Ethereum and Smart Contracts

Anthony Nixon
Universitat Politecnica de Catalunya
anthony.nixon@est.fib.upc.edu

Jeffrey Jedele
Technical University of Munich
jeffrey.jedele@tum.de

1 INTRODUCTION

Software continuously changes the way we communicate, work, learn, play and almost every other aspect of our lives. Progress, however, rarely comes without new drawbacks. One of the drawbacks of our more and more digital lifestyle is that we give an ever increasing amount of control into the hands of large institutions and corporations.

The ways in which this might hurt us are numerous. Google has the power to single-handedly remove inconvenient sites from their search index, thereby practically removing them from the internet for a large part of the population. There are several cases where news distributors have been brought down and made unreachable by cyber attacks. It is easy to imagine non-democratic governments manipulating the results of elections to their liking. Banks and insurance companies increasingly base their decisions on the outputs of closed machine learning systems which only few people understand.

What if we could give these systems back into the hands of the people they influence the most, allowing them to understand exactly how they work and making sure that no single party can manipulate them in unwanted ways?

This is the promise of *Ethereum* and *smart contracts*. The present report explores how to work with this technology given a simple example explained in the next section.

2 SMART TIPPING FOR TUTOR GROUPS

In this section we describe a not completely serious but educative use case from the university context. At heart it is a performance-based remuneration contract, which could be converted to a financial derivative or insurance with minimal context changes.

University courses are typically broken down into lecture and exercise units. The exercises are an essential part in the student's venture of achieving a good grade and are handed off to PhD students besides their actual work without getting enough extra time allocated.

A typical approach to motivate and gratify the tutor in such a situation would be monetary, e.g. by tipping. This however leaves us with the question when the tips will be payed.

In a first option, students tip the tutor at the beginning of the semester. This forces the students to trust the tutor with doing a good job despite already having received his incentive.

In a second option, students tip the tutor at the end of the semester. Now the tutor has to trust the students to actually tip him despite already having received the service.

This is a common situation where both parties do not necessarily know and therefore trust each other upfront. It makes it difficult to agree on a way of conducting a transaction.

Blockchain Technologies Seminar (IN4909), WS 2018, Garching, Germany
2018.

Is there a solution which does not require mutual trust? Imagine a third party, which accepts the tips of the students at the beginning of the semester and tells the tutor how much he would get if he does a good job. But only if the results are satisfactory he is payed out, otherwise the tips are returned to the students.

Both the students and the tutor now only have to trust this third party and not each other. Smart Contracts can be used to build such a third party, and one that is easy to trust on top, since they are deterministic and immutable computer programs with their source code publicly available to everyone.

3 ETHEREUM

Blockchain technology has several important strengths:

- *Decentralization* creates natural resistance against data loss e.g. through natural disasters or hacker attacks.
- *Trustlessness* facilitates transactions between parties that do not know each other.
- *Immutability* prevents data from being manipulated by individual parties.
- *Publicness* makes the data available to all participants and can counteract it from accumulating in corporations and institutions.

Bitcoin introduced the blockchain technology and its advantages for monetary transactions, but it is easy to see that these attributes are interesting for a much wider range of applications. In 2014, Vitalik Buterin wrote the first version of the Ethereum Whitepaper [5], a new type of blockchain on top of which general programs can be build. The native currency is called *Ether (ETH)*.

One way to describe Ethereum is as a "world computer", which is not owned by a single entity and is not located at a single place, but is constituted and shared by everyone who participates in it. The programs developed for this computer are called *smart contracts* and run in a simple, stack-based virtual machine, the Ethereum virtual machine (EVM) . The EVM is defined in the *Yellow Paper* [31], which is mainly maintained by Dr. Gavin Wood.

The deployment model is given by contract accounts. Ethereum has two types of accounts (cf. [18]]):

(1) *externally owned accounts (EOAs)* belong to a person and are controlled by a private key.
(2) *contract accounts* have a public address and an Ether balance like EOAs, but instead of a private key, they are controlled by the associated smart contract.

