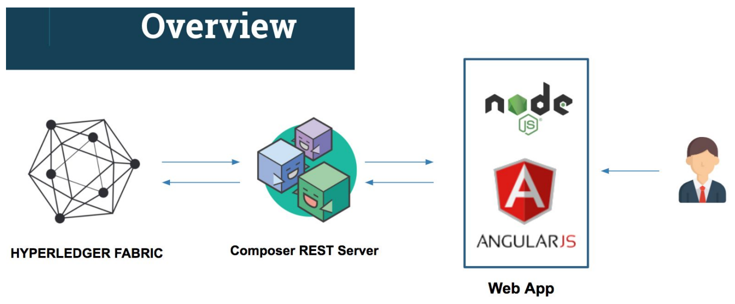
DoxChain - Technical report

Group 3 - Project D
Gabriel Vilen - 4743237 | Guangzhe Huang - 4725824 | Stefan Stojkovski - 4735943
Tamara Covacevich - 4637097 | Valentin Gérard - 4744705

Problem statement

We believe the EU ETS is a good way to reduce climate change but still some problems remain in the current system. It is centralized, companies have to go through an auction market which is regulated by the authorities. The coverage is small, only about 50% of the emission are regulated. In order to reduce the emission effectively, we need a decentralized system with its governance applied through consensus which guarantees transparency, trust and democracy.
Our mission is to build such a system for the industrial area of the port of Rotterdam. We built a prototype of this system using the hyperledger blockchain technology which allowed us to create a custom network. It's built to answer all of the stated problems so emission trading can become easier in practice and easier to audit. This way, even small companies can participate actively and get incentives to reduce their emission.

Architecture



We have implemented an underlying blockchain network using Hyperledger Fabric where we define organizations and channels.

Then on top of that we have connected Hyperledger composer where we define the participants, assets, transaction logic and query logic. By using the Composer with the appropriate network cards for authorization, we instantiate a Node.js REST Server which exposes APIs for communication with the Blockchain network.

Finally, we have created a sample AngularJS frontend application which is used by the client to perform the transactions on the Blockchain and to display the data from it.

Next, we will dive deeper into each layer and the implementation.

Implementation details

To deal with the privacy issue, we create separate channels which act as separate blockchain instances, and within these channels only companies from a specific geofence area can operate.

Moreover, to make sure that the right entities have access to the right resources, there are cryptographic keys that are created for each of the companies. These keys are issued by the Certificate Authority (CA) and in general the keys can be divided in:

- Network admin key (network card) - for the network admin user who has access to all resources associated with his company. This user then issues other keys with specific permissions to other users
- Network users key (network card) - for the specific users within a company with restricted access according to the company policy

The crypto keys enable the regulator to know who did what changes and it makes the audit process easier.

1. Hyperledger Fabric network layer

The underlying layer of each Hyperledger Blockchain solution is the Hyperledger Fabric network. According to the requirements we would need to create separate organization representing each company and channels representing the geo fence areas.

For the purposes of our Proof of Concept project we have decided to create a network with 6 organizations representing 6 companies and 2 channels representing 2 separate geo fence areas.

The configuration for the network is defined in .yaml files:

1. crypto-config.yaml - definition for crypto files creation for the orgs and the orderer
2. configtx.yaml - definitions for the sample network and the relationships between the components
3. Docker-compose-couch.yaml - definitions for all the containers that need to be started for each of the components of the network

For setting up the network we have created a script deploy_network.sh which does all the necessary steps to deploy our network. These steps include:

1. Generate certificates for each organization using the cryptogen tool

2. Generate orderer genesis block
3. Generate channel configuration transaction
4. Generate anchor peer updates for each of the organizations
5. Starting docker containers for each of the components for our network
6. Setting up the orderer, channels and each organization joining the channel
7. Updating anchor peers to make sure that there is communication between the organizations
8. Sample chaincode installation for testing purposes on each peer
9. Chaincode instantiation on the channel which initializes the chaincode on the channel, sets the endorsement policy for the chaincode, and launches a chaincode container for the targeted peer.
10. Doing simple queries to make sure components are communicating
11. Invoking a sample transaction to make sure transactions are being written

After making sure the network is successfully set up and all components are communicating between each other, we need to deploy Hyperledger Composer on each of the peers. To do so, we have created deploy_composer.sh script which does the following:

1. Removing any previously created network cards which are crypto certificates used to access the blockchain network
2. Creating network cards for each of the organizations
3. Importing the network cards for installing composer for each of the organizations
4. Installing Composer runtime for each of the organizations
5. Retrieving crypto certificates for network admins for each organization
6. Starting the business network
7. Creating network cards for for network admins for each organization
8. Importing the network cards for for network admins for each organization
9. Pinging the network on behalf of the network admins for each organization
10. Instantiating REST server with the credentials of one organization

2. Hyperledger Composer

The Composer layer acts as a connection-layer between the underlying Fabric network and Client application. It defines all the business logic and exposes this over an easy-to-use REST API that the client can use to interact with the blockchain network.

