction malleability is explained. Afterwards, we explain Segwit's technical details and lastly illustrate the steps to Segwit's activation in August 2017.

4.1 Transaction Structure

A Bitcoin transaction (without Segwit) has a general serialization which is illustrated in figure 8. The only exception is the *coinbase transaction*, which we are

[28] The Great Firewall of China would amplify this effect even more.

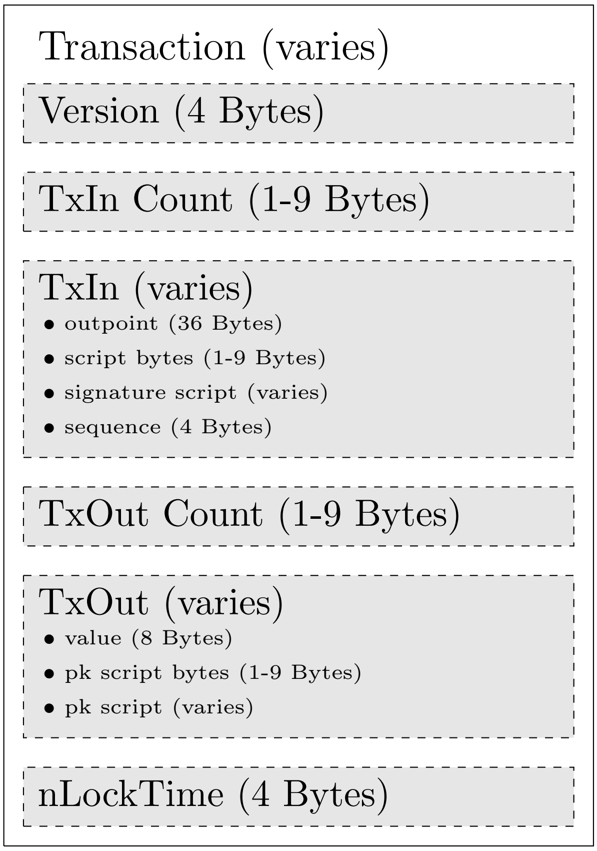


Figure 8: Serialization of a Bitcoin transaction without Segwit. Note: The numbers in brackets show the size of the respective fields.

not going to discuss in more detail due to lack of relevance for the scaling debate. We quickly present the original serialization and show the differences to a Segwit transaction in section 4.4. [9,15,22,47]

Version: The transaction version number indicates the consensus rules under which the transaction was created.

TxIn Count: The number of inputs.

TxIn: The transaction inputs. There are several inputs in a transaction if Bitcoin units are spent from various outputs.

Outpoint: Links the transaction to a previous output. It is composed of a 32-byte transaction ID (TXID)[29], i.e. the SHA256d hash value of the

[29] The transaction ID is the critical part for transaction malleability described in section 4.3.

previous signed transaction and a 4-byte index that indicates which output of the specific transaction is referenced.

Script Bytes: The number of bytes in the signature script. The maximum is 10'000 bytes.

Signature Script: A script, also called scriptSig, which has to satisfy the conditions stated in the public key script (see TxOut). It contains several parameters depending on the transaction type. In the case of a pay-to-address transaction for example, it includes a *secp256k1* signature and a public key. The public key hash (i.e., the Bitcoin address) has to be equal to the public key hash stated in the public key script in the transaction output. The signature is used to prove that someone controls the private key that corresponds to the public key and can hence rightfully spend the Bitcoin units connected to the specific output.

Sequence Number: Part of all transactions which mostly is `0xffffffff`. It is used for the implementation of relative lock time under BIP68 (see section 5.2).

TxOut Count: The number of outputs.

TxOut: The transaction outputs. There are several outputs in a transaction if Bitcoin units are sent to several addresses.

Value: The number of Satoshis[30] to spend. The total value of all outputs must not exceed the total amount of all inputs.

Public Key Script Bytes: The number of bytes in the public key script. The maximum is 10'000 bytes.

Public Key Script: The public key script or scriptPubKey defines conditions that have to be satisfied such that someone can use this output as a subsequent input (see TxIn). In a pay-to-address transaction, it includes the public-key-hash (i.e. the Bitcoin address) such that the

[30] A Satoshi is currently the smallest unit of the Bitcoin currency and equals 0.00000001 BTC.

network knows who is allowed to spend the Bitcoin units in the next transaction.

nLockTime: The *nLockTime* field states the earliest point in time a transaction can be included in a block of the blockchain. It can be either defined as block height or as block time. Originally, *nLockTime* was always set to zero such that a transaction could be included in any block. Since *Bitcoin Core* version 0.11.0., *nLockTime* is set to a recent block number by default.[21]

4.2 P2PKH and P2SH Transactions

To understand the new Segwit transaction types in section 4.4, we quickly review how Pay-to-public-key-hash (P2PKH) and Pay-to-script-hash (P2SH) transactions work.

Imagine Emily wants to send some Bitcoin units to Bob with a P2PKH transaction. Bob takes his private key, generates a corresponding public key via Elliptic Curve Digital Signature Algorithm (ECDSA), double hashes it and encodes it with base58check to get his Bitcoin address. A P2PKH address always has a prefix of 1 as an identifier. He then sends the Bitcoin address to Emily. In the scriptPubKey, she includes both Bob's decoded address and conditions Bob has to meet to spend the Bitcoin units (see figure 9). Emily propagates the transaction to the network, whereby it is added to the blockchain and the output is stored in the unspent transaction output (UTXO) set of full nodes. If Bob wants to spend the UTXO, he creates a new transaction and references the specific output with the transaction ID of Emily's transaction and the index number. Furthermore, he includes his public key and a signature in the scriptSig to satisfy the conditions defined in the scriptPubKey. Bob can now propagate the transaction to the network. The verification procedure requires evaluation of the scriptPubKey and the scriptSig. Each item of the scriptSig and the scriptPubKey is executed one at a time. First, the signature and the public key of the scriptSig are put on the stack. Then, the operations (i.e. the aforementioned conditions) defined in the scriptPubKey are executed. A copy of the public key is created with `OP_DUB` and

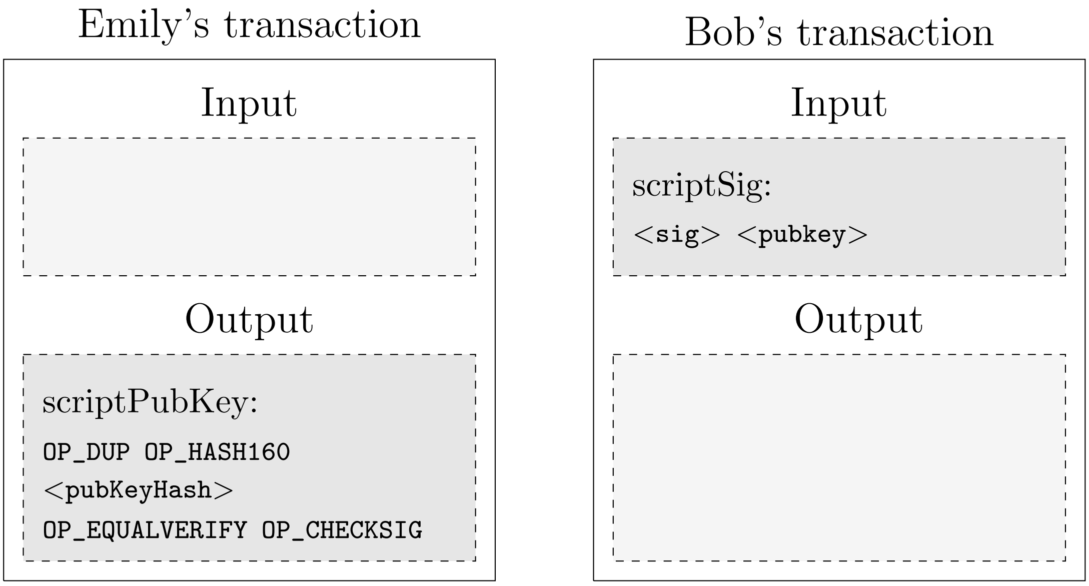


Figure 9: Pay-to-public-key-hash (P2PKH) transaction.

then hashed with `OP_HASH160`. Next, the public key hash Bob gave to Emily is pushed on the stack such that two copies of the public key hash are on top of it. `OP_EQUALVERIFY` verifies whether they are equal. Lastly, `OP_CHECKSIG` checks the signature against the public key. If everything runs successfully, a single TRUE is pushed on top of the stack.

A Pay-to-script-hash (P2SH) transaction allows flexible payout conditions. Any arbitrary conditions can be defined in the redeemScript.[31] P2SH is often used for multi signature payout conditions. For instance, Bob could state in the redeemScript that he cannot spend the Bitcoin units he received from Emily with only one key but two out of three private keys have to sign the transaction. Bob can generate a base58check encoded Bitcoin address corresponding to the specific script, such that Emily knows where to send the Bitcoin units to. These addresses always start with a prefix of value 3.[32] When she wants to send Bitcoin units to Bob's redeemScript, she creates a transaction and includes the hash of the redeemScript and the conditions Bob has to meet in the scriptPubKey and

[31] The redeemScript is the script in which the payout conditions are defined. It is included in the scriptPubKey and has to be provided in the scriptSig if someone intends to spend the respective output.

[32] The prefix of an address became very relevant with the introduction of Segwit addresses which also have a prefix of value 3. We further discuss the details below.

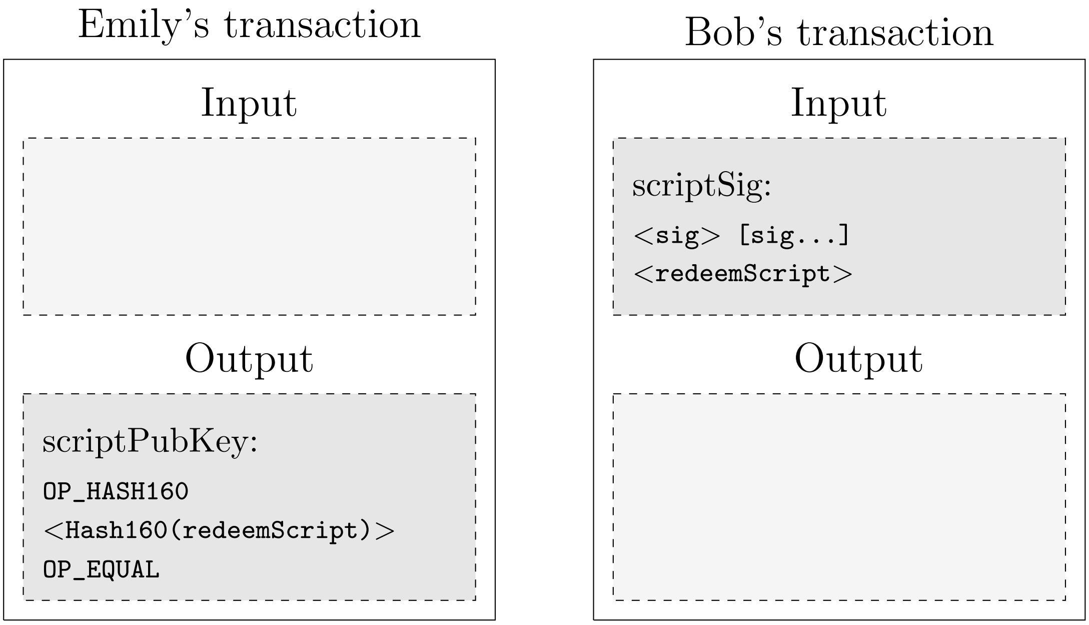


Figure 10: Pay-to-script-hash (P2SH) transaction.

propagates the transaction. If Bob wants to spend the output in a following transaction, he includes the redeem script and the required signatures in the scriptSig (see figure 10). The network will verify if Bob's transaction is valid in a similar manner as before. First, the redeemScript in the scriptSig is put on the stack. Afterwards, the operations in the scriptPubKey are executed. The redeemScript is hashed with `OP_HASH160` and compared to the hash of the redeemScript in the scriptPubKey with `OP_EQUAL`. If both scripts are equal, the redeemScript provided in the scriptSig is analysed and verified. If the script is valid, the transaction can be considered valid.

4.3 Transaction Malleability

As aforementioned, Segregated Witness was originally developed to solve transaction malleability. Therefore, before detailing Segwit we want to explain what transaction malleability is and how it has been exploited. The basic Bitcoin consensus and payment mechanism is not endangered by transaction malleability, but depending on the set up it can affect services built on top of it.[102]

Transaction malleability is related to the transaction ID. As described in section

4.1, the information in a transaction including the signature is hashed to get the transaction ID. To ensure that this ID is unique and distinct, applying the hash function on a transaction has to result in the same ID every time this operation is performed. Unfortunately, it is possible to slightly adapt the signature in a transaction without making it invalid, which leads to a different transaction ID.

A user can choose which parts of a transaction are signed by the signature. There are three options available called signature hash types or sighash flags.

- `SIGHASH_ALL` is the default sighash type and signs all inputs and outputs except the signature script.
- `SIGHASH_NONE` signs all inputs, but none of the outputs.
- `SIGHASH_SINGLE` signs only the output corresponding to a specific input.

The parts of the transaction that are not signed by the signature are modifiable. The signature script is not protected by any of the signature hash types, since it contains the *secp256k1* signature which cannot sign itself. Hence, it is possible to slightly modify the signature in the signature script. This, however, results in a different hash value of the transaction, i.e. a different transaction ID. Very simplified, one can imagine the numbers "32" and "032" to be identical in various applications.[33] However, the additional "0" would change the hash value.[122] Furthermore, additional information can be included in the scriptSig. For example, an `OP_NOP` operation could be added that does not do anything or a combination of `OP_DUP OP_DROP` that first duplicates the signature on the stack and then removes it again.[70] These operations would change the hash value without invalidating the transaction. Transaction malleability is not problematic if a transaction is already included in the blockchain, however, it becomes an issue if the transaction ID is changed before the transaction is added to the blockchain.

[33] More technically speaking, there are two main options to exploit transaction malleability. First, the verification of the signatures relies on OpenSSL. OpenSSL, however, would accept various deviations from the DER standard applied on the signatures. This was fixed in BIP66. Secondly, the Elliptic Curve Signature Algorithm (ECDSA) Bitcoin uses can be abused. Given a signature (r,s), it is possible to derive the complementary signature $(r, -s \bmod n)$ without knowing the private key. The complementary signature would still be valid, produces, however, a different hash.[69]

There are several ways, how transaction malleability can be exploited. For example, a Bitcoin user could request Bitcoin units from an exchange or wallet service and then alter the signature before it enters the blockchain. Consequently, two transactions co-exist which spend the same in- and outputs. Both are valid but both have a unique transaction ID. In case the second transaction enters the blockchain first, the Bitcoin user could contact the exchange or wallet service claiming that the transaction did not succeed. If the wallet service checks the blockchain it will not find the transaction ID of the original transaction and might resend the Bitcoin units. Hence, transaction malleability enables to alter the transaction such that it looks like Bitcoin units have not been sent to a Bitcoin wallet, even though they have.

This is exactly what was claimed to have happened when transaction malleability became very relevant for the first time in the Bitcoin network in February 2014. Users of the Bitcoin exchange platform *Mt. Gox* repeatedly faced problems with the outpayment of their Bitcoin units. On February 07, 2014, *Mt. Gox* released a statement that they halt all Bitcoin withdrawals to better work on the technical issues faced.[101] Three days later, they released another statement in which they made a bug in the Bitcoin code (i.e. transaction malleability) responsible for their problems, even though this issue has been known for a significant amount of time.[51,61] In a nutshell, users were able to withdraw more money then they actually owned, since it seemed like transactions were often not confirmed and *Mt. Gox* re-sent Bitcoin units based on transaction IDs. It is important to highlight that this issue could have been avoided, if *Mt. Gox* had dealt properly with the issue.

A further example how transaction malleability can be exploited are DDoS attacks. An attacker could create mutant transactions and propagate them to several exchanges. If this happens in a high intensity, exchanges might face logistical problems.[34]

Lastly, transaction malleability strongly hinders the use of unconfirmed transaction dependency chains. Theoretically, it is possible to use an output of a transaction A as an input of transaction B before transaction A is included in the blockchain. If, however, the signature of transaction A is modified, the TXID will change and as a consequence, transaction B cannot successfully reference transaction A anymore. This is a problem especially for payment channels as applied, for example, in the Lightning network (see section 6).

4.4 Technical Analysis

In this section we explain the technical details of Segregated Witness. Not only does Segwit solve transaction malleability, but also does it help to increase the number of transactions that can be processed over the network. By solving transaction malleability, Segwit served as an important building block for the further development of second layer solutions like the Lightning network. The way Segwit was implemented enabled also further upgrades, described at the end of this section.

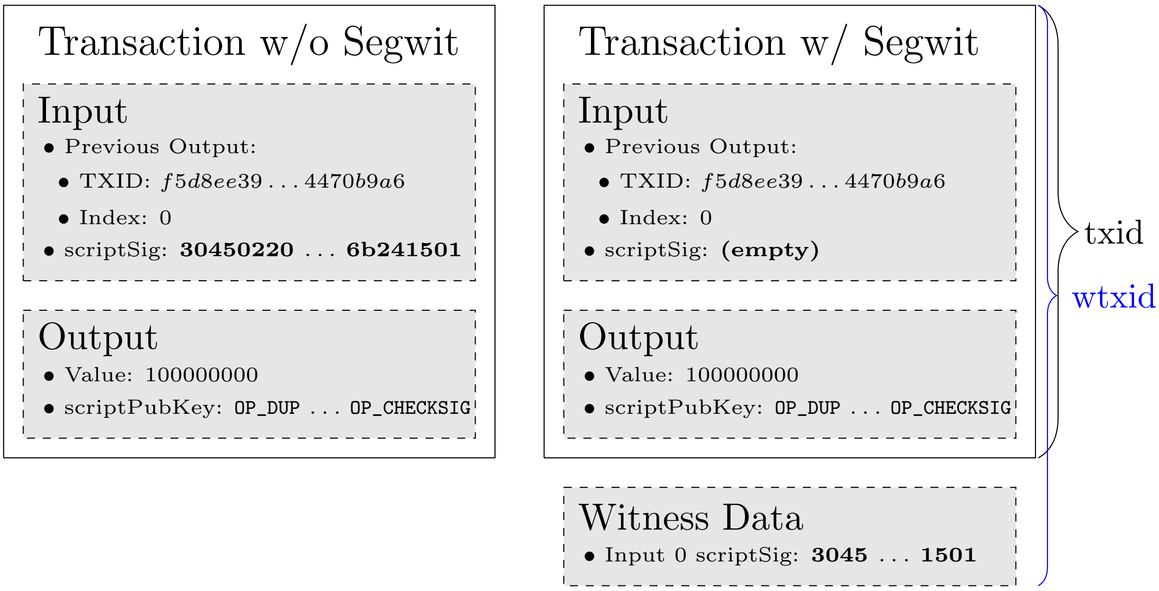

Transaction w/o Segwit
Input
• Previous Output:
• TXID: $f5d8ee39\ldots4470b9a6$
• Index: 0
• scriptSig: **30450220 ... 6b241501**
Output
• Value: 100000000
• scriptPubKey: OP_DUP ... OP_CHECKSIG
Transaction w/ Segwit
Input
• Previous Output:
• TXID: $f5d8ee39\ldots4470b9a6$
• Index: 0
• scriptSig: **(empty)**
Output
• Value: 100000000
• scriptPubKey: OP_DUP ... OP_CHECKSIG
Witness Data
• Input 0 scriptSig: **3045 ... 1501**
txid
wtxid

Figure 11: Simplified structure of a transaction with and without Segwit. The transaction consists of one input and one output and sends one Bitcoin unit.

