ETH zürich

Project group 3

Creating a decentralised market place to incentives grid flexibility actions

Report

Authors:
Hei Kern Leong
Xujia Zhu
Silas Gyger

Sus Dao

https://github.com/FuturICT2/susDAO-eflex

http://flexwise.sgyger.com/

Contents

Introduction

0.1 Future power grids and the need for flexibility

The tradition power grid is primarily powered by spinning generators such as coal, natural gas and hydro power generators. These generators can control its power output by either varying the amount of fuel burnt in the case of thermal generators, or the amount of water going through the turbine in the case of hydro generators. These generators can thus ensure power generation meets power demand by controlling its power output to follow the demand as it raises and falls throughout the day. Power system flexibility is defined as "the ability of a power system to reliably and cost-effectively manage the variability and uncertainty of demand and supply across all relevant timescales, from ensuring instantaneous stability of the power system to supporting long-term security of supply" [1]

In recent years, the energy grid is going through a change in its generation composition. Solar and wind are now the two fastest growing energy sources in 2020 [2]. Solar and wind energy are considered as Variable Renewable Energy Source (VRES). These resources generate power as a function of the prevailing weather conditions. This presents the power grid with the challenge of balancing the grid when both the generation and loads are not controllable. Under low penetration of VRES, conventional spinning generators can compensate for sudden changes in power generation by regulating its own power output in the opposite direction.

Solar production is projected to reach 40% of total electricity in Europe by 2050. [3] When there is a high penetration of VRES, the flexibility requirement would correspondingly increase. An increase in share of VRES would also mean a decrease in proportion of controllable generational units which in turn reduce the flexibility potential since these controllable generational units have been the main source of flexibility provision. This increase in the need for flexibility coupled with the decreased availability from spinning generators presents a challenge to the electric grid.

The penetration of VRES will only continue to increase as the world is moving towards a more environmentally friendly way to power out society. It is thus important to find other sources of flexibility that would allow us to continue to have a reliable power grid to fuel human progress.

0.2 Potential for flexibility by batteries and loads

Other than spinning generators, another source of flexibility are batteries and loads. Batteries Can provide flexibility to the grid by charging itself when there

is a surplus of power and discharge itself when there is a shortage of supply. In some countries, these batteries come in the form of pump hydro facilities where water is pumped up a gradient to be stored behind a dam to be used later. Pump storage power plants can store a large amount of energy and thus provide a lot of flexibility. Pump storage power plants are however very geographical dependent.

Another form of flexibility provision is the use of electrochemical batteries. These batteries are gaining widespread adoption due to the increased popularity of electric vehicles. Grid scale electrochemical batteries are also being installed at an exponential rate with projected worldwide capacity of reaching 150GW by 2030 **??**. Up from 1.2 GW in 2016.

One more form of grid flexibility is in demand response. In the past, the demand was not controlled and the supply would change to follow demand. But as supply becomes less controllable, the role of controlling demand to meet supply is being explored to provide another avenue for flexibility. The addition of electric vehicles and large home batteries in homes further increase the flexibility potential for demand response. According to an IRENA report, [4], behind-the-meter storage and EV to grid technologies would be able to provide substantial flexibility to the electric grid.

0.3 Challenge for scaling the coordination of said flexibility

While there is huge potential in tapping the demand response market, the distributed nature of these flexibility resources means that there is substantial challenge in coordinating and regulating these many small individual flexible resources to provide useful flexibility to the grid.

Currently, the task of aggregating and coordinating these distributed flexibility assets are being undertaken by demand aggregator companies. Companies such as Enel X **cite please** form agreements with commercial and industrial companies to change their loads in exchange for financial benefits. This arrangement puts a lot of power to the aggregation companies as they have overall control how they would like to optimise the demand response.

Other companies such as leap (leap.energ) provide a platform in which users can upload their price for selling flexibility and flexibility users could buy it through the leap market place. This arrangement also puts a lot of power into leap as a company as they monopolise the marketplace as a private company.

