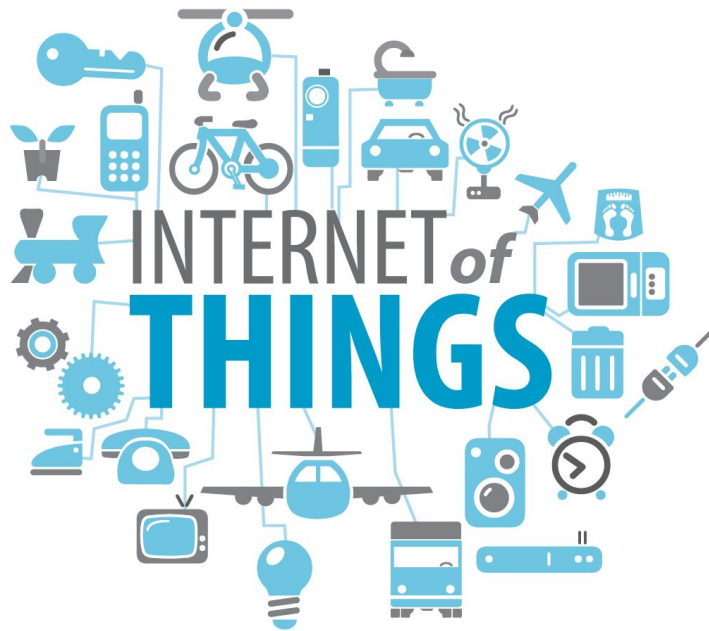


What techniques can ICR3ATE use to make targetwise permitting trustworthy and more cost efficient while creating more economic value?



**HAN** UNIVERSITY  
OF APPLIED SCIENCES

**ICR3ATE**  
getting things prototyped

Opdrachtgever: ICR3ATE  
Auteurs: Nick van Oostrum, Luuk van der Voort en Job Wibbens  
Minor Smart Industry  
Collegejaar: 2020-2021  
Semester: 1

<b>Chapter 1 Plan van Doen</b>	<b>3</b>
<b>Chapter 2 Preliminary research</b>	<b>6</b>
2.1 Micropayments	6
2.2 Immutability of data	7
2.3 Trust inside the network	8
<b>Chapter 3 Blockchain database</b>	<b>11</b>
3.1 Performance	11
3.2 Business	12
3.3 Remarks	12
3.4 Conclusion	13
<b>Chapter 4 Micropayments</b>	<b>14</b>
4.1 Comparison	15
4.2 Conclusion best suitable	16
4.3 Banking of Things (FINN)	21
4.4 Interledger	25
4.5 Stellar	29
4.6 Raiden	34
4.7 Nodle	38
4.8 Dash	42
4.9 Hyperledger Quilt	50
4.10 Paypal	52
4.11 Mollie	54
4.12 Telefonica	55
<b>Chapter 5 Smart Contracts</b>	<b>58</b>
5.1 Comparison table	58
5.2 Conclusion	59
5.3 DAML on top of our HyperLedger fabric	60
5.4 Smart contract of Ethereum	66
5.5 Hyperledger Fabric's Chaincode	68
<b>Chapter 6 Business plan SennetTa</b>	<b>79</b>
<b>Chapter 7 Conclusion</b>	<b>81</b>
<b>Literature list</b>	<b>83</b>
<b>Appendix 1</b>	<b>88</b>
<b>Appendix 2</b>	<b>90</b>
<b>Appendix 3</b>	<b>91</b>

# Chapter 1 Plan van Doen

## Inleiding

Het originele onderzoek vanuit ICR3ATE had een te grote technische complexiteit. Hierdoor hebben wij samen met Manfred van der Voort het onderzoek zodanig vormgegeven dat wij met onze beperkte technische vaardigheden toch van voldoende waarde konden zijn voor het onderzoeken en uitwerken van een technisch bedrijfsconcept.

Om gedegen onderzoek uit te voeren zijn wij begonnen met het opstellen van een Plan van Aanpak. Al snel werd duidelijk dat Manfred van der Voort, van ICR3ATE, meer van een Plan van Doen was. Zodoende hebben wij ons Plan van Aanpak verandert in een Plan van Doen. De belangrijkste onderdelen van dit Plan van Doen zijn de aanleiding van het onderzoek, een beschrijving van de opdrachtgever, de hoofdvraag, de subvragen en de doelstellingen.

Gedurende het project hebben wij de hoofdvraag en subvragen als rode draad gehouden. Tijdens het project zijn wij ook geïntroduceerd aan Mohammed Chohra, Software Developer bij ICR3ATE, om verder onderzoek te doen m.b.t. de technische uitwerking van het concept. Omdat hij Engels spreekt, hebben wij een groot gedeelte van ons onderzoek uitgevoerd en uitgewerkt in het Engels.

## Aanleiding

Het traject van vergunningen aanvragen binnen de agrarische sector is niet optimaal ingericht. Vergunningen voor bijvoorbeeld veestallen worden op dit moment verleend aan de hand van de hoeveelheid m<sup>2</sup> van een bepaald type om te voldoen aan de milieuregels. Dit wordt ook wel middelen gericht vergunnen genoemd (InfoMil, z.d.). Om beter te kunnen meten en weten wat nou daadwerkelijk de emissie uitstoot is kan er gebruik worden gemaakt van sensoren die heel precies kunnen meten wat de uitstoot is. Wanneer vergunningen verleend worden aan de hand van deze daadwerkelijk uitstoot is er sprake van doelgericht vergunnen.

De verwachting vanuit ICR3ATE is dat de vergunningverlening- en handhaving met betrekking tot de publieke ruimte steeds strenger en omvangrijker wordt. Dit komt vanwege de verwachting dat steeds strengere regels en normen zullen worden opgelegd door de overheid (InfoMil, z.d.-b). Door doelgericht te vergunnen kan alle ruimte binnen de regels en normen gebruikt worden zodat de boer optimaal van zijn ondernemers ruimte gebruik kan maken.

## SennetTa

ICR3ATE heeft een concept ontwikkeld waarmee dit doelgericht vergunnen mogelijk wordt. Dit concept heet SennetTa, ofwel a Sensor Network with Trust Added to it. SennetTa gebruikt blockchain technologie in combinatie met IoT-sensoren om een platform te creëren waarbij data vertrouwelijk en betrouwbaar opgeslagen kan worden. Deze informatie kan vervolgens verhandeld worden m.b.v. micropayments. Door blockchain technologie in te zetten wordt het volgende mogelijk:

- Onweerlegbaarheid van sensordata tijdens opslag
- Veiligheid van sensordata tijdens meten en datatransport
- Registratie van data interpretatie raamwerk, inclusief correcties a.g.v. bijvoorbeeld meteorologische omstandigheden, maar ook sensor verloop (en herijkingen)
- Registratie van sensordata representatie raamwerk, inclusief statistische correcties en correlaties
- Mogelijkheid tot verwaarden van de hoogwaardige sensor data, inclusief de raamwerk gegevens (transparantie)

- Mogelijkheid tot het uitvoeren van smart contracts voor economische transacties a.g.v. betrouwbare sensorwaarden

Om SennetTa ook financieel aantrekkelijk te maken wil ICR3ATE de volgende twee zaken waarmaken:

- Het verlagen van de operationele administratieve kosten middels smart contracts.
- Verhogen van de opbrengsten van het netwerk middels micropayments voor de afname van sensor data.

### **De opdrachtgever**

ICR3ATE is in 2015 opgericht door Manfred van der Voort. ICR3ATE bestaat uit een groep van IoT experts, ontwikkelaars, consultants en projectleider. Zij helpen bedrijven, organisaties en professionals met innoveren door het ontwikkelen van baanbrekende prototypes. Hierbij is een nauwe samenwerking met de opdrachtgevers erg belangrijk voor ICR3ATE.

### **Hoofdvraag:**

Welke technieken kan ICR3ATE gebruiken om doelgericht vergunnen vertrouwelijk en kosten efficiënter te maken en hier economische waarde mee toe voegen?

### **Subvragen**

1. Op welke manier kan er economische waarde toegevoegd worden voor de boeren?
2. Hoe kunnen micropayments worden ingeschakeld om data te verwaarden?
3. Op welke manier kan er hogere sensor betrouwbaarheid worden gerealiseerd?
4. Op welke manier kan er een betere data onweerlegbaarheid worden gerealiseerd?
5. Welke database technieken zijn het meest geschikt voor SenneTTa?