3.1 Technicalities

3.1.1 Consensus. Ethereum uses a proof of work (PoW) algorithm called *Ethash* at the time of this writing. It is based on the Keccak hash function. Ethash is designed to be memory-intensive, thereby making it hard to develop application-specific integrated circuits (ASICs) for mining. This aims at counteracting the concentration

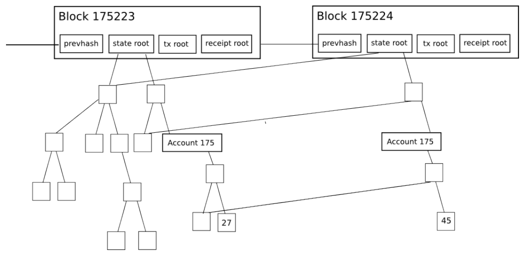

Figure 1: Ethereum Blocks
Two blocks from the Ethereum blockchain. The state trees are expanded and how the updated value of account 175 is persisted using copy-on-write technique. Image taken from [6].

of mining power in pools of large and financially-strong entities that can afford such specialized hardware (cf. [4]).

3.1.2 State Storage. Ethereum can be seen as a state machine. Every transaction changes it from one state to a new state. In contrast to Bitcoin, state is represented on account level rather than transaction level. Technically, Ethereum stores all state in a data structure called *Merkle-Patricia tree* [31, sec. 4.1]. It can be used to efficiently validate the integrity of blocks (Merkle tree) as well as for efficiently querying elements (Patricia tree). The tree nodes are indexed by their hashes and persisted in a key-value-store.

Each block consists out of three such trees: one for transactions, one for transaction receipts and one for the state [6]. The tree structure also provides an efficient copy-on-write mechanism for changing data (cf. Figure 1).

3.2 Gas

Code associated with a blockchain transaction has to be executed by every miner validating a block. Bitcoin allows users to write their own transaction validation scripts, but the language is not Turing complete and severely limited for security reasons (cf. [10]). E.g there are no looping constructs to stop users from writing infinite loops. A blockchain with a general programming model needs a concept to keep resource usage under control.

Ethereum's solution to this problem is called *gas*. In Ethereum, every instruction the EVM has to execute and every byte that has to be stored in the blockchain consumes a defined amount of gas (cf. [23]). Whenever someone initiates a transaction, he decides upon two parameters:

- The *gas price* denotes how much Ether the initiator is willing to pay for each consumed unit of gas.
- The *max gas* value denotes how much gas the transaction may consume in total.

Miners select transactions to include in their blocks based on these parameters, thereby making resource usage on Ethereum a self-contained market where prices can change in relation to real-world operating expenses, e.g. improving computing hardware or power pricing (cf. [1]).

While the transaction executes, the allotted gas amount (max gas) is incrementally consumed. If the transaction completes before the gas is used up, the remaining gas is returned to the initiator. If the gas runs out before the transaction completes, it is aborted and all changes are reverted (the miner still receives the used gas).

3.3 Decentralized Applications

The terms *smart contract* and *decentralized application (DApp)* are frequently encountered when working with blockchains like Ethereum. A DApp typically describes a classical web application where the centralized backend (e.g. a MySQL database) has been replaced by a stack of decentralized smart contracts. There are ventures to decentralize the frontend as well (e.g. the Interplanetary File System (IPFS) [3]), but these are not commonly used as of today and we will not talk more about them in this report.

3.4 Common Use Cases

While smart contracts can be arbitrary programs, some use cases fit the blockchain technology especially well. We introduce some of these below.

3.4.1 Stake/Ownership Management. Probably the oldest and most widely known use case for smart contracts is managing stake or ownership in some underlying asset, e.g. a company or real estate. We call these systems *token systems*. The basic idea is the following: We implement a smart contract that acts as a proxy for this asset and spread the total value out across a specified amount of tokens. Users can then transfer currency to this contract and in return receive tokens which certify their stake in the asset. This is the underlying mechanism of initial coin offerings (ICOs) which have received some attention in recent times. The use case is so common that multiple standard interfaces for token contracts have been defined: ERC20 [29] and ERC721 [11].

3.4.2 Financial Services/Derivatives. This class includes insurance contracts, speculative financial products and performance based remuneration systems amongst others. The example introduced in Section 2 also belongs to this category. The abstract idea is having a pot into which multiple parties pay in currency. Then it gets decided based on external events who receives back how much of the total volume.

