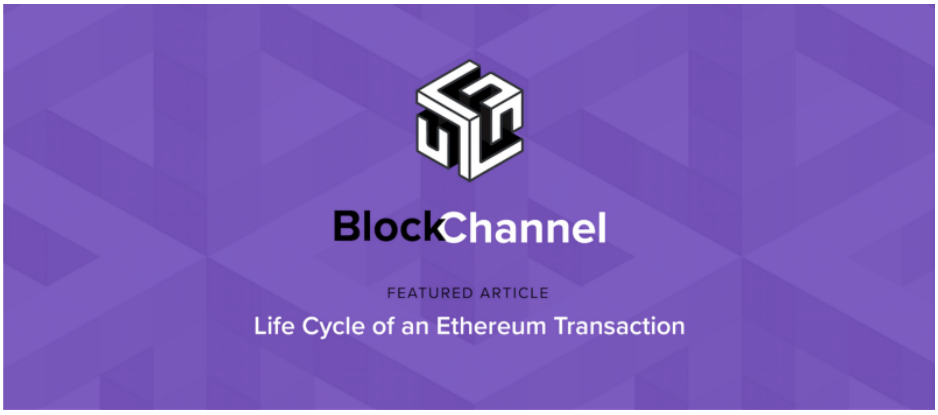
Mahesh Murthy Follow
Techie, Foodie, Traveler, Founder www.zastrin.com
Dec 26, 2017 · 9 min read

Life Cycle of an Ethereum Transaction

Learn How an Ethereum Transaction is Created and Propagated to the Network



Transactions are at the heart of the Ethereum blockchain (or any blockchain for that matter). When you interact with the Ethereum blockchain, you are executing transactions and updating it's state. Have you ever wondered what exactly happens when you execute a transaction in Ethereum? Let's try to understand it by looking at an example transaction. In this post, we will cover the following.

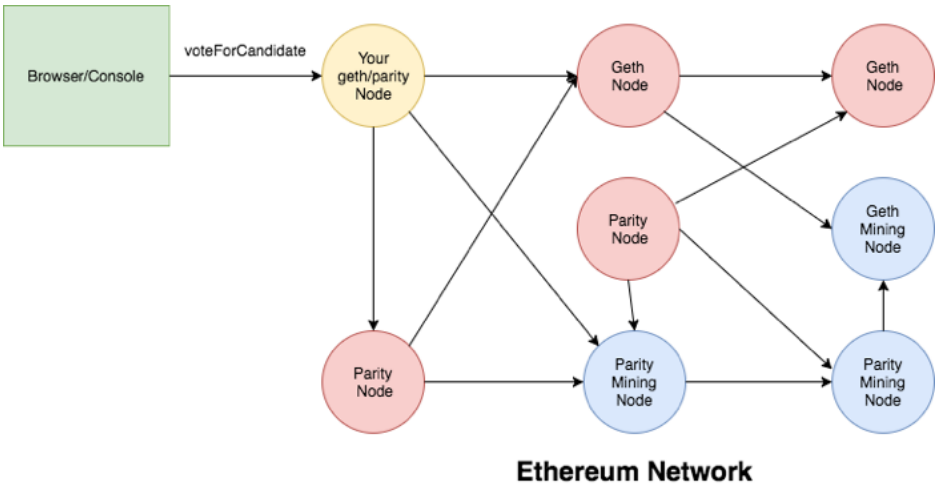
1. An end to end traversal of an Ethereum transaction starting from your browser/console to the Ethereum network and back to your browser/console
2. Understand how transactions work when you use a plugin such as Metamask or Myetherwallet instead of running your own node
3. What to do if you are too paranoid and don't trust any plugins to execute your transaction?

This post assumes that you have a basic understanding of Ethereum and it's components such as accounts, gas and contracts. Here is a good explanation of these concepts. If you are a developer new to Ethereum, you might find this helpful. You can also learn to build a simple dapp here. This post will make more sense if you have executed few transactions yourself. An example of a transaction is you sending some ether to another person or a contract. Another example is if you have interacted with a dapp. For example, if you go here and buy some tokens, that would be a transaction. If you vote for a candidate, that would be another transaction.