1 Decentralising flexibility

The portion of the energy market consisting of the variable energy sources will only increase with time as humanity prepares to switch to purely renewable energy sources. This brings with it the need for means of either matching or flattening this variability, which demand side management conveniently provides. Left unchecked, this results in the power grid increasingly being monitored and controlled by governments and private companies. It can be argued that the mere existence of an infrastructure that allows for the necessary centralized control and tracking of the power grid and its participants brings with it a huge risk of abuse. No institution can be trusted with this power for longer than a few decades; a system is needed which provides the advantages of large scale algorithmic optimization of our power grids without having to trust a central authority to execute these algorithm faithfully and in the general public's interest. Blockchain technology, in particular Ethereum, bears the potential to provide a platform on which sustainable solutions to problems of exactly this kind can be built. This project is an attempt to build a small aspect of such a possible solution, namely a marketplace for energy demand, on Ethereum in the form of smart contract.

1.1 Core Idea

The most immediate solution to decentralizing demand aggregation and redistribution seems to be to simply build a blockchain-equivalent of the service provided by one of the companies mentioned in 0.3 . However, while the Ethereum Virtual Machine can execute Turing Complete code, it still imposes strong constraints on what a program can do in practice. Most notably:

- Ethereum contracts cannot run on their own, and all code executed must end in finite time. All functions must be fulfilled as a response to finite function calls, and a careful incentive structure must be put into place such that there are always parties with an interest in executing the relevant functions.
- Executing code on Ethereum is extremely expensive. Executing optimization algorithms on-chain is hard to impossible to keep profitable.

Because of these reasons, and because of our limited time budget, we decided to provide a *platform* on which demand could be bought and sold, and leave the expensive process of optimizing demand management to be done off-chain by the buyer.

For this purpose, our DApp introduces two new kinds of tokens:

1. Flex Offers (ERC721 compliant): Non-fungible offers of demand that are sold on the market. Each Flex Offer is uniquely associated with a set of data containing information on how much demand it offers, in which time frame it can offer this demand, and so on.
2. Flex Points (ERC20 compliant): In the end, all money generated by selling Flex Offers is given back to the parties that created the Flex Offers. However, not all Flex Offers will get bought on the market. In order to reward the mere *offering* of flexibility, and not only its selling, offerers are rewarded with an intermediary fungible currency called "Flex Points" which can later be traded back to Ethereum.

A person owning a energy-consuming machine can offer flexibility by putting a Flex Offer onto the market, for which people owning an energy-generating machine can bid Ethereum for. The person that bid the most at the end of the bidding period receives the Flex Offer, and can freely choose to enable the demand at a convenient time within the time window given by the Flex Offer. The person that created the Flex Offer is rewarded with Flex Points.

When the person offering flexibility decides to trade her Flex Points for Ethereum, she will receive the same portion of Ethereum currently stored in the contract that makes up her Flex Points on the contract. The Flex Points are then burned.

1.2 An Example

Supposed Joe Mustermann is in ownership of a washing machine and some cloth to wash. Joe needs the washing to be done by the next day but is not concerned when exactly is it done between this evening and tomorrow morning. Joe has a flexible asset that can change its load pattern over a given time. Instead of turning it on at a random time Joe decides, Joe can offer this flexibility to Flexwise. For this flexibility, Joe will be rewarded with points in which he could later convert into money.

Like Joe, many flexibility asset owners can sell their flexibility and this flexibility potential will than be available within Flexwise.

On the demand side, imagine a solar farm that is selling its electricity to the grid. It has signed a contract to supply a certain amount of electricity to a buyer for a time period. The solar farm estimated the supply amount by using the weather forecast. As the time to act on the contract draws near, there is a change in the weather forecast and there will be more sun than usual. To re-balance the agreed upon power, the solar farm can buy flexibility from Flexwise.