The basic idea of Segwit is that the content of the scriptSig is taken out of the old data field and included in an additional data field called *witness* as illustrated in figure 11. The *witness* contains the data required to check the transaction validity but does not have an influence on transaction effects (i.e. it has no effect on inputs and outputs). The transaction ID is still calculated by the same data fields as before, but the scriptSig consists of an empty string under the new serialization. Thus, if someone modifies the signature, the transaction ID will not change. Since the witness data is only used at the time of validation, there is no need to store it on the blockchain. This reduces the total storage requirements of the blockchain considerably.

4.4.1 Serialization of a Segwit Transaction

The serialization of a Segwit transaction is defined in BIP144, as illustrated in figure 12. The data fields are equal to a non-Segwit transaction, except for the *Marker*, *Flag* and the *Witness Script* which are added. The marker field ensures

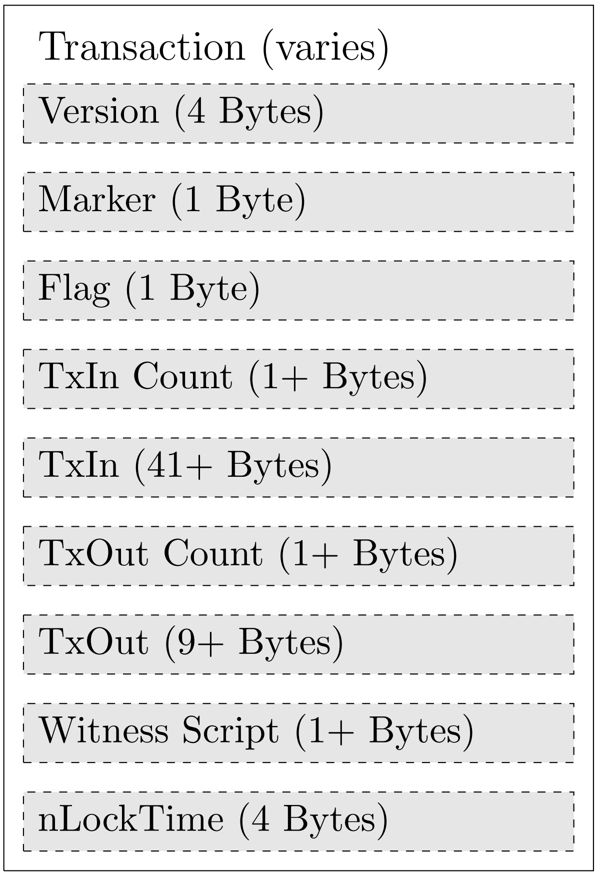


Figure 12: Serialization of a Segwit transaction. Note: The numbers in brackets show the size of the respective fields.

that a node running the old software is not validating a transaction using the new serialization. It must be a 1-byte zero value (`0x00`). The flag field can be interpreted as a bit vector and allows to add more extra non-committed (i.e. not signed) data to a transaction. It contains a 1-byte non-zero value (currently this must be `0x01`).

4.4.2 Commitment to the Signatures

For the consensus protocol it is crucial that the blocks commit to the signatures of the transactions. In other words, a malicious node must not be able to easily modify a signature in a block and relay the modified one. With the old transaction serialization, this is prevented by building a Merkle tree where the transaction IDs of all transactions serve as Merkle leaves. The Merkle root is included in the block header in the *hashMerkleRoot* field as illustrated in figure 7. If someone alters the signature in any transaction, its ID would change which leads to a different Merkle root. Consequently, the block header's hash value is modified as well. With Segwit, the signature is not included in the transaction ID and hence this prevention mechanism fails. Instead, two Merkle trees are built for Segwit transactions.

The first one is the same as in the old serialization. The *txid* used for the leaves of the tree is calculated by applying a double SHA256 hash function on `[nVersion][txins][txouts][nLockTime]`[34]. The second Merkle tree's leaves are represented by the witness transaction IDs (*wtxid*) of all transactions except the coinbase. The *wtxid* is calculated by applying the double SHA256 hash on the new serialization format, i.e.
`[nVersion][marker][flag][txins][txouts][witness][nLockTime]` (see figure 12). *wtxid* thus commits to the signatures. The Merkle root of the second tree is encoded in the scriptPubKey of the coinbase transaction. The coinbase again is included in the first Merkle tree whereby a block successfully commits to

[34] The *TxIn Count* and *TxOut Count* fields depicted in figure 8 are here included in `[txins]` and `[txouts]`.

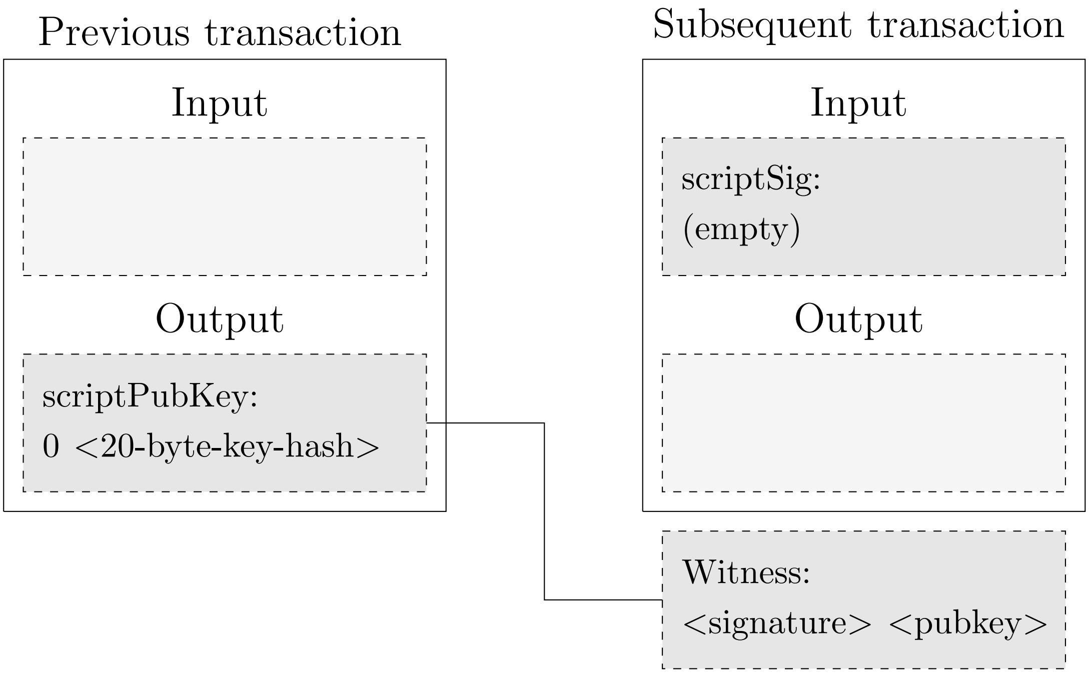


Figure 13: Pay-to-witness-public-key-hash (P2WPKH) transaction.

the signatures of the transactions.[78]

4.4.3 Segwit Transaction Types

The content of the scriptPubKey in a Segwit transaction differs compared to a conventional transaction. It starts with a 1-byte push opcode which can take values from 0 to 16 (version byte) and is followed by a data push byte vector between byte 2 and 40 (witness program). Each value of the version byte has a different meaning and can activate a different operation. Witness validation is triggered depending on pre-defined requirements. Specifically, we distinguish between two cases, *native witness programs* and *P2SH witness programs*.

Fist, witness validation is triggered if the scriptPubKey consists exactly of a version byte and a witness program. Additionally, the scriptSig must be empty. This is called a *native witness program*. We illustrate this on the basis of a pay-to-witness-public-key-hash (P2WPKH) transaction in figure 13. In this case, the scriptPubKey must consist of a version byte taking value 0 and a 20 bytes witness program. If this requirements are fulfilled and the scriptSig is empty, the witness is checked and validated. For a P2WPKH transaction, the witness program equals

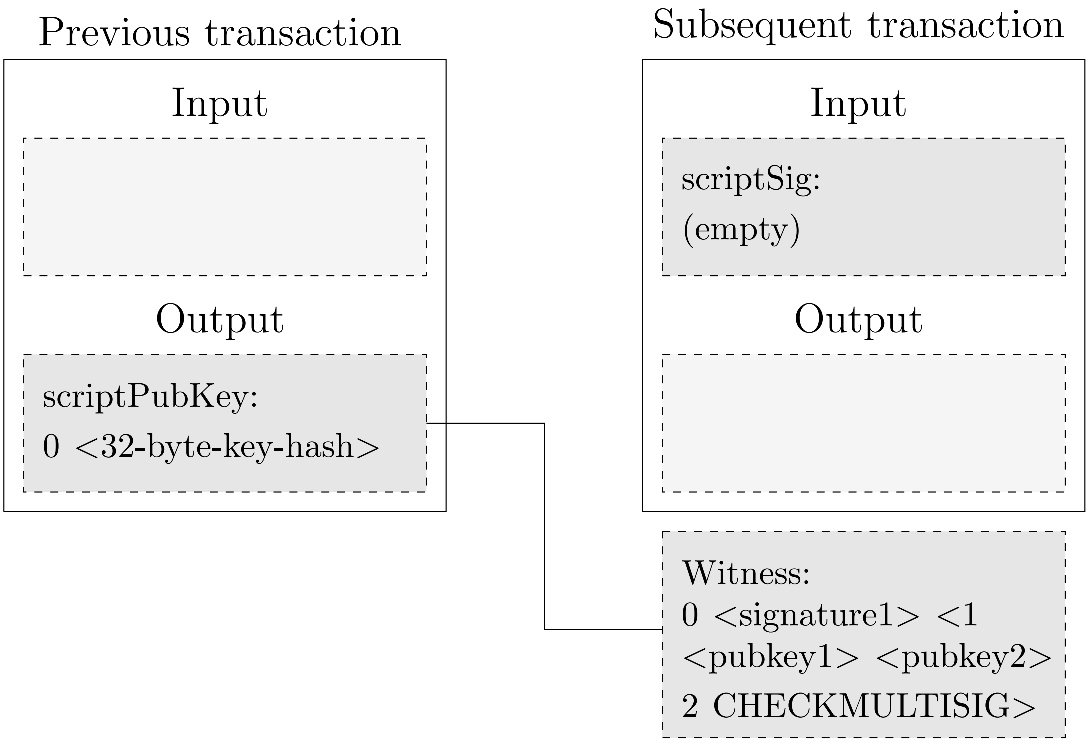


Figure 14: 1-of-2 multisig Pay-to-witness-script-hash (P2WSH) transaction.

the double hash of a compressed public key.[35] The witness field consists exactly of a signature and a public key (each $\leq$ 520 bytes). For the transaction to be valid, the hashed public key in the witness field must match the witness program and the signature must be valid when verified against the public key.

The case of a Pay-to-witness-script-hash (P2WSH) transaction is similar to a P2WPKH transaction. The version byte has value 0 as well, but the witness program consists of 32 bytes, as illustrated in figure 14. The witness program is the SHA256 hash[36] of the serialized witness script.[37] To validate the transaction, the hash of the witness script must match the 32 byte witness program. If this is true, the witness script is examined and validated. Only if everything succeeds, the transaction is valid.

The content used in the scriptPubKey (i.e. either the double hash of the public

[35] `RIPEMD160(SHA256(ECDSA_publicKey))`.

[36] Before Segwit, the P2SH was secured by the HASH160 algorithm. There are, however, security issues if for example one of the signers aims to steal the funds. Using the SHA256 algorithm for P2WSH should resolve this.[13]

[37] The witness script is the counterpart of the redeemScript in case of a Segwit transaction.

key or the hash of the witness script both prefixed with the version byte taking value 0) can be interpreted as native Segwit address. If a user sends this address to someone, support for Segwit and the ability to receive native Segwit transactions is signalled.

If the version byte in the scriptPubKey is 0, but the witness program is neither 20 nor 32 bytes, the script must fail. If it is a number between 1 and 16, no further examination of the witness field takes place. These numbers are reserved for future extensions.

The above described native Segwit transactions are more efficient and use less space than the *P2SH witness programs* we discuss below. However, they only work if the sender and the receiver support Segwit. Since Segwit is implemented as a soft fork, not all users will run Segwit compatible clients from the beginning, wherefore a backwards compatible transaction format has to be implemented.

P2SH witness programs provide backward compatibility. Witness validation is triggered if the scriptPubKey is a P2SH output (without a version byte) and the redeemScript in the scriptSig consists exactly of a version byte and a witness program. There a two forms, P2SH-P2WPKH and P2SH-P2WSH.

The case of a P2SH-P2WPKH is illustrated in figure 15. The redeemScript in the scriptSig is composed of a 0 version byte and the double hash of a compressed public key.[38] The scriptPubKey consists of a 20 byte HASH160 of the redeemScript and the conditions to spend the coins again. This 20 byte hash is used for creating a PS2H-P2WPKH address.[39] It has a prefix of value 3 and is indistinguishable from a non-Segwit P2SH address until the output is spent and the redeem script exposed which makes it backwards compatible.

To validate the transaction, the HASH160 function is applied on the redeemScript and then compared to the 20 byte script hash in the scriptPubKey. If successful the public key and the signature are verified. Compared to the native P2WPKH transaction, more space is used because the scriptSig is not empty. However,

[38] `RIPEMD160(SHA256(ECDSA_publicKey))`.

[39] Under base58check encoding this is: `[one-byte version][20-byte hash][4-byte checksum]`.

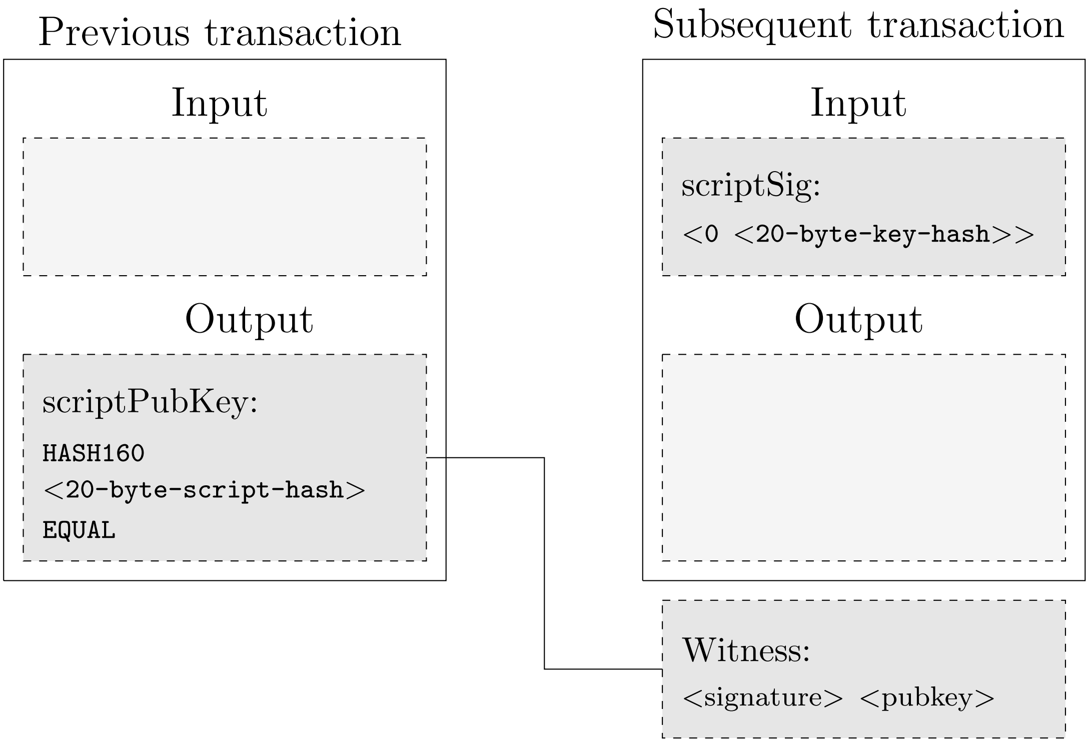


Figure 15: P2SH-P2WPKH transaction.

this transaction type is backwards compatible for all clients since *Bitcoin Core* version 0.6.0. Furthermore, the transaction size of this type is still smaller than a conventional Bitcoin transaction, since the scriptSig only contains a hash value and the signature and public key are part of the witness field.

Moreover, there are P2SH-P2WSH transactions. The scriptPubKey is equal to the P2SH-P2WPKH case. The redeemScript in the scriptSig is composed of a version byte taking value 0 followed by a 32 bytes single SHA256 hash of a witness script. The corresponding address is created using the 20-byte hash in the scriptPubKey with an additional prefix of value 3 and a checksum at the end. As above, the address is indistinguishable from a basic P2SH address until the UTXO is spent. To validate the transaction, the HASH160 function is applied on the redeemScript and the hash value compared to the 20 bytes hash in the scriptPubKey. If they are equal, the witness script is checked. The detailed content of a 1-of-2 multi-signature P2SH-P2WSH script is listed below.

```
scriptPubKey:  HASH160 <20-byte-hash> EQUAL
scriptSig:     <0 <32-byte-hash>>
witness:       0 <signature1> <1 <pubkey1> <pubkey2> 2 CHECKMULTISIG>
```

4.4.4 Anyone-can-spend Output

Since Segwit was implemented as a soft fork, some nodes in the network will be Segwit compatible and others will not. The nodes running old software will not be able to read the Segwit outputs and thus interpret it as an anyone-can-spend output. This leads to several problems we discuss further below.