After setting up the network, we build a .bna archive file that is used to create the business network definition of the Blockchain. To do this, there are several things that need to be defined:

Model file

The model file `org.emission.network.cto` contains the assets, participants, and transactions that are exposed to the client over the REST API. Below are the most important models;

Assets:

`Ett` - an Emission Trade Token that represents the tradable emission of a company

`Market` - an instance of a market

Transactions:

`Buy` - Called when a company wants to buy an amount emission from its market

`Sell` - Similar to Buy but when a company wants to sell emission to its market

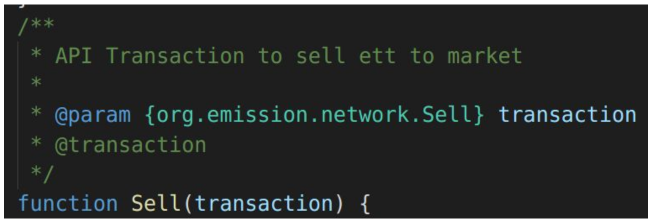
Participants:

`Company` - A company added to the network, can Buy and Sell emission to the market

`Regulator` - A regulator that can regulate, add and remove companies in the network

Script file

The file `logic.js` executes the business logic of the network. This is where the models, defined in the model file, are implemented in code (node.js). The script file can be thought of as a chaincode, smart contract. It implements the above defined transactions as functions, as well as private fields which cannot be accessed over the API.

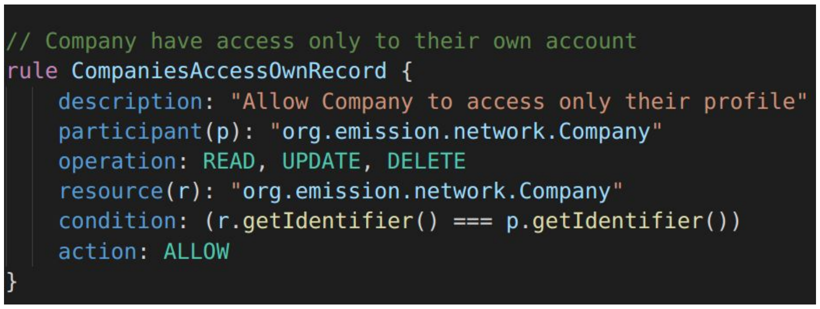


```
/**
 * API Transaction to sell ett to market
 *
 * @param {org.emission.network.Sell} transaction
 * @transaction
 */
function Sell(transaction) {
```

Fig. The @transaction annotation tells Composer that this method is a transaction and will be exposed over the REST API.

Permission file

The permissions.acl defines the permissions of the participants in the network. For instance, a company should not be able to access other companies' Ett, or tamper with the market. In this way, the permission control is easily modeled and no manual checks has to be implemented in the logic file, the permissions are ensured by Composer.

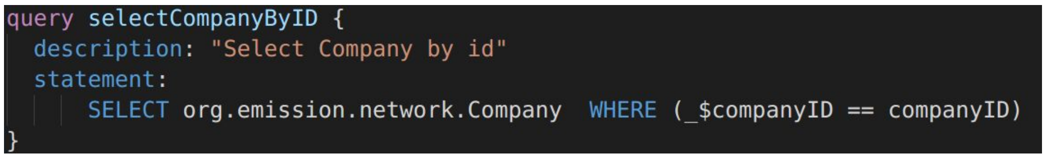


```
// Company have access only to their own account
rule CompaniesAccessOwnRecord {
    description: "Allow Company to access only their profile"
    participant(p): "org.emission.network.Company"
    operation: READ, UPDATE, DELETE
    resource(r): "org.emission.network.Company"
    condition: (r.getIdentifier() === p.getIdentifier())
    action: ALLOW
}
```

Fig. A permission that restrict companies to only access their own profile.

Queries file

The queries file implements queries to the blockchain network that can be called from the REST API and within the script, logic, file. The queries are written in SQL-similar syntax. In this project they are called in the script file to get an object from a reference, for example a company from it's referenced companyID.



```
query selectCompanyByID {
  description: "Select Company by id"
  statement:
      SELECT org.emission.network.Company  WHERE (_$companyID == companyID)
}
```

Fig. A simple query to select a company where the companyID matches the given companyID.

Endorsement policy

In our solution the endorsement policy is defined in such a way that all organizations must endorse a transaction before it is written to the blockchain, i.e. all companies have to agree before anything gets written in the blockchain.

When all of these files are defined, we generate the business network archive (.bna file) that contains this logic. The .bna file is then used when deploying the business network.

3. Web APP

The Web APP servers as a blockchain explorer that provides detailed information about the emission market, company dashboard, etc. The Web uses AngularJS framework for frontend development and Node.js for backend development. The listing in the appendix displays the project structure.

The index.html lives at the root of front-end structure. The index.html file primarily handles loading in all the libraries and Angular elements. The controllers folders contain AngularJS controllers which control the data of AngularJS applications (data binding, data presentation, etc.). All HTML templates are under the folder views.