3.4.3 Unalterable Information Storage. The blockchain is a natural option when information shall be stored such that it is publicly available and tamper-proof. Examples could be scientific data or money flows of charity organizations. Even if the original data must not be publicly available (identity documents, grade reports, health records, ...), the blockchain can still be used to ensure the absence of tampering by storing fingerprints/hashes (cf. [19]).

3.4.4 Decentralized Organization. Smart contracts can be used to make decisions in large, distributed groups where members do not necessarily know and trust each other. One early example of this is DASH, a cryptocurrency that makes use of itself to coordinate its development process (cf. [14]). This technology could e.g. allow for tamper-proof elections in countries with incredible and corrupt governments.

4 WRITING SMART CONTRACTS

In this section, we talk about the development of smart contracts in context of developing the example introduced in section 2.

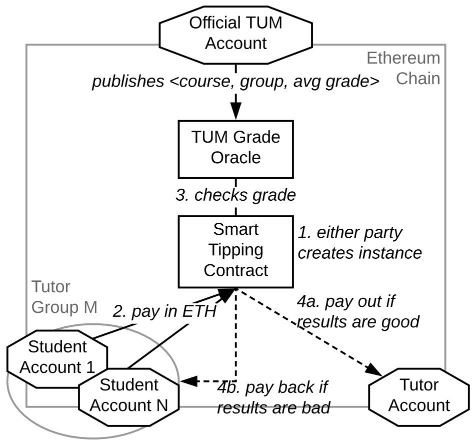

Figure 2: Design of the Smart Tipping System

4.1 Smart Tipping System

The smart tipping system comprises two contracts (cf. Figure 2). The simpler one is an oracle contract used by university employees to write grade reports into the blockchain. We talk more about this necessity in Section 4.3.3. The second one is the actual tipping contract. The workflow is as follows.

(1) Either party creates an instance of a tipping contract at the beginning of the semester. At this point, both parties decide on a target grade determining if the tutoring will be considered successful.
(2) The students pay in their tips. Both parties can check the current volume of the contract at any time.
(3) At the end of the semester, either party triggers the resolution of the contract.
(4) The tipping contract checks the grade results published by the university. If it is better than the defined threshold, the tutor is payed out the accumulated tips. If it is not satisfactory, the money is payed back to the students.

4.2 Development Process

The EVM defines its own byte-code to which smart contracts are compiled. There are several high-level languages to develop smart contracts, the most common one is *Solidity* [27].

The high-level process of developing a DApp involves following steps:

(1) Developing the smart contract code.
(2) Compiling the smart contract code in order to obtain deployable byte-code and interface specifications that can be used from the frontend.
(3) Deploying the contract code to the blockchain.
(4) Writing and deploying the frontend code.

Most DApp developers use the *Truffle Framework* [28] at the time of this writing. It provides tools such as

- Solidity compiler,
- scriptable contract deployment framework,
- unit testing framework,

```
pragma solidity ^0.4.18;

contract TUMOracle {

  address public owner;

  mapping(string => mapping(uint8 => uint8)) grades;

  constructor () public {
    owner = msg.sender;
  }

  function addGrade(string courseId, uint8 groupId, uint8 grade)
  ↪  public {
    require(msg.sender == owner);
    require(grade >= 10 && grade <= 50);
    grades[courseId][groupId] = grade;
  }

  function getGrade(string courseId, uint8 groupId) public view
  ↪  returns (uint8) {
    uint8 grade = grades[courseId][groupId];
    require(grade > 0);
    return grade;
  }

}
```

Listing 1: Grade Oracle Contract

- test chain for local testing ("Ganache") and
- libraries simplifying the integration of smart contracts with frontend code ("Drizzle", "web3").

This report focuses on the development of smart contracts. For the other steps of the development process, the homepage of the Truffle framework is a good starting point.

4.3 Solidity

Solidity is a language with strong influences from JavaScript and C++. The concept of classes has been rebranded as contracts, but conceptually they are similar. We can create instances of contracts and deploy them to the blockchain. A contract can have member variables and methods, methods in turn can have local variables. Listing 2 shows the simple oracle contract from our example use case.