An upgraded node has the possibility to signal support for Segwit while a non-upgraded node has not. Consequently, Segwit transactions cannot be sent to all nodes. A non-upgraded node normally uses either a P2PKH or a P2SH address (see section 4.2). A P2PKH address has a prefix of value 1 and can therefore be distinguished from a Segwit transaction. A Segwit transaction must never be sent to a node using an address with prefix 1. If an address has a prefix of value 3 things become more confusing. It could be either a P2SH address of a non-upgraded node or a *P2SH witness program*.

A non-upgraded node receiving a Segwit transaction cannot interpret the scriptPubKey and reads it as an anyone-can-spend output, i.e. every user would be able to spend the anyone-can-spend output without using a signature. This is prevented by the miners who ensure that the network participants' actions are in line with the consensus protocol. Thus, it is crucial that miners are Segwit compatible and can read the Segwit transactions to ensure that only a respective owner spends a Segwit UTXO.[40]

4.4.5 Indirect Block Size Increase

Segwit transactions need less space in a block compared to conventional transactions since the signatures are included in the witness field and hence, are not

[40] The anyone-can-spend outputs were a side effect of Segwit's implementation as a soft fork.

considered for the one megabyte block size limit. Consequently, a block can include more Segwit transactions than conventional ones.

The one megabyte block size limit is changed to a new block cost limit rule including a block weight. It is defined as

$$\text{Block Weight} = \text{Base Size} * 3 + \text{Total Size} \leq 4'000'000 \text{ bytes} \tag{1}$$

where *base size* is the block size in bytes without any witness related data under the original serialization and *total size* is the block size in bytes including base and witness data under the Segwit transaction serialization illustrated in figure 12. In other words, witness data is counted as one unit of cost and the transaction without the signatures as four units of cost. Practically speaking, the limit for the base size is still one megabyte, the limit for the witness part is three megabytes.

Protocol	Tx Size (bytes)	Witness Size (bytes)	Number of Tx	Base Size (bytes)	Total Size (bytes)	Block Weight	Limit (bytes)	Rate (tx/s)
Standard	500	0	2’000	-	1’000’000	-	1’000’000	3.3
Segwit	250	250	3’200	2’400’000	1’600’000	4’000’000	4’000’000	5.3

Table 1: Transaction capacity comparison between a standard transaction before Segwit and a transaction using Segwit.

To see the effect on the transaction capacity, we illustrate a quick comparison between a conventional transaction and a Segwit transaction in table 1. We assume homogeneous transactions in a block and a transaction size of 500 bytes of which 250 bytes is witness data. Without Segwit, 2’000 transactions can be included in a block which results in a transaction capacity of 3.3 transactions per second. With Segwit, the new limit is a block weight limit of 4’000’000 bytes. A block that consists solely of Segwit transactions can theoretically include 3200 transactions. This results in a $3200 \cdot 250 \cdot 3 = 2'400'000$ bytes base size and a $500 \cdot 3200 = 1'600'000$ bytes total size, which exactly satisfies the block weight limit. The number of transactions per block has therefore increased by 60% and the transaction capacity to 5.3 transactions per second. Depending on the size, type and the fraction of witness data, the number of transactions that can be included in a block may increase between 60% to 100% compared to a block including only conventional transactions.[13] For instance, the block weight of a block including only standard single signature P2PKH transactions is about 1.6 megabytes. If only 2-of-2 multisig transactions are included, the block weight is about two megabytes. In both cases the hard-coded block limit is not exceeded.[109]

Nodes that did not upgrade their software will always see blocks that are within the one megabyte limit since they do not know about the witness data. In our example, the block size measured by an old node would be $250 \cdot 3200 = 750'000$ bytes which well fits in the one megabyte limit.

4.4.6 Other Upgrades Implemented with the Segwit Soft Fork

Since Segwit implied large modifications to the Bitcoin source code, many other upgrades were integrated in the Segwit soft fork as well. For example, BIP143 defines a new transaction digest algorithm for signature verification under Segwit. Originally, the time used to verify the signatures in a block grew quadratically ($O(n^2)$) in the number of signature operations (*sigops*). Doubling the size of a transaction doubled the number of signature operations and the amount of data to be hashed to verify the signatures. A malicious user could exploit this and create a transaction that takes over three minutes to verify.[13] The new transaction digest algorithm allows for linear signature verification ($O(n)$) because each byte of a transaction is hashed at most twice. It is, however, only applicable in witness programs with a version byte taking value 0.[72]

Furthermore, the Segwit related BIP173 was introduced in *Bitcoin Core* version 0.16.0. in February 2018. It defines a new address format for native witness outputs. A Bitcoin address is originally encoded in base58check format which has several shortcomings, e.g. it needs a lot of space in QR codes, decoding it is relatively slow and complicated and the double SHA256 checksum is slow and has no error detection mechanism. With BIP173, *Bech32* is introduced which is a checksummed base32 format.[41] A Bech32 string is maximally 90 characters long. It consists of (i) a human-readable part which conveys everything of importance for the reader, (ii) a separator which is always 1 and (iii) a data part. The data part only consists of alphanumeric characters, excluding *1*, *b*, *i* and *o*. The last six characters form a checksum. The new address format is only compatible with updated software.[130]

4.5 Road to Segwit Activation

Because the path to Segwit's activation was very complicated, we give an overview of the different steps below. As mentioned in section 2, the BIP141 proposal describing Segwit was formalized in December 2015. BIP141 was released under the BIP9 soft fork implementation method. Between November 15, 2016 and November 15, 2017, miners were able to support Segwit through signalling bit 1 in the *nVersion* field of the block header. The one year timespan contained 26 periods consisting of 2016 blocks each. If in any of these periods, at least 95% of the blocks mined (i.e. 1916 blocks) or in other words 95% of the hashing power signalled bit 1, Segwit would have become `LOCKED_IN` and activated 2016 blocks later. Every node running at least *Bitcoin Core* 0.13.1. would have been able to run Segwit. If the threshold had not been reached in any of these 26 periods, BIP141 would have obtained the status `FAILED`.

Because no real progress was made in the activation of Segwit, the *User Activated Soft Fork* (UASF) approach was proposed (BIP148) in March 2017. From August 01, 2017 on, all blocks that did not signal bit 1 would have been rejected by all nodes running BIP148. Nodes that did not run BIP148 would have accepted both blocks mined under the new and under the old rules. Hence, if several nodes had run BIP148 and a miner had produced a block not signalling bit 1 a chain split would have been triggered. Consequently, in the chain where BIP148 would have been applied by the users, 100% of the blocks had signalled readiness for Segwit whereby the 95% threshold would have been reached and Segwit would

[41] Bech32 is also used for a QR-ready protocol for requesting payments over the Lightning network.[30]

have locked in. In other words, if by August 01, 2017, Segwit had not locked in or been activated yet, any node running the BIP148 code would have started rejecting blocks that did not signal bit 1. Because the new rules would have been enforced by the users and not by the miners, it was called a *user activated* soft fork.

Also in March 2017, Sergio Demian Lerner proposed Segwit2Mb.[74] If and only if Segwit was activated, a block size increase to two megabytes should have followed. He stated that the single aim of this proposal was to reunite the Bitcoin community. The proposal was further known as Segwit2x. Lerner suggested a signalling period of four months from April 29 to August 29, 2017. Miners could signal either for deploying Segwit and two megabytes blocks together (signal bit 2) or for deploying Segwit only (signal bit 1). If at least 95% of the hashing power in the network had signalled support for the combined deployment and the Segwit soft fork had been activated, the two megabytes hard fork would have occurred on December 14, 2017.

Jeff Garzik, who already wanted to raise the block size to two megabytes in BIP102, was the lead developer of Segwit2x. A software implementation was developed on the Github repository *btc1*, which differed from the implementation suggested by Demian Lerner. The mechanism to reach 100% support for Segwit was adopted from BIP91 (Segsignal). First, miner publicly stated support by including NYA (New York Agreement) in the coinbase. The actual signalling period was planned to take place between July 21 and August 01, 2017. At least 80% of the blocks had to signal bit 4 in a 336 blocks period (i.e. $\sim$ 2.3 days). Hence, there were five periods in which Segwit could lock in. If successful, blocks not signalling support for Segwit in bit 1 had been orphaned after a 336 blocks grace period. In the Segwit2x case, 90 days (144*90 blocks) after Segwit had activated a hard fork raising the block size to two megabytes would have occurred. Since *Bitcoin Core* publicly opposed a block size increase to two megabytes, all nodes that wanted to support Segwit2x had to switch to an alternative client within these 90 days. Probably due to the upcoming UASF on August 01, 2017, miners already started to signal bit 4 before July 21, 2017. As a consequence, bit

	Segwit2x	**BIP91**	**BIP 148**
Pre-signalling period	> 80% signal bit 4	> 80% signal bit 4	-
Signalling period	> 95% signal bit 1	> 95% signal bit 1	> 95% signal bit 1 Start August 01
Activation Process	• Nodes update to btc1 • > 80% signal bit 4 • BIP91 activates • Reject non-bit 1 blocks	• Nodes update to BIP91 • > 80% signal bit 4 • BIP91 activates • Reject non-bit 1 blocks	• Nodes update to BIP148 • On August 01 BIP148 activates • Reject non-bit 1 blocks
Outcome	• 100% of blocks signal bit 1 • Segwit locks in when 95% reached • Segwit activates after 2016 blocks grace period		
Secondary Purpose	Hard fork 90 days later	None	None

Table 2: Overview of three Segwit activation mechanisms. Namely, Segwit2x, BIP91 Segsignal and BIP148 UASF.[64]

4 already locked in on July 20, 2017 at block height 476768. Up to 90% of all hash power in the network signalled support.[64] The activation of bit 4 made the UASF irrelevant, since miners started to signal bit 1 after the 336 grace period. On August 08, 2017, the necessary threshold required in BIP141 was reached and Segwit locked in. For another 2016 blocks grace period, users had time to upgrade their software. Segwit was finally activated on August 24, 2017.

Furthermore, with bit 4 locking in, Segsignal (BIP91) itself became relevant again. Since Segsignal was implemented in the Segwit2x code, it was not possible to differentiate, whether miners signalled support for Segwit2x, including Segsignal, or only for Segsignal. In other words, it was not clear, whether miners are in favour for Segwit with a block size increase or only for Segwit. On August 16, 2017, the developers of Segwit2x announced that the two megabytes hard fork should take place in mid-November at block height 494'784. However, on November 08, 2017, they stated in a mail to the Segwit2x mailing list that the plans to upgrade to Segwit2x are suspended due to lack of consensus in the Bitcoin community.[8]

More probably, however, is that the suspension of the project was caused by several bugs in the code.[100]

The usage of Segwit transactions steadily increased over time. In July 2018, around 35 to 40% of all transactions were Segwit transactions.[84]

4.6 Discussion

Similar to a block size increase, either Segwit itself or the fact that is was implemented as a soft fork, which for example caused the anyone-can-spend outputs, was heavily debated in the network. We present some arguments that show the benefits of Segwit and others that emphasize the shortcomings.

Without Segwit, signatures accounted for about 50% of the blockchain's storage requirements.[129] However, they are only used at the time of validation and afterwards put unnecessary load on the chain. With Segwit, signatures are omitted whenever possible to reduce the burden on the system. For most nodes there is no need to store the entire history of signatures. Also the transmission of the witness data could be made optional depending on the kind of node a signature is sent to. For example, SPV clients normally download the signatures, even though they are not validating them. Not only do these examples imply a significant cost reduction for storage requirements but also for bandwidth.

With Bitcoin becoming more popular, the unspent transaction output (UTXO) set, which is stored by each full node, continues to grow. With Segwit, a user can validate the transaction when it is received and then store it in the UTXO set without the signatures. This should slow down the increasing storage requirements of the UTXO set.

Furthermore, excluding witness data from the base block size allows for an indirect increase of the block size limit. This results in an effective new limit between 1.6 and 2 megabytes. However, even though not common in practice, Segwit theoretically enables blocks of almost four megabytes in size.[42] Miners and full nodes

[42] For instance, a block that only contains P2WPKH transactions with only 1 input and 1

have to ensure that they can handle these four megabytes blocks. Otherwise, a malicious attacker may disrupt parts of the network.

As aforementioned, all forms of non-intentional malleability become impossible. We have to emphasize, however, that malleability can only be prevented if all inputs of a transaction are Segwit UTXOs. By solving transaction malleability, Segwit allows the creation of unconfirmed transaction dependency chains without counterparty risk. This is critical for second layer protocols such as the Lightning network (see section 6 for a detailed explanation). However, due to the implementation as a soft fork, Segwit and non-Segwit transactions are co-existing in the network which is why all improvements are only applicable to the subset of Segwit transactions. For example, transaction malleability is only prevented for Segwit UTXOs. If Segwit had been implemented as a hard fork, the whole network would have benefited from all improvements.

Something that is often criticised are the anyone-can-spend outputs. Nodes that did not upgrade their software are not able to interpret the meaning of the scriptPubKey in a Segwit transaction and read it as an anyone-can-spend output. Furthermore, old software is not aware of the new witness field and cannot validate it. Theoretically, anyone in the network could spend these anyone-can-spend outputs without providing a corresponding signature. Old software would always consider such a transaction as valid. This is only prevented by the miners who validate the witness data and ensure that the users' actions are in line with the rules. According to Rizun (2017), situations may emerge where miners are incentivised to deviate which would result in security issues.[91]

Another problem that anyone-can-spend outputs entail are zero confirmation (0-conf) double spends. Imagine Emily buys a coffee in Jack's coffee house and pays with Bitcoin units. Emily, as a tech affine girl, has already upgraded her wallet to the new software that supports Segwit transactions. Jack, however, is still running the old software that does not support Segwit transactions. Emily realizes this and plans to exploit the situation. She buys a coffee and pays Jack

output would have a very small base size.

with a Segwit transaction but does not include a signature. Jack will read the output as an anyone-can-spend output and will thus validate the transaction. Emily takes her coffee and propagates another transaction to the network where she references the same UTXO but sends it back to her address. Jack's node will ignore the second transaction since it re-spends the UTXO. However, a miner running a Segwit-compatible software will include the second transaction in a block because the first one is obviously invalid. This scenario is only possible because Jack has not upgraded his software. Thus, all users actively using the Bitcoin network are encouraged to update their software.[26]

The problem above could also concern miners if they do not properly upgrade to Segwit. A miner might accept an under the new rules invalid transaction, like the 0-conf transaction above, and include it in a block. Other miners who did not upgrade to the new rules as well or did not perform a proper validation might continue mining on top of the invalid block. However, miners who create blocks under the new rules will ignore these invalid blocks whereby a fork results in the network. As long as non-Segwit compatible miners do not represent the majority of hash power in the network, the fork will resolve. For the erroneous miner, however, this is tantamount to a waste of resources.

Lastly, the implementation of Segwit is accompanied by some other improvements. BIP143, for instance, replaces quadratic scaling in signature operations with linear scaling. Large transactions and blocks can be generated without potential problems due to signature hashing. Furthermore, the version byte pushed before a witness program enables new script systems. For example, Merkelized abstract syntax trees (MAST), Schnorr signatures (see section 7) or Lamport signatures which enable quantum computing resistance could be implemented through a soft fork using the version byte.

5 Payment Channels

Payment channels allow two people to send transactions to each other with minimal impact on the Bitcoin blockchain. They can transmit transactions off-chain and only add the final balances to the blockchain. In other words, payment channels are Bitcoin balances between two users, where the rest of the network does not care about their transactions until the final balances are added to the blockchain. If the balance of one user increases, the balance of the other user will decrease by the same amount. Payment channel transactions are both cheaper and faster than on-chain transactions. Cheaper, because no transaction fees have to be paid and the network is not burdened with all transaction data. Faster, because they do not require blockchain confirmation.[120]

Payment channels are especially beneficial for transactions that transfer only a small US-Dollar amount of Bitcoin units. A payment worth less than 25 US-Dollars can be called a micropayment.[77] If block space is scarce, it is rather unlikely that micropayments are included in a block due to lower transaction fees compared to transactions processing Bitcoin units worth several thousands of US-Dollars. With payment channels, a user can send as many transactions as desired without having to pay high fees.

Therefore, the main use case for payment channels are micropayments and not transactions sending large amounts. A payment channel always requires the deposit of the amount that is going to be transferred, which is why it is more efficient to just include a transaction sending large amounts of Bitcoin units directly in the blockchain.

5.1 Unidirectional Payment Channels

Unidirectional payment channels were first implemented in the BitcoinJ client in 2013 and became known as Spillman channels.[18] Unidirectional channels only allow for sending transactions in one direction, from A to B but not back. This limits their applicability considerably.

To make our description more intuitive and not too technical we give an example of an unidirectional payment channel between Emily and Jack, illustrated in figure 16. Jack is an art dealer and repeatedly sells expensive paintings to Emily.[43] To ease the payment process, Emily opens up an unidirectional channel. The first step is to create a 2-of-2 multisignature address. Next, Emily sets up a funding transaction where she deposits 10 BTC in the address. However, before broadcasting the funding transaction she creates a refund transaction. This is relevant for the case that Jack is unresponsive and does not sign any transaction spending from the multisig, whereby Emily's funds would be locked. The refund transaction is time locked, references the funding transaction and sends back the funds to Emily at a future point in time, defined as block height in the *nTimeLock* field. Emily sends the refund transaction to Jack who signs it and sends it back to her.[44] She can now broadcast the funding and the refund transaction whereby the channel is opened. In case Jack does not sign the refund transaction the channel creation fails and Emily can use the funds for another purpose.