1. End to end overview of an Ethereum transaction

Let's take the following contract call as an example and traverse through the entire flow of how this function call/transaction gets executed and gets permanently stored on the blockchain. You can find the entire contract here. At a high level, it's a voting contract where you initialize a few candidates contesting in an election and anyone can vote for them. The votes will be recorded on the blockchain.

```
Voting.deployed().then(function(instance) {
  instance.voteForCandidate('Nick', {gas: 140000, from:
web3.eth.accounts[0]}).then(function(r) {
    console.log("Voted successfully!")
  })
})
```



We assume that you have an Ethereum client (Geth or Parity) running locally on your computer which is connected to one of the networks (Testnet or Mainnet) and you have access to the contract address and ABI to execute the transaction.

If you have built a dapp, the above code should look familiar. There is a contract called Voting which has already been deployed on to the blockchain. We instantiate that contract and execute a method called voteForCandidate and pass in the candidate name, gas limit for this transaction and the account that executes this transaction. As the name indicates, this function is used to vote for a candidate and the vote is recorded on the blockchain. Below we will try to deconstruct this call and understand everything that happens when you execute this javascript function.

1. Construct the raw transaction object

The voteForCandidate function call is first converted in to a raw transaction (rawTxn) as shown below. Web3js library is used to build the raw transaction object.

```
txnCount =
web3.eth.getTransactionCount(web3.eth.accounts[0])
var rawTxn = {
    nonce: web3.toHex(txnCount),
    gasPrice: web3.toHex(100000000000),
    gasLimit: web3.toHex(140000),
```

```
    to: '0x633296baebc20f33ac2e1c1b105d7cd1f6a0718b',
    value: web3.toHex(0),
    data:
'0xc7ed014952616d6100000000000000000000000000000000000000000000000000000000000000000'
};
```

Let's try to understand all the fields in the raw transaction object and how they are set.

nonce: Each Ethereum account has a field called nonce to keep track of the total number of transactions that account has executed. Nonce is incremented for every new transaction and this allows the network to know the order in which the transactions need to be executed. Nonce is also used for the replay protection.

gasPrice: Price per unit of gas you are willing to pay for this transaction. If you are executing your transaction on the Mainnet, here is a handy website from ETH Gas Station that recommends what you should set the gas price for your transaction to succeed in a reasonable amount of time. Gas prices are current measured in GWei and range from 0.1->100+Gwei. You will learn more about gas price and it's impact later in this article.

gasLimit: Maximum gas you are willing to pay for this transaction. This value insures that in case of an issue executing your transaction (like infinite loop), you account is not drained of all the funds. Once the transaction is executed, any remaining gas is sent back to your account.

to: The address to which you are directing this function call. This is the contract address (0x633296baebc20f33ac2e1c1b105d7cd1f6a0718b) of the Voting contract in our case.

value: Total Ether you want to send. When we execute `voteForCandidate`, we are not sending any Ether and so the value is 0. If you were executing a transaction to send Ether to another person or a contract, you would set this value.

data: Let's see how this data field is calculated.

You first take the function signature from the ABI `voteForCandidate(bytes32 candidate)` and generate the hash of it.

```
> web3.sha3('voteForCandidate(bytes32 candidate)')

'0xc7ed014922ff9493a686391b70ca0e8bb7e80f91c98a5cd3d285778a
b2e245b3'
```

You take the first 4 bytes of that hash. So, that would be: 0xcc9ab267.

You then take the argument 'Nick' and convert to bytes32 and you get 52616d6100000000000000000000000000000000000000000000000000000000

You combine the two to get the data payload.