The Web communicates with Hyperledger Composer via REST APIs, which are a set of instructions to connect to and interact with the deployed blockchain and business network. The web client sends a HTTP request to perform queries or transactions. Then, the REST server returns the results in JSON format. Consequently, our web users can do operations and transactions easily on DoxChain Web platform.

Appendix

Below transaction flow diagram for Sell and Buy can be found.

SELL TRANSACTION DOXCHAIN

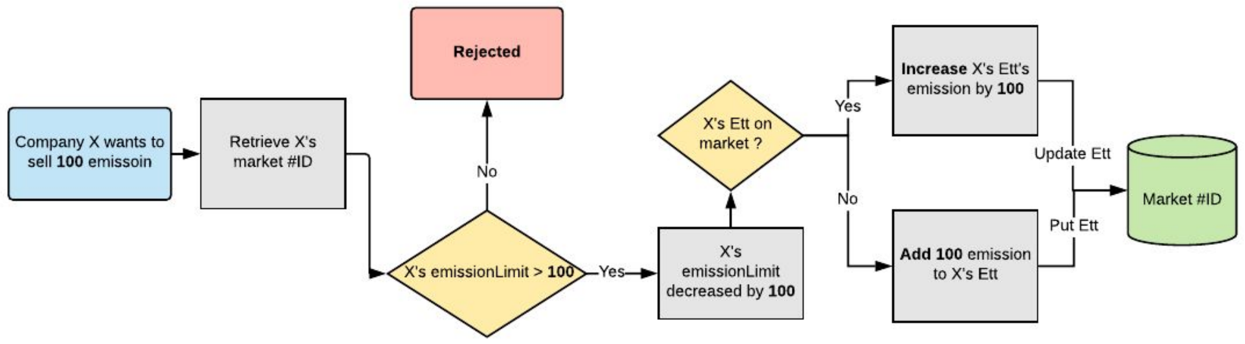

Company X wants to sell **100** emissoin
Retrieve X's market #ID
X's emissionLimit > **100**
Yes
No
Rejected
X's emissionLimit decreased by **100**
X's Ett on market ?
Yes
No
Increase X's Ett's emission by **100**
Add 100 emission to X's Ett
Update Ett
Put Ett
Market #ID

Fig. Transaction logic of a Sell transaction where company X wants to sell 100 emission.

BUY TRANSACTION DOXCHAIN

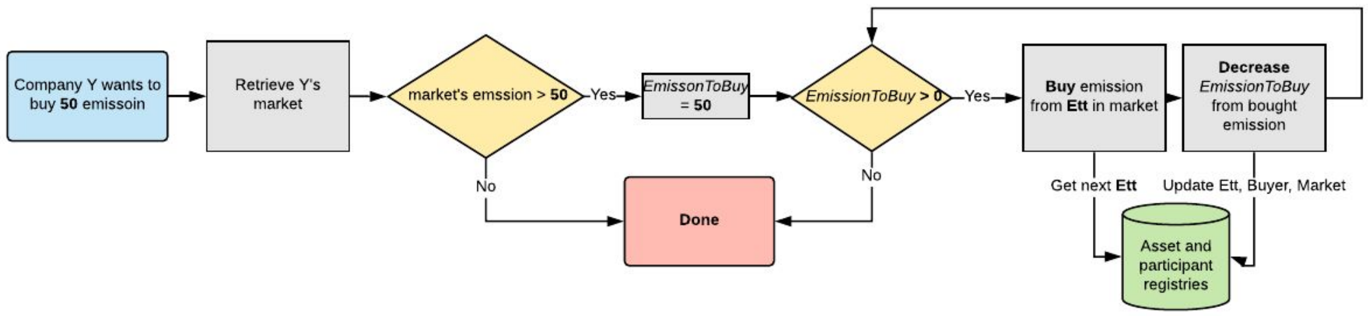

Company Y wants to buy **50** emissoin
Retrieve Y's market
market's emssion > **50**
Yes
No
EmissonToBuy = **50**
EmissionToBuy > **0**
Yes
No
Done
Buy emission from **Ett** in market
Decrease *EmissionToBuy* from bought emission
Get next **Ett**
Update Ett, Buyer, Market
Asset and participant registries

Fig. Transaction logic of a Buy transaction where company Y wants to buy 50 emission.

```
----- index.html
----- server.js
app/
----- controllers/
---------- CompanyCtrl.js
---------- LoginCtrl.js
---------- RegulatorCtrl.js
----- services/
---------- AuthService.js
---------- CompanyService.js
----- js/
---------- routingConfig.js
----- app.js
views/
```

```
----- company.html
----- home.html
----- regulator.html
assets/
----- css/
----- image/
```
Listing. Web Project Directory Structure

Reference

1. Hyperledger Fabric Tutorial
http://hyperledger-fabric.readthedocs.io/en/release/

2. Hyperledger Composer Documentation
https://hyperledger.github.io/composer/introduction/introduction.html

3. The EU Emissions Trading System (EU ETS)
https://ec.europa.eu/clima/sites/clima/files/factsheet_ets_en.pdf

4. Emission Trading
https://en.wikipedia.org/wiki/Emissions_trading