### **Doelstellingen**

#### **Informatie doelstelling**

De informatiedoelen van dit onderzoek bestaan uit het onderzoeken van;

- Initiatieven waarbij hoge sensor data betrouwbaarheid, de onweerlegbaarheid van data en IoT datastream central computing gecombineerd worden. Belangrijk hierbij is het zoeken naar gedocumenteerde implementaties.
- Literatuur waarbij bovenstaande is bereikt zonder blockchain implementatie.
- Literatuur waarbij bovenstaande geïmplementeerd is middels micropayment en blockchain.
- IoT implementaties met behulp van BigChainDB. Hiervan zou een prototype voor SenneTTa ontwikkeld kunnen worden.
- IoT implementaties van IOTA waarmee het verwaarden van sensor data mogelijkheid gemaakt wordt. Bijvoorbeeld via hun Data Marketplace.

#### **Bedrijfsdoelstelling**

ICREATE wil een nieuw product creëren waarmee zij een nieuwe markt willen betreden met een nieuw verdienmodel. Hiermee willen zij de operationele administratie verlagen. Daarnaast willen zij vertrouwen toevoegen aan low cost netwerken. Tot slot willen zij bijdragen aan het Schone Lucht Akkoord en milieuverandering tegengaan.

ICR3ATE wil het volgende bereiken met dit onderzoek:

- Verhogen van de opbrengsten van het netwerk middels micropayments voor de afname van sensor data.
- Mogelijkheid tot verwaarden van de hoogwaardige sensor data, inclusief een schematisch plan om transparantie te bevorderen.
- Het verlagen van de operationele administratieve kosten middels smart contracts waarvoor Blockchain een goede basis biedt”.

## **Methoden en technieken**

### *Deskresearch*

Desk research betekent letterlijk onderzoek dat je vanaf je bureau uitvoert. Je probeert antwoorden te zoeken in bestaande informatie. Deskresearch kan je opsplitsen in twee soorten, namelijk interne deskresearch en externe deskresearch. Bij interne deskresearch wordt er gekeken naar informatie die in de organisatie zelf aanwezig is. Bij externe deskresearch wordt er gekeken naar informatie die buiten de organisatie is, zoals bijvoorbeeld op internet (Scriptium, 2018). In dit onderzoek ligt de voorkeur bij academische bronnen.

Voor dit project wordt er in eerste instantie gebruik gemaakt van interne deskresearch. Er wordt onderzocht welke informatie er al is. Vervolgens wordt deze informatie aangevuld met de resultaten van externe deskresearch. Hierbij wordt er gebruik gemaakt van

### *Fieldresearch*

Ter aanvulling op het deskresearch wordt er een interview gehouden bij FINN, Banking of Things. Dit is een initiatief van de ING om IoT betalingen autonoom te maken. Het doel van dit gesprek is om resultaten van het deskresearch te toetsen en om op nieuwe inzichten te komen.

Tot slot worden prototypes getest in het multidisciplinair lab van ICR3ATE. Dit is echter wel afhankelijk van de corona maatregelen.

## **Zoekstrategie**

Er worden betrouwbare zoekmachines gebruikt om er zeker van te zijn dat er waardevolle en betrouwbare kennis wordt verzameld. In dit onderzoek is er gebruik gemaakt van Google Scholar en de databanken van HAN studiecentra. Aanvullende informatie is via het reguliere Google verkregen. Daarnaast is er zowel in het Engels als in het Nederlands gezocht om het aantal resultaten te vergroten. Ook is er gebruik gemaakt van de literatuurlijst van gevonden bronnen om meer bruikbare literatuur te vinden.

## **Bepaling bruikbaarheid en selectiemethode**

Voor dit onderzoek zijn er geen bronnen gebruikt die zijn gepubliceerd voor 2014. Wel is er een bron gebruikt waarvan de publicatiedatum onbekend was. Omdat deze bron is gebruikt om een tijdloos begrip te definiëren is dit door de onderzoekers goedgekeurd. Bij de eerste zoekresultaten zijn de uittreksels gelezen. Dit is gedaan om vast te stellen of de inhoud van de bron de lading van dit onderzoek dekt. Daarnaast is er gebruik gemaakt van de zoekfunctie om voor ons relevante begrippen te zoeken in de tekst en hier globaal omheen te lezen. Daarna zijn de auteurs van de artikelen onderzocht. Hierdoor worden onderzoeken die zijn geschreven vanuit een vooroordeel uitgesloten.

## **Samenwerking met ICR3ATE**

Zowel in dit onderzoek als bij ICR3ATE wordt er gewerkt aan de ontwikkeling van SennetTa. Het is dus noodzakelijk dat er nauw contact wordt gehouden met ICR3ATE om bevindingen met elkaar te delen. Om dit te bereiken wordt er elke week een contactmoment ingepland. Daarnaast worden documenten online gedeeld met behulp van Google Drive. In deze documenten kan iedereen aanpassingen doen en opmerkingen plaatsen.

In de eerste twee weken wordt er vooral onderzoek gedaan naar de bestaande informatie bij ICR3ATE. Dit dient als basis voor het verdere onderzoek. Om een goed beeld te krijgen van de organisaties zullen er ook meetings worden gehouden met medewerkers van verschillende afdelingen.

# Chapter 2 Preliminary research

## 2.1 Micropayments

### **What are micropayments?**

According to Párhonyi, Nieuwenhuis and Pras (n.d.), micropayments are transactions of small amounts of money at low transaction costs. Micropayments are processed very quickly and there is very little chance of the money being lost or stolen. It can be used, among other things, to facilitate ad-free content and to protect it against spam. It can also support Peer-to-Peer networks (Chiesa et al., 2017).

There are different micropayment systems. In order to know which properties a system needs to offer a certain solution, different characteristics have to be taken into account (Párhonyi, Nieuwenhuis & Pras, s.d.):

### **Token-based of account-based**

Token-based systems use electronic money such as cryptocurrencies. Account-based means that people have an account with, for example, a bank. This enables them to transfer money to other people through the bank.

### **Usability**

This relates to the degree of ease of use for both existing and new users/customers. This is usually expressed in the form of the interface and the underlying hardware and software systems.

### **Anonymity**

The anonymity is only relevant to the customers.

### **Scalability**

This relates to the possibility of increasing the number of users and the volume of transactions. Is it necessary for the system to grow rapidly or will the number of users and transactions remain the same?

### **Validity**

This refers to whether a payment system can carry out transactions without a digital contact with a third party such as a bank or trader as in the case of cryptocurrencies. Online validity means that there is always a third party involved in the transaction. Semi-online validity means that sometimes a third party is involved. Offline means that no third party is needed, as for example in the case of cash payments.

### **Security**

Security must ensure that transactions are securely transferred from one party to another. The most important points to take into account are immutability, authentication, authorisation, data integrity and confidentiality.

### **Interoperability**

Systems are interoperable if they can work together with other systems without problems. This is therefore very important, because you want micropayments to be easy and automatic, so systems must be able to cooperate well with each other. Standardisation can help with this.

## 2.2 Immutability of data

### What is immutability of data?

According to Lumen (n.d.) immutability is the idea that data or objects should not be modified after they are created. Once a file is closed, it cannot be modified. Immutability makes it impossible to falsify or manipulate data stored in a network. Achieving immutability through blockchain will increase trust between organisations since there will be a shared source of truth (Taçoğlu, n.d.).

Achieving immutability through blockchain will change the auditing into a cost-effective, efficient and quick process. There is no need for a third party, like a bank or government, to protect and validate data and transactions (Doubleday, 2018) This cost-effective process and micropayments will allow SennetTa to have small transactions while keeping profitable.

### How does BigchainDB ensure the immutability of data?

BigchainDB achieves immutability by a combination of seven ways, as stated by Bigchaindb (n.d.) themself:

#### 1. Replication

Data is shared and cut into different parts which will be saved in different places. The amount of replications can be set by the customer. The more replications, the more difficult it will be to change or delete all replicas.

#### 2. Internal watchdogs

All nodes check their data for changes. If an unallowed change happens, a pre-programmed action will be taken. If a valid block is deleted, the nodes will automatically put it back for example.

#### 3. External watchdogs

Customers can also choose to let self chosen third-parties monitor and audit their data. If their data is publicly-readable, the public can check the data and search for irregularities.

#### 4. Cryptographic signatures

Cryptographic signatures are used throughout BigchainDB as a way to check if messages (transactions, blocks and votes) have been tampered with enroute, and as a way to verify who signed the messages. Each block is signed by the node that created it. Each vote is signed by the node that cast it. A creation transaction is signed by the node that created it. Transfer transactions can contain multiple fulfillments (one per asset transferred). Each fulfillment will typically contain one or more signatures from the owners.

