Ethereum

Video Description:

1. Brief description.

2. Timestamp.

3. Link to Ethereum White Paper.

4. Social media link.

5. Hand written White paper.

	☐ Introduction to Bitcoin &
	☐ existing concepts.
	☐
	☐ Ethereum
	☐
	☐ Applications
	☐
	☐ Miscellanea & Concerns
	☐
	☐ Conclusion
	☐
	☐
	☐
	☐

	✓ Timestamp.
0.00	☐ Introduction.
	☐
	☐ 1. Introduction to Bitcoin &
	☐ existing concepts.
	☐
	☐ 2. Ethereum
	☐ a. Philosophy
	☐ b. Ethereum Accounts
	☐ c. Messages & Transaction
	☐ i) Transaction
	☐ ii) Messages
	☐ d. Ethereum state transition
	☐ func
	☐

	☐ e. Code execution.
	☐ f. Blockchain & mining
	☐
	☐ 3. Application.
	☐
	☐ 4. Miscellanea & Concerns
	☐
	☐ 5. Conclusion.
	☐
	☐
	☐
	☐
	☐
	☐

Intro:

It is very hard to find any crypto-currency enthusiast who is not aware of 'Eth' or any blockchain development enthusiast who is unaware of smart contracts written for Ethereum.

This makes Ethereum very interesting & important, which one needs to dissect in depth.

So, if you are a trader or developer already into the ecosystem & trying to make an entry into the ecosystem you should carefully pay attention to this video.

Welcome to coinfor whitepaper. In today's episode we will carefully dissect & understand Ethereum white paper.

Introduction to Bitcoin & existing concepts.

The concept of digital currency has been around for decades. In 1980's & 1990's e-cash protocols were introduced but failed. due to its reliance on centralized intermediary.

In 1998, computational puzzle & decentralized consensus was introduced. Later in 2005 concept of reusable proof on concept was introduced. But they all had some shortcomings.

In 2009, Satoshi Nakamoto introduced a working example of digital currency combining public key cryptography with consensus algorithm - called Proof of Work.

Bitcoin solved many existing problems. but had limitations too.

Ethereum.

The intent of creating Ethereum was to build an alternate protocol for decentralized applications

Ethereum provides the base for creating decentralized applications with particular emphasis on rapid development time, security for small & rarely used applications & ability to interact with different applications.

Ethereum is able to achieve all of this through an abstract foundation layer: A blockchain with a built in turing complete programming language for writing smart contracts.

Philosophy

Ethereum follows 5 design principles:

Simplicity. Universality. Modularity. Agility & non discrimination & non censorship.

The philosophy of Ethereum is the protocol should be as simple as possible, even at the cost of some data storage & time inefficiency

Any average programmer should be able to follow the & implement the entire specification. Any optimization should not be included if it introduces complexity, unless it provides substantial benefit.

Universality is a fundamental part of Ethereum philosophy. Instead of providing features, Ethereum provides a turing complete scripting language which can be used to define any smart contract. Want to create your own currency - you can. Want to setup a full scale daemon sure you can.

The parts of Ethereum protocol should be designed to be as modular and seperable as possible. The goal is even though a feature is being used by Ethereum, its still usable by other protocols.

Agility: Although the details of Ethereum are not set in stone. the foundation is very judicious on making modifications. But if they find any process can significantly improve the protocol, they will pounce at the opportunity.

Finally, the 5th philosophy of Ethereum is non discrimination & non censorship. Ethereum will only regulate direct harm & not undesirable applns. For example, a programmer can even run infinite loop as long as he is ready to pay the gas fees.

Ethereum Accounts:

In the next section let us to understand what are Ethereum accounts.

In ethereum, the state is made up of objects called "accounts" with each account having 20-byte address.

Ethereum account contains four fields:

1) Nonce
2) Ether Balance
3) Contract code
4) Storage

Think of nonce as a counter which is used to make sure each tx can be processed only once.

Ether balance denotes the current balance of an account.

Contract code is the execution logic of a smart contract.

The last field is: Account storage which is empty by default.

