NFT

What is NFT

- Fungible means ersetztbar.
- Unique token on a public blockchain
- Guarantees that a digital asset is unique and not interchangeable
- Can be any digital data that can be hashed (only hash is stored on-chain)
- With NFT: proof of ownership (you can copy the digital data, but the ownership remains)

Difference fungible/non-fugible

Fungible assets are the assets that you can swap with another similar entity. For example, you can swap currency or shares with similar values. You can exchange a one-dollar bill with any other one-dollar bill as all of them represent same value. On the other hand, non-fungible assets are the opposite and cannot be swapped for one another. For example, a house could be easily considered a non-fungible asset as it would have some unique properties. When it comes to the crypto world, representation of assets in the digital form would definitely have to consider the aspects of fungibility and non-fungibility.

Examples

Popular NFT collection is CryptoPunks. Owner is stored in the Ethereum blockchain and can be traded decentralized.

- Collectible media (football, basketball players)
- Jack Dorsey sold first twitter post
- Tickets
- NFT items in games, e.g., CS:GO skins
- Artists sell music as NFT

Fan Token

Fan Tokens are not NFTs!
They are Fungible Tokens, so replaceable.

Ethereum

ERC-20

Token contract keeps track of fungible tokens. Can be used as vault for NFTs. The ERC-20 token standard supported the implementation of a standard API or Application Programming Interface for tokens in smart contracts. So, ERC20 serves as a standard protocol for the Ethereum blockchain and enables users to share, exchanging or transfer tokens.

Code Functions

Mandatory

```
#How many tokens are in circulation. (read)
function totalSupply() public view returns
       (uint256)

#How many tokens has the address. (read)
function balanceOf(address _owner) public view
       returns (uint256 balance)

function transfer(address _to, uint256 _value)
       public returns (bool success)
function transferFrom(address _from, address _to,
       uint256 _value) public returns (bool
       success)
function approve(address _spender, uint256 _value)
       public returns (bool success)
function allowance(address _owner, address
       _spender) 9public view returns (uint256
       remaining)
```

Optional (But recommended)

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
```

Events

```
#Triggered by transfer. Smart Contract cannot
       react to Event. Client outside of contract
       needed.
event Transfer(address indexed _from, address
       indexed _to,uint256 _value)
```

```
#Needed if you want to give someone else
       permission to transfer.
event Approval(address indexed _owner, address
       indexed _spender, uint256 _value)
```

Statements

```
#Is used to check for code that should never be
       false. Failing assertion probably means
       that there is a bug. Uses up all the
       remaining gas and reverts all the changes
       made.
assert()

#is used to validate inputs and conditions before
       execution. Reverts back all the changes
       made to the contract but does refund all
       the remaining gas fees we offered to pay.
require()

#is used abort execution and revert state changes
revert()
```

Implementation

OpenZeppelin - many default contracts, very good source.

Dividends

Loop over accounts does not work. TotalDrop always increasing. Every account knows if bonus payed out. Call updateAccount() on every transfer. User claims bonus.

```
function claimBonus() public payable {
  Account storage account =
        updateAccount(msg.sender);
  uint256 sendValue = account.bonusWei;
  account.bonusWei = 0;
  msg.sender.transfer(sendValue);
}
uint256 public totalDrop = 0;
uint256 public rounding = 0;
struct Account {
  uint256 lastAirdropWei;
  uint256 bonusWei;
  uint256 valueToken;
}
mapping(address => Account) public accounts;

function() public payable {
  uint256 value = msg.value + rounding;
  rounding = value % totalSupply;
  uint256 weiPerToken = (value - rounding) /
        totalSupply;
  totalDrop += weiPerToken;
}

function updateAccount(address _addr) internal {
  Account storage account = accounts[_addr];
  uint256 weiPerToken = totalDrop -
        account.lastAirdropWei;
  if(weiPerToken != 0) {
    account.bonusWei += weiPerToken *
          account.valueToken;
    account.lastAirdropWei = totalDrop;
  }
}
```

Considerations

Random Numbers

There is no random number in Ethereum, because every EVM (Node) needs to come to the same result.

Alternative

1. Random Numbers from Oracle (External source)
2. Commitment schemes

Commitment Scheme - Conflip

1. Alice flips coin, adds it to a random number
   - e.g.: tail#randomnumber1234
   - hashes it
   - commitment = sha256(tail#randomnumber1234)
2. Alice sends the commitment to Bob and tells bob to flip the coin.
3. Bob flips coin and sends head to Alice.
4. Alice reveals the random number, Bob can verify that the commitment was tail.
5. Both same, Alice wins, both different, Bob wins.
6. Alice: tail, Bob: head → Bob wins.

Step 4) here you can try to cheat! Alice knows the result before Bob, so she can just not send the number to Bob. Therefore, add a amount to the commitment, so Alice pays before the commitment and loses the amount if she does not send the number.

Blockhash

Only up to 256 blockhashes from the past can be accessed. Deduct / add tokens / ETH from the past address. Miner can influence the random value in a sense.