#### 5. Full or partial backups

BigchainDB allows their customers to make backups. This can be done online as well as offline. Online could mean in another blockchain or cloud storage system. Offline could mean on paper, USB or even CD.

#### 6. Strong security

Security policies can be adopted by node owners.

#### 7. Node diversity

To avoid any actor or event to control enough nodes, BigchainDB makes sure nodes have high diversity. There are four kinds of node diversity:

- Jurisdictional diversity

Nodes are spread across multiple places with different legal jurisdictions. This way it will be hard to use legal means to control enough nodes.

- Geographic diversity  
Nodes are physically spread across the globe. In case a natural disaster happens like an earthquake or flood, there will be enough nodes left to prevent problems.
- Hosting diversity  
Nodes are distributed across multiple hosting providers to prevent one hosting provider being able to influence enough nodes.
- Operation system diversity  
A bug in an operating system can be exploited to control nodes. Therefore, nodes use different operating systems so a bug in one operating system won't be enough to control enough nodes.

BigChainDB is more than only tamper-resistant; once you store data, it can't be changed or altered (Chohra, personal communication, November 6, 2020). They use the old centralized database (MongoDB) and add to it the decentralization in each server and chaining the data in blocks for immutability and security. BigchainDB is less complex and more clear compared to their competitors. This is because they use a valid path which is based on the MongoDB principles.

## 2.3 Trust inside the network

### **What is Byzantine Fault Tolerance?**

#### *Byzantine Fault Tolerance*

BigchainDB 2.0 is Byzantine Fault Tolerant. This makes the BigChainDB 2.0 database more decentralized and more secure (BigchainDB blog, 2018).

For the correct operation of BigchainDB 2.0 the nodes need to reach consensus on the next block. If some nodes crash, fail, go off protocol or show any form of arbitrary behavior that is called a Byzantine fault. Byzantine Fault Tolerance ensures BigchainDB 2.0 that the database can still come to consensus even when one third of the nodes fail in any way (BigchainDB blog, 2018).

To provide the database with Byzantine Fault Tolerance BigchainDB 2.0 uses Tendermint.

#### *Tendermint*

From BigchainDB 2.0 onwards Tendermint will be building the blocks, doing the voting and reaching consensus in the database, in contrast with BigchainDB doing that in the early versions of BigchainDB (BigchainDB blog, 2018). Consensus or overall system reliability is reached by using Tendermint software to ensure that all nodes or state machines in the system reach agree on the current state of the BigchainDB. Tendermint has no notion of what is inside the state machine (BigchainDB blog, 2018). Implementation of the state machine is done by BigChainDB software, BigchainDB code is used to construct valid transactions and to check if a given transaction is valid (BigchainDB blog, 2018).

### **How can BFT be useful for SennetTa?**

Using BigChainDB 2.0 with Byzantine Fault Tolerance lets SennetTa be functional even when some nodes in the database fail to register data. Failures of nodes or data placed in the database by ICR3ATE won't result in a total shutdown of SennetTa.

#### *Improved trust*

How can Tendermint add value to the SennetTa project?

Working with Tendermint in BigchainDB 2.0 has two big benefits for the SennetTa project. One is performance, one is scalability.

#### *Performance*



With the current implementation of Tendermint, thousands of transactions can be made per second on up to 64 nodes on machines distributed around the globe (Buchman, 2016). Latencies will be mostly in the one to two second range. The time between which a transaction is submitted to the time it's included in a finalized block is around one or two seconds for a global network (BigchainDB blog, 2018).

### *Scalability*

Using the consensus protocol Tendermint makes it possible to scale and interoperate blockchains. Cosmos (n.d.) is a decentralized network of independent parallel blockchains powered by consensus algorithms like Tendermint. To enable SennetTa to scale, when more and more nodes get introduced to the project, Cosmos provides the opportunity to use multiple heterogeneous blockchains (Cosmos, n.d.). The connection between blockchains is achieved through a protocol called Inter-Blockchain Communication protocol (IBC). This makes it possible for the different blockchains to transfer value or data to each other (Cosmos, n.d.).

BigchainDB is compatible with IBC because it has fast finality by using Tendermint as its consensus protocol.

IBC works as follows (Copied from: <https://cosmos.network/intro>):

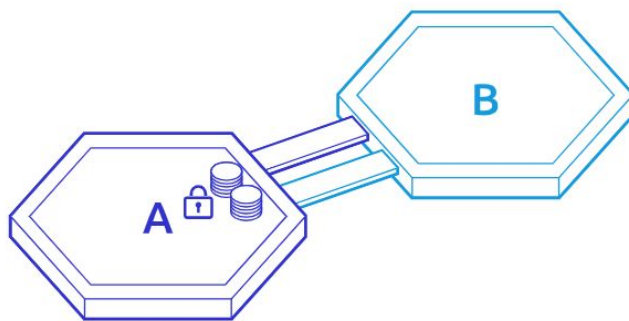
The principle behind IBC is fairly simple. Let us take an example where an account on chain A wants to send 10 tokens (let us call them ATOM) to chain B.

### **Tracking**

Continuously, chain B receives the headers of chain A, and vice versa. This allows each chain to track the validator set of the other. In essence, each chain runs a light-client of the other.

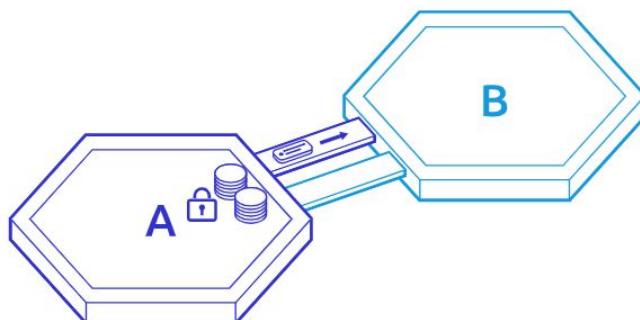
### **Bonding**

When the IBC transfer is initiated, the ATOM are locked up (bonded) on chain A.



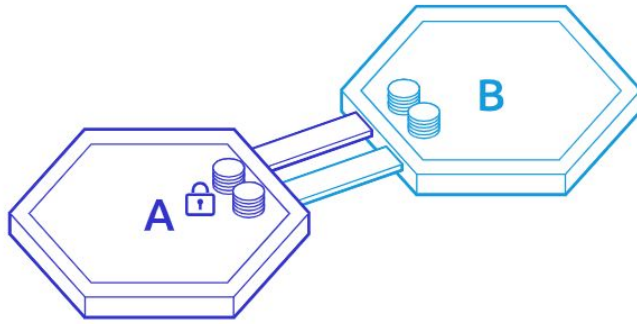
### **Proof Relay**

Then, a proof that the 10 ATOM are bonded is relayed from chain A to chain B.



### **Validation**

The proof is verified on chain B against chain A's header and, if it is valid, then 10 ATOM-vouchers are created on chain B.



Note that the ATOM that have been created on chain B are not real ATOM, as ATOM only exist on chain A. They are a representation on B of ATOM from chain A, along with a proof that these ATOM are frozen on chain A.

For a more comprehensive description of the IBC protocol see:

<https://github.com/cosmos/cosmos-sdk/tree/master/docs/ibc>

# Chapter 3 Blockchain database

This research was made in cooperation with ICR3ATE's software engineer. In this chapter, we will try to make a comparison between BigChainDB and IOTA. Both of them have their advantages and disadvantages, and both of them will be a nice solution for our new storage systems.

We will try to define the principle of each solution alone. They are all working on a decentralized storage system to have a secure, immutable, and private database, taking into account the speed and scalability.

- For IOTA, they created a new technology called Tangle to replace the blockchain structure. The tangle is not a block chain (a list of blocks which each block refers by a hash to the next one), it is more a Graph (Directed Acyclic Graph). They tried to eliminate the Miners by making any person who would do a transaction, validate two last ones. And they put that their system is scalable as many more users use it.
- On the other hand, BigchainDB has taken another approach. They use the old centralized database (MongoDB) and add to it the decentralization in each server and chaining the data in blocks for immutability and security.

Both methods are innovative, but for BigChainDB it is less complex and more clear as they use a valid path which is based on the MongoDB principles. IOTA is trying to do a new way of how the data is stored in a decentralized network.