2 Overall Flow

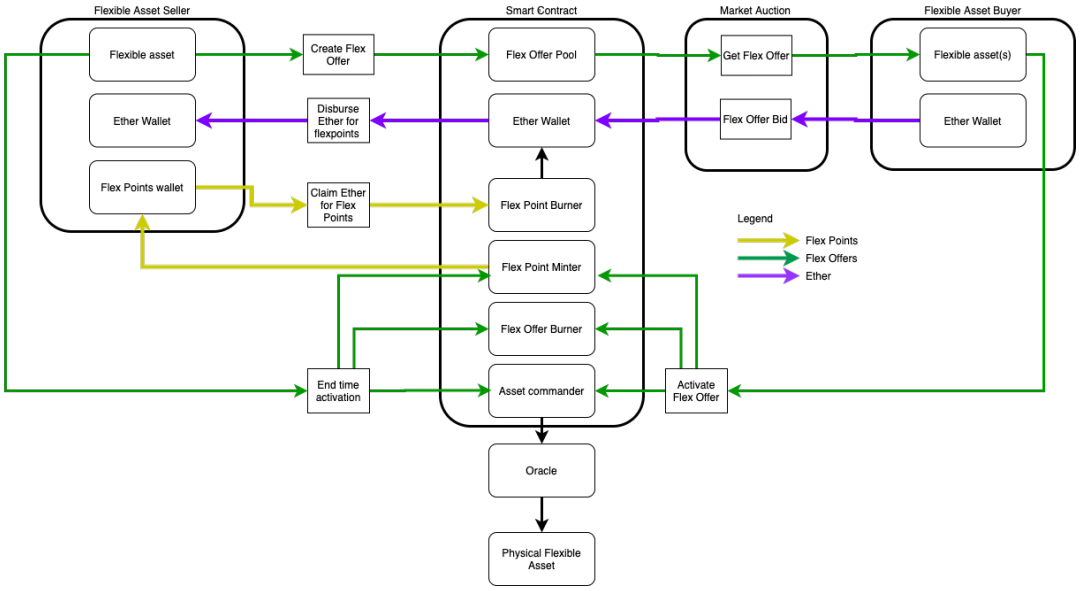

Figure 1: Overview of Flexwise

Flexwise comprise of three main interacting parties. The first party is the flexible asset owner. A flexible asset is any asset that has the ability to shift consumption or generation over a time period. This could include dish washers, washing machines, batteries etc. The owner of the flexible asset can formulate and provide a flexible offer in the form of a non-fungible token. This token is termed as a Flex Offer.

2.1 Flex Offer Market

A Flex Offer is designed as a non-fungible Ethereum token using the ERC721 standard. Each Flex Offer contains information that describes the flexibility it provides. When an asset owner creates a Flex Offer, he gives up the right to manually operate his machine for the time period of the Flex Offer. This Flex Offer is then put into the Flex Offer pool where buyers can bid for the Flex Offers. Flex Offers are made available to buyers and final ownership is being decide by way of an open auction.

2.1.1 Flex Offer Lifespan

A Flex Offer goes through a linear lifespan that is illustrated by figure 2. When a Flex Offer is created, the contract first checks that the start time is sufficiently far

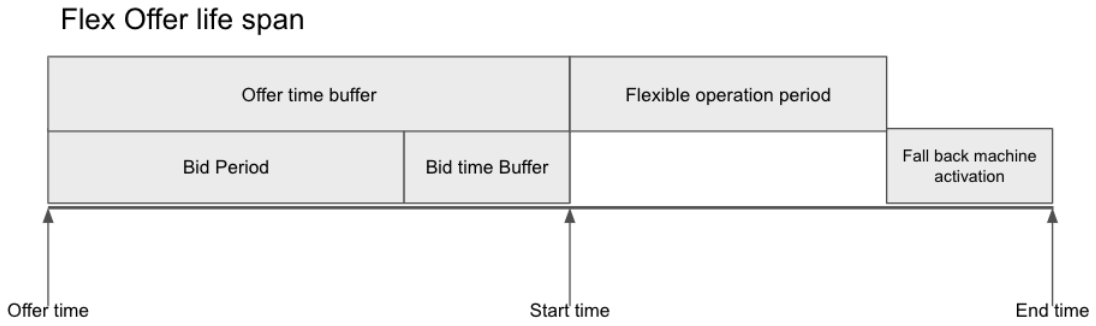

Figure 2: a Flex Offer token lifespan

away which allows room for the Flex Offer to be auctioned off. Once the Flex Offer is available, any buyer will be able to bid for the Flex Offer. The Flex Offer will be assigned ownership to the next highest bidder as long as it is within the bid period. During each bid, buyers have to stake their bid in Ether. If a bid is being surpassed, the Ether that was staked in the previous bid would be returned to the previous highest bid. Once the bid period is over, the person owning the Flex Offer will assume control of the flexible asset for the agreed upon time period. The Ether staked by the last highest bidder would thus be transferred permanently to the smart contract. A bid time buffer is implemented to ensure that buyers have sufficient planning time with confirmed successful bids to balance their demand.

During the flexible operation period, the owner of the Flex Offer can choose whenever the flexible asset will be activated. If the owner did not activate it till the end of the flexible period, the asset may activate itself automatically. Once the flexible asset is being activated, the Flex Offer has completed its life cycle and is burned.

