Decentralized Marketplace for Applications

Aakash Jain
Roll Number - 2019101028

Ishaan Shah
Roll Number - 2019111028

Zeeshan Ahmed
Roll Number - 2019111024

Mentor: Sambhav Solanki

Submitted as part of course: Distributing Trust and Blockchains (Monsoon 2020) by Dr. Sujit Prakash Gujar.

***Abstract*—This project focuses on building a secure peer to peer, decentralized app store which supports hosting and purchasing applications. We propose *Dapp Store*, which seeks to become the primary hub of managing your applications on any device, while maintaining the necessary standards for user authentication & licensing, cryptocurrency as means of payment, and data encryption & protection.**

I. Introduction

A. Motivation

Currently, users have little choice in deciding where to get and upload their desired applications for their devices. The front-runners in this industry, Google Play Store and Apple App Store, have managed to achieve a virtual duopoly, giving them authority and control over developers and users alike. Developers have to pay these companies in order to distribute applications on their platform and be chosen as relevant results, and comply by their sometimes limiting policies. Moreover, the applications shown to the users are prone to bias and user advertisements, the privacy policy & terms of use have often come under scrutiny, and these stores are infamous for their practice of collection and harvesting of user data and their search trends [1].

There are presently no viable alternatives to these applications. Marketplaces developed by competing companies such as Samsung Galaxy Store and Huawei App Gallery haven't been very successful and suffer from the same issues as the aforementioned front-runners. With our decentralized, distributed applications, we can provide a fair and equal marketplace that is able to cater to the needs of all the users, while providing genuine results that are truly dependent on users. Moreover, licensing, user data regulation, and protection of the applications will have the same industry-standard as conventional application stores.

B. Who will be the beneficiaries?

The prime beneficiaries of the Dapp Store will be the users, who wish to use a fair and unbiased platform to browse for applications, and employ the latest blockchain developments for app licensing and user authentication. Being self-regulated, the users would be able to contribute towards the application rating system, malicious content detection, and also towards the distributed storage system, by becoming seeders. In addition, independent app developers and small businesses will also benefit from the decentralized nature of the Dapp Store and its distributed manner of hosting. There isn't any notable central authority, corporation, or bank exerting its control and regulations for these applications to abide by, giving greater flexibility, freedom, and profit to the developers.

C. How big will the impact be of the problem?

The Dapp Store has a very large potential for growth. While it is entering a relatively saturated industry, with many application marketplaces present currently, there haven't been any successful endeavors that incorporate blockchain technologies. This peer to peer application doesn't face many significant challenges when scaling up. As with many blockchain, distributed applications, both its functionality and success depend on the nature of involvement of its users. As long as there are enough users engaged in hosting, developing, and purchasing applications, the Dapp Store can hope to be a prominent competitor for Google and Apple as an application marketplace.

II. Need for Blockchain

A. Complexity in case of Absence of Blockchain

The Dapp Store bases its success in the market almost exclusively due to the lack of a proper decentralized application marketplace currently present. In case our application was centralized, it wouldn't be any different from the vast number of app stores that are currently present, and would be competing with both a larger number of similar applications, and also more renowned and prominent ones. Users and developers alike would have to deal with the hassle of communication and bureaucracy with companies and banks, and such a system would only be a sub-standard imitation of the current market giants.

B. Advantages of Blockchain-Based Solution

By using a decentralized approach to the above problem, we are able to detach any and all corporate influence from the marketplace. The Dapp Store would be run and maintained sustainably by the people using it. The storage of applications, proof of ownership, transactions, and application rating system all employ the concepts of distributed trust, and the result of this blockchain-based set of solutions is a truly open-source, community supported, scalable decentralized software application market that would resonate with the demands of both developers and users.

III. Solution

A. Architecture

We aim to build a distributed application based on Smart Contracts [2]. We plan to run our application on the Ethereum blockchain. We plan to use the BitTorrent protocol for the underlying transfer of files between users of the application. Whenever a person buys an application the transaction gets logged on the chain. The buyer can then start downloading the files needed for the application from the peers who have the application stored on their node. Whenever the users run the application a licensing check is done which confirms if the owner does actually owns the application and has not acquired it by any illegal means. More details on how the application authentication works has been given in Section IV.