## 3.1 Performance

When it comes to performance, the researcher Albert Fuentes Contreras has confirmed that in his thesis in 2019 has found that BigchainDb is much more efficient and faster than IOTA. As quoted by him: "In comparison with IOTA, BigchainDB reached a Throughput around 400 times better, if we compare both performances using the highest measured throughput. During the tests there were no issues with respect to the CPU overload that was experienced during the IOTA tests."

Also, IOTA have different claims that has been proven wrong, one of them is that they claimed before that their network can reach 1000tx/s, which has been proven to only 400x/s (2019), and now they claims with the last changes it gets to 1500 tx/s (and can get to 4000tx/s).

As IOTA tried to create a new way of storing data from scratch, they used different new technologies for security, storing data, and hashing which has been proven hackable and has a lot of security issues ( we talked about that at the IOTA presentation). Also, IOTA stills not a decentralized database as it has a coordinator in each of its nodes to verify transactions ( they are working on a new way to integrate Coordicide to make their node fully decentralized).

On the other hand, BigChainDB uses a more efficient way which is to not reinvent the wheel ( they use MongoDB for transactions, and add the blockchain properties on top of it).

In this performance section, BigChainDB looks more mature and they are taking an easy road to improve decentralized storage. IOTA is more a whole innovative solution. They had a lot of research papers and worked on different sides of the blockchain from creating new hashing algorithms, a new approach, a new consensus mechanism, and even a new hash algorithm.

## 3.2 Business

If we talk about business, we will talk about partnerships, real use cases of the solutions, and community.

Starting with the community, BigchainDB looks to have a bigger open source community that contribute to the project comparing to IOTA (also can be seen as a result of the IOTA innovative approach as a not simple developer can work on it, only research or people with a high level of expertise in the blockchain, cryptography, database, and security field.).

For partnership, both of them have a lot of partners in different fields. But, IOTA has much more in a different field, and they enter into more associations that are working together for the future of storage.

For the real use cases, which we mean by it the use of that database in real-life projects. IOTA is much more mature at this point. They created different projects with different partners in a different field (automobiles, electronics, supply chain), which they do more real experiments for their solution (thus the issues that have been found with it which are on the progress to overcome).

## 3.3 Remarks

IOTA has been created for the IoT field (you can get that for its name), and this is an advantage for IoT systems.

For Documentation, IOTA is much richer and has different documentations compared to BigChainDB.

In the Online presence, IOTA has much more partners, more organization, different communication channels and more online active :Youtube videos, Podcasts, Blogs,...

One of the things that can be taken into consideration is the variation of business model using IOTA coin (they have a cryptocurrency token for doing payment (especially micro-payment), so you can use their network to store data, and recompose client with tokens if a condition is valid (they did a project with Jaguar which pay drivers if they share some of the data with them such as weather and road conditions)).

- With this a lot of use cases and business models emerges.
- That does not mean that we can't use BigChainDB for this purpose, but it will be slower than using IOTA for both data management and micropayment.

One of the things that you can get from a first look of both of them, that BigChainDB is only a plugin for MongoDB and IOTA is a new and innovative way to store data in a decentralized manner (This will play a role on the future partnership with different companies as being one of the pioneer in the IOTA storage systems) (you get also the feeling that IOTA are giving a lot of wrong promises, as they are trying to solve a lot of problem at once and using new solutions each time rather than using traditional one and improve on it).

Both of them have a lot of integration solutions, which make them easy to be integrated into the application with different languages (Python, Java, and Javascript..) and they are both open sources.

### 3.4 Conclusion

Both solutions are good for decentralized storage. Their main principle is different as they attack the same issues from different angles and with a different perspective.

To compare between them is hard and only future improvements and innovation will show which is better than the other.

To do a comparison, we will create a table and see how we can use them.

We will give notes on scale from 1 to 5.

	Performance	Security	Immutability	Trust	Micropayment	Privacy
BigChainDB	5	5	5	4	4	5
IOTA	4	3	5	5	5	5

In the end, ICR3ATE has chosen Hyperledger Fabric as their database solution. This decision was made after this research, which is why it is not included in this paper.

Hyperledger Fabric has a richer documentation and a lot of partners. Similar projects like SennetTa have chosen Hyperledger Fabric as well. The combination of these factors has led ICR3ATE to the decision to use Hyperledger Fabric instead of BigChainDB or IOTA.

# Chapter 4 Micropayments

In this research paper we look at different micropayment systems. The following micropayment systems will be discussed:

- Banking of Things (FINN)
- Interledger
- Stellar
- Raiden
- Nodle
- Dash
- Quilt
- Paypal
- Mollie
- Telefonica

The different micropayment systems will be looked at via a quick introduction, a price table, links to the website and official documentation. We will discuss how it works, what the pros and cons are of the micropayment system and how it can be implemented. When available we will showcase real life use cases from existing companies.

## 4.1 Comparison

We have made measurements from the information we have worked out below. We rank 1 to 5. 1 is worse, 5 is best. This comparison table is not objective.

	Price	Easy to use	Trust in company	Security	Documentation	Speed	Remark	Average
<b>FINN</b>	<b>1</b>	<b>3,5</b>	<b>5</b>	<b>U</b>	<b>4</b>	<b>N/A</b>	<b>1<sup>1</sup></b>	<b>3,4</b>
<b>Interledger</b>	<b>N/A</b>	<b>3</b>	<b>4</b>	<b>4</b>	<b>5</b>	<b>N/A</b>	<b>2<sup>2</sup></b>	<b>4</b>
<b>Stellar</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>3,5</b>	<b>2,5</b>	<b>4</b>	<b>3<sup>3</sup></b>	<b>3,7</b>
<b>Raiden</b>	<b>3</b>	<b>3,5</b>	<b>U</b>	<b>2</b>	<b>3</b>	<b>2</b>	<b>4<sup>4</sup></b>	<b>2,7</b>
<b>Nodle</b>	<b>5</b>	<b>2</b>	<b>U</b>	<b>3,5</b>	<b>4</b>	<b>5</b>	<b>5<sup>5</sup></b>	<b>3,9</b>
<b>Dash</b>	<b>4</b>	<b>3,5</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>5</b>	<b>6<sup>6</sup></b>	<b>4,1</b>
<b>Quilt</b>	<b>N/A</b>	<b>3</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>N/A</b>	<b>7<sup>7</sup></b>	<b>3,75</b>
<b>PayPal</b>	<b>2,5</b>	<b>2</b>	<b>5</b>	<b>4</b>	<b>2,5</b>	<b>3</b>	<b>8<sup>8</sup></b>	<b>3,1</b>
<b>Mollie</b>	<b>2,5</b>	<b>2</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>3</b>	<b>9<sup>9</sup></b>	<b>3,25</b>
<b>Telefonica</b>	<b>N/A</b>	<b>3</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>3</b>	<b>10<sup>10</sup></b>	<b>3,6</b>

**Unknown: U**

**Not Available: N/A**

<sup>1</sup> FINN, it is expensive(€100-€2500), reliable company(linked to ING), security and speed unknown.

<sup>2</sup> Interledger, clear documentation, exact costs and speed unknown.

<sup>3</sup> Stellar, Good overall score, messy documentation, minimum balance(Lumen)

<sup>4</sup> Raiden, fast off-chain transactions, slower on-chain transactions. No transaction fee needed. Only 1 time deposit. Only useful if the off-chain fast transactions are deemed secure enough.

<sup>5</sup> Nodle, Nodle pays fees. Fast transactions, Unsure whether it can be used outside their app. People can't cash out Nodle Cash.

<sup>6</sup> Dash, Low transaction fees. Expensive to run a masternode, €5-€30 per month to buy a masternode hosting service. Own currency; DASH. Fast transactions. Blockchain is self funded and self governed. Good option.

<sup>7</sup> Quilt, STREAM transactions can be very useful for SennetTa. Price and speed depend on currency that is used. Built on Hyperledger and Interledger protocol. Good security with automated code scans.

<sup>8</sup> PayPal: This is a good solution if transactions are above ~5 EUR. Documentation is a bit messy

<sup>9</sup> Mollie: This is a good solution if transactions are above ~5 EUR. Documentation is

<sup>10</sup> Telefonica: Looks promising, but no prices available. Very tricky because prices could break it or make it

## 4.2 Conclusion best suitable

Looking at the table above we think Interledger, Dash, Hyperledger Quilt and Stellar are the systems that are best suitable for SennetTa's needs. Nodle falls off because its currency is unable to be cashed out, which makes it useless for the farmers that want their money in Euros instead of an online cryptocurrency. Further research can be done to develop the implementation possibilities of these systems. For Interledger and Hyperledger Quilt it is needed to find a micropayment currency or system that can be used in these protocols.