2.2 Flex Points

At the same time of Flex Offer burning, Flex Points are being minted and credited to the Flex Offer creator. Flex Points are fungible ERC20 tokens that form the incentive structure for providing Flex Offers. Flexwise's goal is to use demand response to meet the need for flexibility, therefore value is placed on the offering of Flex Offers and not the eventual price being obtained by the Flex Offer at the auction. To calculate the amount of Flex Points being minted for each Flex Offer, the following equation is being used.

$$\text{Flex Points} \propto \left\lfloor \frac{t_{idle}}{t_{total}} e_{total} \right\rfloor \tag{2.1}$$

This gives value to long shifting periods and high power consumption/generation.

Flex Points are awarded to the Flex Offer creators only upon the activation of the flexible asset. This is so that Flex Offers that do not keep their contract promises can be penalised by means of a smaller amount of Flex Points minted for the Flex Offer.

2.2.1 Flex Point Market Place

As Flex Offers are being created and consumed, the contract would accumulate Ether from the auction. It would also correspondingly have minted Flex Points to the Flex Offer creators. At any point of time, Flex Point owners will be able to exchange their Flex Points for Ether. For the exchange, each Flex Point will be valued as a proportion to the total number of Flex Points minted by the contract. This fraction will be multiplied the total amount of Ethers in the contract wallet to eventually get the amount of Ethers that would be exchanged for the Flex Points. The following the the equation governing the exchange rate:

$$\frac{\text{Flex Point to exchange}}{\text{Flex Point total supply}} \times \text{Ether total supply in contract wallet} \tag{2.2}$$

By construction, this manner of withdrawal is designed to regulate the supply and demand for Flex Offers. When there is a low supply for Flex Offers, the demand of Flex Offers would cause the Flex Offers to be auctioned off at a higher price. This would increase the value of Flex Points in Ethers. This increase in value of Flex Points would in turn drive up incentives for flexible asset owners to offer more Flex Offers since this is the only way to get Flex Points. The increase in supply of Flex Offers will then cause the auction price to drop. By means of open market regulation using the Flex Point exchange rate, the contract was design such that the amount of supply for Flex Offers will follow the demand, removing the need for a coordinating intermediary.

3 Implementing Proof of Concept

In implementing the above concept, the team has decide to scope the project to include functions that would demonstrate the key working principles of Flexwise. In this chapter, the report details the key scoping decisions made by the team. It will also describe the technology stack the team used so that interested parties may replicate a similar system.

3.1 Project Assumptions and Scope

In order to keep the project within a manageable scope, the team decided to make the following assumptions:

3.1.1 Load Profiles

As a basic experiment in offering flexibility, all Flex Offers are assumed to be in the form of a single square step with power = height and the duration = width of square wave. The team believes this to be a fair approximation as any arbitrary load profiles can be generated by a combination of Flex Offers. This Flex Offers would then likely have to be sold as a package but the mechanism of this is out of scope of this project.

3.1.2 Defaulting Offers

The team assumes that Flex Offerers always follow up on their promise and there will be no halfway interruption of Flex Offers where the owner of the asset turns it on manually or goes offline. In production, one possible solution to the Flex Offer defaulting problem would be to set aside a number of Flex Offers so that the Flex Offers can be replaced in case of default. Alternatively, a heavy fine can be imposed to make it very expensive for asset owners to default on their contracts. If is however of note that penalisation should not be done for emergency situations.

3.1.3 Time Spans

To allow for faster debugging, demonstration and simulation of market trends, the time periods within the lifespan of a Flex Offer 2 are shortened. As of the current project implementation, the offer time buffer is set to 60 seconds while the Bid time Buffer has been set to 10 seconds. In actual deployment, these time can be varied to fit the needs of different flexibility market requirements. As an additional feature, Flex Offers with different offer time buffers and bid time buffers can be offered as separate products that can be traded in the Flexwise decentralised market place.

3.1.4 Oracles

We assume the existence of oracles that can verify claims made by participants about the state of the outside world. In particular, we assumed the claims of devices turning on to be correct without verifying it.

At the point of writing, there are no such oracles in existence. A potential approach would be to turn all electricity meters into oracles that can verify such claims on request. This wouldn't remove the potential for fraud completely, but at least decreases it to the level that is generally accepted today (as utility companies rely on the accuracy of this claims as well).

Similarly, we have not fleshed out methods which which devices creating the demand could interact with the smart contract, or how an actual user interface of such a device could look like. Perhaps in the future, new houshold devices will be connected to a blockchain by default; another option would be to provide a third-party device that could be placed between the washing machine and the power grid.

