Chapter 25

Eventual Consistency & Bitcoin

How would you implement an ATM? Does the following implementation work satisfactorily?

Algorithm 25.1 Naïve ATM

```
1: ATM makes withdrawal request to bank
2: ATM waits for response from bank
3: if balance of customer sufficient then
4:    ATM dispenses cash
5: else
6:    ATM displays error
7: end if
```

Remarks:

- A connection problem between the bank and the ATM may block Algorithm 25.1 in Line 2.
- A *network partition* is a failure where a network splits into at least two parts that cannot communicate with each other. Intuitively any non-trivial distributed system cannot proceed during a partition *and* maintain consistency. In the following we introduce the tradeoff between consistency, availability and partition tolerance.
- There are numerous causes for partitions to occur, e.g., physical disconnections, software errors, or incompatible protocol versions. From the point of view of a node in the system, a partition is similar to a period of sustained message loss.

25.1 Consistency, Availability and Partitions

Definition 25.2 (Consistency)**.** *All nodes in the system agree on the current state of the system.*

Definition 25.3 (Availability)**.** *The system is operational and instantly processing incoming requests.*

Definition 25.4 (Partition Tolerance)**.** *Partition tolerance is the ability of a distributed system to continue operating correctly even in the presence of a network partition.*

Theorem 25.5 (CAP Theorem)**.** *It is impossible for a distributed system to simultaneously provide Consistency, Availability and Partition Tolerance. A distributed system can satisfy any two of these but not all three.*

Proof. Assume two nodes, sharing some state. The nodes are in different partitions, i.e., they cannot communicate. Assume a request wants to update the state and contacts a node. The node may either: 1) update its local state, resulting in inconsistent states, or 2) not update its local state, i.e., the system is no longer available for updates. □

Algorithm 25.6 Partition tolerant and available ATM

```
if bank reachable then
  Synchronize local view of balances between ATM and bank
  if balance of customer insufficient then
    ATM displays error and aborts user interaction
  end if
end if
ATM dispenses cash
ATM logs withdrawal for synchronization
```

Remarks:

- Algorithm 25.6 is partition tolerant and available since it continues to process requests even when the bank is not reachable.
- The ATM's local view of the balances may diverge from the balances as seen by the bank, therefore consistency is no longer guaranteed.
- The algorithm will synchronize any changes it made to the local balances back to the bank once connectivity is re-established. This is known as eventual consistency.

Definition 25.7 (Eventual Consistency)**.** *If no new updates to the shared state are issued, then eventually the system is in a quiescent state, i.e., no more messages need to be exchanged between nodes, and the shared state is consistent.*

Remarks:

- Eventual consistency is a form of *weak consistency.*
- Eventual consistency guarantees that the state is eventually agreed upon, but the nodes may disagree temporarily.
- During a partition, different updates may semantically conflict with each other. A *conflict resolution* mechanism is required to resolve the conflicts and allow the nodes to eventually agree on a common state.

- One example of eventual consistency is the Bitcoin cryptocurrency system.

25.2 Bitcoin

Definition 25.8 (Bitcoin Network)**.** *The Bitcoin network is a randomly connected overlay network of a few tens of thousands of individually controlled* ***nodes****.*

Remarks:

- The lack of structure is intentional: it ensures that an attacker cannot strategically position itself in the network and manipulate the information exchange. Information is exchanged via a simple gossip protocol (nodes tells their neighbors about new messages).

- Old nodes re-entering the system try to connect to peers that they were earlier connected to. If those peers are not available, they default to the new node behavior.

- New nodes entering the system face the bootstrap problem, and can find active peers any which way they want. If they cannot find an active peer, their node will look for active peers from a set of authoritative sources. These authoritative sources are hard-coded in the Bitcoin source code.

Definition 25.9 (Cryptographic Keys)**.** *Users can generate any number of private keys. From each private key a corresponding public key can be derived using arithmetic operations over a finite field. A public key may be used to identify the recipient of funds in Bitcoin, and the corresponding private key can spend these funds.*

Remarks:

- Bitcoin supports the ECDSA and the Schnorr digital signature algorithms to verify ownership of bitcoins.