### Substantiation of the comparison and rating by topic

Below you can find the substantiation of the figure for each solution.

#### FINN

The reason why FINN scored a 1 at **price** is because you must take a subscription. These subscriptions cost between €100 and €2500. Beside subscription costs there are transaction fees of 2% - 4,75% per transaction. FINN is reasonably **easy to use** (3,5), there is a makers dashboard where you can see statistics. But there are not many use cases comparable to SenneTa. FINN is a Dutch company connected to ING with a professional team, that's the reason why they score a 5 at **trust**. There is not enough information about **security** to make a well-founded decision. FINN scores a 4 at **documentation**, the documentation is clearly arranged but could be more detailed. There is no information about the transaction **speed** of FINN.

#### Interledger

There is no clear information about the **price** of Interledger. You'll have to pay a fee to connectors of the transaction, but the amount is unknown. Interledger scores a 3 at **easy to use**, the reason for this is that they have a clear architecture, protocol (ILPv4) and cross currency payments are enabled. Interledger is not bound to a single company, currency or blockchain, that is why it scores a 4 at **trust**. Because Interledger is a blockchain based open protocol chain it scores a 4 at **security**. There is a lot of documentation about Interledger, it is well organized and easy to understand, that is why it scores a 5 for **documentation**. There is no clear information about the **speed** of Interledger.

#### Stellar

Stellar scores a 4 at **price**, this is due to the low costs (0,00001 lumen) per transaction. But there is also a transaction fee that must be paid to the facilitator of the transaction. The Stellar software makes sure you pay the lowest fee possible. Due to the Horizon API, Stellar scores a 4 at **easy to use**. The Horizon API ensures a relatively easy implementation. Stellar is a non-profit organisation supported by the Stellar Development Foundation and some big companies like IBM and Deloitte have already worked with Stellar. Therefore it scores a 4 at **trust**. A disadvantage of Stellar is the **documentation**, it scores a 2,5. This is due to the huge collection of documents, this makes it very chaotic. Because Stellar is doing a restructuring in its documentation a lot of information occurs more than once. Processing a transaction only takes a (few) second(s) therefore it scores a 4 at **speed**.

#### Raiden

Price: 3.

[The Raiden Network has no transaction fees](#), however a one-time on-chain deposit is needed to start a payment channel.

Easy to use: 3.5.

The Raiden Network provided the code for different implementation possibilities, which can be found [here](#).



Trust in company: Unknown.

It is hard to decide whether we can trust Raiden or not.

Security: 2.

Due to the Raiden Network doing a lot of their transactions off-chain, it might be less useful for the security level ICR3ATE is looking for.

Documentation: 3.

Raiden has a lot of documentation available [here](#).

Speed: 2.

The off chain transactions of the Raiden Network are super fast and instantaneous. Their on-chain transactions in the Raiden Network are slower and will take 15 second to several minutes to be confirmed.

## **Nodle**

Price: 5

Nodle pays the fees in the Nodle Cash app. It is unclear what the fees are when Nodle Cash is set up in another suite. There's a side note however. At the moment it is impossible to cash out the acquired Nodle Cash.

Easy to use: 2

Nodle Cash is easily used in the Nodle Cash app. It is unclear whether Nodle Cash can be used in SennetTa.

Trust in company: Unknown.

It is hard to decide whether we can trust Nodle or not.

Security: 3,5.

Security and performance is in the hands of Validators → Delegated Proof of Stake.

Documentation: 4.

There's clear documentation available about Nodle and it's SDK and API documentation [here](#).

Speed: 5.

Using Nodle Cash for transactions can be very fast. Nodle can process up to a million micropayments per day.

## **Dash**

Price: 4.

Dash has transaction fees that are a [fraction of a cent](#). Running a masternode costs a lot, €86060. Letting a Dash community member run the masternode for you, makes this way cheaper. €4,50-30 per month can be paid for the hosting service.

Easy to use: 3,5

Dash has a lot of information on how to set up the different transaction types they offer, which can be found [here](#).

Trust in company: 4.

Dash is funded and governed by its members and masternodes. Because the masternodes are financially invested in Dash, it is highly unlikely that they want to screw with the network.

Security: 4.

It is run in a decentralized blockchain with well thought out security measures. Dash allows financial privacy with their PrivateSend transactions.

Documentation: 4.

Dash has a lot of information available on their [website](#).

Speed: 5.

Dash InstantSend transactions are fully confirmed within two seconds. The PrivateSend transactions take some more time to process.

## **Quilt**

Price: N/A.

There's no clear information about the fees for using Quilt.

Easy to use: 3.

There's [documentation](#) available to set up STREAM to be used for payment and messaging applications.

Trust in company: 4.

Quilt is a part of the Hyperledger and Interledger protocol. Interledger is a well established protocol.

Security: 4.

Hyperledger Quilt uses automated scans to help finding zero-days and prevent vulnerabilities.

Documentation: 4.

Quilt has a lot of information available on their [website and on their wiki page](#).

Speed: N/A.

The speed of the transactions in Quilt will depend on the micropayment system or currency that is used.

## **PayPal**

*Price: 2,5*

Micropayments cost 5% of the transaction plus 0,05 EUR. The break even point is at 11,70 EUR. After that, the regular pricing will be more profitable. Paypal scores a 2,5 because it is significantly higher than some other kryptovaluta in this research. On the other side, PayPal does not have any start up costs.

*Easy to use: 2*

PayPal is not specialized in blockchain. In their documentation, there is no step by step guide on how to implement PayPal in a project like SenetTa.

*Trust in company: 5*

PayPal is a well known and secure company. You can assume PayPal will be here for the following years.

*Security: 4*

PayPal does not use blockchain technology. It uses a centralized database. However, there are no relevant cases of a security breach yet to be found.

*Documentation: 2,5*

The documentation of PayPal is not structured very well. The same information can be found on different pages. Sometimes you'll find outdated information. Sometimes you will not find information in their documentation, but you'll find it when you do a Google search.

*Speed: 3*

A transaction between PayPal accounts is quite instant. However, a transaction by credit card or iDeal to PayPal can take up to one business day.

*Average: 2,8*

The average is quite high because of the trust in the company. PayPal is a good payment processor, but might not fit in the requirements of SenetTa

## **Mollie**

*Price: 2,5*

Transaction costs with Mollie are 0,29 EUR per transaction. No set up costs. It is higher than some crypto valuta, but still considered a good option because with Mollie you don't have to deal with cryptocurrencies.

*Easy to use: 2*

Mollie is not specialized in blockchain. In their documentation, there is no step by step guide on how to implement Mollie in a project like SenetTa.

*Trust in company: 5*

Mollie is well known in the Netherlands. It is a company that'll likely stay for the following years.

*Security: 4*

Mollie does not use blockchain technology. It uses a centralized database. However, there are no relevant cases of a security breach yet to be found.

*Documentation: 3*

Mollie's documentation is better structured than PayPal's documentation. However, there is no information about using Mollie in combination with blockchain or smart contracts.

*Speed: 3*

Mollie immediately notifies users if their payment was successful. Payouts are done between 1-2 days after the transaction.

*Average: 2,8*

Mollie is a bit like PayPal. It's a good payment solution provider, but might not fit the needs of the SenetTa project.

## **Telefonica**

*Price: N/A*

Telefonica does not show any prices on their websites. This is likely because prices are in tune with the desires and needs of the customer.

*Easy to use: 3*

Telefonica has blockchain solutions with smart contracts embedded. They put a lot of effort in making the system as simple as possible. However, since it is a Spanish company, there might be a language barrier. The documentation is in English.

*Trust in company: 4*

Telefonica is the biggest blockchain provider in Spain. They are one of the biggest companies in the world in their main business, telecommunication. They have put a lot of effort in developing and marketing the blockchain platform. This indicates that they take the project seriously, and it's not a one week thing.

*Security: 4*

Telefonica runs on a decentralized blockchain, ensuring immutability, transparency and efficiency. Some big companies praise Telefonica for their security. However, these interviews are Spanish spoken, so not sure.

*Documentation: 4*

Thankfully, their documentation is in English and well structured. The information in their documentation seems very good, but it's hard to decide for non developers.

*Speed: 3*

In one interview, Telefonica is praised for their speed. This might not be a trustworthy indication because it's made by themselves.

*Average: 3,4*

Telefonica is a good option for SenetTa. One drawback could be that it is a Spanish company, and communication might be influenced by this.