```
function transfer(address _to, uint256 _value)
      public returns (bool) {
  //since this is a lucky coin, the value
        transferred is not what you expect
  luckyTransfer();
  previousTransferBlockNr = block.number;
  previousTransferAddress = msg.sender;

  uint256 val -= potIncrease;
  pot += potIncrease;
}
function luckyTransfer() private {
  if(block.number != previousTransferBlockNr &&
        (block.number - previousTransferBlockNr)
        < 256 ) {
    uint256 rnd = uint256(
      keccak256(
        block.blockhash(previousTransferBlockNr)
      )
    );
    if(rnd % 200 == 0) { //.5%
      balances[previousTransferAddress] += pot;
      Emit Transfer(this, previousTransferAddress,
            pot);
      //tokens are from pot, thus no tokens are
            created from thin air
      pot = 0;
      previousTransferBlockNr = 0;
      previousTransferAddress = 0;
    }
  }
}
```

ERC-721

There is either one NFT or there are none. The standards in ERC-721 tokens focus on critical aspects for deciding ownership and approaches for the creation of tokens. The standards also dictate the approaches for destroying and transferring the tokens.

Pros

- Trade items without middleman
- Works 24/7
- Only digital

Cons

- Technical complexity
- NFTs on Ethereum not environment friendly
- Only digital
- Owning tokens does not necessarily mean owning copyright (unless it is explicitly transferred)

Implementation

```
#Balance of NFTs of an owner
function balanceOf(address _owner) external view
       returns (uint256);

#Queries the owner of a specific NFT
function ownerOf(uint256 _tokenId) external view
       returns (address);

#Transfer token of owner, or of approved owner in
       a safe manner (calls onERC721Received)
function safeTransferFrom(address _from, address
       _to, uint256 _tokenId, bytes data) external
       payable;
function safeTransferFrom(address _from, address
       _to, uint256 _tokenId) external payable;

#Make sure you have the right address
function transferFrom(address _from, address _to,
       uint256 _tokenId) external payable;

#Set approve that other address can transfer NFTs
function approve(address _approved, uint256
       _tokenId) external payable;
function setApprovalForAll(address _operator, bool
       _approved) external;

#Check approval
function getApproved(uint256 _tokenId) external
       view returns (address);
function isApprovedForAll(address _owner, address
       _operator) external view returns (bool);

#ERC165 - mandatory! XOR of all function signatures
function supportsInterface(bytes4 interfaceId)
       external view returns (bool);
return interfaceId == type(IERC721).interfaceId ||
       interfaceId ==
       type(IERC721Metadata).interfaceId ||
       interfaceId == type(IERC165).interfaceId;
```

Events

```
event Transfer(address indexed _from, address
       indexed _to, uint256 indexed _tokenId);
event Approval(address indexed _owner, address
       indexed _approved, uint256 indexed
       _tokenId);
event ApprovalForAll(address indexed _owner,
       address indexed _operator, bool _approved);
```

Metadata

```
function name() external view returns (string
       _name);
function symbol() external view returns (string
       _symbol);
function tokenURI(uint256 _tokenId) external view
       returns (string);
```

Optional

Make NFT discoverable

```
function totalSupply() external view returns
       (uint256);
function tokenByIndex(uint256 _index) external
       view returns (uint256);
function tokenOfOwnerByIndex(address _owner,
       uint256 _index) external view returns
       (uint256);
```

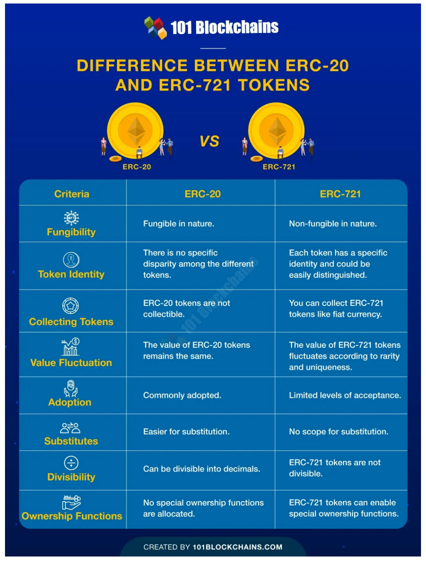
Imports