- It is hard to link public keys to the user that controls them, hence Bitcoin is often referred to as being *pseudonymous.*

Definition 25.10 (Bitcoin Currency)**.** *Bitcoin, the currency, is an integer value that is transferred in Bitcoin transactions. This integer value is measured in Satoshi; 100 million Satoshi are 1 Bitcoin.*

Definition 25.11 (Transaction)**.** *A transaction is a data structure that describes the transfer of bitcoins from spenders to recipients. It consists of inputs and outputs. Outputs are tuples consisting of an amount of bitcoins and a spending condition. Inputs are references to outputs of previous transactions.*

Remarks:

- New transactions refer to old transactions, which refer to even older transactions. Every transaction can publicly followed back to coinbase transactions of blocks (see Definition 25.18)
- A recipient with a public/private key pair can be paid by a transaction whose output's spending condition locks the payment with the public key. It can be unlocked and spent in the future if the recipient signs a future transaction with the private key.
- Inputs reference the output that is being spent by a (h, i)-tuple, where h is the hash of the transaction that created the output, and i specifies the index of the output in that transaction.
- Spending conditions are scripts that offer a variety of options. Apart from a single signature, they may include conditions that require multiple signatures, the result of a simple computation, or the solution to a cryptographic puzzle. However, Bitcoin spending scripts are not Turing complete.
- Transactions can be gossiped by any node in the network and are processed by every node that receives them through the gossip protocol.
- Outputs exist in two states: unspent and spent. An output is originally unspent, and can be spent at most once.
- The set of unspent transaction outputs (UTXO) is part of the shared state of Bitcoin. Every node in the Bitcoin network holds a complete replica of that state. Local replicas may temporarily diverge, but consistency is eventually re-established.

Algorithm 25.12 Node Receives Transaction (Naïve)

```
Receive transaction t
for each input (h, i) in t do
  if output (h, i) is not in local UTXO set or signature invalid then
    Drop t and stop
  end if
end for
if sum of values of inputs < sum of values of new outputs then
  Drop t and stop
end if
for each input (h, i) in t do
  Remove (h, i) from local UTXO set
end for
for each output o in t do
  add o to local UTXO set
end for
Forward t to neighbors in the Bitcoin network
```

Remarks:

- Note that the effect of a transaction on the state is deterministic. In other words if all nodes receive the same set of transactions in the same order (Definition 15.8), then the state across nodes is consistent.
- The outputs of a transaction may assign less than the sum of inputs, in which case the difference is called the transaction *fee*. The fee is used to incentivize other participants in the system (see Definition 25.18)
- Notice that so far we only described a local acceptance policy. Nothing prevents two nodes to locally accept different transactions that spend the same output.
- Transactions are in one of two states: unconfirmed or confirmed. Incoming transactions from the broadcast are unconfirmed and added to a pool of transactions called the *memory pool*.

Definition 25.13 (Doublespend)**.** *A doublespend is a situation in which multiple transactions attempt to spend the same output. Only one transaction can be valid since outputs can only be spent once. When nodes accept different transactions in a doublespend, the shared state across nodes becomes inconsistent.*

Remarks:

- Doublespends may occur naturally, e.g., if outputs are co-owned by multiple users who all know the corresponding private key. However, doublespends can be malicious as well – we call these doublespend-attacks: An attacker creates two transactions both using the same input. One transaction would transfer the money to a victim, he other transaction would transfer the money back to the attacker.
- Doublespends can result in an inconsistent state since the validity of transactions depends on the order in which they arrive. If two conflicting transactions are seen by a node, the node considers the first to be valid, see Algorithm 25.12. The second transaction is invalid since it tries to spend an output that is already spent. The order in which transactions are seen, may not be the same for all nodes, hence the inconsistent state.
- If doublespends are not resolved, the shared state diverges. Therefore a conflict resolution mechanism is needed to decide which of the conflicting transactions is to be confirmed (accepted by everybody), to achieve eventual consistency.

Definition 25.14 (Proof-of-Work)**.** *Proof-of-Work (PoW) is a mechanism that allows a party to prove to another party that a certain amount of computational resources has been utilized for a period of time. A function* $\mathcal{F}_d(c, x) \rightarrow \{true, false\}$*, where difficulty* d *is a positive number, while challenge* c *and nonce* x *are usually bit-strings, is called a Proof-of-Work function if it has following properties:*

1. $\mathcal{F}_d(c, x)$ *is fast to compute if* d*,* c*, and* x *are given.*

2. *For fixed parameters d and c, finding x such that $\mathcal{F}_d(c, x) = true$ is computationally difficult but feasible. The difficulty d is used to adjust the time to find such an x.*

Definition 25.15 (Bitcoin PoW function)**.** *The Bitcoin PoW function is given by*

$$\mathcal{F}_d(c, x) \to \text{SHA256}(\text{SHA256}(c|x)) < \frac{2^{224}}{d}.$$

Remarks:

- This function concatenates the challenge c and nonce x, and hashes them twice using SHA256. The output of SHA256 is a cryptographic hash with a numeric value in $\{0, \ldots, 2^{256} - 1\}$ which is compared to a target value $\frac{2^{224}}{d}$, which gets smaller with increasing difficulty.
- SHA256 is a cryptographic hash function with pseudorandom output. No better algorithm is known to find a nonce x such that the function $\mathcal{F}_d(c, x)$ returns true than simply iterating over possible inputs. This is by design to make it difficult to find such an input, but simple to verify the validity once it has been found.

Definition 25.16 (Block)**.** *A block is a data structure used to communicate incremental changes to the local state of a node. A block consists of a list of transactions, a timestamp, a reference to a previous block and a nonce. A block lists some transactions the block creator ("miner") has accepted to its memory pool since the previous block. A node finds and broadcasts a block when it finds a valid nonce for its PoW function.*

Algorithm 25.17 Node Creates (Mines) Block

block $b_t = \{coinbase\_tx\}$
while size(b_t) $\leq$ 1 MB **do**
  Choose transaction t in the *memory pool* that is consistent with b_t and local UTXO set
  Add t to b_t
end while
Nonce $x = 0$, difficulty d, previous block b_{t-1}, timestamp $= t_s$
challenge $c = (merkle(b_t), hash(b_{t-1}), t_s, d)$
repeat
  $x = x + 1$
until $\mathcal{F}_d(c, x) = true$
Gossip block b_t
Update local UTXO set to reflect b_t

Remarks:

- The function $merkle(b_t)$ creates a cryptographic representation of the set of transactions in b_t. It is compact and has a fixed length no matter how large the set is.

- With their reference to a previous block, the blocks build a tree, rooted in the so called *genesis block*. The genesis block's hash is hard-coded in the Bitcoin source code.

- The primary goal for using the PoW mechanism is to adjust the rate at which blocks are found in the network, giving the network time to synchronize on the latest block. Bitcoin sets the difficulty so that globally a block is created about every 10 minutes in expectation.

- Finding a block allows the finder to impose the transactions in its local memory pool to all other nodes. Upon receiving a block, all nodes roll back any local changes since the previous block and apply the new block's transactions.

- Transactions contained in a block are said to be *confirmed* by that block.

Definition 25.18 (Coinbase Transaction)**.** *The first transaction in a block is called the coinbase transaction. The block's miner is rewarded for confirming transactions by allowing it to mint new coins. The coinbase transaction has a dummy input, and the sum of outputs is determined by a fixed subsidy plus the sum of the fees of transactions confirmed in the block.*

Remarks:

- A coinbase transaction is the sole exception to the rule that the sum of inputs must be at least the sum of outputs. New bitcoins enter the system through coinbase transactions.

- The number of bitcoins that are minted by the coinbase transaction and assigned to the miner is determined by a subsidy schedule that is part of the protocol. Initially the subsidy was 50 bitcoins for every block, and it is being halved every 210,000 blocks, or 4 years in expectation. Due to the halving of the value of the coinbase transaction, the total amount of bitcoins in circulation never exceeds 21 million bitcoins.

- It is expected that the cost of performing the PoW to find a block, in terms of energy and infrastructure, is close to the value of the reward the miner receives from the coinbase transaction in the block.

Definition 25.19 (Blockchain)**.** *The longest path from the genesis block (root of the tree) to a (deepest) leaf is called the blockchain. The blockchain acts as a consistent transaction history on which all nodes eventually agree.*

Remarks:

- The path length from the genesis block to block b is the height h_b.

- Only the longest path from the genesis block to a leaf is a valid transaction history, since branches may contradict each other because of doublespends.

- Since only transactions in the longest path are agreed upon, miners have an incentive to append their blocks to the longest chain, thus agreeing on the current state.
- The mining incentives quickly increased the difficulty of the PoW mechanism: initially miners used CPUs to mine blocks, but CPUs were quickly replaced by GPUs, FPGAs and even application specific integrated circuits (ASICs) as bitcoins appreciated. This results in an equilibrium today in which only the most cost efficient miners, in terms of hardware supply and electricity, make a profit in expectation.
- If multiple blocks are mined more or less concurrently, the system is said to have *forked*. Forks happen naturally because mining is a distributed random process and two new blocks may be found at roughly the same time.