## 4.3 Banking of Things (FINN)

<https://makethingsfinn.com/>

FINN, also known as Banking of Things, is a company that provides secure payments to smart devices. They use autonomous payments for smart devices and machine to machine interaction (FINN, n.d.-b). FINN provides several services, for example pay by car, pay-per-use and as said before autonomous payments (FINN, n.d.-f).

FINN provides several tools and information which can help a company to use their micropayment system. There is a toolkit, development kit, use cases, blogs, documentation, GitHub repository and they use Slack to assist you (FINN, n.d.-g).

### Autonomous payments

The most interesting things for SennetTa are autonomous payments and pay-per-use. Autonomous payments can be used for many things, for example pay for gas, food or parking without leaving the connected vehicle. Moreover, autonomous payments can also eliminate the need for customers to do in-store payments, because this payment system can automate this process (Wong & Kim, 2016).

The IoT payment platform from FINN (n.d.-a), facilitates smart devices to make autonomous payments. According to FINN (n.d.-a) the advantages of using this payment platform are increased potential customer base, increased perceived value and improvement of customer relations.

FINN has been interviewed by the research company Forrester for their report 'The dawn of autonomous finance' (FINN, 2020). An interesting Figure is shown in the report, this Figure (1) shows the difference between autonomous executions and executions done by humans.

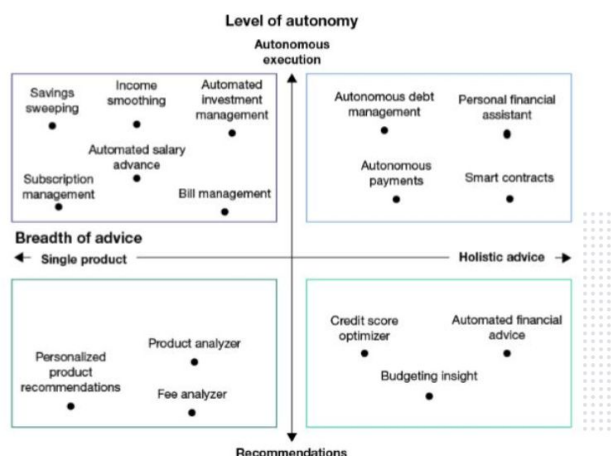


Figure 1. Autonomous Finance Includes Everything from Product Tips To Personal Financial Assistance. Taken from *Research report: Forrester's 'The dawn of autonomous finance'* by FINN, 2020 (<https://makethingsfinn.com/resources/blog/research-forrester-autonomous-payment-finance>).

### Pay per use

Beside autonomous payments, FINN also offers a pay-per-use micropayment solution. According to FINN (n.d.-i) "letting customers pay for their actual usage is called pay-per-use". Advantages of offering pay-per-use are increasing the perceived value of your business, creating long-term customer relations and it increases your potential customer base (FINN, n.d.-i).

It is also possible to make your product available as a service, called product-as-a-service, this is another usage-based business model. This model puts even more emphasis on long-term relationships with customers. In the context of SenetTa: you don't make someone pay for the data once, but you make sure that the data is always available and up to date. You can, for example, link a subscription to this service whereby customers pay a weekly or monthly fee. The advantages are increasing revenue streams, improving customer relations and giving competitive edges (FINN, n.d.-i).

### Costs

There are costs attached to the use of finn, divided into fixed and variable costs. There are three different subscriptions to choose from, namely Pro, Premium and Enterprise. All subscriptions offer (FINN, n.d.-d):

- License to use FINN platform
- Payments & Payout enabled
- Inviting & onboarding (KYC) of X suppliers
- X product types
- Support

The main differences are the amount of suppliers and products you can use, this is shown in Table 1.

Table 1  
*Differences between FINN subscriptions*

	<b><i>Pro</i></b>	<b><i>Premium</i></b>	<b><i>Enterprise</i></b>
<b>Price per month</b>	€ 100	€ 1000	€ 2500
<b>Inviting &amp; onboarding X suppliers</b>	2 suppliers	20 suppliers	Unlimited suppliers
<b>Amount of products</b>	5 products	10 products	Unlimited products
<b>Support</b>	Slack	Slack, mail & chat	24/7 Slack, mail, chat & phone

FINN. (n.d.-h)

There is also an option to try it for free, here you get a license to use the FINN platform, 1 product type with max 5 devices, testing without processing payments and Slack support (FINN, n.d.-d).

It is also possible to make trigger based payments, this also involves costs. This is only possible for pro subscriptions and above. The specifications are shown in Figure 2.

Number of IoT devices	Number of payment triggers per month	Percentage per trigger
< 100	< 40.000	4,75% *
< 1.000	< 400.000	3,75% *
< 5.000	< 2.000.000	3,25% *
< 15.000	< 7.500.000	2,75% *
< 50.000	< 25.000.000	2,25% *
> 50.000	> 25.000.000	2,00% *

\*Payment fee will be deducted at payout, transactions overview can be downloaded in the FINN dashboard.

Figure 2. Trigger based payment fees. Taken from *Get going with our IoT payment platform* by FINN, n.d.-d. (<https://makethingsfinn.com/pricing>).

### Implementing FINN

FINN developed a toolkit which can be used to add autonomous payments to IoT devices. The basis of the toolkit is the open source SDK. This SDK is available in C++, Android Things, NodeJS and Python. FINN's SDK enables trigger-based payments for smart devices, for example sensors. According to FINN it is important to think: 'if this, then that' (FINN, n.d.-c). This 'if this, then that' actions can be defined in the FINN Maker Dashboard, which will be discussed in more detail later.

In Figure 3 there is a SDK feature overview (FINN, n.d.-e).

SDK feature overview	NodeJS	Python	Java	Android Things	ESP32
Secure connection to our API	✓	✓	✓	✓	✓
Retrieving & triggering actions	✓	✓	✓	✓	✓
Essential logging	✓	✓	✓	✓	✓
Pairing with Bluetooth Low Energy (BLE)	✓	✓	✓	✓	✓
Pairing with QR Code	✓	✓	✓	✓	✓
Remote debugging	✓	✓	✓	✓	✓
Offline action queueing	✗	✗	✗	✓	✓
Configure IoT WiFi	✓	✗	✗	✓	✓

Figure 2. SDK feature overview. Taken from *IoT SDK's* by FINN, n.d. (<https://docs.bankingofthings.io/iot-sdk/overview>).

All the different programmes have their own FINN [Github](#) environment where development is possible. The most used languages for writing codes for FINN are Swift, Python, Java, C++ and Kotlin (Github, n.d.). The [Readme](#) file at Github shows more detailed how to start with the SDK from FINN, for example there are some pre-created codes which can be used.

FINN uses a communication tool called Slack, this can be used to ask questions about developing your autonomous payment system. This is the link to the Slack page:

[https://ing-bankingofthings.slack.com/join/shared\\_invite/enQtNDEyODg3MDE1NDg4LWJhNGFiOTFh](https://ing-bankingofthings.slack.com/join/shared_invite/enQtNDEyODg3MDE1NDg4LWJhNGFiOTFh)

[ZmVINGQwMTM4ZjQzMmZmZDk5ZGZiNjNIZTVjZjNmYjE0Y2MxZjU5MWQxNmY5MTgzYzAxNmFiNGU#/">ZmVINGQwMTM4ZjQzMmZmZDk5ZGZiNjNIZTVjZjNmYjE0Y2MxZjU5MWQxNmY5MTgzYzAxNmFiNGU#/](#)

There are some **hardware requirements** set by FINN, namely (FINN, n.d.-c):

- A smartphone
  - - Iphone (5s or later) with iOS 11 or higher
  - - Android device with API 21 or higher
- An IoT device powered by Raspberry Pi or another Ubuntu OS (debian) powered
- The possibility to pay with iDEAL, Giropay, or Sofort. You will make a payment of 1 cent to validate your bank account.

### Use cases

There are a few use cases displayed at the website of FINN, an example is the 'connected car'. The connected car has the software of FINN integrated in its system, one of the solutions FINN offers here is automatic toll payment on a tollway. The advantage of this is that the car does not have to stop.

Another use case of FINN is the pay-per-use tool renting with Sijperda Verhuur. The software of FINN made it possible for customers to return their rented tools 24/7. Technology in the tools and lockers calculate the actual usage and the costs are paid automatically without the intervention of people. For more use cases [click here](#).