2. Sign the Transaction

If you remember, you used web3.eth.accounts[0] to execute the transaction. The Ethereum network needs to know that you actually own that account to make sure someone else doesn't execute this transaction on your behalf. The way to prove this to the network is by signing the transaction using the private key corresponding to that account. The signed transaction looks like this:

```
const privateKey =
Buffer.from('e331b6d69882b4ab4ea581s88e0b6s4039a3de5967d88d
fdcffdd2270c0fd109', 'hex')

const txn = new EthereumTx(rawTxn)
txn.sign(privateKey)
const serializedTxn = txn.serialize()
```

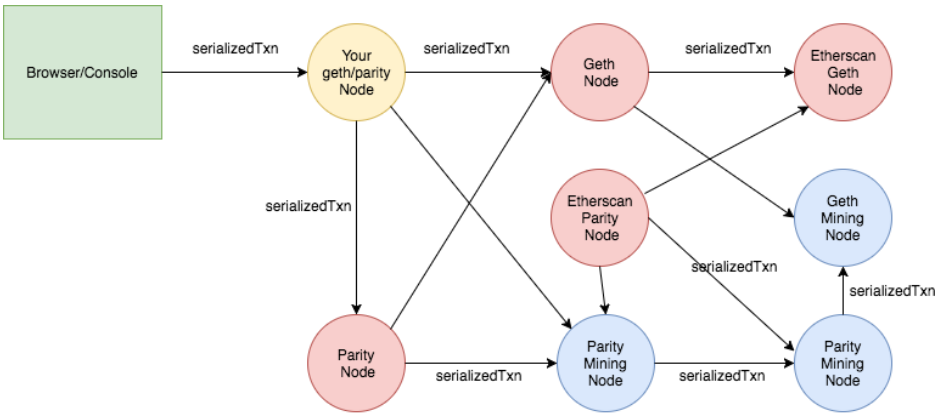
3. Transaction is validated locally

The signed transaction is submitted to your local Ethereum node. You local node will then validate the signed transaction to make sure that it was really signed by this account address.

4. Transaction is broadcast to the network

The signed transaction is broadcast by your geth/parity node to it's peers who in turn broadcast to their peers and so on. Once the transaction is broadcast to the network, your local node also outputs the transaction id which you can use to track the status of your transaction. This transaction id is just the hash of the signed transaction object.

```
transactionId = sha3(serializedTxn)
```



Signed Transaction Propogates to the network

If you execute the transaction on a public Ethereum network, the best way to track the status of your transaction is on etherscan.io. In the picture above, if you notice there are a couple of nodes marked as Etherscan nodes. The folks at Etherscan run a few nodes and they have a nice frontend webapp connected to it. When your transaction is picked up by their nodes, you can see your pending transaction on their website.

One other thing to remember is, not all nodes will accept your transaction. Some of these nodes might have a setting to accept only

transactions with certain minimum gas price. If you have set a gas price lower than that limit, that node will just ignore your transaction.

5. Miner Node accepts the transaction

As you can see in the picture, the Ethereum network has a mix of miner nodes and the non miner nodes. As you probably know, the miners are the ones who do the job of including your transaction in the block. Miners maintain a transaction pool where your transaction gets added to before they start evaluating it.

If you notice the picture above, the miners store all the transactions in the pool sorted by gas price. The higher the gas price, the more likely the transaction is included in the next block. This is the common configuration for a miner node (to optimize for higher pay). However, a

miner can configure her node to sort the transactions however they like (say they want to help the network by mining only low gas transactions).

In the picture above, you see how our voteForCandidate transaction is at the bottom of the miner's pool? Once all the high gas fee transactions have been mined and included in the block, the miners will work our transaction.

One other thing to note is, the miner's pool can hold a finite number of transactions. Let's say there is a hot ICO sale going on or a really popular dapp (like Crypto Kitties) takes off . People submit their transactions with high gas price hoping that the miners pick up their transaction before any other transactions. If higher gas price transactions fill up the pool, your low price transaction is discarded. There is no hope for our candidate Nick to receive any vote for a while. We might even have to rebroadcast the transaction in such cases.