```
#import interface instead of all functions
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
import "@openzeppelin/IERC721.sol";
import "@openzeppelin/IERC721Metadata.sol";

contract BlChNFT is IERC721 {

}
```

ERC20 vs. ERC721



101 Blockchains

DIFFERENCE BETWEEN ERC-20 AND ERC-721 TOKENS

Criteria	ERC-20	ERC-721
Fungibility	Fungible in nature.	Non-fungible in nature.
Token Identity	There is no specific disparity among the different tokens.	Each token has a specific identity and could be easily distinguished.
Collecting Tokens	ERC-20 tokens are not collectible.	You can collect ERC-721 tokens like fiat currency.
Value Fluctuation	The value of ERC-20 tokens remains the same.	The value of ERC-721 tokens fluctuates according to rarity and uniqueness.
Adoption	Commonly adopted.	Limited levels of acceptance.
Substitutes	Easier for substitution.	No scope for substitution.
Divisibility	Can be divisible into decimals.	ERC-721 tokens are not divisible.
Ownership Functions	No special ownership functions are allocated.	ERC-721 tokens can enable special ownership functions.

CREATED BY 101BLOCKCHAINS.COM

Location Transparency

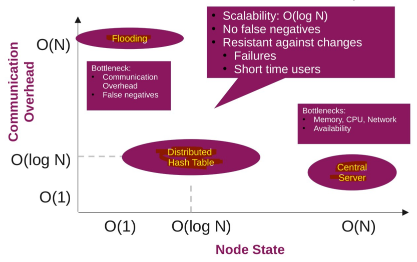
Distributed Management/Retrieval of Data

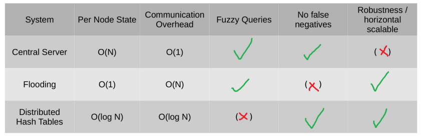
Challenges:

- Location: Where shall the item be stored?
- Location: How can the item be found?
- Scalability: keep the complexity for communication and storage scalable
- Robustness and resilience in case of faults and frequent changes

Strategies for Data Retrieval

Strategies to store and retrieve data items in distributed systems:



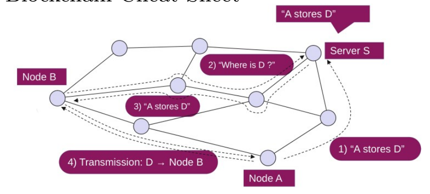


System	Per Node State	Communication Overhead	Fuzzy Queries	No false negatives	Robustness / horizontal scalable
Central Server	O(N)	O(1)	✓	✓	(✗)
Flooding	O(1)	O(N)	✓	(✗)	✓
Distributed Hash Tables	O(log N)	O(log N)	(✗)	✓	✓

Central server

e.g., service registry, reverse proxy - although main use case is load balancing. Simple strategy: Central Server (can be powerful - vertical scaling!). Server stores information about locations:

1. Node A (provider) tells server that it stores item D
2. Node B (requester) asks server S for the location of D
3. Server S tells B that node A stores item D
4. Node B requests item D from node A



Advantages

- Search complexity of O(1) - "just ask the server"
- Complex and fuzzy queries are possible
- Simple and fast

Problems

- No Scalability
  - O(N) node state in server
  - O(N) network and system load of server
- Single point of failure or attack
- (Single) central server not suitable for systems with massive numbers of users

Flooding search

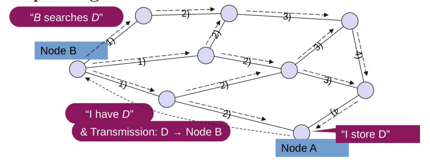
e.g., layer 2 broadcasting, wireless mesh networks, Bitcoin Fully-distributed Approach and ppposite approach of central server. No information on location of a content.
Retrieval of data:

- No routing information for content
- Necessity to ask as much systems as possible / necessary
- No guarantee to reach all nodes
- Flooding: high traffic load on network, scalability issues

How it works

There is an search fee to prevent spamming.

1. Node B (requester) asks neighboring nodes for item D
2. Nodes forward request to further nodes (breadth-first search / flooding)
3. Node A (provider of item D) sends D to requesting node B