### Pros and cons

The pros and cons according to FINN (n.d.-b) and our own research are:

#### Pros

- Easily scalable
- Bank-grade security
- Real-time data
- A percentage of the transaction amount must be paid to FINN (max 4.75%, min 2%)
- When you have the enterprise subscription there is 24/7 support
- Clear overview of statistics, products, suppliers and the possibility to link your mobile application in the makers dashboard

#### Cons

- Relatively few/comparable use cases
- Paid subscription required (min €100, max €2500 per month)
- IoT device must be powered by Raspberry Pi or another Ubuntu OS (debian)
- Not all features are available while working with NodeJS, Python and Java



## 4.4 Interledger

<https://interledger.org/>

### Overview (Introduction)

Interledger is a blockchain-based open protocol chain that enables the sending of value across independent payment networks. Due to the minimal protocol and open architecture interoperability is enabled for any value transfer system. Interledger is not bound to a single company, currency or blockchain (Interledger, n.d.-b). So while using Interledger you can send XRP to someone who wants to receive ETH and USD to someone who wants EUR (Interledger, n.d.-c). When Interledger started they made design goals, these are: neutrality, interoperability, security, simplicity and end-to-end Principle (Interledger, n.d.-d).

### Price table

There is no clear price table found. To read more about the payment part of Interledger see:

- [SPSP Pull Payments](#)
- [Payment Pointers](#)
- [Web Monetization](#)
- [W3C Web Payments](#)

### Link to the official website and documentation

Official website: <https://interledger.org/>

Documentation: <https://interledger.org/overview.html>

Github: <https://github.com/interledger>

### How it works

The center of Interledger is the [Interledger ILPv4](#), The ILPv4 stands for 'Interledger Protocol v4'. ILPv4 is the set of rules which define how nodes in the system should send values over the Interledger network. ILPv4 is a request/response protocol, these requests and responses are ILPv4 packets. When a packet goes from source to destination it is divided into multiple packets, the content of the packets is not visible for the nodes participating in the transaction. ILPv4 has three different sorts of packets, namely Prepare, Fulfill and Reject (Interledger, n.d.-c).

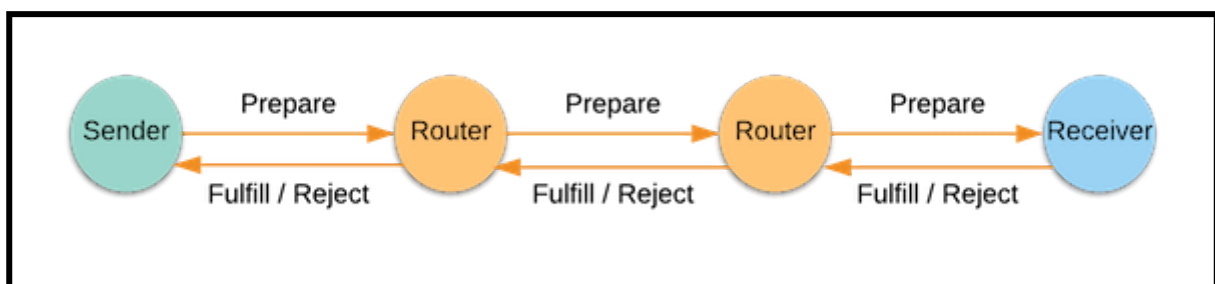


Figure 3. Prepare, Fulfill and Reject overview. Taken from *Interledger Overview* by Interledger, n.d.-c (<https://interledger.org/overview.html>).

As can be seen in Figure 3 the sender sends a Prepare packet to the receiver, the receiver sends a Fulfill or Reject packet back to the sender. When the sender receives a Fulfill packet it will continue to send the other Prepare packets until the value is fully transferred.

While processing this value transactions ILPv4 does not rely on a single payment network. It is possible to connect with an ILPv4 router at any time so you can join the network. These value transactions are being sent in very small data packets which makes the transactions fast, secure and

inexpensive (Interledger, n.d.-c). For technical specs [see here more](#). For a Protocol for Interledger Payments [see here more](#).

The Interledger Protocol can be implemented in two modes, the atomic mode and the universal mode. The atomic mode is used between parties that trust each other and the universal mode is used for parties that don't trust each other ([Frankenfield, 2020](#)).

### Architecture

Interledger is a standard way of bridging financial systems, heavily inspired by the architecture of the Internet. So it's not a blockchain, token or a central service (Interledger, n.d.-a). The Interledger architecture distinguishes 3 types of nodes, a Sender, Connector and Receiver. A node can represent different things, for example a person, a company, a device or a software program. A note from Interledger (n.d.-a): 'When a node represents a device or a software program it's usually connected to a node that represents the device's owner or the person running the software'.

When making a transaction the Sender and the Receiver are called an account. Parties with only one account are only capable of sending or receiving a packet. Parties with two or more accounts are Connectors, these Connectors can facilitate payments to or from everyone they are connected to. So Connectors provide a service of transferring packets and money, they take a certain risk while doing this. In order to justify this risk Connectors can charge fees and make a profit from these services. Due to the competition between the Connectors they try to offer the best balance between speed, reliability, coverage and cost (Interledger, n.d.-a).

The Interledger protocol suite is divided into several layers, which can be seen in Figure 4. To read more about the Architecture and layer stack of Interledger [click here](#).

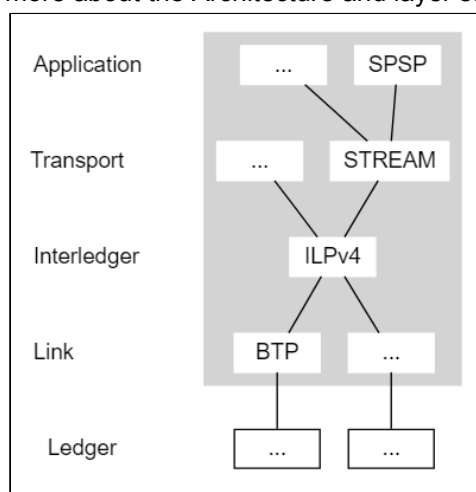


Figure 4. Interledger layer stack. Taken from *Architecture Overview* by Interledger, n.d.-a (<https://interledger.org/rfcs/0001-interledger-architecture/>).

## Implementation (Examples)

### Testnet wallets

#### Rafiki Money

A user-facing demo wallet that can make Interledger payments to a variety of peers (supports USD, testnet only). Must make an account. [More info](#)

### Connectors

#### Java Connector

An Interledger Connector implemented in Java. This project is actively maintained by [Xpring](#) and recommended for new production deployments. [More info](#)

## Rust Connector

An Interledger Connector implemented in Rust. This project is not actively maintained, but has a robust feature-set and is easy to use. [Code/info](#)

## Javascript Connector

An Interledger Connector implemented in Javascript . This project has been battle-tested in various production deployments. [Code/info](#)

## Libraries

### Interledger Java

Build ILP applications that send and receive payments natively in Java using [Quilt](#) , which is a Java implementation of Interledger.

### Interledger RS

Build ILP applications that send and receive payments natively in Rust using [interledger.rs](#) , which is a Rust implementation of Interledger.

### ILP-over-HTTP

Implementations of [ILP-over-HTTP](#) , which is a bilateral communications protocol for server-to-server ILP connections.

- Java: [ILP-over-HTTP Link](#)
- Rust: [ILP-over-HTTP](#)
- Javascript: [ilp-plugin-http](#)

### Interledger STREAM

Reliably send packetized money and data over Interledger using [STREAM](#) .

- Java: [STREAM Java](#)
- Rust: [STREAM RS](#)
- Javascript: [STREAM JS](#)

## Use cases

There are several very diverse use cases, for example the Bill & Melinda Gates Foundation has an open source payment system for emerging markets. Interledger is used to increase financial inclusion through interoperability. Another example is Coil, they use Interledger micropayments and the proposed Web Monetization standard to let people pay for the number of pages read of an article. More of these examples can be found [here](#) (at the end of the article).

## Pros and cons

### Pros

- Easy to use with different valuta
- Reduced costs due to competition between Connectors
- Packetizing Value - splitting up larger transfers into many lower-value packets which increases the network's efficiency, security, and interoperability
- Lots of documentation
- Interledger does not rely on a single payment network

### Cons

- It is possible for everyone to connect to an ILPv4 router and join the network
- No clear overview about the costs of Interledger

- As a Connector there is a risk you will lose money
- Depending on Connectors for making transactions

**Conclusion**

Interledger is a blockchain-based open protocol chain that enables the sending of value across independent payment networks. It supports transactions between different valuta and cryptocurrencies. There is much documentation over Interledger, but it is still clearly arranged. A disadvantage is that there is no clear overview about the costs of using Interledger or the costs per transaction. Due to the competition between connectors you will likely get the best price and fastest transaction inside Interledger.