3.1.5 Profitability for Participants

Care was taken to make the main beneficiaries of all function calls the callers themselves, and to make no party dependent on the action of another (given the assumptions). However, for no step was the price of executing code on the Ethereum Virtual Machine taken into account, which would likely eclipse the monetary gain from creating a Flex Offer. The ratio between monetary return and extra effort spent by the user creating a Flex Offer is questionable as well: There are few household devices for which the cost of electricity is a concern as-is; to expect the monetary return of the Flex Offer, which would make up at most a fraction of this cost, to warrant thinking about the parameters of a Flex Offer is unrealistic.

This doesn't doom the project completely though. As Electric Vehicle adoptions increase and more households install heat pumps and battery storage for their solar cells, these new devices may start becoming significant flexible assets. Devices outside of households with much higher load requirements and fewer time constraints could also turn out net positive. Off-chain, there have been many successful projects with a similar mechanism such as [5, 6]. Furthermore, upgrades planned for Ethereum that decrease contract code execution prices substantially, potentially making the return net positive even for smaller devices.

3.2 Implementation

The Flexwise DApp is implemented in Solidity and runs on Ethereum. It comprises of two main contracts: the Flex Offer contract and the Flex Point contract.

3.2.1 Flex Offer Contract

The Flex Offer contract is the main contract in Flexwise. It holds the pool of Flex Offers and also the Ethers given by the bidders. In addition, the Flex Offer contract generates and manages the Flex Point contract. The Flex Offer contract holds several data structures to store the information needed for Flexwise to function. Flex Offers are saved in a struct **(flex_offer_data)** with the following fields:

- **Address** of the owner of the flexible asset
- **Power** draw of the asset
- **Duration** of the power draw
- **Start time** of the flexibility period
- **End time** of the flexibility period
- **Current bid price** on the Flex Offer

These Flex Offer data structures are in turn stored in a mapping that maps the Flex Offer id to a Flex Offer data struct.

Another mapping implemented in the project is one that links an external address to an array of Flex Offer ids. These Flex Offer ids are the Flex Offers that this particular address created.

The Flex Offer token inherits OpenZeppelin's implementation of the ERC721 standard [7] and extends it with the following functions:

1. mint_flex_offer_to(uint power, uint duration, uint start_time,uint end_time)

- Requirements to transact
  - Time of creation and flexibility time period must respect the offer time buffers
- Event emitted
  - flexOfferMinted event that returns the minted Flex Offer id and the address which created the Flex Offer

2. BidForFlexOffer(uint256 flexOfferId)

- Requirements to transact
  - Time of bid must respect the bid time buffers
  - Bid must be higher then the current bid attached to the Flex Offer
- Event emitted
  - flexOfferBidSuccess event that returns the minted Flex Offer id, the address which gave the bid and the amount of Ether the bid contained

3. ActivateFlexOffer(uint256 flexOfferId)

- Requirements to transact
  - Caller of this function must be the current owner of the Flex Offer
  - Time of activation must be within the flexibility period in the Flex Offer
- Event emitted
  - flexOfferActivation event that returns the minted Flex Offer id, the address that activated the Flex Offer and the Flex Points generated from the activation.

4. EndTimeActivate(uint256 flexOfferId)

- Requirements to transact
  - Caller of this function must be the creator of the Flex Offer
  - Time of activation must be after the end of flexibility period (This is the case where the bidder bought the flexibility but did not activate the machine. The original owner is then free to operate the machine after the flexibility period is over.)
- Event emitted
  - flexOfferActivation event that returns the minted Flex Offer id, the address that activated the Flex Offer and the Flex Points generated from the activation.

5. GetMyTotMintedFlexOffers(address _address) This function returns the total number of Flex Offers created by the caller's address

6. GetMyFlexOffer(address _address, uint256 i) This function allows the caller to retrieve each Flex Offer id by index

3.2.2 Flex Point Contract

The Flex Point contract is generated by the Flex Offer contract and serves to manage the flow of Flex Points within flexwise. The Flex Point contract will not hold Flex Points at any time as they will be given to the Flex Offer creator once minted. Also, when the Flex Points are being traded in for Ethers, the Flex Points will be burned in the process. The Flex Points contract inherits its properties from the ERC20 standard implementation by OpenZeppelin [8] and have the following functions as extensions.