If the set up has been successful, Emily can start to use the deposit to send payments over the channel until either the deposit is exhausted or the time lock of the refund transaction is reached and the channel closed. Emily buys a first painting from Jack and pays him 5 BTC. She creates a transaction using the multisig as input and includes two outputs, one where she sends 5 BTC to Jack and the other one where she sends 5 BTC to herself. Jack does not sign and broadcast this transaction. Thus, it is not added to the blockchain yet. At some later point in time, Emily purchases a second painting which costs 2 BTC. In total she has spent 7 BTC. She updates the payment channel by creating a new transaction where she sends 3 BTC to herself and 7 BTC to Jack, which reflect the current balances in the payment channel. Lastly, Emily buys a third painting

[43] Obviously, this example is contradictory to the remark above that payment channels are mostly meaningful for small payments. However, for the sake of clarity, we decided to show an example with few payments and integer Bitcoin amounts. The basic mechanism is equivalent for smaller and more frequent payments. A more accurate real life example would be a music streaming service where you pay a small amount of Bitcoin units for each song you listen to.

[44] It is crucial that Jack does only see the hash of the funding transaction - which is referenced as input of the refund transaction - but has no knowledge about other parts of the transaction. Otherwise, he could exploit transaction malleability to harm Emily through breaking the link between the funding and the refund transaction whereby Emily's fund would be locked.

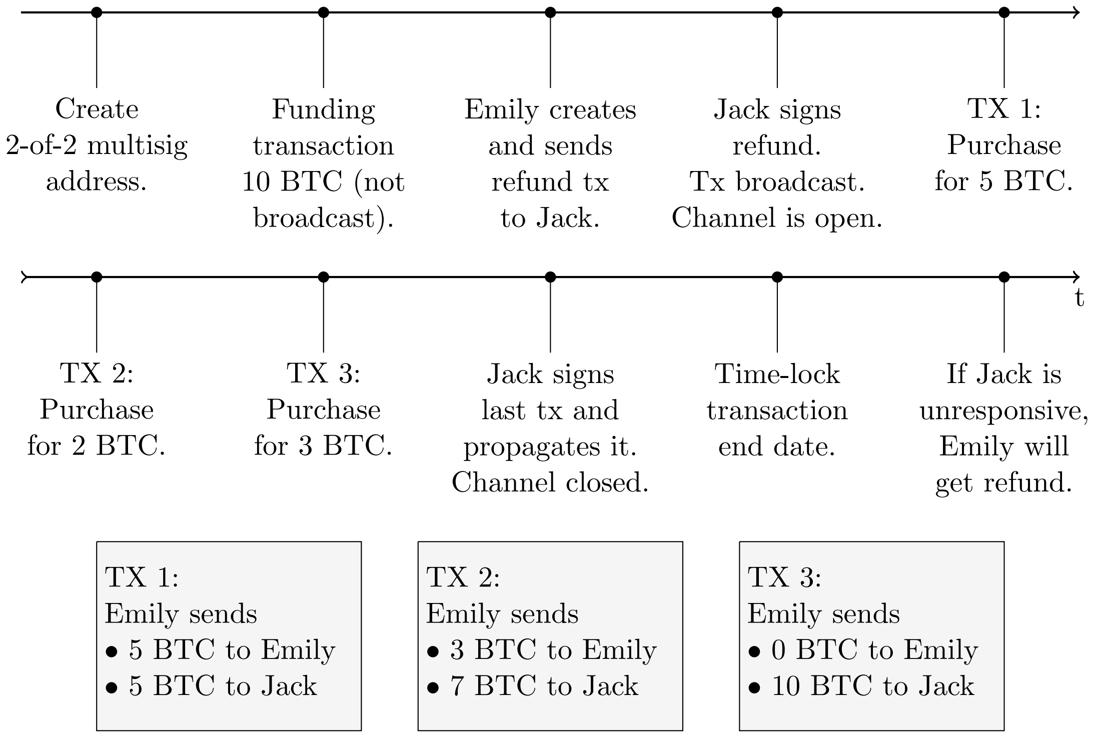

Create 2-of-2 multisig address.
Funding transaction 10 BTC (not broadcast).
Emily creates and sends refund tx to Jack.
Jack signs refund. Tx broadcast. Channel is open.
TX 1: Purchase for 5 BTC.
t
TX 2: Purchase for 2 BTC.
TX 3: Purchase for 3 BTC.
Jack signs last tx and propagates it. Channel closed.
Time-lock transaction end date.
If Jack is unresponsive, Emily will get refund.
TX 1: Emily sends • 5 BTC to Emily • 5 BTC to Jack
TX 2: Emily sends • 3 BTC to Emily • 7 BTC to Jack
TX 3: Emily sends • 0 BTC to Emily • 10 BTC to Jack

Figure 16: Timeline of an unidirectional payment channel between Emily and Jack.

and has to pay another 3 BTC to Jack. In total, she has now purchased paintings worth 10 BTC. She updates the payment channel a last time and sends 10 BTC to Jack but nothing to herself. Jack can now sign the last transaction and propagate it to the network, whereby it is added to the blockchain.

Only one of these transactions can be added to the blockchain, since Emily always references the same input. If Jack acts rationally, he will always sign the last transaction in which he receives the highest amount of Bitcoin units. If Jack is unresponsive and does not sign any transaction, Emily will get the deposit back through the refund transaction.

As most conventional payment channels, also unidirectional channels are prone to transaction malleability. If Jack receives the funding transaction Emily broadcasts quickly, he could alter Emily's signature and propagate a modified transaction with a different transaction ID to the network. If Jack's transaction is included into the blockchain before Emily's, the link between the funding transaction and the refund transaction is broken. Due to the different transaction ID of the funding transaction, the refund transaction does not correctly reference the funding transaction anymore. As a consequence, Emily could not use the refund transaction to get her money back and her funds would be locked in the 2-of-2 multisig address.[48]

This was improved by the implementation of BIP65 at the end of 2015 which introduced CheckLockTimeVerify (CLTV).[107] CLTV allows to include an if/else condition into the scriptPubKey.

5.2 Basic Bidirectional Payment Channels

In contrast to an unidirectional payment channel, a bidirectional channel allows to send Bitcoin units in both directions. There are several different proposals of bidirectional channels. We first describe a basic mechanism how a bidirectional channel works and then present some concrete examples.

5.2.1 Basic Mechanism

Imagine Emily and Jack got to know each other better and rented an apartment together. They are afraid that one of the two might do much more of the house work and therefore decide to pay each other whenever someone did some work at home. Both agree to pay each other in Bitcoin units. Since there are many transactions going to be sent between the two, it would be inefficient to add every single transaction to the blockchain, which is why they decide to create a bidirectional payment channel.

The procedure to set up the channel is the same as above. Figure 17 illustrates a basic example. Emily creates a 2-of-2 multisig address, where she sends a funding transaction to and creates a refund transaction that Jack signs. We define the absolute lock time of the refund transaction to be 20 days. Again, Emily deposits 10 BTC.[45]

The channel is now set up and they begin with the house work. Jack washes the

[45] Emily and Jack can also fund the channel together. For understanding the basic mechanism, this is a negligible detail, though.

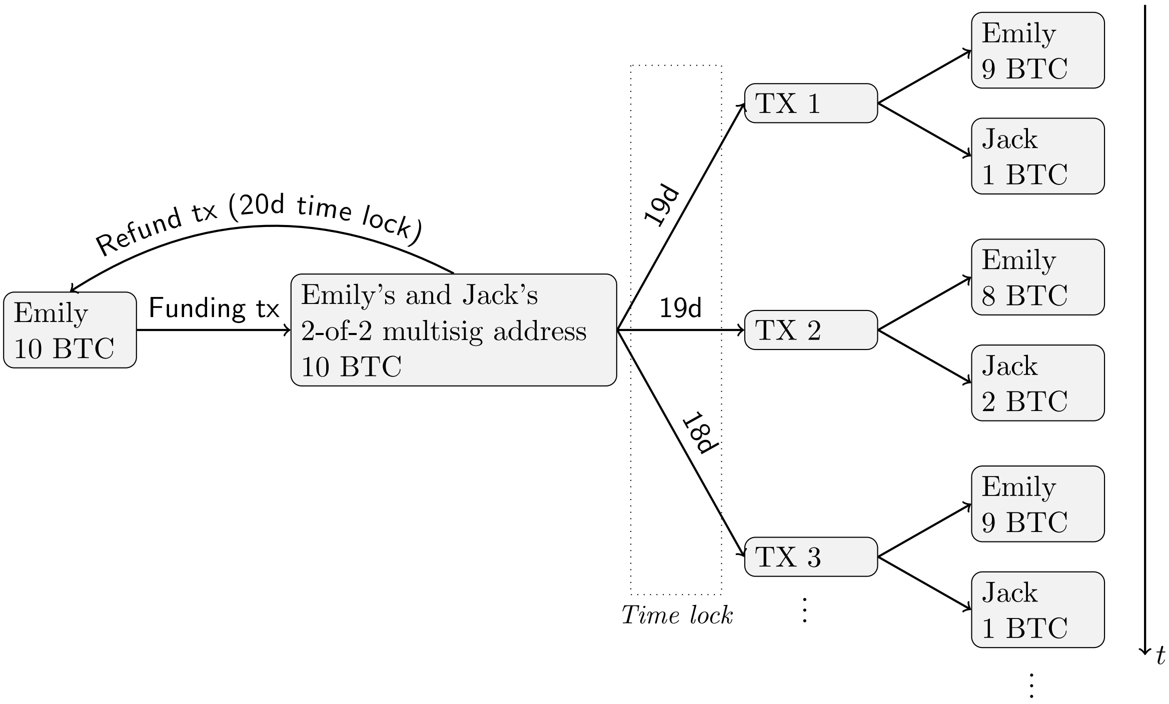


Figure 17: Basic mechanism of a bidirectional payment channel.

dishes after lunch for what Emily sends 1 BTC to him. Emily creates the first transaction in the channel where she sends 9 BTC to herself and 1 BTC to Jack. Additionally, she time locks the transaction through setting the absolute lock time to 19 days. She signs it and sends it to Jack. In the afternoon, Jack vacuum-cleans and gets another Bitcoin. Emily creates another time lock transaction where she sends 8 BTC to herself and 2 BTC to Jack and sets the absolute lock time to 19 days again. In the evening, Emily cooks dinner wherefore she is owed 1 BTC by Jack. He creates a transaction in which he sends 9 BTC to Emily and 1 BTC to himself including a time lock of only 18 days.

The rationale of applying the time locks is the following: the most current state of the channel is described by the third transaction. Thus it is crucial that this state can be included in the blockchain before transactions one and two. With the time locks, the first two transactions can only be included after 19 days, the third one already after 18 days. As soon as the third transaction is included in the blockchain the others become invalid, since they spend the same outputs. Hence, every time a payment is sent in the opposite direction than the payment before, the time lock is reduced. It can be hold equal for payments going in the same direction, since a rationale counterparty will always prefer to sign and broadcast the latest transaction in which the highest amount of Bitcoin units is received, i.e. Jack prefers transaction two over transaction one.

This channel set up does not allow for infinitely open channels. Every switch of direction will bring the absolute lock time closer to the present. Even if the forward shift of the time lock is minimized to an interval that is still secure, only a certain maximal amount of transactions can be processed depending on the initial time lock chosen for the refund transaction.

Emily and Jack can close the channel in two ways. First, the latest state of the channel can be propagated, whereby they receive their channel balances after the time lock period expired. If they are inpatient, they can also cooperatively close it through creating a new transaction without a time lock sending the respective balances to the two parties.

5.2.2 BIP68, 112 and 113

For the development of (bidirectional) payment channels, BIP68, 112 and 113 were of great importance. The implementation took place in *Bitcoin Core* version 0.12.1. in April 2016. Together the proposals allow payment channels to be open as long as the parties want, which should increase the efficiency of bidirectional channels tremendously.[11]

BIP68 introduces relative lock time contrary to the absolute lock time described above. Each Bitcoin transaction has a sequence number field for each input as illustrated in figure 8. BIP68 repurposed this field such that mining an input can be prevented until the referenced output has achieved a certain age in the blockchain. This can be defined in either a number of blocks or a timespan. In other words, with absolute lock time a user can only define a fixed date in the future when the transaction can be included in the blockchain whereas with BIP68 the validity of a transaction depends on a specific output's age in the chain. The bits of the sequence number are set according to some prerequisites such that a user can signal whether a corresponding transaction can be included in any block or whether it is locked by a relative lock time.[36]

BIP112 replaces the NOP3 opcode[46] with `OP_CHECKSEQUENCEVERIFY` (CSV). CSV checks whether the age of a referenced output is reached. It will either validate or fail the transaction. If it fails, the transaction cannot be included in a block.[37]

Lastly, there is BIP113. The consensus rules in Bitcoin do not mandate strict ordering of block timestamps. For example, block B which is mined after block A can still have an earlier timestamp than block A (see section 3.1). This may become problematic for the relative lock time implementation. A transaction that includes an input with a specific time lock would be valid under the timestamp in block A, but would be invalid under the timestamp in block B, even though B is mined after A. Furthermore, miners could be incentivised to manipulate the timestamp in a block such that they can include transactions which by network time have not yet matured. With BIP113 the relative lock time of an input is compared against the median timestamp of the previous eleven blocks and not against the timestamp of the respective block. The consensus rules guarantee this value to monotonically increase.[68]

5.2.3 Bilateral Channels with Relative Lock Time

The relative lock time can be applied on the above described bidirectional channel. In this example, Emily and Jack both fund the 2-of-2 multisig address with 5 BTC each as depicted in figure 18. Before starting to send transactions to each other they create an additional *kick-off* transaction that sends the balances to a multisig. It reflects the opening of the channel but is not broadcast to the blockchain. The kick-off transaction's output is referenced as input of the microtransactions[47]. The microtransactions are only valid, if the kick-off transaction is included in the blockchain and is confirmed by a pre-defined number of other blocks, i.e. it has achieved the age defined in the relative lock time.

In our example, Emily owes Jack 2 BTC and creates the first microtransaction,

[46] Opcodes are *"operation codes from the Bitcoin Script language which push data or perform functions within a pubkey script or signature script"*.[16]

[47] We define microtransactions as transactions sent in the payment channel.

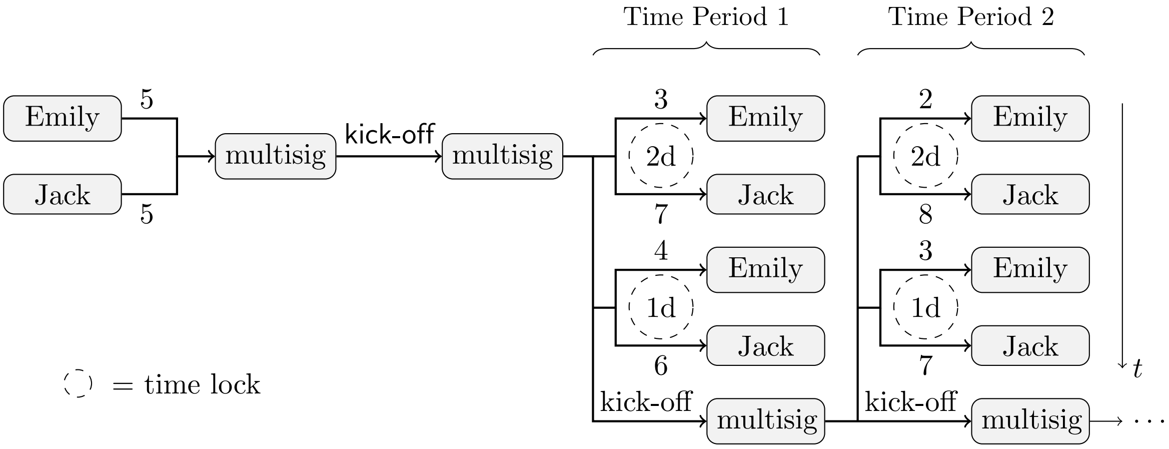


Figure 18: Bidirectional payment channel using relative lock time.

where she sends 3 BTC to herself and 7 BTC to Jack. The transaction is locked by a relative lock time of two days, i.e. both can only claim the funds if they propagate the kick-off transaction and then wait for another two days. Next, Jack wants to send Emily 1 BTC and creates another microtransaction where he sends 6 BTC to himself and 4 BTC to her. The direction of the channel switches and the time lock is preponed by one day. For now they cannot switch the channel direction again since the relative lock time would be reduced to zero days. Thus, they create another kick-off transaction spending the current balances and referencing the output of the previous kick-off transaction. Both sign but do not broadcast it to the network. The channel is now open for another two days. In time period 2, Emily owes Jack 2 additional BTC which is why she creates a microtransaction that sends 2 BTC to Emily and 8 BTC to Jack. She references the new kick-off transaction which serves as the new reference point for the relative lock time. Whenever it is not possible anymore to create additional microtransactions in a specific time period, Emily and Jack create a new kick-off transaction and can thereby leave the channel open as long as they want.

Since Emily receives more Bitcoin units in the first time period, she might try to cheat on Jack and propagate the kick-off transaction corresponding to time period 1. However, she has to wait for one day until she can claim the 4 BTC. In the meantime, Jack can propagate the subsequent kick-off transaction and thus invalidates the microtransactions of the first time period.

The channel can be closed either unilaterally or bilaterally. Emily and Jack can agree on a closing transaction that spends the funds immediately back to them. If one party intends to close the channel unilaterally, he or she broadcasts the kick-off transactions and the latest microtransaction of the respective time period and will receive the funds after the relative lock time period.[104]

5.3 Poon-Dryja Payment Channels

Poon-Dryja payment channels were presented by Joseph Poon and Thaddeus Dryja in their paper about the Lightning network.[88] We illustrate the creation of such a channel in figure 19. As before, Emily and Jack agree on a funding transaction they do not sign yet. Emily deposits 4 BTC and Jack 6 BTC in a 2-of-2 multisig address. Next, they both create their first commitment transactions.