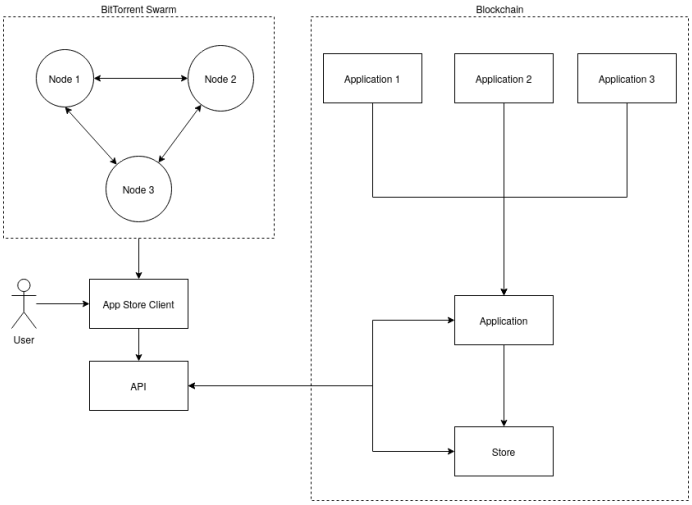

Fig. 1. Architecture Diagram of the Dapp Store

IV. Analysis

A. Problems

1) Licensing and proof of ownership: The idea of P2P transfer of files is more than a decade old, BitTorrent works on the same principles but allows anyone to download/upload files, and further, there are no restrictions on the files that are uploaded. To maintain a network where people will have to pay the developer for downloading files, one needs to have restrictions on file usage and file upload.

2) Ability to take down applications that are harmful: Original software can still be malicious, which means an adversary can cause problems on the user's device. This leads to a requirement for a mechanism that allows the network to remove files on demand, due to the decentralization, an authority can't be assigned to this task.

3) Protecting users from malicious copies of the application: Other than the case where the developer is an adversary there can be cases where a node storing the files is an adversary that supplies a malicious file to the user. A check has to be added that will verify if the files received are the same ones as the developer has distributed.

4) Why people would want to be a part of the network: In the case of a centralized app store, the commission for every purchase served as an incentive for hosting the app store and supplying the users with the files. In the case of decentralization, there needs to be some form of incentive that the developers have to provide to assure that their software stays on the nodes and is accessible by the users.

B. Decentralized Approach

When developing a decentralized app store, the above are a few problems that one has to tackle. Along with the ways that we used to work around the problems, we have also explained the approach for interacting with the users and the architecture for the database.

We use smart contracts to maintain the marketplace along with an API which is used to communicate with the devices. The storage for the actual files for the applications will be off-chain but will still be decentralized. We propose to use the BitTorrent protocol [4] which is already a well established peer to peer decentralized communication protocol to share the application files. To make sure that their applications are always present on the network, the developers have to incentivize the nodes to store their applications on their node.

1) Licensing and proof of ownership: To ensure the app is being used only by the person who bought it we need create some kind of licensing system which verifies the ownership of the application on each run of the application. We can solve this issue by storing the addresses of all the buyers of an application in the contract of the application. However this gives rise to a new problem, one can easily create a new address with and transfer Ether into it such that after buying the app minimal Ether will be left in the account. Then they can easily share the private key for this new dummy account with anyone and they too can use it. We counter this problem by allowing the user to transfer the ownership of the application to a new address given they have access to the original address. This adds a risk of losing the ownership of the application.

1) When an user buys an application from the store, the smart contract appends the user's address to a list.
2) During authentication, the user has to make a call to the smart contract to authentic the usage. The smart contract checks if the address is present in the list.
3) If the authentication is successful, the user is allowed to use the application.

With the scheme described above in place, the user can still share the newly generated private key with anyone. However, we also give an option to change public-private key, hence the person who the original buyer has given the private key can change the keys and effectively revoke access to the original buyer. All of this functionality can be implemented by the app developers using a library which will publicly available so that each developer doesn't have to reinvent the wheel.

2) Ability to take down applications that are harmful: We also need to have an ability to take down applications that are harmful for the community. We would need to have some

kind of voting system which would allow the chain to blacklist particular applications. We also need to make sure that no single entity can control which apps should be blacklisted as this is against the idea of decentralization. We directly can't use votes given by any user because then anyone can create sufficient number of addresses and register onto the network to vote an application to be harmful. It would make more sense to count the votes cast by the miners of the chain as they have to do some amount computation to to cast a vote and no single entity can singlehandedly blacklist a particular app.

3) Protecting users from malicious copies of the application: As we will be using decentralized storage, we need to make sure the files received by the user from any node are not corrupted or malicious. To solve this issue we store the hash of the binary files on the chain whenever the developer pushes their application. Once the files get downloaded on the user end, we use the same hashing algorithm and verify that the download files are the same as the ones that developer uploaded.