Broadly speaking, there are two types of accounts:

a. Externally owned accounts — Controlled by private keys
b. Contract accounts. — Controlled by contract code.

In an externally owned account, messages are sent by creating & signing a tx.

In a contract code, everytime a msg is recieved, the code is activated allowing it to read & write to internal storage, send other messages or create contract in return.

The main fuel which powers all these tx is Eth, which is used to pay tx fees. This tx fees is called 'gas' which needs to be paid everytime there is a tx.

Messages & Transaction.

The term 'tx' refers to a signed data packet that stores a message to be sent from an externally owned account. Tx contains:

1) Receiver of msg.
2) Signer which is sender.
3) Amount of Eth to be transferred from sender to receiver.
4) Optional data field.
5) STARTGAS - represents the max number of computational step the tx execution is allowed to take.
6) GASPRICE represents the fee sender will pay for each step.

Messages

Messages are virtual objects that exist in Ethereum execution environment. Contracts have the ability to send messages to other contracts.

A message consists of following fields:

1) The sender of message.
2) The recipient of message.
3) The amount of Eth to be transferred.
4) Optional data field.
5) STARTGAS value.

Essentially tx & messages are very similar in Ethereum except that msg are produced by a contract & not external actors.

Let's say for eg: A contract code is being executed. Whenever there is an 'CALL' Opcode the contract produces & executes a msg.

This leads the msg to the recievers for executing the code.

Thus. contracts have a relationship with other contract, very similar to external actors.

Also. gas allowance assigned by a tx applies to total gas consumed.

For eg: If total gas is 1000 Eth.

A sends tx to B with 1000 Eth.
B consumes 600 Eth sends msg to C with 400 Eth
C consumes 300 Eth for execution.
Now B can spend on 100 Eth before running out of gas

Ethereum State Transition Function.

Let us understand what Ethereum .... is.

If you have a function say: Apply

Apply (S, Tx)

The func.n S' can be defined as:

1. Tx is well formed. (Values, Signature, nonce) return error. Valid

2. Calculate tx fees STARTGAS * GAS PRICE

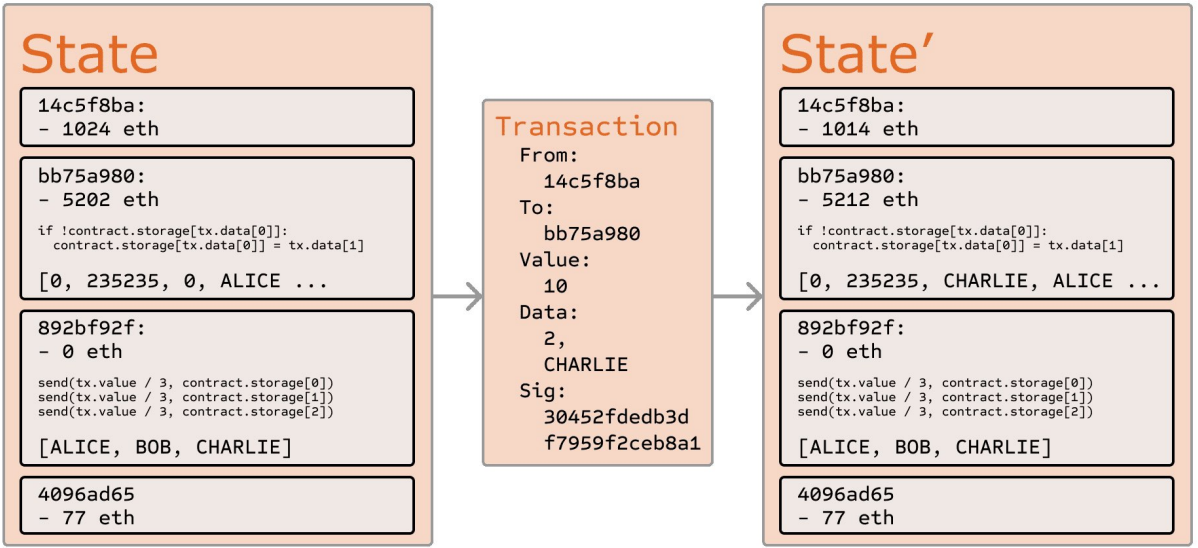
   You subtract the value from sender. Return error if not enough balance.