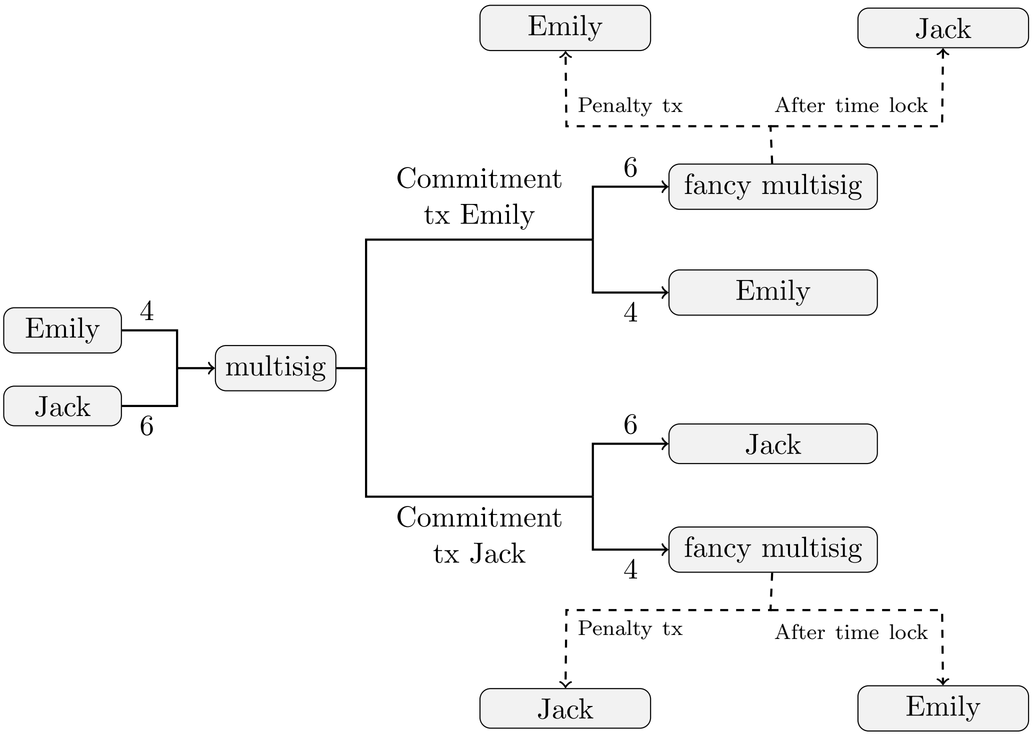


Figure 19: Creation of a Poon-Dryja payment channel.

Emily deposited 4 BTC in the channel and thus includes an output that sends 4 BTC to herself and another output that sends 6 BTC (Jack's deposit) to a multisig address that we call a fancy multisig address. The funds in the fancy multisig can be claimed through a penalty transaction we discuss below or by Jack after a CSV-time lock period (e.g. 1000 blocks), which means that the child transaction (i.e. the payout to Jack) is only valid when the parent transaction (i.e. Emily's commitment transaction) has reached a specific age in the blockchain. Jack's commitment transaction is the same but mirrored. He sends 6 BTC to himself and 4 BTC to the fancy multisig address. Emily can unlock the funds in the fancy multisig as soon as Jack's commitment transaction reached the blockchain age defined by the CSV-time lock. Both sign their own commitment transaction, send it to each other but do not broadcast it. Lastly, they both sign the funding transaction and propagate it to the network. After the funding transactions are included in the blockchain and secured by a certain number of other blocks the channel is officially opened.

If Jack signs and broadcasts Emily's commitment transaction, she immediately receives 4 BTC whereas Jack has to wait for another 1000 blocks until he could claim the 6 BTC from the fancy multisig address. Analogously, Emily could sign and broadcast Jack's commitment transaction.

If Jack wants to send 1 BTC to Emily they have to update the channel. New commitment transactions are created, where both send 5 BTC to the fancy multisig and 5 BTC to themselves. Jack, however, has an incentive to broadcast the old commitment transaction where he receives 6 BTC. To prevent this, a penalty transaction is created right before the new commitment transactions are shared with each other. Emily adds a penalty transaction, that sends the funds from her old commitment transaction's fancy multisig to herself and Jack adds a penalty transaction that sends the funds from his old commitment transaction's fancy multisig to himself. First, they both share the signed penalty transactions with each other and only afterwards the new commitment transactions.

Assuming Jack broadcasts Emily's old commitment transaction within the new set up, Emily receives 4 BTC immediately while Jack had to wait for the CSV-time lock period to end until he receives the Bitcoin units. Emily sees the old commitment transaction in the blockchain and can claim the funds in the fancy multisig output before Jack can by broadcasting the penalty transaction. She receives all the funds in the channel as a punishment for Jack since he broadcast an old channel state. If, on the other hand, Emily signed Jack's old commitment transaction and broadcast it, Jack could claim all funds in the channel by broadcasting the penalty transaction. Consequently, it is suboptimal for a rational party to deviate from the consensus protocol. [123]

The channel can be kept open as long as there are enough funds available. Closing it is possible unilaterally or bilaterally. If Jack and Emily cooperate on closing the channel, they create a closing transaction which both sign and then broadcast it to the network. In this case it is indistinguishable from a regular 2-of-2 on-chain multisig address spend. If for example Jack wants to close the channel unilaterally, he can sign Emily's latest commitment transaction. Emily will receive her balance immediately and Jack after the time lock period. In contrast to broadcasting an old commitment transaction, Emily is not able to claim all the funds in the channel for herself since they have not created the penalty transactions yet.

5.4 Duplex Payment Channels

Lastly, we present Duplex payment channels developed by Christian Decker and Roger Wattenhofer in 2015.[46]

Duplex channels are established through creating pairs of unidirectional micropayment channels, one for each direction. Additionally, time locks are applied to make previous states invalid similar as illustrated in figure 18. Normally, there is a trade-off between a large time lock such that the channel can be kept open longer and a small one to reduce the risk that the funds are locked in the channel for a long period of time. Duplex payment channels propose the use of an *invalidation tree* to hold the single time locks small but still allow for long opened channels. The top of the tree is represented by the channel set up, the nodes are multisig outputs and the leafs a pair of unidirectional micropayment channels.

We give a short example of Emily and Jack who use a Duplex payment channel. They create two funding transactions where both agree to send 5 BTC each to a 2-of-2 multisig. Before signing them, they create valid refund transactions and send them to each other. As soon as the set up is completed, the first branch or subtree in the invalidation tree is created by sending a *reset request* and a *reset response* message to each other. This establishes a pair of unidirectional micropayment channels. One going from Jack to Emily that includes 5 BTC Jack can spend and the other one going from Emily to Jack that includes 5 BTC Emily can spend. The subtree has an implicit time lock of ten days.[48] Both can now start sending transactions to each other in the respective channels as described in section 5.1. They use the existing channels until no funds are left in one of the two channels. For example, Jack has transferred 5 BTC to Emily and Emily 3 BTC to Jack. In total, Jack's balance in the channel now adds up to 3 BTC and Emily's to 7 BTC. To further use the Duplex channel, Jack sends a *reset request* message to Emily. Emily stops performing updates to her unidirectional channel and sends a *reset response* message back. They create a new subtree and transmit the balances from the last state of the first branch. The new subtree has an implicit time lock of nine days. Thus, it can be broadcast before the first subtree, whereby the first branch is successfully invalidated.

This process can be repeated until the channel reaches its lifetime, i.e. it is not possible to further prepone the time lock to invalidate another subtree. In this case, a channel can be cooperatively extended by a refresh process. *Refresh request* and *refresh response* messages are sent between the two parties. This transmits the funds to new channels, creates a new invalidation tree and invalidates the old invalidation tree by making use of an atomic multiparty opt-in transaction that is committed to the blockchain. The opt-in transaction spends the latest multisig output of the old tree and creates a new multisig output, called root output. All transactions in the new invalidation tree are only valid if the opt-in transaction is valid, i.e. all parties have signed it.

Closing the channel works similar as in the other bidirectional channels. Both parties can agree on a *teardown transaction* that can be broadcast immediately and sends the latest channel balances to them. If a party closes the channel unilaterally, the latest branch including all transactions of this branch is broadcast

[48] The time lock can be arbitrarily chosen and does not have to be ten days.

and committed to the blockchain. Consequently, the blockchain is burdened much more in case of an uncooperative closing.

Decker and Wattenhofer also proposed to implement Duplex micropayment channels in an off-chain payment network. However, this has not been established so far.

5.5 Discussion

Although payment channels themselves are not very controversial and the advantages seem to outweigh the disadvantages, there are still some shortcomings.

For example, if a user maintains many open channels, a lot of funds are locked which cannot be used for other purposes. Furthermore, a node has to create a new channel with each user it wants to transfer payments.

On the other hand, there are many advantages of payment channels. Firstly, they are cheap, since transaction fees have to be paid only for the opening and closing transactions. Secondly, they are fast. For an on-chain transaction to be confirmed, one has to wait until it is included in the blockchain.[49] In a payment channel, there is instant confirmation, or to be more precise, transaction confirmation is as fast as internet speed allows. Thirdly, payment channels enable to send as many transactions as internet bandwidth capacity allows. And lastly, privacy is enhanced. Since transactions are processed off-chain, they are not publicly visible. Only the final balances can be seen in the blockchain.

6 Lightning Network

In the following section, we present the Lightning network which consists of many different bidirectional payment channels. Additionally, Hashed Time Locked Contracts (HTLC) allow to route a payment through the network from node A to

[49] To increase security, a user preferably waits until two or three additional blocks are mined on top.

B without the requirement that the two nodes maintain a payment channel with each other.

The Bitcoin blockchain is a so called gossip protocol, in which all transactions are broadcast to all participants. The necessity that every participant is informed about each transaction in the network is, however, questionable. The tremendous amount of required resources to process all transactions on-chain might severely limit the ability of Bitcoin to become a prevalent method of payment. The Lightning network aims to scale Bitcoin in processing many transactions off-chain and thus to reduce the burden on the blockchain.[88]

There are several teams working on implementations of the Lightning network (see also section 2). Our explanation is mostly based on the Basis of Lightning Technology (BOLT)[31] consensus, but sometimes simplified.

6.1 Hashed Time Lock Contract (HTLC)

Before we discuss the Lightning network, we describe a further important building block of payment channel networks called hashed time lock contracts (HTLC). HTLCs were initially intended to enable atomic swaps[50] between different cryptocurrencies or blockchains,[82] but they can also be used to route a payment through a payment channel network.

Imagine there is Jack who wants to send 1 BTC to Emily. They do not have a direct payment channel between each other but they both maintain a channel to Bob, as illustrated in figure 20. Using a HTLC allows that Jack can send some Bitcoin units to Emily via Bob. First, Emily creates a secret string of numbers R, called pre-image, calculates the SHA256 hash value $H(R)$ of it and sends it to Jack (step 1). The pre-image R is only known by Emily. Jack then creates a transaction including a 1 BTC payment to Bob and adds the condition that Bob can only claim the funds if he is able to provide the pre-image R, i.e. applying the hash function on R must match the hash value $H(R)$ Jack received

[50] With atomic swaps, nodes can process a transaction between two different blockchains as long as there exists a similar hash function across the chains.

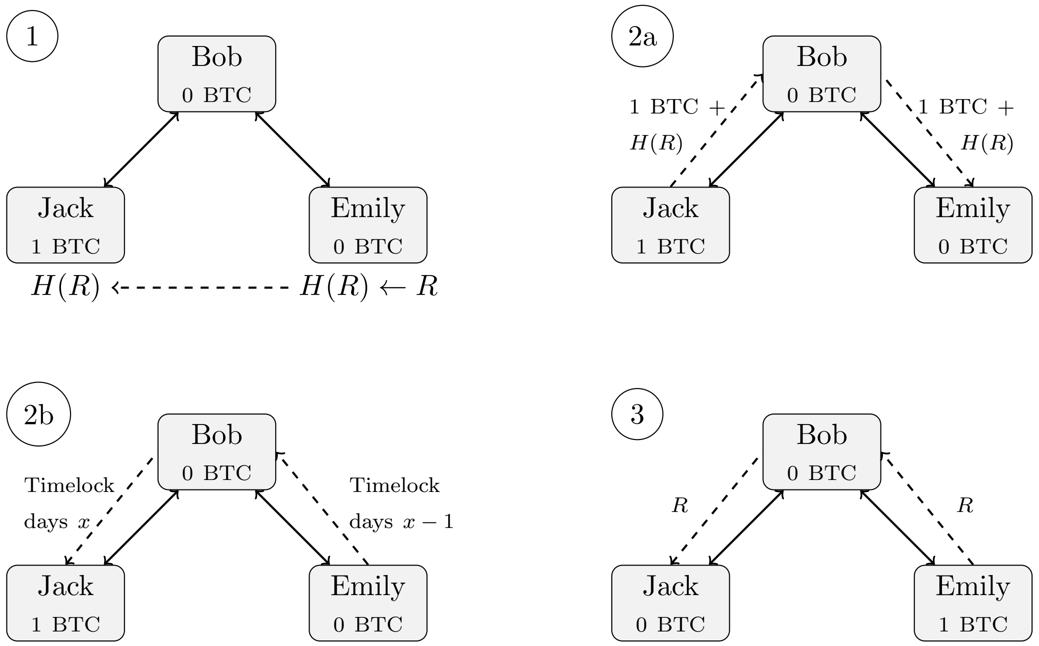


Figure 20: Illustration of Hashed Timelock Contracts.

from Emily (step 2a). Additionally, Jack includes a time locked refund option to the transaction (step 2b). If the Bitcoin units are not claimed by Bob in a certain period of time x, Jack will receive the Bitcoins back. Bob sends a similar transaction with the same payout condition to Emily. Emily knows the pre-image R and can thus finalize the transaction and claim the Bitcoin units from Bob (step 3). In doing so, she discloses the secret whereby Bob can claim the funds from Jack as well and Jack's payment has successfully reached Emily.

The question may arise, why Bob should provide the service of forwarding the Bitcoin units without having an advantage in doing so. In a payment channel network, nodes that forward transactions can be compensated by receiving a small transaction fee that the sender or receiver pay.

In our example, only one hub exists, i.e. there is only one intermediary between Jack and Emily. Theoretically, HTLCs work with an arbitrarily large number of hubs.

6.2 Technical Explanation

The Lightning network combines Poon-Dryja payment channels described in section 5.3 with hash time lock contracts from section 6.1. In our example, Jack wants to send 1 BTC to Emily. Even though he does not maintain a payment channel with her, he can send her a payment via existing channels making use of the Lightning network. In our example, Jack transmits his payment through the open channels between him and Bob, Bob and Carol and lastly Carol and Emily (see figure 22 on page 65).

The single payment channels are based on Poon-Dryja channels with an additional hashed time lock contract (HTLC) output. We examine the channel between Jack and Bob in more detail. If Jack and Bob do not have an open channel yet, they establish it by initializing a connection and subsequently sending an *open_channel* and an *accept_channel* message to each other. They create the funding transactions - 5 BTC each in our example - and both versions of the commitment transactions. Jack then sends the funding transaction's outpoint and the signature for his commitment transaction to Bob via a *funding_created* message. Bob will sign his commitment transaction and the funding transaction and sends it back to Jack via a *funding_signed* message. Jack signs the funding transaction as well and broadcasts it to the network. They wait until the funding transaction is added to the blockchain and has reached a pre-defined number of confirmations (block depth) before the channel is officially opened. They confirm this by sending a *funding_locked* message to each other.[33]

Assume for now that they already processed some transactions between each other and the balances are 4 BTC to Jack and 6 BTC to Bob. As aforementioned, Jack wants to send 1 BTC to Emily over the Lightning network via a HTLC. Hence, he has to set up a HTLC between himself and Bob, too. The respective commitment transactions contain a third, HTLC like output. We illustrate Jack's commitment transaction in figure 21.

The first two outputs are analogous to the Poon-Dryja channel in figure 19. There is a regular output that sends 3 BTC back to Jack himself and reflects the current

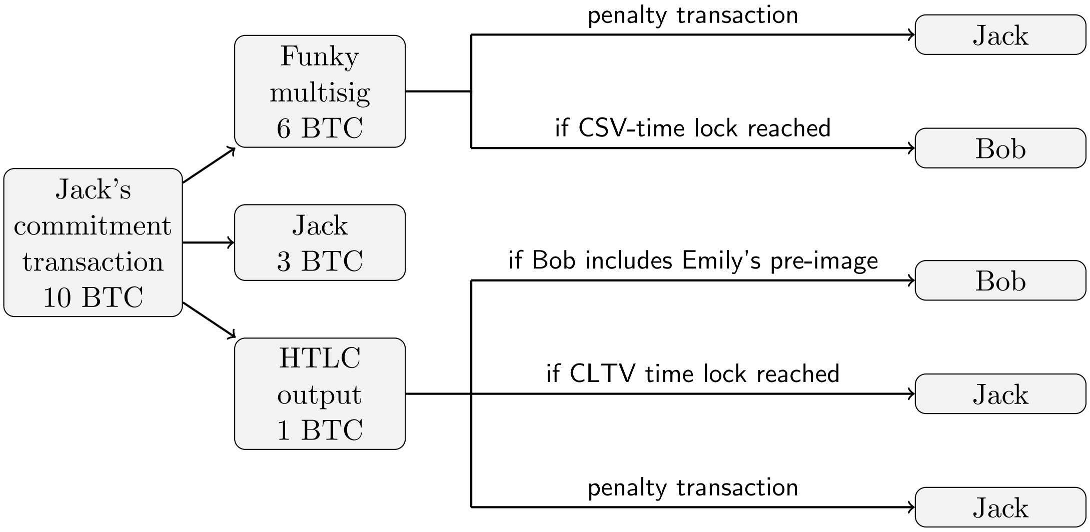


Figure 21: Jack's commitment transaction in his payment channel with Bob in the Lightning network.

balances Jack possesses in the channel minus the HTLC output. Furthermore, 6 BTC are sent to the fancy multisig output which represents Bob's current balance in the channel.[29]

Lastly, there is a new output worth 1 BTC which is essentially the hashed time lock contract. There are three ways how this output can be unlocked. (1) The output can be claimed by Bob if the signatures and the pre-image that Emily produced as described in section 6.1 are provided. (2) There is a regular CLTV time lock. If Bob does not provide the pre-image - maybe because he did not receive it from Emily - Jack gets his funds back at a pre-defined date, e.g. in two weeks. (3) There is a penalty transaction in case a user broadcasts an old channel state to get more Bitcoin units. It is similar to the penalty transaction in the Poon-Dryja channel as illustrated in section 5.3. Before Jack and Bob share their new commitment transactions, a HTLC penalty transaction is created and shared to invalidate the old HTLC states. Therefore, the HTLC output's Bitcoin units can be "stolen" in case the other party tries to broadcast an old state.[124]

Closing a payment channel is initiated by a *shutdown* message sent by either node. It includes a scriptPubKey that states where the funds should be sent to. If the channel is empty of HTLCs and the *shutdown* was successful, they send *closing_singed* messages to each other. These messages include a proposal for the

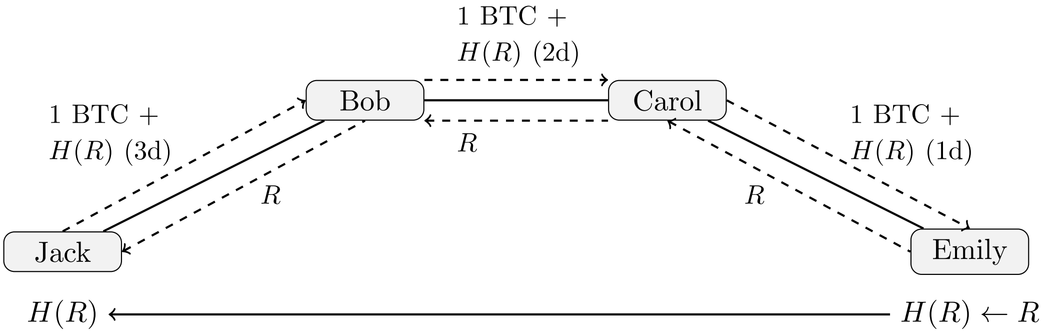


Figure 22: Basic Structure of a subset of the Lightning network. Note: The term in brackets shows the respective time lock in days.

blockchain transaction fee each pays and the signatures needed for the closing transaction. If they agree on a closing transaction, both parties can spend the Bitcoin units immediately. If a party does not agree, they send new *closing_signed* messages to each other until they cooperate or one party fails the channel. It is also possible to close the channel unilaterally. In this case, one party just broadcasts the latest commitment transaction whereby the other party receives the funds immediately and the closing party after the time lock period. [33]

All of this works off-chain. Consensus is achieved through the threat of on-chain enforcement. Transactions only hit the blockchain if a new channel is opened, an existing one closed or a participant is uncooperative.