4) Why people would want to be a part of the network?: As we are storing all the files in a decentralized fashion, developers need to have a way so that their application files are always available on the network. To make sure that nodes on the network are motivated to store an developers app, the developer can optionally allot some percentage of the application price to go to the seeders on the network. For example suppose an application costs 1 ETH, and the developer allotted 10% of the application cost to go to the seeders of the network, then the 0.1 ETH will be distributed amongst the seeders based upon their seeding ratio. The amount to be transferred to each seeder can be calculated using the following formula:

$$P = P_A \times \frac{B}{100} \times \frac{TransferredBytes}{TotalBytes} \tag{1}$$

Where P is the amount to be transferred to the seeder, P_A is the application price and B is the cut decided by the developer to give to the seeders.

V. Required blockchain protocol

The blockchain that is required to be used is different from the original idea proposed in the Bitcoin white paper [3]. Due to requirements such as the change in incentive, The blockchain protocols must change.

A. Incentive for storage

In the original bitcoin protocol, the incentive for mining was the providing the miners with mining fee along with the tips given by the users. Since we are going to use pre-existing currency, we cannot use this mechanism. Incentive can be tackled by allowing the developers to set a percentage cut which indicates the fraction of the price that is transferred to the nodes that are supplying the files.

The amount transferred to the developer is less than the price that is being paid. The rest of the amount goes to the node that provided the files. This is a feature that leads to modification of the underlying protocols since the profit must go to the node that provided the files.

B. Consensus for File removal

A person is allowed to make any number of addresses on the blockchain. For privacy reasons, there is no way of tracking ownership of the key to a person. This means the traditional system of voting can be eliminated. but consensus is one of the core ideas of bitcoin. Bitcoin protocols are amended and added based on the consensus [5]. In the original protocol, the vote is backed up by proof of work, i.e. the vote is caste when the miner sets a parameter in the block they mined.

Similar mechanism must be used for deciding the fate of the apps. Each block mined includes a parameter which is a list of app ids that the successful miner votes to be taken off the chain. After 2021 blocks, if there is an app listed on the majority of the blocks, the app is removed from the store and the miners get no incentive for storing and supplying the app.

VI. Smart Contracts

The above section covered the implicit changes and requirements that a blockchain must satisfy, now we can look into the details of smart contract around which the dapp store will be built.

A. Hierarchy

The hierarchy of contracts is simple. There is one single *Store* contract that handles the complete market. It serves as the point of contact for the API. It holds the details of the applications in a map of structures. Each app has a unique id which is generated randomly upon the submission of the app.

Application is a base class that can be inherited for other apps. This concept has not been tackled in this paper.

```
/// @title App store
/// @author Aakash Jain, Ishaan Shah, Zeeshan Ahmed
/// @notice View, sell, and buy applications
contract Store {
    /// @dev Stores the details of an application
    struct AppDetails {
      //...other details
      Application applicationContract;
    }
    /// @dev mapping for all the applications
    mapping(uint256 => AppDetails) private
        applications;

    /// @dev list of all the hash of the files so
        far.
    mapping(bytes32 => bool) private hashList;

    /// @notice Triggered to store the details of a
        listing on transaction logs
    event ApplicationAdded();

    /// @notice Function to add application to the
        app store.
    function createAppListing() public payable {}

    /// @notice Function to buy a listing and
        accepts the money to store in the contract
    function buyApp() external payable {}

    /// @notice Verifies if the user has purchased
        required app.
    function checkPurchased() public view  {}
}
```

The following is the *Application* contract.

```
/// @title Base class for applications.
/// @author Aakash Jain, Ishaan Shah, Zeeshan Ahmed
/// @dev Parent class for the other applications.
contract Application {
  /// @dev structure to contain details about users
      for verification.
  struct User;
  mapping(address payable => User) public users;

  /// @dev structure to store all the details about
      the product
  struct Details;
  Details public details;

  /// @notice Triggered to store the details of a
      listing on transaction logs
  event PurchaseMade();

  /// @notice Function called by the buyer to make a
       bid.
  function buy() public payable {}

  /// @notice Fetches the details of the auction.
  function fetchDetails() public view {}

  /// @notice Verifies if the user has purchased the
       app.
  function checkPurchased() public view {}

  /// @notice Transfers the ownership of app to the
      new address
  function transferOwner() public {}
}
```

B. Application Details

Keeping track of applications has to be done in the app store. Minimal information is stored for the API to directly fetch all the apps and display it to the user. This can be made more effective by creating a search query function.

```
/// @dev Stores the details of an application
struct AppDetails {
  uint listingID;
  string appName;
  string appDesc;
  uint256 price;
  address payable developerID;
  uint256 downloads;
  bytes32 fileHash;
  Application applicationContract;
}
/// @dev mapping for all the applications
mapping(uint256 => AppDetails) private applications;
```

The contract *Store* exists only to interact with the API and provide the necessary functionality from the *Application* contract.