4.3.1 Storage. Solidity provides simple types like integers and complex types like structs, arrays and hashmaps (which are called mappings). Noteworthy are the absence of floating point numbers (fixed point numbers are in development but not yet supported at the time of this writing, cf. [26]). Integer size can be defined very granuarly from 8bits to 256bits in 8bit steps. This is important because each stored byte has costs gas.

The EVM has two types of memory: *local memory* and *contract storage*. Local memory is volatile and supports the execution of smart contract methods. It uses a stack for function arguments and linearly addressable memory for local variables. Contract storage is persistent and part of the contract account. Conceptually it can be thought of as a 256bit-addressable byte array, but it is implemented as a key-value-store to account for how sparsely[1] this array will be populated (cf. [20]).

[1]A 256bit-addressable byte array can hold more than 10^{68}GB.

```
import "./TUMOracle.sol";

contract Tipping {

    // interface for the grade results
    address tumOracle;

    constructor (address _tumOracle) public {
        tumOracle = _tumOracle;
    }

    function resolve(string _course, uint8 _group) public {
        TutorGroup storage t = tutorGroups[_course][_group];
        TUMOracle oracle = TUMOracle(tumOracle);
        uint8 grade = oracle.getGrade(_course, _group);
        if (grade <= t.gradeGoal) {
        ...
    }
}
```

Listing 2: Inter-Contract Calls

4.3.2 Error Handling. The EVM has two error modes: *expected errors* and *unexpected errors* (cf. [21]).

Expected errors are caused by failing parameter validations (`require()` function) and by explicit calls to `revert()`. When an expected error happens, all changes are reverted and the *remaining gas is returned to the transaction initiator.*

Unexpected errors are caused by failing `assert()` calls, array access with negative index, etc. In this case, changes to the chain are reverted and *all remaining gas is used up.*

It is good practice to check user inputs wherever possible using `require()` statements.

4.3.3 Isolation. The reader might wonder why the presented system uses a separate grade oracle contract instead of directly accessing already existing interfaces of the university. It turns out that this is not possible. Every execution of contract code must be executed by different miners on different computers every time the containing block is validated. The execution must deterministically yield the same result, otherwise reaching consensus about the validity of transactions would not be possible.

This has two practical implications:

- The EVM provides no mechanisms for randomness.
- There is no access to external systems. Contracts can only work with data stored in the blockchain.

The second point creates the need for so-called *oracle accounts*. These are simple smart contracts that proxy institutions and serve as a gateway to get external data into the blockchain (cf. [13]).

4.3.4 Inter-Contract Communication. To create more complex systems (e.g. using data from oracle contracts), contract methods must be able to call other contract methods. To do so, two things are necessary: Firstly, the code of the callee contract is needed by the Solidity compiler in order to properly compile and link the call. Secondly, the address of the deployed instance of the callee contract must be known so that calls can be forwarded. Given these two things, a proxy object for the instance can be instantiated and used like a local object. Typically the address of the callee is provided as constructor parameter. Listing 2 outlines this process using the relevant parts of the smart tipping contract.

```
contract Event {

    event AmountIncreased(uint32 tutorGroupId, uint newTotal);

    function payIn(uint32 _tutorGroupId) public payable {
        TutorGroup storage t =
        ↪  tutorGroupIdToTutorGroup[_tutorGroupId];
        Student storage s = t.numStudentToStudent[t.numStudents++];
        s.addr = msg.sender;
        s.amount = msg.value;
        t.amount += s.amount;
        emit AmountIncreased(_tutorGroupId, t.amount);
    }

}
```

Listing 3: Emission of Events

```
tippingContract.events.AmountIncreased({
    filter: {tutorGroupId: 5},
    fromBlock: 0
}, function(error, event) {
    alert(`You're now getting ${event.newTotal}ETH`);
});
```

Listing 4: Event Handling Using the Web3 Library

4.3.5 Reads and Writes. Calls to smart contracts can either be read-only operations or modify the state of the blockchain. Read operations do not trigger a transaction, are executed synchronously and do not consume gas (cf. [16]).