So far we only looked at the payment channel between Jack and Bob and presented the HTLC within the channel. The HTLC is responsible that a transaction can be transferred over the network without maintaining a direct channel to the receiving party. Below, we discuss how this basically works.

Emily creates a HTLC by sending Jack an *update_add_htlc* message. She produces a random string of numbers R called pre-image and sends its hash value $H(R)$ to Jack via secure off-chain communication. Jack then sends an *update_add_htlc* message to Bob and creates a transaction in which he sends 1 BTC to Bob, including $H(R)$ and a refund transaction with a time lock of three days.[51] Thus, if Bob does not claim the funds from Jack, Jack can reclaim his Bitcoin unit.

[51] Although the time lock should be defined in a number of blocks and not in days, we use a time definition for the sake of simplicity.

Subsequently, Bob creates the same transaction with Carol as receiver, but with a time lock for the refund transaction of two days. Lastly, Carol sends 1 BTC to Emily including a refund transaction with a time lock of one day. All transaction outputs have in common that they can only be claimed by the receivers if they can provide the pre-image R. Emily knows R and provides it to Carol via an *update_fulfill_htlc* message whereby the funds are transferred to Emily. Carol now knows the pre-image R and therefore pulls the Bitcoin unit from Bob. Lastly, Bob does the same with Jack. The transfer of the Bitcoin unit has been successfully completed without hitting the blockchain.[33]

There may be situations, in which a party is uncooperative. For example, Carol might send the funds to Emily but Bob does not send the funds to Carol. Since Carol knows R, she can broadcast Bob's commitment transaction to the blockchain, whereby Bob has unwillingly sent 1 BTC to Carol through a blockchain transaction. If Bob realizes that the transaction hit the blockchain he could claim his funds (off-chain) from Jack since he knows R from the blockchain. However, if Bob did not pay attention, Jack could get the funds back after three days, whereby Bob had actually paid Emily. Consequently, participants in the network are incentivised to be cooperative.

Moreover, Emily might never send the pre-image R. In this case, Carol gets her payment back after the time lock period, i.e. after one day, Bob after two days and Jack after three days. The HTLC is then removed via an *update_fail_htlc* message. It becomes clear why the time lock periods of the different nodes have to differ. Imagine the time lock between Bob and Carol was one day and the one between Carol and Emily two days. Bob could reclaim the funds from Carol before Carol can from Emily. If Emily sends the pre-image R between the first and second day, she could claim the Bitcoin unit from Carol. In such a situation, Carol would have paid for Jack unintentionally.

6.3 Key Generation

If for each commitment transaction the corresponding private and public keys had to be stored, a massive amount of storage capacity would be required. This can be enhanced by working with hierarchical deterministic wallets[52], where both parties in a payment channel pre-generate the keys via a merkle tree. For example, both parties pre-generate one million keys where each key is a child of a previous key. The use of the keys follows some deterministic process to be defined. Bob could use the last key in the merkle tree as a master key to generate subkeys which are only used for the first day. As soon as the first day ended, Bob can share only his master key such that Jack does not have to store every single key. After the second day, the first master key is redundant to store as well since it is a child of the second master key. This process enables to reduce the amount that needs to be stored for validation purposes to a minimum. Thus, core channels in the Lightning network can process a tremendous amount of transactions with negligible storage costs.[88]

6.4 Network

Obviously, if each node in the network opened a channel with each other node, the scalability problem would not be solved. Scaling Bitcoin can be achieved successfully by relying on a large network of different nodes who maintain only a few payment channels with other nodes. If a user intends to send Bitcoin units to another user, the payment is sent via different hubs without new channels being created. Thus, sending funds becomes possible off-chain via a multi-hub system without a central clearing system.

This set up requires that many nodes in the network deposit a certain amount of redundant Bitcoin units in the channels which can be used in the HTLCs to process other parties' payments. These funds may be locked in some payment channels for a certain amount of time if a party is uncooperative. Since nodes

[52] See Berentsen and Schär (2017) or BIP32[128] for a detailed description of HD wallets.

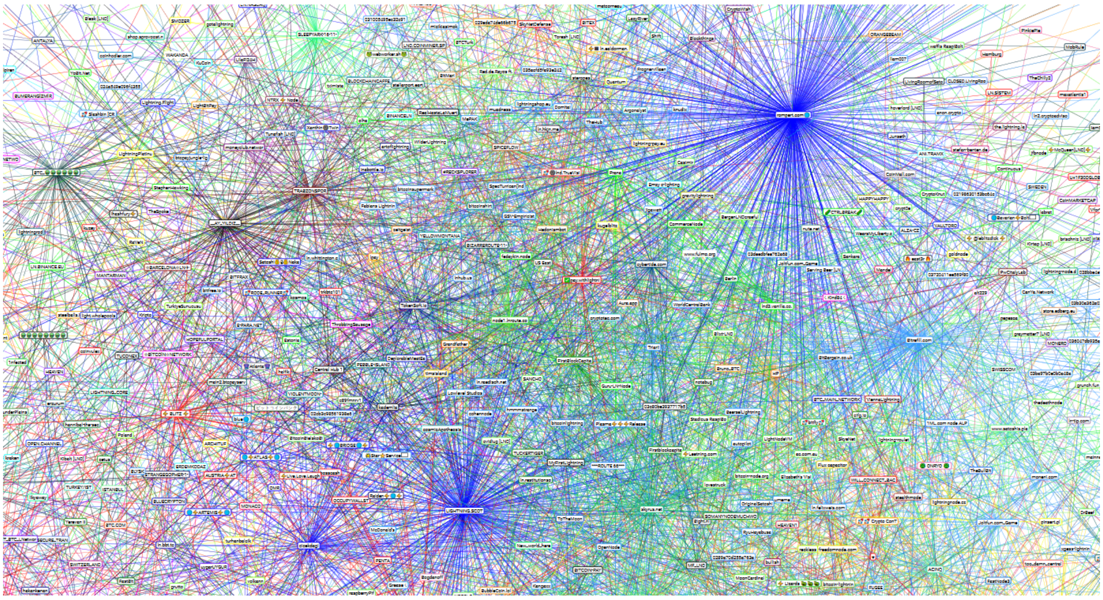


Figure 23: Illustration of a subset of the Lightning network (Source: `https://lnmainnet.gaben.win/`, 01.07.2018).

probably demand a compensation for the time-value of the locked funds, fees have to be paid to send transactions. Furthermore the fees reflect other costs like bandwidth. With a large and interconnected network, these fees might become very small, especially for micropayments.

If a fee market existed, some hubs may emerge who specialize on high interconnectivity and a 24/7 uptime. They need a lot of capital to meet all obligations but earn money with the fees paid in the network. If the Lightning network should scale to billions of transactions per day, centralized hubs are non-negligible. These centralized hubs, however, are a big point of criticism since the fundamental value of a decentralized system is challenged. According to some authors, it is even impossible for the Lightning network to scale as a peer-to-peer network.[55] For one payment, each intermediary hub has to lock the same amount of funds. Thus, the more hubs in a route, the more Bitcoins are locked in the Lightning network. Consequently, there is a trade-off between more channels per user, more deposited capital or centralized hubs. What is currently observed in the network is a clear tendency to central hubs maintaining payment channels to many nodes.[53] This

[53] Poon and Dryja described a network structure with centralized hubs already in their white paper, where they stated that "*eventually [...] the network will look a lot like the correspondent banking network*" and further that "*the core [...] routes can be online all the time while nodes at the edges, such as average users, would be connected intermittently*".[88]

can nicely be seen in a visualization of the Lightning network in figure 23.

A way to improve this situation is the idea that payments can be split over multiple routes such that single nodes do not need to deposit as much capital in the channels. This would be especially helpful for larger payments.

6.5 Routing

To route a payment from an origin node to a final node over multiple hops, onion routing packets are used.[54] Onion routing allows for anonymous communication through a network of nodes. Hops which receive a packet can verify its integrity and learn to which node they should forward it. Since the packets are obfuscated at each hop, a hop cannot learn the length of the route or their position within it and which other nodes are part of the route except for its predecessor and successor.[32]

We quickly discuss the basic mechanism of the onion routing protocol. The origin node first constructs a route through the network. The maximum route length in the Lightning network is limited to 20 hops. It will then collect the public key of each node and generate secrets and different decryption keys for each hop. The origin node has access to all keys whereas the single hops only have access to the specific keys. The origin node will then create a packet (onion) with different encryption layers. The innermost layer is related to the last node in the route and the uttermost layer to the first node. When a packet is forwarded through the network, a hop can only decrypt its respective layer of the packet, or in other words, can peel only one layer of the onion, hence the name onion routing. Therefore, a node has only access to information that is intended for it and cannot gain any other information about the route and who else is part of the network (except for the predecessor and the successor). If the final node wants to send a message back, it uses the same route but each node encrypts the message. The origin node can decrypt the message again since it has access to all keys.

[54] Onion routing is especially known from the TOR-browser ("The onion router"), which uses it for anonymity purposes.

With the current onion routing protocol, scaling the Lightning network is limited. Channel states updates are currently broadcast to every node in the network. This puts a massive burden on the network and may become problematic if there exist between 10'000 and 1'000'000 open channels.[94] There are also other routing proposals that could replace onion routing. One of them is Flare that would allow scaling to a network size of at least several hundred thousands of nodes.[89] Otherwise, the issue could be mitigated through centralized hubs. Nodes would not have to maintain as many payment channels but predominantly have open channels to the central hubs whereby the total number of channels existing in the network could be reduced.

6.6 eltoo

Below, we present a proposal of a new update mechanism for the Lightning network called *eltoo*.[55] Due to the Poon-Dryja payment channels used, a lot of data has to be stored to invalidate old transactions. Furthermore, if a node forgets about the latest update of a channel and propagates an old transaction without bad intentions - for example if after a backup the most recent updates are missing - it loses all the money in the channel. Christian Decker, Rusty Russell and Olaoluwa Osuntokun used this as a motivation to propose the *eltoo* update mechanism in April 2018.[56][45]

We quickly sketch the eltoo protocol. As before, there is a funding transaction to put funds in a 2-of-2 multisig. Additionally, update and settlement transactions exist. An update transaction is used to update the balances and the settlement transaction to broadcast them to the network. The update and the settlement transactions of a current state are always linked to each other through corresponding private and public keys for each state. The funding transaction's and each update transaction's output consists of an if/else condition, i.e. there are two ways how the output can be spent. On the one hand, a settlement transac-

[55] The name *eltoo* is derived from the phonetic spelling of *L2* which means layer-two, i.e. off-chain solutions to process transactions.

[56] eltoo is not included in any Lightning implementation so far.

tion can be created, which comes with a CSV-time lock that defines how long the output is locked after the funding or the previous update transaction is included in the blockchain. On the other hand, new update transactions can be created during the time lock period which are immediately valid to be spent again. A new update transaction spends either the funding transaction's or the previous update transaction's output and invalidates the old settlement transaction, since the output of the old update transaction is referenced in the new one and thus cannot be referenced in the old settlement transaction. Before a new update transaction is created, the two parties agree on a new settlement transaction.

With each new payment, a new update transaction is generated. Thus, a chain of update transactions is created where each update transaction's input references the preceding update transaction's output. The latest update and settlement transaction represent the current state of balances. In this regard, the authors introduce `SIGHASH_NOINPUT` that could be implemented with a soft fork.

Normally, it is not possible to modify the inputs of a transaction, since they are signed by the signature. Hence, any change would make the transaction invalid. As described in section 4.3, the parts that the signature commits to are defined by the sighash flag. Introducing the new sighash flag `SIGHASH_NOINPUT` would allow to alter the inputs without invalidating the transaction. Without the new sighash flag, the whole chain of update transactions had to be added to the blockchain if a user wants to broadcast the final balances. With `SIGHASH_NOINPUT`, the latest update transaction's input can directly reference the funding transaction when committed to the blockchain. Therefore, only the latest update and settlement transaction is needed to broadcast the current balances.

Furthermore, they introduce *state numbers* to provide a transaction order, because otherwise an older update transaction could be modified to spend a newer ones' output. This is implemented by using the `CHECKLOCKTIMEVERIFY` (CLTV) opcode in the output script. An update transaction can only spend the output of another update transaction if it has a higher *state number*. In general, `OP_CLTV` checks if the Unix timestamp defined in *nLockTime* is larger than a pre-specified time. For the purpose of *state numbers*, past timestamps are employed which are not used for regular `OP_CLTV` anymore. An update transaction is only valid if its *nLockTime* is at least by one bigger than the *nLockTime* of the predecessor. Unix timestamps start at 500'000'000 and are currently above 1'500'000'000. Therefore, over 1 billion updates are possible.

`SIGHASH_NOINPUT` and *state numbers* enable the enforcement of the latest state of the channel without taking care of any intermediate transactions, i.e. a node does not have to store any intermediate update and settlement transactions.

Nothing prevents a party from propagating an old update transaction with different balances. However, only after the update transaction's CSV-time lock is achieved, the settlement transaction will be valid and the channel's Bitcoin units paid out. In the meantime, the other party could immediately broadcast a newer update transaction that references the old update transaction and thus invalidates the old settlement transaction. This is always possible until no other update transaction is available anymore that is signed by both parties. To close the channel, the latest update and settlement transaction is committed to the blockchain.

6.7 Scalable Funding

In October 2017, Conrad Burchert, Christian Decker and Roger Wattenhofer proposed a new mechanism to raise efficiency in existing payment channel networks.[39] They highlight two shortcomings. First, existing payment channel networks (e.g. the Lightning network) require to lock a lot of funds in the channels and second, the limited space in the Bitcoin blockchain does not enable Bitcoin to become a wide-spread method of payment. In the Lightning network each channel's opening and closing transaction enters the blockchain, which, according to the authors, the Bitcoin blockchain is not capable to deal with in case the user numbers increase largely. They propose a new layer in between the Bitcoin blockchain and the payment channel network. A three layered system would emerge with the first and third layer already existing. The first layer locks the funds into a shared

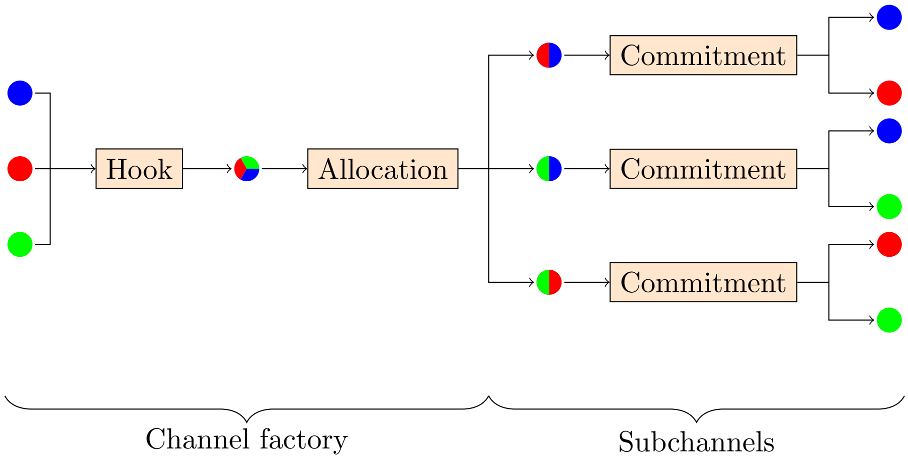


Figure 24: Illustration of a three party channel factory with three subchannels. (Source: Burchert et al. (2017)) Note: Each colour represents one participant of the channel factory. A circle with several colours indicates who is involved in the specific action.

ownership between a group of nodes and enforces off-chain consensus by using the blockchain. The second layer enables trustless off-chain channel funding by introducing shared accounts of groups of nodes which would reduce the cost of opening two-party channels. The third layer performs the transfer of Bitcoin units in the payment network.[57]

With the three layers solution, opening and closing transactions of two-party payment channels do not appear in the blockchain, except in case of disputes. Only one transaction per user enters the blockchain if one participates in the system for the first time (and another one if the group is dissolved). Subsequently, many channels can be opened without hitting the blockchain. According to the authors up to 90% of blockchain space could be saved for a group of 20 users with 100 channels in between them and up to 96% if Schnorr signatures are implemented (see section 7).

The second layer they introduce consists of so called channel factories. These factories can quickly fund a payment channel between two parties. We show an illustration of a three party channel factory in figure 24.[58] First, the three parties fund a 3-of-3 multisig through a *hook transaction*. Before signing and broadcast-

[57] The three layered system is an extension of the Lightning network but not included in any Lightning implementation so far.

[58] This could be analogously extended to a larger group.

ing it, an allocation and commitment transactions are created according to the initial balances to ensure that the funds will not become locked in the multisig if a party is uncooperative (similar to a refund transaction). The allocation transaction distributes the funds in the multisig to the single two-party channels and effectively replaces on-chain funding transactions. To update the allocation transaction, the authors propose an invalidation tree similar to the one used in Duplex micropayment channels (see section 5.4). The commitment transactions send the respective balances in a two-party channel back to the owners.