C. Transaction records

Using smart contracts, it is very simple to keep track of the apps that have been purchased. The API can keep listening to events that are triggered when the app purchase is successful and the transaction has passed. This serves as a method to interact with the API and ensuring that every purchase is kept on the chain.

```
/// @notice Triggered to store the details of a
    listing on transaction logs
/// @param listingID Unique ID for the application.
/// @param appName Name of the appplication.
/// @param price Price set by the developer.
/// @param developerID Address of the developer for
    transferring funds.
event ApplicationAdded (
  uint indexed listingID,
  string appName,
  uint price,
  address developerID
);
```

D. Preventing duplicacy

An adversary can upload an application that already exists on the market and try to take the profit away. This can easily be prevented by ensuring that the hash of the files for each app.

If the hash is repeated then the application is denied.

```
    mapping(bytes32 => bool) private hashList;
    /// @notice Function to add application to the
        app store.
    /// @dev Triggers the event for logging
    /// @param appName Name of the item
    /// @param appDesc Description of the item set
        by seller
    /// @param price Price set by the seller
    /// @param filePtr The pointer which will allow
        user to download the app.
    /// @param developerCut Value which will be
        transferred to the developer.
    function createAppListing(
      string memory appName,
      string memory appDesc,
      uint256 price,
      string memory filePtr,
      uint256 developerCut,
      uint256 fileHash
    ) public payable {
      if(hashList[fileHash]) {
        return;
      }
      // generating a new random id.
      uint id = (keccak256(abi.encodePacked(
                block.difficulty, block.timestamp,
                    players)));
      // storing the newly created application in
          the map.
      listings[id] = Listing(
        id,
        appName,
        appDesc,
        price,
        msg.sender,
        0,
        fileHash,
        new Application(
          appName, appDesc, price, developerID,
              filePtr, developerCut)
      );
      appCount += 1;
      hashList[id] = true;

      emit ListingCreated(id, appName, price, msg.
          sender);
    }
```

E. Verification of ownership

The underlying concept used in the address mechanism for verification is the same as the public key encryption. The address is a public key where as the wallet holds the secret key. With the help of smart contracts, the public key verification can be inherently captured in the calls that are made to smart contracts.

```
// Part of the Store contract
    /// @notice Verifies if the user has purchased
        required app.
    /// @param appId Id of the app which is being
        checked.
    /// @param user Address of the user that needs
        verification.
    /// @return If the user has purchased the app.
    function checkPurchased(uint256 appId, address
        user)
      public view returns (bool)
    {
        return applications[appId].checkPurchased(
            user);
    }
// Part of the Application contract
  /// @dev structure to contain details about users
      for verification.
  struct User {
    bool hasBought;
  }
  /// @notice Verifies if the user has purchased the
       app.
  /// @param user Address of the user that needs
      verification.
  /// @return If the user has purchased the app.
  function checkPurchased(address user) public view
      returns (bool) {
    return users[user].hasBought;
  }
```

The current decentralized systems rely on a username and a password. One who is willing to share the password with someone also gives them the power of changing the same.

The problem of having multiple users in the same account can not be solved given that the identity of the user is anonymous. Since the condition of having multiple users can't be improved in a decentralized network, we add an additional feature to keep it on par with the centralized counterpart.

Anyone with the private key has the ability to transfer the ownership to a new address, creating the risk of the original user losing access to the application they shared.

```
  /// @notice Transfers the ownership of app to the
      new address
  /// @param user Original address of the
      application.
  /// @param newUser New address of the application.
  function transferOwner(address user, address
      newUser) public {
    if(users[user].hasBought) {
      users[user].hasBought = false;
      users[newUser].hasBought = true;
    }
  }
```

The APIs that are created will handle the authentication by making calls to the smart contract with the user's account, a privelage which is granted only if you have the private key for the address.

VII. Conclusion

The proposed solution, the Dapp Store, is a P2P, decentralized application marketplace which supports the selling and buying of applications. The system relies on bittorent for the file transfer protocol, and an API that enables communication between the user and the store's contract. The Dapp Store is our solution to exclude large corporations and banks from the equation and make a transparent, fair, and user-friendly application that is both meant for and regulated by users and developers alike.

References

[1] Viennot, Garcia, Nieh(2014): A measurement study of google play, 2014 ACM International Conference on Measurement and Modeling of Computer Systems

[2] V Buterin(2014): A next-generation smart contract and decentralized application platform

[3] Nakamoto, Satoshi(2009) Bitcoin: A Peer-to-Peer Electronic Cash System

[4] B Cohen(2003): Incentives Build Robustness in BitTorrent

[5] Bitcoin Source Code: https://github.com/bitcoin/bitcoin