Write operations trigger transactions and do consume gas. Two things are important when working with transactions: Firstly, writes to the blockchain are not only triggered by explicit assignments to the contract storage but also when events are emitted (cf. Section 4.3.6). Secondly, transactions are executed asynchronously since they have to be mined to become valid. As one result, calling a write operation does not return a value but a *transaction receipt* [31, sec. 4]. At the time of this writing, Solidity however allows defining return values for write operations and the compiler does not catch when they are used. This is a common source of confusion amongst new Solidity developers.

4.3.6 Events. Since transactions are executed asynchronously, we need an asynchronous mechanism to communicate their results and effects to other systems. Solidity provides such a mechanism by the means of *events* [9], which make use of the underlying Ethereum logs [31, sec. 4]. Solidity code can define and emit structured events, which client libraries can subscribe and react to using the publish/subscribe paradigm. Listings 3 and 4 show how this mechanism can be used to notify a tutor about new tips.

4.4 Ethereum Development Considerations

4.4.1 Compilation and Debugging. Development in the Ethereum ecospace is still in its infant stages. New iterations of the solidity compiler and companion tools such as the *Truffle Framework* are regularly being released.

To ensure that contracts behave as expected it is important to add a pragma statement, e.g. '`pragma solidity 0̂.4.18;`', at the beginning of solidity files which defines the compiler release the code conforms to.

The solidity compiler's error handling tends to lack meaningful and descriptive messages. It can take patience and thoughtful code review in order to track down difficult syntactic or run-time errors. This is likely due to the high frequency of releases and hopefully improvements will come in time.

The current Truffle toolset includes a solidity runtime debugger but the functionality is extremely limited to what is standard in more mature languages. One noticeable limitation is the inability to view individual variable states in local memory. At this time, the best method for tracking variables remains console log statements or emitting events as described in the previous section.

4.4.2 Building UIs. The "Drizzle" frontend integration tools are a fast way to quickly build UI interactions with contracts. There are two flavors, Drizzle-React and Drizzle-React-Components that build on top of the Ethereum web3 framework in the sequence *Web3 -> Drizzle-React -> Drizzle-React-Components.* Although the Components are plug and play React elements, they suffer from severe limitations such as the inability to assign transaction values to messages. With this in mind, it is often the case that developers will have to go deeper towards web3 in the tech stack until they obtain the granularity needed for the contract's use case. Another tip is to keep an eye on current API definitions as recently there have been changes between version releases.

4.4.3 Testing Contracts. Because deploying contracts to the main Ethereum chain costs gas and suffers from validation delay it is advisable to use a local test-chain such as Ganache or that included in the Truffle Console. This drastically speeds up development time, however, most modern browsers use the Metamask wallet management extension and in its current state it does not work well with local test-chains. The connection and management of multiple accounts is clunky and cumbersome and can slow down progress.

Finally, it is important to realize that documentation and sources of reference regarding DApp development is widely available but often out-dated. A tutorial or resource may be correct for an outdated pragma/compiler release, but inefficient or broken in current releases. Adding to the confusion is the multitude of higher-level tools and frameworks which often have different approaches to the same task. It is advisable to refer to multiple sources and evaluate the merits and suitability of the procedure before diving in and committing significant development time.

5 CHALLENGES IN DAPP DEVELOPMENT

This section will discuss particular challenges as they apply to both DApp development on Ethereum, and blockchains in the broader context.

5.1 Dangers of Immutability

Along with all the benefits of immutable ledgers, one of the foremost disadvantages is an inability to modify existing code logic. Once a contract is deployed, it is incapable of having patches or bug-fixes applied to it. The only manner in which contract behavior can be modified is to deploy a new contract and have all actors redirect calls to the address of the new contract. Furthermore, whatever interfaces remain public and active on the previous contract will still be accessible, as well as any unintended side effects or transactions which were confirmed via the flawed code.

An important case study regarding the dangers of flawed contract code being migrated onto the main Ethereum chain is that of the DAO hack. [30]

The DAO was a successful crowdfunding initiative launched via token sale which took place in May of 2016. The premise was to create a trustless and autonomous venture capital platform where stakeholders could vote on investment opportunities and have fund allocations and management functions executed solely by Ethereum contract interactions.

A major vulnerability concerning the handling of recursive calls allowed a user to siphon off 3.6 million Ether of the DAO holdings (around a third of the total value). The fiat value of Ether at the time of the attack put the stolen amount at approximately 50 million USD.