A user's payment channels might become unbalanced at some point, i.e. there are lots of funds in some channels but only little in others. The channel factory allows to rebalance the channels by setting up a new allocation transaction without hitting the blockchain. Furthermore, old channels can be removed, new ones created and funds can be transferred between channels.

Moreover, higher order systems can be introduced to include more users in a group and increase the interconnectivity. Larger groups can be created from several subgroups such that they overlap and network wide transactions are possible.

If all users in a channel factory agree to cooperatively close the factory, they can just create and broadcast a closing transaction where each user receives its respective balance immediately. The channel factory can also be closed unilaterally, since each node can broadcast the allocation transaction at any point in time to the Bitcoin blockchain. In this case, the participants can continue to use the subchannels, but the option of moving funds between the channels is lost. However, no user has a personal advantage of propagating the allocation, since an uncooperative closure requires that the fees of including the transaction into the blockchain have to be paid by the closing party.

6.8 Discussion

The Lightning network is a promising solution if it comes to scaling Bitcoin. In theory, it allows to process almost an indefinite amount of transactions off-chain whereas the Bitcoin blockchain is only needed for opening and closing transactions and for enforcing off-chain consensus. Contrary to the blockchain, where transactions need several minutes up to several hours to be confirmed, if there is high congestion on the network, instant transaction confirmation is possible on the Lightning network. This allows to use Bitcoin as a prevalent method of payment. Due to the use case for micropayments, it is easy to buy a coffee in a coffee house or pay for an ice cream with Bitcoins. Furthermore, the Lightning network enables atomic swaps or cross-chain transactions, i.e. nodes can process a transaction between two different blockchains as long as there exists a similar hash function across the chains.

There are, however, also some issues that have to be further worked on. On the one hand, going off-chain prevents the centralization threat that could come with a block size increase. On the other hand, for the Lightning network to scale as promised under current conditions, centralized hubs are non-negligible. Thus, off-chain centralization arises, which is not in line with Bitcoin's core value of being a decentralized system.

Moreover, the use of onion routing is a controversially discussed topic. Since only the first node knows the route through the network, a node within the route cannot make adjustments to the path if an issue in the route arises (e.g. if a channel closes or a node goes offline). For a dynamic network in which channel states are constantly changed and nodes may go online and offline all the time, this might be problematic if decentralization should be maintained simultaneously.

Furthermore, anonymity on the Lightning network is not equally guaranteed as in the TOR browser protocol. Contrary to TOR, a node in the Lightning network is not connected to any other node but only maintains a limited number of payment channels to other nodes, i.e. the interconnectivity is much smaller. The state of all these channels is broadcast to the whole network. Thus, a node knows for certain the subsequent hop of its successor and the preceding hop of its predecessor if the successor and the predecessor only maintain one additional channel. Nevertheless, anonymity is still way stronger than on a public ledger like the blockchain. What is more, the onion routing protocol limits the scalability of Bitcoin. Centralized hubs could serve as a solution, which is, however, criticised by many users.

Also, a node has to be constantly online to monitor the blockchain and make sure that no old channel state is broadcast. This is not a realistic requirement for all nodes participating in the Lightning network. As a solution, some authors propose to outsource this task to third parties.[43] The issue of loosing all funds in case of broadcasting an old transaction and the requirement to store old transactions could be solved by the introduction of *eltoo*.

Lastly, we want to highlight that the main use case of the Lightning network are micropayments. Large payments should still be processed on the blockchain directly. As long as payments cannot be split, each node has to deposit a large amount of Bitcoin units in a channel to enable the transmission of these transactions. On the other hand, sending micropayments directly on the blockchain has been economically meaningless due to relatively high transaction fees for small payments.

7 Other Concepts to Improve Scaling in Bitcoin

Above, we discussed in detail some relevant concepts that aim to scale Bitcoin. They are, however, by far not the only ones with this purpose which is why we list some other important proposals below. Some of them will probably never be implemented in the Bitcoin code, but others like Schnorr signatures are supported by large parts of the community. Not all proposals are directly about increasing the number of transactions, but help to scale Bitcoin in some way. Since scaling Bitcoin is not only limited by the network design but also by bandwidth or storage capacities, these proposals are of similar importance. Below, we list some of them and give a brief description what they are about.

Schnorr signatures: For creating the private key's corresponding signature, Bitcoin uses the *Elliptic Curve Digital Signature Algorithm* (ECDSA).[59]

[59] See Berentsen and Schär (2017) for an intuitive but not too technical explanation.

There exist a lot of other algorithms for cryptographic purposes. One of them using elliptic curves as well is called Schnorr. Schnorr signatures are compatible with all features of ECDSA in Bitcoin. Moreover, they are non-malleable. If someone does not have access to a private key, it is not possible to modify the signature without invalidating it. There are several other advantages of Schnorr signatures. They can be smaller and faster verifiable than ECDSA signatures. Furthermore, signature aggregation would be enabled. In case of multisig outputs, Schnorr signatures allow to combine n public keys and n signatures into one overall public key and signature. This increases for example the efficiency of certain applications used in payment channel networks.[14]

Sidechains: The concept of sidechains attracted a lot of attention due to a paper about *pegged sidechains* published in 2014 written by several well known *Bitcoin Core* developers.[7] Sidechains intend to provide another blockchain besides the Bitcoin blockchain. Users could "move" their Bitcoin units to a sidechain that exhibits different features without having to create a new coin.[60] They can use the coins on the sidechain and enjoy different functionalities without affecting the main Bitcoin blockchain. Any arbitrary design could be implemented for a sidechain depending on the use case, e.g. use of smart contracts, privacy features or using sidechains as a test environment before a new feature is implemented on the main chain. Moreover, a sidechain with a higher transaction throughput could be created to improve scalability of Bitcoin. However, the same restrictions as for any blockchain design remain, which might be deficient to scale Bitcoin sufficiently. Before sidechains can be implemented, there are still some security issues to be solved in order to safely move funds between the chains.

Block interval: The difficulty to create a new block in the Bitcoin blockchain is

[60] Technically, the coins are not moved. A user who wants to make use of a sidechain sends some Bitcoin units to a specific address. The funds are then locked in this address and are out of his control. The user proves in the sidechain that these funds really are locked whereby the same amount of coins is created on the sidechain. The user can then use the coins in the sidechain until he wants to transfer them back to the Bitcoin blockchain. He does so in locking up the funds in the sidechain and proving it to the Bitcoin blockchain, whereby the funds locked before can be used in the Bitcoin blockchain again.

adjusted every 2016 blocks such that the block interval is about ten minutes on average. If ceteris paribus the interval is reduced, more blocks are generated and the number of transactions processed can be increased. However, with a lower block interval, the probability of double spends and orphaned blocks is increased. Furthermore, storage requirements rise. These problems have to be taken into account before the interval is shortened.

Block Relay: If Bitcoin scaled by creating larger blocks, block relay has to be improved as well. Otherwise, less well connected nodes are disadvantaged. There a several proposals to increase block relay. For instance, the *Fast Internet Bitcoin Relay Engine* (FIBRE) is a protocol that promises very fast block relay with delays close to the physical limit of the speed of light.[54] BIP152 discusses compact blocks as a more efficient way to relay blocks over the network and save bandwidth.[42] Some proposals deal with the problem that transactions are propagated through the network twice, the first time when someone propagates the transaction itself and the second time when the block is relayed. This can be improved if nodes only fetch the transactions of a newly mined block they do not possess yet.[3]

MAST: Merkelized Abstract Syntax Trees (MAST) detailed in BIP114[71] would besides many other improvements like more privacy or improved smart contracts applicability reduce transaction size. With MAST, a Bitcoin script can be split up into several separate segments which are placed in a merkle tree. The merkle tree ensures that a segment belongs to the script without the need of providing the whole script. Code that is not executed can be replaced by a simple hash which saves space in a transaction. Thus, Bitcoin scripts with many complex payout conditions can be created that only use a small amount of data. MAST could be implemented through the versionbits enabled by Segwit.[14,63]

Batching: Batching is the idea of combining several separate transactions to a single one. To give a very simple example, imagine Bob who is in a bar with some friends and orders some beers. They stay the whole evening and Bob pays with Bitcoins every time they order a new round. Consequently, several transactions are created. Instead, Bob could just pay at the end of the evening and create only one single transaction. This puts significantly less burden on the blockchain. Batching is not only possible if transactions are sent to the same counterparty, but also if he sends payments to different recipients. For example, if Bob pays his bills at the end of the month, he could create single transactions for each bill. However, he could also create only one transaction with several different outputs. Adding some outputs to an existing transaction uses much less additional bytes than creating a whole new transaction.[62] As illustrated in figure 3, there is still a lot of potential to combine different payments in a single transaction. Every user is responsible for applying batching and take some burden off the blockchain. Nevertheless, batching is only a slight improvement to scaling and not the ultimate remedy.

Sharding: Sharding is about splitting up the tasks of consensus to different groups of nodes. Each group validates only a fraction of incoming transactions. This reduces the number of transactions that has to be processed by each individual node and reduces storage requirements for a single node. Sharding is often referred as horizontal scaling, since the total number of transactions in the whole network can be increased if the network grows.[67] In Bitcoin, there is no active debate ongoing about sharding. Contrarily, it is discussed and developed in Ethereum.[52,103]

Fundamental changes: There are also more radical changes to the protocol proposed to scale Bitcoin. The probability that they are implemented is rather small. For example, the *GHOST* (Greedy Heaviest Observed Subtree) protocol "*selects at each fork in the chain the heaviest subtree rooted at the fork*".[97] This would allow to reduce the block interval without orphan blocks becoming a serious problem. *PHANTOM* that is developed by the same authors like the *GHOST* protocol intends to replace the block structure by a *blockDAG* (directed acyclic graphs of blocks).[98] Moreover, Bitcoin-NG (Next Generation) proposes a Byzantine fault tolerant blockchain protocol that could deal with lots of transactions and shares the same trust model as Bitcoin by using a specific form of *leader selection* and *transaction serialization*.[53] Some of these proposals are rather old and never gained a lot of attention in the Bitcoin community.

8 Conclusion

In our paper, we aim to give a detailed insight into the Bitcoin scaling debate and discuss some of the most relevant proposals. If Bitcoin shall become a widespread method of payment, the basic protocol introduced by Satoshi Nakamoto in 2008[81] has to be enhanced with new technical concepts. We focus on a block size increase, Segwit, payment channels and the corresponding payment channel networks. These topics are by far not the only concepts that aim to improve scaling. In our opinion, however, they are the most advanced ones which have played a dominant role in the scaling debate.

To our knowledge, there is no extensive paper that provides a detailed description of the aforementioned concepts. A vast number of blogs, threads in forums, articles in online magazines, explanations in Github repositories and few papers discuss the Bitcoin scaling debate, but the interested user has to arduously collect the information from many different sources. Hence, our main contribution is to provide a detailed overview of the Bitcoin scaling debate and discuss the different concepts. Our goal is to make the scaling debate comprehensible for the average Bitcoin user. We, therefore, try to provide a good mixture between technical details and intuition. Since the scaling debate is a very controversial topic we abstain from subjectively assessing the applicability of the proposals. We only present arguments of different interest groups that are used in the context of the scaling debate.

We conclude by saying that there is no ultimate answer to the question of how Bitcoin should be scaled. All proposals have some advantages and disadvantages. Which arguments outweigh at the end is strongly dependent on what kind of values a user shares. Most probably there is no best proposal which solves the scaling debate itself, but a combination of different ideas can make the difference. Consequently, it is crucial that many researchers keep working on Bitcoin and on the scalability of blockchains in general.

References

[1] 1ml. *Statistics*. URL: https://1ml.com/statistics (visited on 15/07/2018).

[2] Gavin Andresen. *Increase maximum block size*. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki (visited on 16/05/2018).

[3] Gavin Andresen. *O(1) Block Propagation*. URL: https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2 (visited on 12/07/2018).

[4] Gavin Andresen. *One-dollar lulz*. 2016. URL: http://gavinandresen.ninja/One-Dollar-Lulz (visited on 30/03/2018).

[5] Gavin Andresen. *Two million byte size limit with sigop and sighash limits*. 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0109.mediawiki (visited on 16/05/2018).

[6] Gavin Andresen. *Will a 20MB max increase centralization?* 2015. URL: http://gavinandresen.ninja/does-more-transactions-necessarily-mean-more-centralized (visited on 02/04/2018).

[7] Aadam Back et al. *Enabling blockchain innovations with pegged sidechains*. 2014.

[8] Mike Belshe. *Segwit2x Final Steps*. 2017. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-segwit2x/2017-November/000685.html (visited on 30/04/2018).

[9] Aleksander Berentsen and Fabian Schär. *Bitcoin, Blockchain und Kryptoassets: eine umfassende Einführung*. 2017.

[10] Marc Bevand. *Running a Bitcoin full node on $5 a month*. 2017. URL: http://blog.zorinaq.com/full-node-on-5-dollars/ (visited on 18/05/2018).

[11] Bitcoin Core. *Bitcoin Capacity Increases FAQ*. 2015. URL: https://bitcoincore.org/en/2015/12/23/capacity-increases-faq/ (visited on 05/04/2018).

[12] Bitcoin Core. *Bitcoin Core version 0.9.0 released.* 2014. URL: https://bitcoin.org/en/release/v0.9.0#rebranding-to-bitcoin-core (visited on 09/05/2018).

[13] Bitcoin Core. *Segregated Witness Benefits.* 2016. URL: https://bitcoincore.org/en/2016/01/26/segwit-benefits/ (visited on 27/05/2018).

[14] Bitcoin Core. *Segregated witness: the next steps.* 2016. URL: https://bitcoincore.org/en/2016/06/24/segwit-next-steps/ (visited on 11/07/2018).

[15] Bitcoin Developer Reference. *Block Headers.* 2018. URL: https://bitcoin.org/en/developer-reference (visited on 02/05/2018).

[16] Bitcoin Glossary. *Opcode.* URL: https://bitcoin.org/en/glossary/op-code (visited on 16/06/2018).

[17] Bitcoin Roundtable. *Bitcoin Roundtable Consensus.* 2016. URL: https://medium.com/@bitcoinroundtable/bitcoin-roundtable-consensus-266d475a61ff (visited on 14/05/2018).

[18] BitcoinJ. *Working with micropayment channels.* URL: https://bitcoinj.github.io/working-with-micropayments (visited on 12/06/2018).

[19] Bitcoinwiki. *Block.* 2018. URL: https://en.bitcoin.it/wiki/Block (visited on 02/05/2018).

[20] Bitcoinwiki. *Block hashing algorithm.* 2015. URL: https://en.bitcoin.it/wiki/Block_hashing_algorithm (visited on 02/05/2018).

[21] Bitcoinwiki. *Timelock.* 2017. URL: https://en.bitcoin.it/wiki/Timelock (visited on 11/06/2018).

[22] Bitcoinwiki. *Transaction.* 2018. URL: https://en.bitcoin.it/wiki/Transaction (visited on 03/05/2018).

[23] BitFury Group. *BitFury Report On Block Size Increase.* 2015.

[24] BitInfoCharts. *Cryptocurrency statistics.* URL: https://bitinfocharts.com/ (visited on 15/07/2018).

[25] Bitmain. *UAHF: A contingency plan against UASF (BIP148)*. 2017. URL: https://blog.bitmain.com/en/uahf-contingency-plan-uasf-bip148/ (visited on 13/04/2018).

[26] BitPay. *What happens if I don't upgrade my Bitcoin node for Segwit?* 2017. URL: https://support.bitpay.com/hc/en-us/articles/115004141766-What-happens-if-I-don-t-upgrade-my-Bitcoin-node-for-Segwit- (visited on 28/05/2018).

[27] Blueadept. *Bi-directional micro payment channels with single party bitcoin locking?* 2014. URL: https://bitcointalk.org/index.php?topic=814770.msg9185225#msg9185225 (visited on 15/05/2018).

[28] BOLT. *Base Protocol*. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/01-messaging.md#reference-1 (visited on 28/06/2018).

[29] BOLT. *Bitcoin Transaction and Script Formats*. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#requirements (visited on 30/06/2018).

[30] BOLT. *Invoice Protocol for Lightning Payments*. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md#examples (visited on 01/07/2018).

[31] BOLT. *lightning-rfc*. URL: https://github.com/lightningnetwork/lightning-rfc (visited on 03/07/2018).

[32] BOLT. *Onion Routing Protocol*. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md (visited on 30/06/2018).

[33] BOLT. *Peer Protocol for Channel Management*. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md (visited on 29/06/2018).

[34] Danny Bradbury. *What the 'Bitcoin Bug' Means: A Guide to Transaction Malleability*. 2014. URL: https://www.coindesk.com/bitcoin-bug-guide-transaction-malleability/ (visited on 20/05/2018).

[35] Ryan Browne. *Big transaction fees are a problem for bitcoin — but there could be a solution.* 2017. URL: https://www.cnbc.com/2017/12/19/big-transactions-fees-are-a-problem-for-bitcoin.html (visited on 23/04/2018).

[36] BtcDrak and Mark Friedenbach. *Relative lock-time using consensus-enforced sequence numbers.* 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki (visited on 15/04/2018).

[37] BtcDrak, Mark Friedenbach and Eric Lombrozo. *CHECKSEQUENCEVERIFY.* 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki (visited on 15/06/2018).

[38] Jean-Pierre Buntinx. *Pro's and Con's on Bitcoin Block Pruning - Bitcoin News.* 2015. URL: https://news.bitcoin.com/pros-and-cons-on-bitcoin-block-pruning/ (visited on 03/04/2018).

[39] Conrad Burchert, Christian Decker and Roger Wattenhofer. "Scalable Funding of Bitcoin Micropayment Channel Networks". In: *Stabilization, Safety, and Security of Distributed Systems.* 2017, pp. 361–377.

[40] Celean. *UDP flood DDoS attacks against XT nodes.* 2016. URL: https://www.reddit.com/r/bitcoinxt/comments/3iumsr/udp_flood_ddos_attacks_against_xt_nodes/ (visited on 09/05/2018).

[41] Bitcoin Classic. *How do SegWit and FlexTrans compare?* URL: https://bitcoinclassic.com/devel/FlexTrans-vs-SegWit.html (visited on 15/07/2018).