For Bitcoin

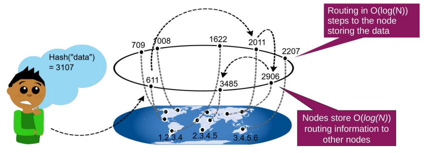
Bitcoin has all data in the block, so it searches itself.

Distributed indexing / Hash Tables

Tor, Bittorrent, IPFS, Apache Cassandra, Dynamo, Atek.

Goal is scalable complexity for:

- Communication effort: O(log(N)) hops
- Node state: O(log(N)) routing entries


Routing in O(log(N)) steps to the node storing the data
Hash("data") = 3107
709
1008
1622
2011
2207
611
3485
2906
1,2,3,4
2,3,4,5
3,4,5,6
Nodes store O(log(N)) routing information to other nodes

1. Hash data
2. Every node knows log(N) neighbors (for 1000, min. 3)
3. First node asks neighbors
4. Neighbors references next neighbor, and so on.
5. Until node is found which is responsible for data item.

Approach

- Data and nodes are mapped into same address space
- Nodes maintain routing information to other nodes

Problems

- Maintenance of routing information required
- Fuzzy queries not primarily supported

Characteristics

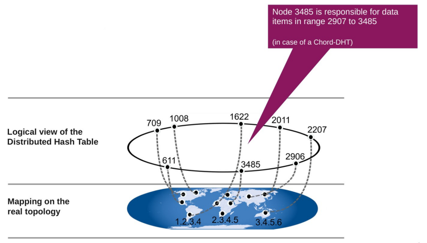
- Reliability / Scalability
- Equal distribution of content among nodes.
- Assignment of responsibilities to new nodes.
- Re-assignment and re-distribution of responsibilities in case of node failure or departure.
- Consistent hashing → nodes responsible for hash value intervals.
- More peers = smaller responsible intervals.
- Hash Table is something different.

Mapping of nodes and data

- Peers and content are addressed using flat identifiers: E.g., Address is public key (256bit) or SHA256 of public key. Content ID = SHA256(content)
- Nodes are responsible for data in certain parts of the address space
- Association of data to nodes may change since nodes may disappear

Storing

- Each node is responsible for part of the value range:
  - Often with redundancy (overlapping of parts)
  - Continuous adaptation
  - Real (underlay) and logical (overlay) topology are uncorrelated

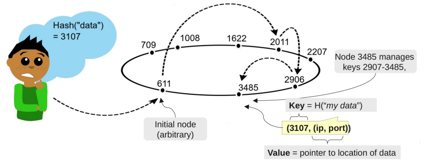

Node 3485 is responsible for data items in range 2907 to 3485
(in case of a Chord-DHT)
Logical view of the Distributed Hash Table
Mapping on the real topology
709
1008
1622
2011
2207
611
3485
2906
1,2,3,4
2,3,4,5
3,4,5,6

Storing / Looking up data

- Store data = first, search for responsible node (Not necessarily known in advance)
- Search data = first, search for responsible node

Routing to Data Item

- Start lookup at arbitrary node of DHT
- Routing to requested data item (key)
- K/V-pair is delivered to requester
- Requester analyzes K/V-tuple (and downloads data from actual location - in case of indirect storage)



Indirect vs Direct:
Indirect is one step more: e.g.: Download from IP, Port. Good for larger files.
Direct is good for smaller files.

Join

1. Calculation of node ID (normally random / or based on PK)
2. New node contacts DHT via arbitrary node (bootstrap node)
3. Lookup of its node ID (routing)
4. Copying of K/V-pairs of hash range (in case of replication)
5. Notify neighbors

Leave

Failure

- Use of redundant K/V pairs (if a node fails)
- Use of redundant / alternative routing paths
- Key-value usually still retrievable if at least one copy remains
- need for keep-alive messages

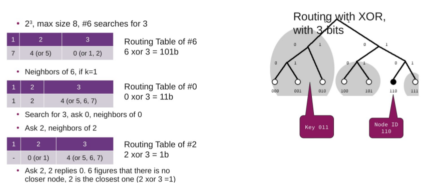
Departure

- Copying of K/V pairs to corresponding nodes
- Friendly unbinding from routing environment