7. ClaimEthWithFlexPoint(uint pointAmount)

- Requirements to transact
  - The caller must own equal or more Flex Points that he/she is claiming
- Event emitted
  - ethForFlexPoint event that returns the caller address, the ether amount being claimed and the amount of Flex Points used to claim the ether.

3.3 Front-end design

The graphic interface demonstrates how an user can interact with the smart contract. It mainly consists of three columns: overview of the market, creation of Flex Offers, and the bid interface. Upon entry to the Dapp, MetaMask will pop up and ask for connecting to the Dapp with a wallet.

The overview of the market provides users with the essential information about the market of Flex Offers, as shown in Figure 3. This page displays some important statistics of the smart contract: the total number of Flex Offers and Flex Points, the amount of cryptocurrency (in Wei) possessed by the smart contract, and the exchange rate from Flex Point to Wei. More importantly, it shows the recently created Flex Offers. Each offer has four status: selling, pause, active, and expire. When a user creates a Flex Offer, it will directly be put into the market for sale and thus in the "selling" state. A countdown according to Figure 2 is set to show the deadline of bid for the Flex Offer. The history of the bids for the offer is displayed next to the time line (moving the mouse above the bid shows the bidder address). Once the bid is closed, the Flex Offer is switched to "pause" which is a time buffer to terminate the bid process. Then, the offer is in the "Activate" state, meaning that the successful bidder of the Flex Offer can choose to activate the Flex Offer, i.e., turn on the machine. A countdown is set to indicate the remaining time for the owner of the Flex Offer to activate it. Once the Flex Offer period passes or the owner of the Flex Offer do not turn on the machine on time, the Flex Offer turns to the state "Expire".

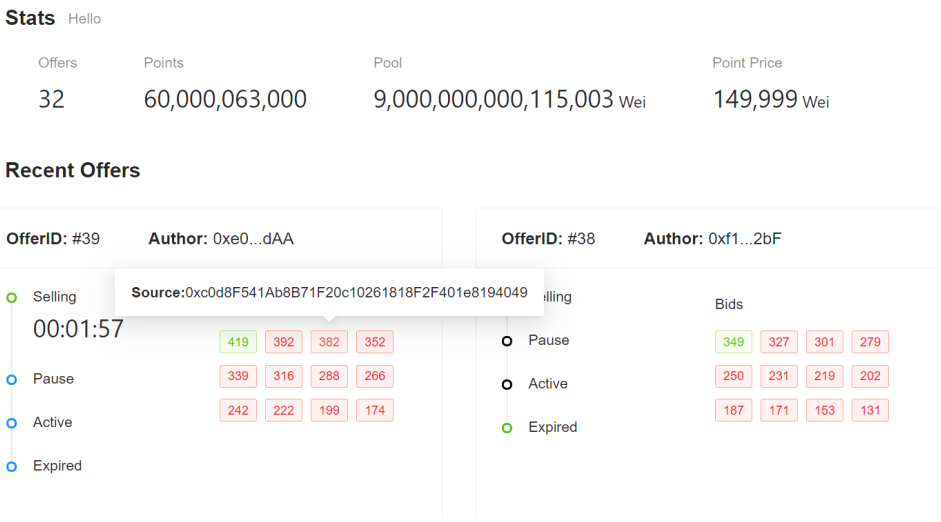

Figure 3: Market of Flex Offers

The bid page helps users to bid for Flex Offers. Through a search box (with the the Flex Offer ID as search key), the user can get the information about the target Flex Offer (see Fig. 4). If the user wants to own the Flex Offer, he/she should place a bid. The creation of a bid is a transaction that sends the bid price from the user wallet to the smart contract. As a result, the user should have enough amount of cryptocurrency to bid for a high price. What's more, the new price should be superior to the current bid. Otherwise, the transaction would fail and no Wei is sent to the smart contract. If the user places a successful bid, he/she becomes the owner of the Flex Offer, and the previous bid will be returned to the associated bidder.