[42] Matt Corallo. *Compact Block Relay.* 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki (visited on 11/07/2018).

[43] Kyle Croman et al. *On scaling decentralized blockchains (A position paper).* 2016.

[44] Cryddit. *Permanently keeping the 1MB (anti-spam) restriction is a great idea.* 2015. URL: https://bitcointalk.org/index.php?topic=946236.msg10388435#msg10388435 (visited on 04/05/2018).

[45] Christian Decker, Rusty Russell and Olaoluwa Osuntokun. *eltoo: A Simple Layer2 Protocol for Bitcoin*. 2018. URL: https://blockstream.com/eltoo.pdf.

[46] Christian Decker and Roger Wattenhofer. *A fast and scalable payment network with bitcoin duplex micropayment channels*. 2015.

[47] Developer Guide. *Bitcoin Developer Guide*. 2018. URL: https://bitcoin.org/en/developer-guide (visited on 03/05/2018).

[48] Developer Guide. *Micropayment Channel*. URL: https://bitcoin.org/en/developer-guide#micropayment-channel (visited on 12/06/2018).

[49] Digital Currency Group. *Bitcoin Scaling Agreement at Consensus 2017*. 2017. URL: https://medium.com/@DCGco/bitcoin-scaling-agreement-at-consensus-2017-133521fe9a77 (visited on 09/04/2018).

[50] Elementsproject. *Segregated Witness*. 2015. URL: https://elementsproject.org/elements/segregated-witness/ (visited on 11/05/2018).

[51] Enochian. *New Attack Vector*. 2011. URL: https://bitcointalk.org/index.php?topic=8392.msg122410#msg122410 (visited on 21/05/2018).

[52] Ethereum Wiki. *On sharding blockchains*. URL: https://github.com/ethereum/wiki/wiki/Sharding-FAQs (visited on 12/07/2018).

[53] Ittay Eyal et al. *Bitcoin-NG: A Scalable Blockchain Protocol*. 2015.

[54] FIBRE. *Fast Internet Bitcoin Relay Engine*. URL: http://bitcoinfibre.org/ (visited on 11/07/2018).

[55] Jonald Fyookball. *Mathematical Proof That the Lightning Network Cannot Be a Decentralized Bitcoin Scaling Solution*. 2017. URL: https://medium.com/@jonaldfyookball/mathematical-proof-that-the-lightning-network-cannot-be-a-decentralized-bitcoin-scaling-solution-1b8147650800 (visited on 26/06/2018).

[56] Jeff Garzik. *Block size increase to 2 MB*. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki (visited on 16/05/2018).

[57] Jeff Garzik. *Segregated Witness in the context of Scaling Bitcoin.* 2015. URL: https://www.mail-archive.com/bitcoin-dev@lists.linuxfoundation.org/msg03051.html (visited on 11/05/2018).

[58] Jeff Garzik, Tom Harding and Dagur Valberg. *Dynamic maximum block size by miner vote.* 2015. URL: https://github.com/jgarzik/bip100/blob/master/bip-0100.mediawiki (visited on 16/05/2018).

[59] John M. Griffin and Amin Shams. *Is Bitcoin Really Un-Tethered?* 2018. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3195066.

[60] Samuel Haig. *Satoshi Nakamoto's Confidant Gavin Andresen Throws Support Behind Bitcoin Cash.* 2017. URL: https://news.bitcoin.com/satoshi-nakamotos-confidant-gavin-andresen-throws-support-behind-bitcoin-cash/ (visited on 14/05/2018).

[61] Nermin Hajdarbegovic. *Price Drops as Mt. Gox Blames Bitcoin Flaw for Withdrawal Delays.* 2014. URL: https://www.coindesk.com/price-drops-mt-gox-blames-bitcoin-flaw-withdrawal-delays/ (visited on 18/05/2018).

[62] David A. Harding. *Saving up to 80% on Bitcoin transaction fees by batching payments.* 2017. URL: https://bitcointechtalk.com/saving-up-to-80-on-bitcoin-transaction-fees-by-batching-payments-4147ab7009fb (visited on 22/07/2018).

[63] David A. Harding. *What is a Bitcoin Merklized Abstract Syntax Tree (MAST)?* 2017. URL: https://bitcointechtalk.com/what-is-a-bitcoin-merklized-abstract-syntax-tree-mast-33fdf2da5e2f (visited on 11/07/2018).

[64] Matthew Haywood. *All roads lead to Segwit — Segwit2x, BIP 91 Segsignal and UASF.* 2017. URL: https://medium.com/@wintercooled/segwit2x-segsignal-and-the-uasf-all-roads-lead-to-segwit-d66fedf7fba (visited on 13/04/2018).

[65] Mike Hearn. *The Resolution of the Bitcoin Experiment*. 2016. URL: https://medium.com/@octskyward/the-resolution-of-the-bitcoin-experiment-dabb30201f7#.xccfiy7ms (visited on 28/03/2018).

[66] Impulse. *Inter-Channel Payments*. Tech. rep. 2015.

[67] Yaoqi Jia. *Op Ed: The Many Faces of Sharding for Blockchain Scalability*. 2018. URL: https://bitcoinmagazine.com/articles/op-ed-many-faces-sharding-blockchain-scalability/ (visited on 12/07/2018).

[68] Thomas Kerin and Mark Friedenbach. *Median time-past as endpoint for lock-time calculations*. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki (visited on 15/06/2018).

[69] Evan Klitzke. *Bitcoin Transaction Malleability*. 2017. URL: https://eklitzke.org/bitcoin-transaction-malleability (visited on 21/05/2018).

[70] Dimitry Laptev. *Bitcoin: transactions, malleability, SegWit and scaling*. 2017. URL: https://medium.com/@x0100/bitcoin-transactions-malleability-segwit-and-scaling-258af8ed9cbf (visited on 24/05/2018).

[71] Johnson Lau. *Merkelized Abstract Syntax Tree*. 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0114.mediawiki (visited on 11/07/2018).

[72] Johnson Lau and Pieter Wuille. *Transaction Signature Verification for Version 0 Witness Program*. 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki#cite_note-wiki_1 (visited on 26/05/2018).

[73] Sergio Damian Lerner. *New Bitcoin vulnerability: A transaction that takes at least 3 minutes to verify*. 2013. URL: https://bitcointalk.org/index.php?topic=140078 (visited on 17/05/2018).

[74] Sergio Damian Lerner. *Segwit2Mb - combined soft/hard fork - Request For Comments*. 2017. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-March/013921.html (visited on 07/04/2018).

[75] Taariq Lewis. *SF Bitcoin Devs Seminar: Scaling Bitcoin to Billions of Transactions Per Day.* 2015. URL: https://www.youtube.com/watch?time_continue=16&v=8zVzw912wPo (visited on 09/05/2018).

[76] Lightning Developers. *Lightning Protocol 1.0: Compatibility Achieved.* 2017. URL: https://medium.com/@lightning_network/lightning-protocol-1-0-compatibility-achieved-f9d22b7b19c4 (visited on 15/05/2018).

[77] Liquidity.Network. *Understanding off-chain transactions in blockchain for fun and profit.* 2017. URL: https://medium.com/@liquidity.network/understanding-off-chain-transactions-in-blockchain-for-fun-and-profit-591e7e27ccc0 (visited on 09/06/2018).

[78] Eric Lombrozo, Johnson Lau and Pieter Wuille. *Segregated Witness (Consensus layer).* 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki#cite_note-2 (visited on 25/05/2018).

[79] Gregory Maxwell. *I do not support the BIP 148 UASF.* 2017. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-April/014152.html (visited on 14/05/2018).

[80] Daniel Morgan. *The Great Bitcoin Scaling Debate — A Timeline.* 2017. URL: https://hackernoon.com/the-great-bitcoin-scaling-debate-a-timeline-6108081dbada (visited on 24/03/2018).

[81] Satoshi Nakamoto. *Bitcoin : A Peer-to-Peer Electronic Cash System.* 2008.

[82] Tier Nolan. *Alt chains and atomic transfers.* 2013. URL: https://bitcointalk.org/index.php?topic=193281.0.

[83] Pinar A. Ozisik et al. *Graphene: A New Protocol for Block Propagation Using Set Reconciliation.* URL: https://www.youtube.com/watch?time_continue=10570&v=BPNs9EVxWrA (visited on 30/07/2018).

[84] P2sh. *SegWit Usage.* URL: https://p2sh.info/dashboard/db/segwit-usage?orgId=1 (visited on 15/07/2018).

[85] Stephen Pair et al. *Industry Letter.* 2015. URL: https://bitcoinxt.software/industry-letter.pdf (visited on 09/05/2018).

[86] Daniel Palmer. *Scalability Debate Continues As Bitcoin XT Proposal Stalls.* 2016. URL: http://www.coindesk.com/scalability-debate-bitcoin-xt-proposal-stalls/ (visited on 06/04/2018).

[87] PayPal. *PayPal Reports Fourth Quarter and Full Year 2017 Results.* 2018. URL: https://www.businesswire.com/news/home/20180131006195/en/PayPal-Reports-Fourth-Quarter-Full-Year-2017 (visited on 08/05/2018).

[88] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.* 2016.

[89] Pavel Prihodko et al. *Flare: An Approach to Routing in Lightning Network.* 2016.

[90] Jamie Redman. *Fork Watch: Block 478558 Initiates 'Bitcoin Cash' Split – First Blocks Now Mined.* 2017. URL: https://news.bitcoin.com/fork-watch-first-bitcoin-cash-block-mined/ (visited on 24/04/2018).

[91] Peter Rizun. *SegWit Coins are not Bitcoins.* 2017. URL: https://www.youtube.com/watch?v=VoFb3mcxluY&t=1723s (visited on 29/05/2018).

[92] Pete Rizzo. *Is Segregated Witness the Answer to Bitcoin's Block Size Debate?* 2015. URL: https://www.coindesk.com/segregated-witness-bitcoin-block-size-debate/ (visited on 11/05/2018).

[93] Meni Rosenfeld. *Elastic block cap with rollover peanlties.* 2015. URL: https://bitcointalk.org/index.php?topic=1078521.0 (visited on 16/05/2018).

[94] RustyReddit. *What is the status of the Lightning Network?* 2017. URL: https://www.reddit.com/r/Bitcoin/comments/714x2k/what_is_the_status_of_the_lightning_network/ (visited on 01/07/2018).

[95] Satoshi. *Increase block size limit.* 2010. URL: https://bitcointalk.org/index.php?topic=1347.msg15366#msg15366 (visited on 04/05/2018).

[96] Shaolinfry. *Moving towards a user activated soft fork activation.* 2017. URL: https://bitcointalk.org/index.php?topic=1805060.0 (visited on 14/05/2018).

[97] Yonatan Sompolinsky and Aviv Zohar. *Secure High-Rate Transaction Processing in.* 2013.

[98] Yonatan Sompolinsky, Aviv Zohar and Computer Science. *Phantom.* 2018.

[99] Jimmy Song. *Bitcoin Cash: What You Need to Know.* 2017. URL: https://medium.com/@jimmysong/bitcoin-cash-what-you-need-to-know-c25df28995cf (visited on 15/04/2018).

[100] Jimmy Song. *Segwit2x Bugs Explained.* 2017. URL: https://bitcointechtalk.com/segwit2x-bugs-explained-8e0c286124bc (visited on 21/05/2018).

[101] Jon Southurst. *Mt. Gox Halts ALL Bitcoin Withdrawals, Price Drop Follows.* 2014. URL: https://www.coindesk.com/mt-gox-halts-bitcoin-withdrawals-price-drop/ (visited on 18/05/2018).

[102] Emily Spaven. *Bitcoin Exchanges Under 'Massive and Concerted Attack'.* 2014. URL: https://www.coindesk.com/massive-concerted-attack-launched-bitcoin-exchanges/ (visited on 18/05/2018).

[103] William Suberg. *Vitalik Buterin: Sharding Scaling Improvement 'Is Coming' To Ethereum Network.* 2018. URL: https://cointelegraph.com/news/vitalik-buterin-sharding-scaling-improvement-is-coming-to-ethereum-network (visited on 12/07/2018).

[104] Federico Tenga. *Understanding Payment Channels.* 2018. URL: https://blog.chainside.net/understanding-payment-channels-4ab018be79d4 (visited on 17/06/2018).

[105] Theymos. *It's time for a break: About the recent mess & temporary new rules.* 2016. URL: https://www.reddit.com/r/Bitcoin/comments/3h9cq4/its_time_for_a_break_about_the_recent_mess/ (visited on 09/05/2018).

[106] Peter Todd. *Near-zero fee transactions with hub-and-spoke micropayments.* 2014. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-December/006988.html (visited on 15/05/2018).

[107] Peter Todd. *OP_CHECKLOCKTIMEVERIFY*. 2014. URL: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki (visited on 11/06/2018).

[108] Kyle Torpey. *Antpool Will Not Run SegWit Without Bitcoin Block Size Increase Hard Fork*. 2016. URL: https://bitcoinmagazine.com/articles/antpool-will-not-run-segwit-without-block-size-increase-hard-fork-1464028753/ (visited on 14/05/2018).

[109] Anthony Towns. *Capacity increases for the Bitcoin system*. 2015. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-December/011869.html (visited on 28/05/2018).

[110] Unicode. *Miscellaneous Symbols*. URL: http://www.unicode.org/charts/PDF/U2600.pdf (visited on 28/06/2018).

[111] Kiran Vaidya. *Bitcoin's implementation of Blockchain*. 2016. URL: https://medium.com/all-things-ledger/bitcoins-implementation-of-blockchain-2be713f662c2 (visited on 02/05/2018).

[112] ViaBTC. *Statement on Bitcoin User Activated Hard Fork*. 2017. URL: https://medium.com/@ViaBTC/statement-on-bitcoin-user-activated-hard-fork-6e7aebb67e67 (visited on 15/04/2018).

[113] Visa. *Annual Report 2017*. Tech. rep. 2017. URL: https://s1.q4cdn.com/050606653/files/doc_financials/annual/2017/Visa-2017-Annual-Report.pdf.

[114] Visa. *Visa Inc. Facts & Figures*. Tech. rep. 2017. URL: https://usa.visa.com/dam/VCOM/global/about-visa/documents/visa-facts-figures-jan-2017.pdf.

[115] Aaron van Wirdum. *After Scaling Bitcoin, a Lightning Winter Release Is Now Within Reach*. 2016. URL: https://bitcoinmagazine.com/articles/after-scaling-bitcoin-a-lightning-winter-release-is-now-within-reach-1476814566/ (visited on 15/05/2018).

[116] Aaron van Wirdum. *Chinese Mining Pools Call for Consensus; Refuse Switch to Bitcoin XT*. 2015. URL: https://cointelegraph.com/news/chinese-mining-pools-call-for-consensus-refuse-switch-to-bitcoin-xt (visited on 04/04/2018).

[117] Aaron van Wirdum. *Chinese Mining Pools Propose Alternative 8 MB Block Size*. 2015. URL: https://cointelegraph.com/news/chinese-mining-pools-propose-alternative-8-mb-block-size (visited on 09/05/2018).

[118] Aaron van Wirdum. *Monero Just Hard Forked — and It Resulted in Four New Projects*. 2018. URL: https://bitcoinmagazine.com/articles/monero-just-hard-forked-and-it-resulted-four-new-projects/ (visited on 18/05/2018).

[119] Aaron van Wirdum. *The Future of "Bitcoin Cash:" An Interview with Bitcoin ABC lead developer Amaury Séchet*. 2017. URL: https://bitcoinmagazine.com/articles/future-bitcoin-cash-interview-bitcoin-abc-lead-developer-amaury-sÃl'chet/ (visited on 14/04/2018).

[120] Aaron van Wirdum. *The History of Lightning: From Brainstorm to Beta*. 2018. URL: https://bitcoinmagazine.com/articles/history-lightning-brainstorm-beta/ (visited on 15/05/2018).

[121] Aaron van Wirdum. *The Long Road to SegWit: How Bitcoin's Biggest Protocol Upgrade Became Reality*. 2017. URL: https://bitcoinmagazine.com/articles/long-road-segwit-how-bitcoins-biggest-protocol-upgrade-became-reality/ (visited on 11/05/2018).

[122] Aaron van Wirdum. *The Who, What, Why and How of the Ongoing Transaction Malleability Attack*. 2015. URL: https://bitcoinmagazine.com/articles/the-who-what-why-and-how-of-the-ongoing-transaction-malleability-attack-1444253640/ (visited on 20/05/2018).

[123] Aaron van Wirdum. *Understanding the Lightning Network, Part 1: Building a Bidirectional Bitcoin Payment Channel*. 2016. URL: https://bitcoinmagazine.

com/articles/understanding-the-lightning-network-part-building-a-bidirectional-payment-channel-1464710791/ (visited on 17/06/2018).

[124] Aaron van Wirdum. *Understanding the Lightning Network, Part 3: Completing the Puzzle and Closing the Channel*. 2016. URL: https://bitcoinmagazine.com/articles/understanding-the-lightning-network-part-completing-the-puzzle-and-closing-the-channel-1466178980/ (visited on 21/06/2018).

[125] Aaron van Wirdum. *Why ViaBTC Rejects SegWit Soft Fork in Favor of Block Size Hard Fork: Interview With Haipo Yang*. 2016. URL: https://bitcoinmagazine.com/articles/why-viabtc-rejects-segwit-soft-fork-in-favor-of-block-size-hard-fork-interview-with-haipo-yang-1479409475/ (visited on 14/05/2018).

[126] Halle Wittenberg. *on Block*. 1994. URL: https://medium.com/@octskyward/on-block-sizes-e047bc9f830 (visited on 04/04/2018).

[127] Pieter Wuille. *Block size following technological growth*. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0103.mediawiki (visited on 16/05/2018).

[128] Pieter Wuille. *Hierarchical Deterministic Wallets*. 2012. URL: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki (visited on 25/06/2018).

[129] Pieter Wuille. *Segregated Witness for Bitcoin*. 2015. URL: https://www.youtube.com/watch?time_continue=2160&v=fst1IK_mrnghttp://diyhpl.us/wiki/transcripts/scalingbitcoin/hong-kong/segregated-witness-and-its-impact-on-scalability/ (visited on 11/05/2018).

[130] Pieter Wuille and Gregory Maxwell. *Base32 address format for native v0-16 witness outputs*. 2017. URL: https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki.