Mining, Nodes, and Blocks

A **node** is a device/program that communicates with the Ethereum network. Nodes are also known as **clients**. Software that can act as an Ethereum node include Parity and Go-ethereum (`geth`). This software mostly, if not always, also provides **wallet** functionality (software that allows users to perform transactions on the blockchain). Some node software also allows other programs to indirectly interact with the blockchain; this is provided through some remote procedure call (RPC) facility. RPC is usually implemented in the form of web APIs (HTTP) or Unix sockets.

In general, we can divide node software into two types: full nodes and light(weight) nodes. **Full nodes** verify block that is broadcast onto the network. That is, they ensure that the transactions contained in the blocks (and the blocks themselves) follow the rules defined in the Ethereum specifications. They maintain the current state of the network (as defined according to the Ethereum specifications). Transactions and blocks that do not follow the rules are not used to determine the current state of the Ethereum network. For example, if A tries to send 100 ether to B but A has 0 ethers and a block includes this transaction, the full nodes will realize this does not follow the rules of Ethereum and reject that block as invalid. In particular, the execution of **smart contracts** is an example of a transaction. Whenever a smart contract is used in a transaction (e.g., sending ERC-20 tokens), all full nodes will have to run all the instructions to ensure that they arrive at the correct, agreed-upon next state of the blockchain.

There are multiple ways of arriving at the same state. For example, if A had 101 ether and gave a hundred of them to B in one transaction paying 1 ether for gas, the end result would be the same as if A sent 100 transactions of 1 ether each to B paying 0.01 ether per transaction (ignoring who received the transaction fees). To know if B is now allowed to send 100 ether, it is sufficient to know what B's current balance is. Full nodes that preserve the entire history of transactions are known as **full archiving nodes**. These must exist on the network for it to be healthy. Nodes may also opt to discard old data; if B wants to send 100 ether to C, it doesn't matter *how* the ether was obtained, only that B's account contains 100 ether. **Light nodes**, in contrast, do not verify every block or transaction and may not have a copy of the current blockchain state. They rely on full nodes to provide them with missing details (or simply lack particular functionality). The advantage of light nodes is that they can get up and running much more quickly, can run on more computationally/memory constrained devices, and don’t eat up nearly as much storage. On the downside, there is an element of trust in other nodes (it varies based on client and probabilistic

methods/heuristics can be used to reduce risk). Some full clients include features to have faster syncs (e.g., Parity's warp sync).

As part of the Ethereum protocol, new transactions can be added to the blockchain in quantized units known as "blocks"; transactions thus added to the blockchain are said to be **confirmed**. One of the rules that blocks must follow (at present, with proof-of-work) is that, in addition to the transactions to be confirmed, a proof of work is included with the block itself. Generating this proof of work is known as **mining**. Generating this proof of work does not itself require executing the transactions; miners can connect to a full node and get the work to be completed which is only dependent on the *contents* of the block to be added to the blockchain, not the execution of the contents.

Each block represents a snapshot in time. Because there is often short-term disagreement (**soft forks**) as to what the current state of the blockchain is, after a transaction is confirmed in a block, it is often recommended to wait for a few more blocks to be built on top of that block before assuming that the transaction is final. The longest chain is the agreed upon state of the chain. The more blocks that have been added after a particular block, the less likely it is that the block will be "undone". The purpose of choosing the longest chain to add blocks to the blockchain is to arrive at **consensus** -- agreement. The reason why mining is necessary at the moment is so that there is a cost to extend the chain. Without this cost, a malicious account owner M can send a transaction to A in block `n` and then A making a decision based on the transaction having taken place (maybe M bought coffee and now A serves coffee to M). But then M can go back and create a new block `n'` where the transaction did not take place and add a block `n' + 1` so that this becomes the new "official" version of history (since it's longer than the chain that only contained `n`). But if it takes time for M to add blocks -- in the form of computational work that takes time -- M has to do three times as much work to add block `n`, `n'`, and `n' + 1` to the blockchain as adding only `n` (which M can leave to other miners to do). If M were to succeed in this endeavour, M could send the same ether originally sent to A to someone else at a later date, thus spending the same ether more than once. This re-spending of the same ether (or tokens, or whatever) is known as a **double-spend**.

Miners (the people who do the mining), as a reward for their work in preventing agents like M from being duplicitous, are given **block rewards** (currently 3 ETH per block) and **transaction fees** paid by people who wish to have their transactions added to the blockchain.