Another trick to bump your transaction up the pool is to resubmit your transaction with a higher gas price but using the same nonce value. That way, when the new transaction is received by the miners, the new higher gas price transaction overwrites your old transaction. If the nonce value is different, it is considered a different transaction (You will end up casting 2 votes to Nick). Here is an excellent article from Jim McDonald explaining this topic in depth.

6. Miner Node finds a valid block and broadcasts to the network

The miner eventually picks our transaction to include in the block along with other transactions. The miner can only pick so many transactions to include in the block because Ethereum has set a block gas limit. i.e, the sum of all the gas limits on the transactions can not exceed the block gas limit. You can see the current gas limit at ethstats.net.

Once the miners select the transactions to include in the block, the transactions are validated and included in a pending block and the Proof of Work begins. One of the miner nodes eventually finds a valid block (by solving the PoW puzzle) and adds that block to the blockchain. Just like the raw transaction was broadcast by your local node which was

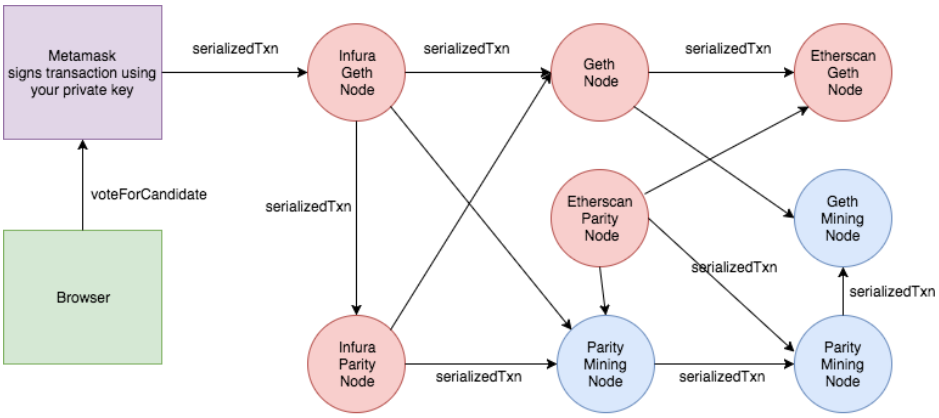
received by all other nodes, the miner node broadcasts this valid block to other nodes.

7. Local Node receives/syncs the new block

Eventually your local node will receive this new block and syncs it's local copy of the blockchain. Upon receiving the new block, the local node executes all the transactions in the block.

If you use truffle to execute your transaction, truffle constantly polls the blockchain for confirmation. Once it sees the transaction is confirmed, it executes the code inside the then() block and prints the console log (per our example).

2. Using Metamask instead of the local node



Using Metamask instead of running a local node

If you install the MetaMask browser plugin, you can manage your accounts in your browser. The keys are stored only on your browser, so you are the only one who has access to your account and the private key. When you execute a transaction in your browser, the plugin takes care of converting your function call in to a raw transaction and sign the transaction using your private key. Metamask runs their own nodes which they use to broadcast your transaction (Behind the scenes

Metamask uses nodes hosted by Infura). This way, you don't even have to run your own Ethereum node.

3. Offline signing

What if you are not comfortable using a plugin or if you are worried that your local geth node might be compromised? There is a secure solution to solve that issue.

If you notice, the first 2 steps don't require you to be online at all. If you want to absolutely make sure your transaction isn't tampered with, you can use a computer which is offline to convert the function call in to a raw transaction and use your private key to sign the transaction. You can then copy the signed transaction string and use a computer that is online to broadcast it to the network. There are many services like Etherscan and Infura you can use to broadcast your signed transaction to the network.

Another safe solution is to use a hardware wallet such as Ledger or Trezor. These wallets store your private key and the code to sign the transaction is programmed in to the hardware itself. They need to be connected to the internet only to publish your signed transaction.

Hopefully this gave you a better idea of how Ethereum transactions work. If you find something incorrect in the post, please leave a comment and I will fix it.

Thanks Jim McDonald for proofreading this article.

Self promotion: If you are interested in learning to build Ethereum dapps, check out my site zastrin.com!