As alluded to at the beginning of this section, from a technical standpoint, there is no remedy within the parameters of Ethereum's intended operation. However, given the gravity of the situation, during a 28 day holding period coded into the original contract, community leaders contemplated external and oligopolistic methods to artificially change the state of the blockchain. The two main ways of doing this are a soft-fork and hard-fork. In the soft-fork scenario, DAO transactions would be blacklisted going forward. In the hard-fork, the DAO transactions would be reverted and a parallel chain (fork), through consensus and political influence, would become the "new" Ethereum going forward. [7] In the end, a hard-fork was executed and this split the community between those who felt it was in the social interest to interfere externally with the immutability of the Ethereum chain, and those that felt the blockchain was intended to be a departure from such governing interferences. The abandoned fork still operates today as an independent entity under the moniker of Ethereum-Classic, alongside the main Ethereum. [15]

5.2 Speed and Scalability

In today's atmosphere of cloud computing, distributed systems, and large scale infrastructure investment; critical bottlenecks in speed and scalability are often addressed further along the development process for many applications. However, due to the need for validation and replication of blocks by miners, transaction time and number of op-code executions becomes a crucial consideration from the onset of design in blockchain applications.

As seen in figure 3, Ethereum in its current state has an extremely low transaction throughput. Although the figure compares digital currency transactions, the mechanisms for DApp confirmations is the same. The throughput issue can often be even more crippling during high loading times such as during an ICO or token launch.

With the scaling and speed limitations in mind, DApp development currently favors systems developed with a hybrid approach of handling non-pertinent transactions and procedures off chain, while committing stake and value transactions to Ethereum storage. An example of this paradigm would be CryptoKitties, which stores ownership and DNA (a seed which identifies a unique permutation) on the blockchain but hosts all digital assets on a centralized server.

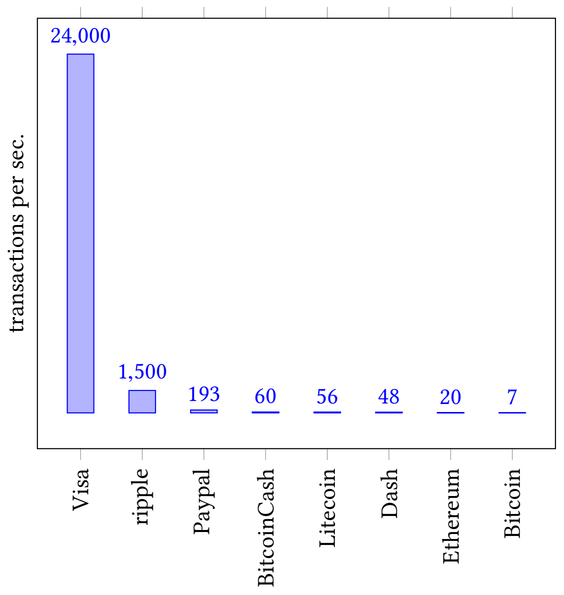

Figure 3: Transactions Speed of Digital Currencies[17]

When the DNA is combined with the centralized data it generates the user's unique kitty.[22]

5.2.1 Other Solutions to Scalability and Throughput. Although developers have found ways to mitigate scalability issues through careful program design, there is an ongoing push to improve the underlying capabilities of the Ethereum network and increase transaction speed through incremental changes to the underlying technology. In the Ethereum sphere there are two such solutions which are gaining heavy support. First is shifting the consensus method from proof of work (PoW) to proof of stake (PoS) and the other is sharding and side-chaining.

Ethereum is currently using PoW where consensus is driven by hash rate. In a PoS system, validators are delegated based on their value stake in the system - with the rationale that attacks on the system serve only to devalue the bad actor's assets in this case. In addition to scalability benefits, PoS also has the following advantages: less energy consumption, lower transaction costs needed to motivate miners, decreased risk of hash cartels, reduced centralization risk, and discouraging many 51% attacks.[12]