Kademlia

Each Kademlia node and data item has unique identifier. Keys are located on the node whose node ID is closest to the key. Knows neighbors well, further nodes not that much.

Example Search



- 2^3, max size 8, #6 searches for 3

1	2	3
7	4 (or 5)	0 (or 1, 2)

Routing Table of #6
6 xor 3 = 101b

- Neighbors of 6, if k=1

1	2	3
1	2	4 (or 5, 6, 7)

Routing Table of #0
0 xor 3 = 11b

- Search for 3, ask 0, neighbors of 0
- Ask 2, neighbors of 2

1	2	3
-	0 (or 1)	4 (or 5, 6, 7)

Routing Table of #2
2 xor 3 = 1b

- Ask 2, 2 replies 0. 6 figures that there is no closer node, 2 is the closest one (2 xor 3 =1)

Sybil Attacks

- Create large number of identities
- Larger than honest nodes
- Isolate nodes

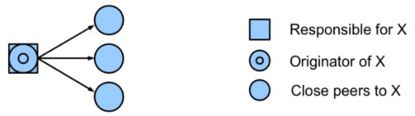
Prevention

- Creation of identities costs money
- Always assume data from other nodes may be missing
- Chain of trust / reputation

Redundancy

Direct Replication

- Originator peer is responsible
- Periodically refresh replicas
- Example: tracker that announces its data
- Problem: Originator offline → replicas disappear. Content has TTL



Indirect Replication

- The closest peer is responsible, originator may go offline vs any close peers are responsible.
- Periodically checks if enough replicas exist.
- Detects if responsibility changes.
- Problem: Requires cooperation between responsible peer and originator
- Problem: Multiple peers may think they are responsible for different versions → eventually solved

Consistency in Replicas

DHTs have weak consistency. If two changes are at the same time, one get overwritten. Requires a coordinator. (Leader Election with tools)

Similarity Search

Levenshtein Distance

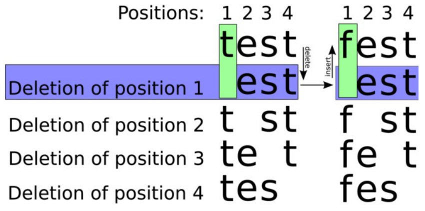
Example d(test,east) = 2 (remove a, insert t).
Main idea: pre-calculate errors:

		T	E	S	T
	0	1	2	3	4
E	1	1	1	2	3
A	2	2	2	2	3
S	3	3	3	2	3
T	4	3	4	3	2

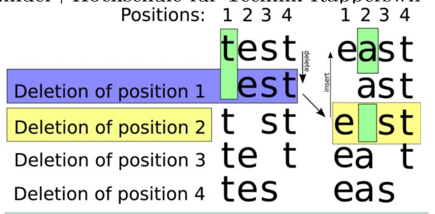
If Letters are the same: Calculate left or top and add one or select diagonal top-left and add 0. Select minimum.
If letters are not the same: select the minimum of either left, top, or diagonal top-left and add 1.

FastSS

FastSS pre-calculates with deletions only. Neighbors for test with ed 2: test, est, tst, tet, tes, ts, tet, te, tes. 11 neighbors → 11 more queries, indexed enlarged by 11 entries.
Example d(test,fest)=1



Example d(test,east)=2



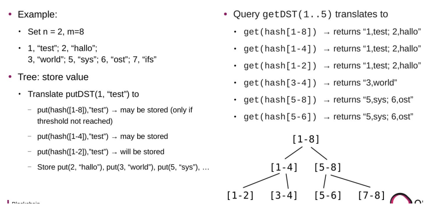
Range Queries

Problem: random insert vs. sequence insert

Solution: Over-DHT

DST: stores data on each level (redundancy) up to a threshold

Example DST



- Example:
  - Set n = 2, m=8
  - 1, "test"; 2, "hallo"; 3, "world"; 5, "sys"; 6, "ost"; 7, "ifs"
- Tree: store value
  - Translate putDST(1, "test") to
    - put(hash([1-8]),"test") → may be stored (only if threshold not reached)
    - put(hash([1-4]),"test") → may be stored
    - put(hash([1-2]),"test") → will be stored
    - Store put(2, "hallo"), put(3, "world"), put(5, "sys"), …
- Query `getDST(1..5)` translates to
  - `get(hash[1-8])` → returns "1,test; 2,hallo"
  - `get(hash[1-4])` → returns "1,test; 2,hallo"
  - `get(hash[1-2])` → returns "1,test; 2,hallo"
  - `get(hash[3-4])` → returns "3,world"
  - `get(hash[5-8])` → returns "5,sys; 6,ost"
  - `get(hash[5-6])` → returns "5,sys; 6,ost"

HTLC