3. Initialize GAS = STARTGAS to pay for bytes in tx.

4. Transfer the tx to receiver.
If receiver's account doesn't exist, create one; & if its another contract execute it - till its completed or it runs out of gas.

5. If the value transferred failed because Sender did not have enough money, or code execution ran out of gas, the State changes are reverted except payment of fees - which goes to miners account.

6. Refund remaining fee to sender and fees paid to miner.


State
14c5f8ba:
- 1024 eth
bb75a980:
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ...
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65
- 77 eth
Transaction
From:
14c5f8ba
To:
bb75a980
Value:
10
Data:
2,
CHARLIE
Sig:
30452fdedb3d
f7959f2ceb8a1
State'
14c5f8ba:
- 1014 eth
bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ...
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65
- 77 eth

Code Execution.

The code in Ethereum is written as low-level byte code called Ethereum virtual machine code.

The code is a series of bytes & each byte represents an operation. The execution of this byte code is is an infinite loop with a program counter starting at 0.

This loop keeps executing till it encounters STOP or ERROR or RETURN instruction. The operations have access to three types of spaces to store data:

a. Stack : Last In First out containes where values can be pushed or popped.

b. Memory : Infinite expandable byte array.

c. Storage : This is for long term storage unlike stack & memory which will reset after computation ends.

The formal execution code of Ethereum is very simple.
It can be defined by a tuple :

(blockstate, transaction, message code, memory, stack, pc, gas)

For eg: ADD operation pops two items off the stack, pushes their sum into stack, reduces qos by 1, increments program counter by 1. SStore pops two items off the stack inserts second item into contract storage at index specified by first item.

Blockchain & Mining.

Ethereum draws inspiration from bitcoin and thus has many similarities. However it also has some key diff. A major diff bw Bitcoin & Ethereum architecture is, unlike Bitcoin which contains a copy of tx list, Ethereum also contains both copy of tx list & most recent state.

Apart from that, Block numbr & difficulty is also stored in the Block.

The basic block validation in Ethereum works as follows:

1. Check previous reference block if its valid or not.

2. Check timestamp. Should be greater than previous block & less than 15 minutes into the future.

3. Check if block number, tx root, uncle root, gas limit are valid.

4. Check if proof of work on block is valid.

Execution logic:

1 Let S[0] be the state at the end of the previous block.
2 Let TX be the block's transaction list, with n transactions. For all i in 0...n-1, set S[i+1] = APPLY(S[i],TX[i]). If any application returns an error, or if the total gas consumed in the block up until this point exceeds the GASLIMIT, return an error.
3 Let S_FINAL be S[n], but adding the block reward paid to the miner.
4 Check if the Merkle tree root of the state S_FINAL is equal to the final state root provided in the block header. If it is, the block is valid; otherwise, it is not valid.

On first glance this might look very inefficient. But in reality the efficiency can be compared to that of Bitcoin.

The reason why it is efficient is the state is stored in tree like structure. Only a small part of tree is changed & therefore some data can be referenced by pointers.

This is accomplished by using a special tree called 'patricia tree' & modification of merkle tree allowing not just modification but also insert & delete.

Applications:

In general there are 3 category of applications built on top of Ethereum.

1. Financial category.
   - Sub currency. Financial derivatives. Saving wallets etc.

2. Semi Financial.
   - Self enforcing bounties for solutions to computational problems.

3. Non Financial.
   - Online voting. Decentralized governance.

Miscellanea & Concerns.

GHOST

FEES

Computation & Turing completeness

Currency & Issuance

Mining Centralization

Scalability

Conclusion

Ethereum protocol was originally conceived as upgraded version of crypto currency with features like escrow. withdrawal limits. gambling markets etc.

Ethereum does not support appln directly. but provides support for Turing complete language: which opens up the arena to so many appln.

The best part is Ethereum is not just limited to financial appln but many non financial appln. too.