Sharding and side-chaining are techniques of delegating main chain transactions and then recombining without loss of trust. Sharding accomplishes this often by using sub-pools of miners on the main chain, while side-chaining takes the transactions completely off the main chain. In the first quarter of 2018 Loom Network launched to much fanfare as a viable side-chain solution to Ethereum's scaling issues. Loom uses two layers of consensus to give DApps access to their own blockchain while maintaining trust for write-back to the main Ethereum network via a transfer gateway. As of Feb 4, 2019 Loom Network has a market cap of more than 26 million USD showing increasing acceptance of the technology for throughput critical applications.[8]

5.3 Independence and Decentralization

Even though the state of a DApp is independent and decentralized, in many cases to be truly useful programs will inevitably intersect with central, trusted authorities. This may come in the form of relying on data from a centralized database, triggering events from third party news sources, or even something seemingly innocuous such as inputting a tracking number (issued by a postal office). In all cases, these interactions have the potential to carry with them the instabilities, self-interest, and bias of the authority or agency and degrade the autonomy of a DApp. Developers must put careful thought into all factors which could affect the integrity of their system. Significant risk can be mitigated with proper vetting of trusted authorities and relying on outside influence only when absolutely necessary for the functioning of the contract.

Another point of consideration regarding independence is the governing legal frameworks in the region of an application's target market. For many years blockchain technologies enjoyed a rather hands-off approach by regulators which was lauded by many as a boon for the advancement and rapid growth of the technology.[32] However, after the proliferation of fraud and illicit ICO offerings in the wake of the 2017 cryptocurrency bubble there has been an upswell in support for regulation and control in the industry. It has been estimated that around 80% of ICOs launched in 2017 were scams.[2] In the era of regulatory uncertainty, some countries have taken a proactive approach by quickly drafting and enacting into law pro-business regulation regarding digital assets. Topping the list of countries favorable to blockchain related development are the countries of Malta, and Switzerland. In 2018, the parliament of Malta approved regulatory frameworks addressing blockchain technologies while Swiss investors enjoy tax-free status on crypto investments.[25]

6 CONCLUSION

Undoubtedly, blockchain technology is not only impacting monetary institutions but affecting the way the world approaches trust-based interactions in general. Ethereum, analogous to a "world computer" is at the forefront of the current revolution.

We have discussed common use cases for modern DApps and how distributed immutable ledgers are a solution to long-standing problems across multiple domains. In the context of our prototype Smart Tipping for Tutor Groups DApp we have covered the basics of Ethereum development from a practical perspective.

Finally, we discussed current challenges in the domain of DApp development touching on dangers of immutability, scalability considerations, and state of the market and push for regulation. It is our hope that the next generation of Ethereum smart contracts will democratize and empower the masses creating a more just and transparent society. As such, it is an exciting time to be a blockchain developer.

REFERENCES

[1] Sunny Aggarwal. 2017. Understanding Ether vs Gas. Retrieved 2019-01-13 from https://medium.com/sunnya97/understanding-ether-vs-gas-82ce2f1dc560
[2] Ana Alexandre. 2018. New Study Says 80 Percent of ICOs Conducted in 2017 Were Scams. Retrieved 2019-02-04 from https://cointelegraph.com/news/new-study-says-80-percent-of-icos-conducted-in-2017-were-scams
[3] Juan Benet. 2013. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* abs/1407.3561 (2013). arXiv:1407.3561 http://arxiv.org/abs/1407.3561

[4] Vitalik Buterin. 2013. Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alternative. Retrieved 2019-02-07 from http://www.hashcash.org/papers/dagger.html

[5] Vitalik Buterin. 2014. A Next Generation Smart Contract & Decentralized Application Platform. Retrieved 2019-01-13 from https://www.weusecoins.com/assets/pdf/library/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

[6] Vitalik Buterin. 2015. Merkling in Ethereum. Retrieved 2019-02-07 from https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

[7] Noelle Acheson Coindesk. 2018. Hard Fork vs. Soft Fork. Retrieved 2019-02-04 from https://www.coindesk.com/information/hard-fork-vs-soft-fork

[8] CoinMarketCap. 2019. Loom Network Charts.