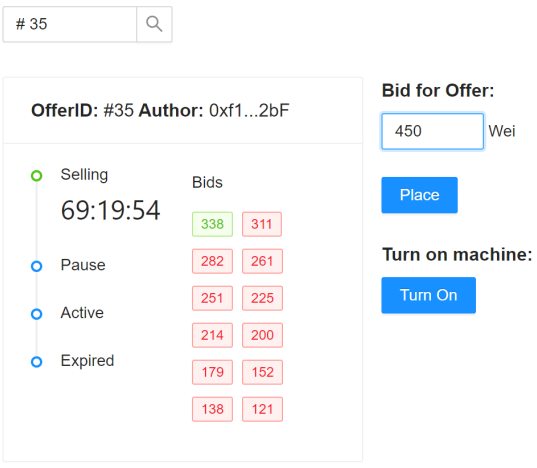

Figure 4: Bid for Flex Offers

The Flex Offer page allows users to create Flex Offers and keep track of them. Therefore, this page consists of two main components: a form for creating Flex Offers and a table summarizing the Flex Offers created by the user, as illustrated

by Fig. 5. To create a Flex Offer, the user fill the form by providing the power of the machine, the duration of the operation, and the start and end time of the flexibility period. As shown by Figure 2, the start time should leave enough time for bid, and the duration of the operation should be smaller than the flexibility period (i.e., end time minus start time). This is checked by both the front-end and the smart contract. In case the user bypasses the front-end verification to send an invalid Flex Offer, the transaction would fail. Once a new Flex Offer is created successfully, it will be added to the table below the form. This table summarizes the Flex Offers created by the user to keep track of them. Apart from the information provided by the user upon creation of Flex Offers, the table also shows the ID, the status, the bid price, and the current holder of Flex Offers. Right after its creation, a Flex Offer has the status "wait for bid". After a bidder places a price for the offer, the status changes to "bidding". The bid price and the bidder address are then updated accordingly. A countdown is set to indicate the time that the creator of the offer can turn on the machine if the holder of the offer does not activate it on time or the offer is not sold. When the countdown goes to 0, an alert window will pop up notifying the user that he/she can activate the associated Flex Offer. The status of the Flex Offer then changes to "to be activated", and a link "Activate" is shown in the action column. Clicking on the link, the user will activate the Flex Offer, and the status becomes "manually activated". When the machine receives the message from the blockchain to operate, the status becomes "running" and the starting time will also be shown. After the washing machine completes the task, the status becomes"accomplished".

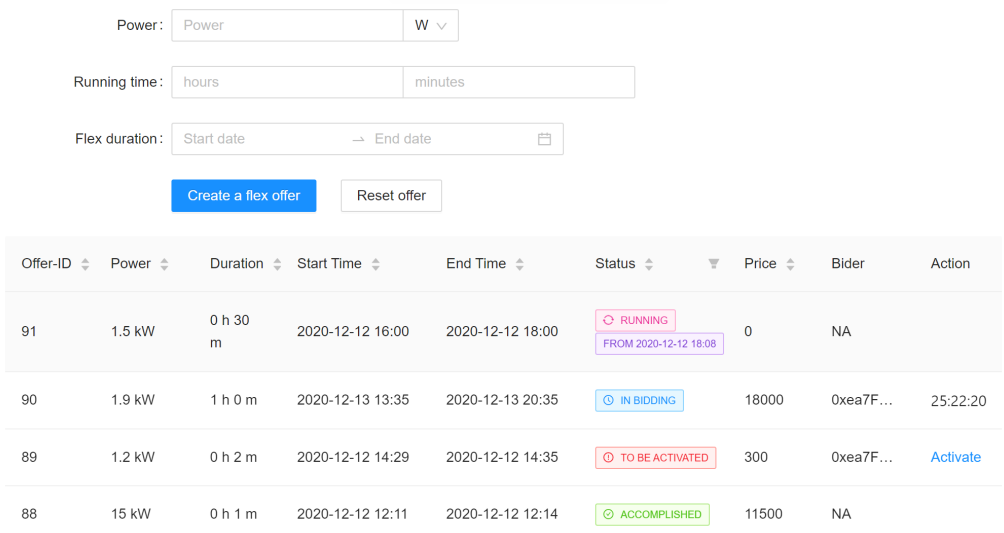

Figure 5: Seller of Flex Offers

The Flex Offer page also contains a virtual washing machine for illustration. This machine is created based on the code [9]. It contains several parts: a door, some functional buttons, and three clothes initially put aside the machine. The door can be opened/closed, and the clothes can be moved into/out of the chamber. The washing machine is set to listen to the blockchain to simulate the real process.

When a Flex Offer is activated, the smart contract will emit an event. Then, the machine will check if the activated offer is contained in the table of the user's Flex Offers. If yes, it will operate for the required time.