Algorithm 25.20 Node Receives Block

Receive block b_t
For this node, the current head is block b_{max} at height h_{max}
For this node, b_{max} defines the local UTXO set
From b_t, extract reference to b_{t-1}, and find b_{t-1} in the node's local copy of the blockchain
$h_b = h_{b_{t-1}} + 1$
if $h_b > h_{max}$ and $is\_valid(b_t)$ **then**
    $h_{max} = h_b$
    $b_{max} = b$
    Update UTXO set to reflect transactions in b_t
end if

Remarks:

- Algorithm 25.20 describes how a node updates its local state upon receiving a block. Like Algorithm 25.12, this describes the local policy and may also result in node states diverging, i.e., by accepting different blocks at the same height as current head.
- Unlike extending the current path, switching paths may result in confirmed transactions no longer being confirmed, because the blocks in the new path do not include them. Switching paths is referred to as a *reorg*.

Theorem 25.21. *Forks are eventually resolved and all nodes eventually agree on which is the longest blockchain. The system therefore guarantees eventual consistency.*

Proof. In order for the fork to continue to exist, pairs of blocks need to be found in close succession, extending distinct branches, otherwise the nodes on the shorter branch would switch to the longer one. The probability of branches being extended almost simultaneously decreases exponentially with the length of the fork, hence there will eventually be a time when only one branch is being extended, becoming the longest branch. □

Definition 25.22 (Consensus Rules)**.** *The is_valid function in algorithm 25.20 represents the consensus rules of Bitcoin. All nodes will converge on the same shared state if and only if all nodes agree on this function.*

Remarks:

- If nodes have different implementations of the *is_valid* function, some nodes will reject blocks that other nodes will accept. This is called a *hard fork*, which is different than a regular fork. A regular fork happens because different nodes see different blocks that are mined at around the same time. Hard forks happen because the rules of Bitcoin itself have changed.

- Getting all nodes to change their implementation of *is_valid* together, at the same time, so that new features can be added to the Bitcoin system, is difficult, as there is no centralized authority to coordinate such an upgrade.

- In Bitcoin, hard forks are distinguished from soft forks:

Definition 25.23 (Hard/Soft Fork)**.** *If the set of valid transactions is expanded, we have a hard fork. If the set of valid transactions is reduced, we have a soft fork.*

Remarks:

- As all nodes cannot upgrade at the same time, miners can create blocks that have more restrictive *is_valid* rules and older nodes will still accept them as they accept broader rules. This way, rules can still be changed without having to upgrade all nodes at the same time. Miners, on the other hand, have to upgrade almost at the same time.

25.3 Layer 2

Definition 25.24 (Smart Contract)**.** *A smart contract is an agreement between two or more parties, encoded in such a way that the correct execution is guaranteed by the blockchain.*

Remarks:

- Contracts allow business logic to be encoded in Bitcoin transactions which mutually guarantee that an agreed upon action is performed. The blockchain acts as conflict mediator, should a party fail to honor an agreement.

- The use of scripts as spending conditions for outputs enables smart contracts. Scripts, together with some additional features such as timelocks, allow encoding complex conditions, specifying who may spend the funds associated with an output and when.

Definition 25.25 (Timelock)**.** *Bitcoin provides a mechanism to make transactions invalid until some time in the future:* ***timelocks****. A transaction may specify a locktime: the earliest time, expressed in either a Unix timestamp or a blockchain height, at which it may be included in a block and therefore be confirmed.*

Remarks:

- Transactions with a timelock are not released into the network until the timelock expires. It is the responsibility of the node receiving the transaction to store it locally until the timelock expires and then release it into the network.

- Transactions (and blocks) with future timelocks are invalid. Upon receiving invalid transactions or blocks, nodes discard them immediately and do not forward them to their neighbors.

- Timelocks can be used to replace or supersede transactions: a timelocked transaction t_1 can be replaced by another transaction t_0, spending some of the same outputs, if the replacing transaction t_0 has an earlier timelock and can be broadcast in the network before the replaced transaction t_1 becomes valid.