[9] ConsenSys. 2016. Technical Introduction to Events and Logs in Ethereum. Retrieved 2019-01-15 from https://media.consensys.net/technical-introduction-to-events-and-logs-in-ethereum-a074d65dd61e

[10] Albert Costill. 2016. Bitcoin scripting and how it can be improved. Retrieved 2019-01-13 from https://due.com/blog/bitcoin-scripting-and-how-it-can-be-improved/

[11] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. 2018. ERC-721 Non-Fungible Token Standard. Retrieved 2019-01-14 from https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md

[12] et al. James Ray. 2018. Proof of Stake FAQs. Retrieved 2019-02-04 from https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs

[13] Adam Gall. 2018. Building your first Ethereum Oracle. Retrieved 2019-01-14 from https://medium.com/decentcrypto/building-your-first-ethereum-oracle-1ab4cccf0b31

[14] Juan S. Galt. 2015. DASH – The First Decentralized Autonomous Organization? Retrieved 2019-01-14 from https://cointelegraph.com/news/dash-the-first-decentralized-autonomous-organization

[15] Alyssa Hertig. 2017. Ethereum's Two Ethereums Explained. Retrieved 2019-02-04 from https://www.coindesk.com/ethereum-classic-explained-blockchain

[16] Rob Hitchens. 2018. Calls vs. transactions in Ethereum smart contracts. Retrieved 2019-01-15 from https://blog.b9lab.com/calls-vs-transactions-in-ethereum-smart-contracts-62d6b17d0bc2

[17] Crypto Coin Junky. 2018. The Fastest Cryptocurrency Transaction Speeds for 2018. Retrieved 2019-02-04 from https://medium.com/@johnhinkle_80891/the-fastest-cryptocurrency-transaction-speeds-for-2018-498c1baf87ef

[18] Hu Kenneth. 2018. Ethereum account. Retrieved 2019-01-13 from https://medium.com/coinmonks/ethereum-account-212feb9c4154

[19] Lukas Marx. 2018. Storing Data on the Blockchain: The Developers Guide. Retrieved 2019-01-14 from https://malcoded.com/posts/storing-data-blockchain

[20] Steve Marx. 2018. Understanding Ethereum Smart Contract Storage. Retrieved 2019-01-13 from https://programtheblockchain.com/posts/2018/03/09/understanding-ethereum-smart-contract-storage/

[21] Maurelian. 2017. The use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM. Retrieved 2019-01-14 from https://media.consensys.net/when-to-use-revert-assert-and-require-in-solidity-61fb2c0e5a57

[22] Greg McMullen. 2017. Do you really own your CryptoKitties? Retrieved 2019-02-04 from https://medium.com/@gmcmullen/do-you-really-own-your-cryptokitties-d2731d3491a9

[23] Ameer Rosic. 2018. What is Ethereum Gas: Step-By-Step Guide. Retrieved 2019-01-13 from https://blockgeeks.com/guides/ethereum-gas-step-by-step-guide/

[24] Vaibhav Saini. 2018. Getting Deep Into EVM: How Ethereum Works Backstage. Retrieved 2019-01-13 from https://hackernoon.com/getting-deep-into-evm-how-ethereum-works-backstage-ac7efa1f0015

[25] Michael K. Spencer. 2018. Top 8 Crypto & Blockchain Countries. Retrieved 2019-02-04 from https://medium.com/futuresin/top-8-crypto-blockchain-countries-2c526e00e951

[26] Solidity Team. 2018. Solidity 0.5.2 Documentation - Types. Retrieved 2019-01-14 from https://solidity.readthedocs.io/en/v0.5.2/types.html

[27] Solidity Team. 2018. Solidity Documentation. Retrieved 2019-01-14 from https://solidity.readthedocs.io/

[28] Truffle Team. 2018. Truffle Suite. Retrieved 2019-01-14 from https://www.truffleframework.com

[29] Fabian Vogelsteller and Vitalik Buterin. 2015. ERC-20 Token Standard. Retrieved 2019-01-14 from https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

[30] Wikipedia/Various. 2019. The DAO (organization). Retrieved 2019-02-04 from https://en.wikipedia.org/wiki/The_DAO_(organization)#History

[31] Gavin Wood. 2018. Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper). Retrieved 2019-01-13 from https://ethereum.github.io/yellowpaper/paper.pdf

[32] Peter Yeoh. 2017. Regulatory issues in blockchain technology. *Journal of Financial Regulation and Compliance* 25 (2017), 196–208.