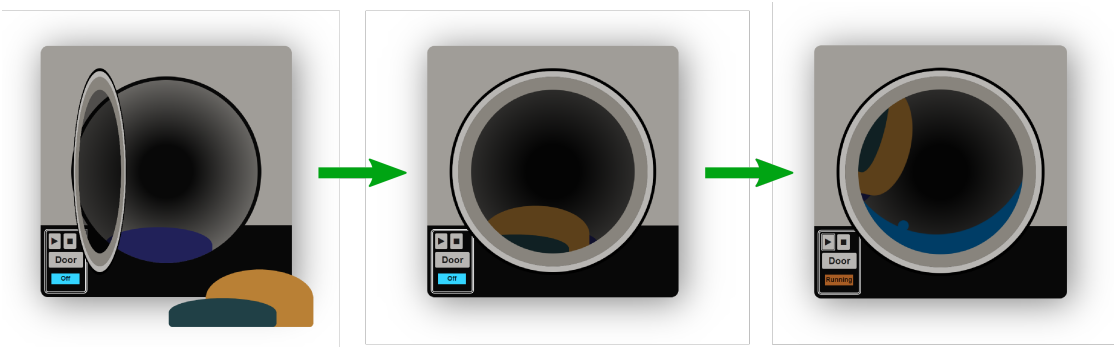

Figure 6: Virtual washing machine

3.4 Deployment and Simulation

3.4.1 Private Ethereum Blockchain

We have setup a private Ethereum blockchain and deployed the contracts to it. Anyone may connect to it using the following credentials:

RPC URL	http://ganache.sgyger.com/
Port	80
Chain ID	5777

We provide private keys for accounts containing some Ethereum by clicking on "Show Keys" in the web gui.

3.4.2 Auto Bidders and Auto Offerers

In order to test our system with multiple market participants, we've written bots to simulate both bidders and sellers. The bidder bots (called "autobidders") react to events emitted by the contract. For each Flex Offer, they randomly select a target bid price, and then start bidding against each other in small steps until their target bid price is surpassed. The seller bot (called "autoofferer") creates new Flex Offers every couple of seconds.

Conclusion

In this project, our focus was two-fold. On the one hand, we explored the problem of flexible energy provision. We designed and laid down a first solution to handle Flex Offers in a decentralized manner. This benefits from the advantages of the blockchain technology which allows for both security and anonymity. On the other hand, we explored various tools for the development of decentralized applications, namely Solidity, Truffle, Ganache, MetaMask, React, as well as different packages for interacting with the blockchain such as `Web3.js` and `Web3.py`. Through this project, we understand the general process for Dapp developments and learned the necessary tools by doing. At the end of the project, we managed to implement a functional proof of concept with the main designed features.

In the future work, one could elaborate the idea of handling the flexible energy consumption by focusing on a more detailed and mature design. Technical feasibility of cooperating with the IoT technology can be further investigated, which would open a large potential for future developments. Furthermore, one could look into the economic and social effects of the system, especially the Flex Offer buyer's financial profits, the market scale of Flex Offers, and the behaviors of different participants in the system. The associated studies could help discover potential weaknesses of the system and improve the design. In this respect, advanced and large-scale simulations and surveys would be necessary and helpful.

Bibliography

[1] Simon Mueller, Peerapat Vithayasrichareon, and Enrique Gutierrez. *Data amp; Publications*. 2018. URL: https://webstore.iea.org/download/direct/1041?fileName=Status_of_Power_System_Transformation_2018.pdf.

[2] Adrian Whiteman. "Renewable energy highlights (1 July 2020)". en. In: (), p. 2.

[3] Peter Feldhaus et al., eds. *Transformation of Europe's power system until 2050*. Jan. 2010. (Visited on 11/27/2020).

[4] "Electricity storage and renewables: Costs and markets to 2030". en. In: (2030), p. 20.

[5] *VPP explained: What is a Virtual Power Plant?* Aug. 2020. URL: https://www.next-kraftwerke.com/vpp/virtual-power-plant.

[6] *Leap. The marketplace for grid flexibility*. URL: https://leap.energy/.

[7] *Open Zeppelin'es ERC721 Implementation*. URL: https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC721.

[8] *Open Zeppelin'es ERC20 Implementation*. URL: https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC20.

[9] *7/26 CSS Code Challenge - Washing Machine Animation*. URL: https://codepen.io/benjyring/pen/bXBNWW.