Definition 25.26 (Singlesig and Multisig Outputs)**.** *When an output can be claimed by providing a single signature it is called a* ***singlesig output****. In contrast the script of* ***multisig outputs*** *specifies a set of m public keys and requires k-of-m (with $k \leq m$) valid signatures from distinct matching public keys from that set in order to be valid.*

Remarks:

- Many smart contracts begin with the creation of a 2-of-2 multisig output, requiring a signature from both parties. Once the transaction creating the multisig output is confirmed in the blockchain, both parties are guaranteed that the funds of that output cannot be spent unilaterally.

Algorithm 25.27 Parties A and B create a 2-of-2 multisig output o

1: B sends a list I_B of inputs with c_B coins to A
2: A selects its own inputs I_A with c_A coins
3: A creates transaction $t_s\{[I_A, I_B], [o = c_A + c_B \rightarrow (A, B)]\}$
4: A creates timelocked transaction $t_r\{[o], [c_A \rightarrow A, c_B \rightarrow B]\}$ and signs it
5: A sends t_s and t_r to B
6: B signs both t_s and t_r and sends them to A
7: A signs t_s and broadcasts it to the Bitcoin network

Remarks:

- t_s is called a *setup transaction* and is used to lock in funds into a shared account. If t_s is signed and broadcast immediately, one of the parties could not collaborate to spend the multisig output, and the funds become unspendable. To avoid a situation where the funds cannot be spent, the protocol also creates a timelocked *refund transaction* t_r which guarantees that, should the funds not be spent before the timelock expires, the funds are returned to the respective party. At no point in time one of the parties holds a fully signed setup transaction without the other party holding a fully signed refund transaction, guaranteeing that funds are eventually returned.

- Both transactions require the signature of both parties. The setup transaction has two inputs from A and B respectively which require individual signatures. The refund transaction requires both signatures because of the a 2-of-2 multisig input.

Algorithm 25.28 Simple Micropayment Channel from s to r with capacity c

$c_s = c$, $c_r = 0$
s and r use Algorithm 25.27 to set up output o with value c from s
Create settlement transaction $t_f\{[o], [c_s \to s, c_r \to r]\}$
while channel open **and** $c_r < c$ **do**
  In exchange for good with value δ
  $c_r = c_r + \delta$
  $c_s = c_s - \delta$
  Update t_f with outputs $[c_r \to r, c_s \to s]$
  s signs and sends t_f to r
end while
r signs last t_f and broadcasts it

Remarks:

- Algorithm 25.28 implements a Simple Micropayment Channel, a smart contract that is used for rapidly adjusting micropayments from a spender to a recipient. Only two transactions are ever broadcast and inserted into the blockchain: the setup transaction t_s and the last settlement transaction t_f. There may have been any number of updates to the settlement transaction, transferring ever more of the shared output to the recipient.

- The number of bitcoins c used to fund the channel is also the maximum total that may be transferred over the simple micropayment channel.

- At any time the recipient R is guaranteed to eventually receive the bitcoins, since she holds a fully signed settlement transaction, while the spender only has partially signed ones.

- The simple micropayment channel is intrinsically unidirectional. Since the recipient may choose any of the settlement transactions in the protocol, she will use the one with maximum payout for her. If we were to transfer bitcoins back, we would be reducing the amount paid out to the recipient, hence she would choose not to broadcast that transaction.

25.4 Weak Consistency

Eventual consistency is only one form of weak consistency. A number of different tradeoffs between partition tolerance and consistency exist in literature.

Definition 25.29 (Monotonic Read Consistency). *If a node u has seen a particular value of an object, any subsequent accesses of u will never return any older values.*

Remarks:

- Users are annoyed if they receive a notification about a comment on an online social network, but are unable to reply because the web interface does not show the same notification yet. In this case the notification acts as the first read operation, while looking up the comment on the web interface is the second read operation.

Definition 25.30 (Monotonic Write Consistency). *A write operation by a node on a data item is completed before any successive write operation by the same node (i.e., system guarantees to serialize writes by the same node).*

Remarks:

- The ATM must replay all operations in order, otherwise it might happen that an earlier operation overwrites the result of a later operation, resulting in an inconsistent final state.

Definition 25.31 (Read-Your-Write Consistency). *After a node u has updated a data item, any later reads from node u will never see an older value.*

Definition 25.32 (Causal Relation). *The following pairs of operations are said to be causally related:*

- *Two writes by the same node to different variables.*
- *A read followed by a write of the same node.*
- *A read that returns the value of a write from any node.*
- *Two operations that are transitively related according to the above conditions.*

Remarks:

- The first rule ensures that writes by a single node are seen in the same order. For example if a node writes a value in one variable and then signals that it has written the value by writing in another variable. Another node could then read the signalling variable but still read the old value from the first variable, if the two writes were not causally related.

Definition 25.33 (Causal Consistency)**.** *A system provides causal consistency if operations that potentially are causally related are seen by every node of the system in the same order. Concurrent writes are not causally related, and may be seen in different orders by different nodes.*

Chapter Notes

The CAP theorem was first introduced by Fox and Brewer [FB99], although it is commonly attributed to a talk by Eric Brewer [Bre00]. It was later proven by Gilbert and Lynch [GL02] for the asynchronous model. Gilbert and Lynch also showed how to relax the consistency requirement in a partially synchronous system to achieve availability and partition tolerance.

Bitcoin was introduced in 2008 by Satoshi Nakamoto [Nak08]. Nakamoto is thought to be a pseudonym used by either a single person or a group of people; it is still unknown who invented Bitcoin, giving rise to speculation and conspiracy theories. Among the plausible theories are noted cryptographers Nick Szabo [Big13] and Hal Finney [Gre14]. The first Bitcoin client was published shortly after the paper and the first block was mined on January 3, 2009. The genesis block contained the headline of the release date's The Times issue "*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*", which serves as proof that the genesis block has been indeed mined on that date, and that no one had mined before that date. The quote in the genesis block is also thought to be an ideological hint: Bitcoin was created in a climate of financial crisis, induced by rampant manipulation by the banking sector, and Bitcoin quickly grew in popularity in anarchic and libertarian circles. The original client is nowadays maintained by a group of independent core developers and remains the most used client in the Bitcoin network.

Central to Bitcoin is the resolution of conflicts due to doublespends, which is solved by waiting for transactions to be included in the blockchain. This however introduces large delays for the confirmation of payments which are undesirable in some scenarios in which an immediate confirmation is required. Karame et al. [KAC12] show that accepting unconfirmed transactions leads to a non-negligible probability of being defrauded as a result of a doublespending attack. This is facilitated by *information eclipsing* [DW13], i.e., that nodes do not forward conflicting transactions, hence the victim does not see both transactions of the doublespend. Bamert et al. [BDE$^+$13] showed that the odds of detecting a doublespending attack in real-time can be improved by connecting to a large sample of nodes and tracing the propagation of transactions in the network.

Bitcoin does not scale very well due to its reliance on confirmations in the blockchain. A copy of the entire transaction history is stored on every node in order to bootstrap joining nodes, which have to reconstruct the transaction history from the genesis block. Simple micropayment channels were introduced by Hearn and Spilman [HS12] and may be used to bundle multiple transfers between two parties but they are limited to transferring the funds locked into the channel once. Duplex Micropayment Channels [DW15] and the Lightning Network [PD15] were the first suggestions for bidirectional micropayment channels in which the funds can be transferred back and forth an arbitrary number of times, greatly increasing the flexibility of Bitcoin transfers and enabling a number of features, such as micropayments and routing payments between any two endpoints.

This chapter was written in collaboration with Christian Decker.

Bibliography

[BDE+13] Tobias Bamert, Christian Decker, Lennart Elsen, Samuel Welten, and Roger Wattenhofer. Have a snack, pay with bitcoin. In *IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, 2013.

[Big13] John Biggs. Who is the real satoshi nakamoto? one researcher may have found the answer. http://on.tcrn.ch/l/R0vA, 2013.

[Bre00] Eric A. Brewer. Towards robust distributed systems. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 2000.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, September 2013.

[DW15] Christian Decker and Roger Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2015.

[FB99] Armando Fox and Eric Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems*. IEEE, 1999.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 2002.

[Gre14] Andy Greenberg. Nakamoto's neighbor: My hunt for bitcoin's creator led to a paralyzed crypto genius. http://onforb.es/1rvyecq, 2014.

[HS12] Mike Hearn and Jeremy Spilman. Contract: Rapidly adjusting micro-payments. https://en.bitcoin.it/wiki/Contract, 2012. Last accessed on November 11, 2015.

[KAC12] G.O. Karame, E. Androulaki, and S. Capkun. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In *Conference on Computer and Communication Security (CCS)*, 2012.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[PD15] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network. 2015.