## 4.5 Stellar

### Overview (Introduction)

Stellar is an open source network for currencies and payments which handles millions of transactions between different valuta's and cryptocurrencies each day. The purpose of Stellar is to let all the world's financial systems work together on a single network. The Stellar software runs across a decentralized, open network and is not owned by a company. A non-profit company based in the U.S., named [Stellar Development Foundation](#) guides and supports the Stellar technology (Stellar, n.d.-e). Stellar is based on and depending on blockchain to keep the network in sync. Stellar says: 'it is much faster, cheaper and more energy-efficient than typical blockchain-based systems (Stellar, n.d.-e).

### Link to the official website and documentation

Official website: <https://www.stellar.org/?locale=en>

Documentation: <https://developers.stellar.org/docs/>

Github: <https://github.com/stellar/new-docs/issues>

### Price Table

To prevent ledger spam and to keep the network efficient, Stellar charges a small transaction fee and they require a minimum balance on accounts. The minimum balance is for most accounts a couple of Lumen tokens, [Lumen](#) is the built-in token of Stellar. The formula to calculate your minimum balance is:

- Minimum Balance = (2 + # of entries + # of sponsoring entries - # of sponsored entries) \* base reserve

For more info about the minimum [balance click here](#).

The transaction fee that must be paid can be calculated with the formula which is shown in Figure 5.

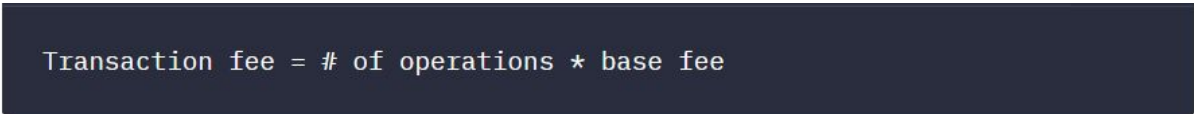

$$\text{Transaction fee} = \# \text{ of operations} * \text{base fee}$$

Figure 5. Transaction fee formula. Taken from *Fees* by Stellar, n.d.-c

(<https://developers.stellar.org/docs/glossary/fees/> ).

The base fee of Stellar for a given ledger 'is determined dynamically using a version of a [VCG](#) auction'. While making a transaction to the network, you have to specify the maximum base fee you're willing to pay per operation. So you'll never pay more than this price, but the network will make sure you'll pay the lowest possible fee based on network activity. When network activity is below capacity, you pay the network minimum, this is currently 100 stroops (0.00001 XLM) per operation (Stellar, n.d.-c).

### [Low Fees](#)

Micropayments don't work with non-micro fees: you can't transact in pennies when it costs you dimes. Stellar network fees are near zero.

### How it works

On the website of Stellar there is a learning section, this consists of 6 parts. Below there will be a short summary of each part, there will also be a link to the full text.

### Introduction

Stellar is a decentralized system that can be used for trading all sorts of money in a transparent and efficient way. Stellar is a cryptocurrency-adjacent, but the goal of the Stellar software is to enhance the existing financial system, not to undermine or replace it. The stellar native digital currency is Lumen. Stellar wants to make tradition forms of money more useful and accessible by creating

tokens. Stellar makes a digital representation of for example a euro, on Stellar this is called a 'euro token' (Stellar, n.d.-e).

#### *How does it work?*

Stellar is a system for tracking ownership, it uses a ledger to do so. Accountants used to do it. As an alternative there is a network of independent computers which is self-monitoring including rechecks. There is no central authority, but the ledgers are verified and updated every five seconds. The ledger stores two things for every account holder, namely what they own and what they want to do with what they own (Stellar, n.d.-e).

To keep everything in sync. There is a unique algorithm called the [Stellar Consensus Protocol](#). This protocol is used to be better configurable, faster and more energy efficient, for more details there is a [peer-reviewed paper](#). Nodes are the computers which run the core Stellar software. These nodes publish and check the ledgers. The Stellar network is verified by hundreds of nodes across the globe. The way the nodes communicate and the nodes themselves are [public information](#) (Stellar, n.d.-e). For more about this subject [see here](#).

#### Speed:

One-click access to content doesn't mean much if you have to wait around for a payment to settle. Stellar transactions take seconds to clear.

### **The power of Stellar**

'Stellar is an open, interoperable payment and currency system'. In this section the power of Stellar will be explained according to Stellar.

Stellar allows users to create a redeemable, tradable representation of any asset, these are called tokens. A token does not necessarily have to be a representation of a valuta or cryptocurrency, it can also be a representation of gold, shares or an hour of labour. Stellar (n.d.-h) gives the following example: 'In a sense, Stellar offers a generalized toolkit for anyone to do what [Tether](#) did for the Dollar with their USDT, or what Coinbase is doing with their [USDC](#)'.

Stellar allows senders and receivers to only send or receive the valuta/token they want. So a person can send dollars to someone while this person wants euros, Stellar ensures that this person will receive euros. This is called a path payment, the advantage of this is that both sides do not risk any exchange risks or delays. A schematic overview is shown in Figure 6.

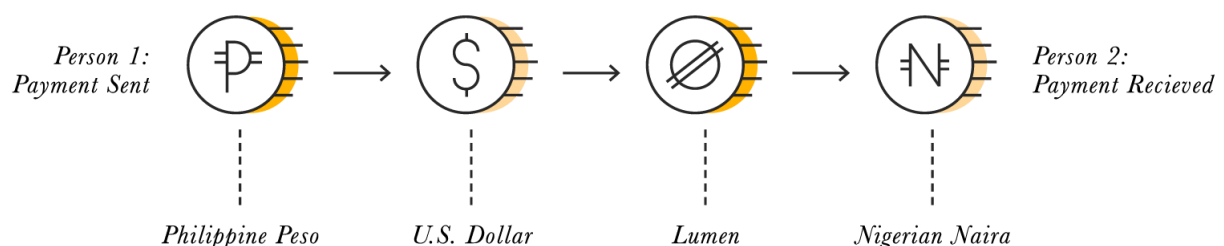


Figure 6. Path payment. Taken from *The power of Stellar* by Stellar, n.d.-h (<https://www.stellar.org/learn/the-power-of-stellar>).

To read more detailed how it works [click here](#).

### **Stellar for Remittances**

'Traditional remittances using MTOs can take days to confirm a payment and rely on logal agents to pay out in cash'. Agents and MTOs are not in real-time connection, it takes time. The banks or MTOs that can transfer money real-time are rarely able to move money to or from emerging economies. For

companies who want to serve foreign countries must establish in-country accounts and local bank relations. These banks charge fees which costs you money (Stellar, n.d.-f). This process is visualised in Figure 7.

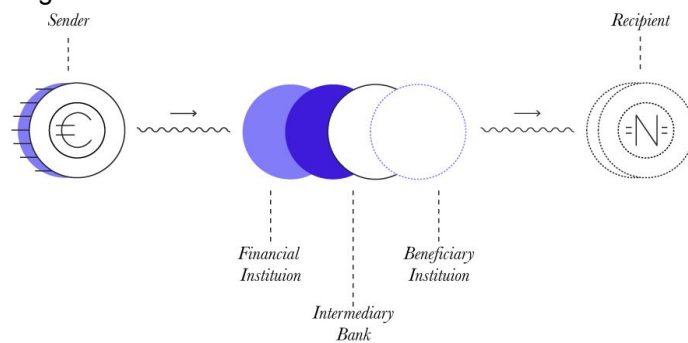


Figure 7. Traditional remittance pathway. Taken from *Stellar for remittance* by Stellar, n.d.-f (<https://www.stellar.org/learn/stellar-for-remittances>).

### How can Stellar help?

'Stellar records the movement of secure, digital representations of world currencies, helping businesses facilitate cross-border transactions without intermediaries like correspondent banks, but rather through two efficient payment standards' (Stellar, n.d.-f). For companies who use Stellar it is possible to connect directly to a network of licensed fiat (traditional currency) acceptance and distribution partners called [anchors](#). Businesses using Stellar can focus on their customers experience because the anchors take care of deposits, redemptions and compliance. Stellar (Stellar, n.d.-f) says the following things about Anchors:

- Anchors are licensed and regulated financial institutions, money service businesses, or fintech companies that provide fiat on/off ramps to the anchor country's banking system.
- Anchors often also provide 'stablecoins', one-to-one fiat-backed tokens, that users can redeem for fiat at any time.
- Anchors handle local regulatory processes such as KYC/AML.

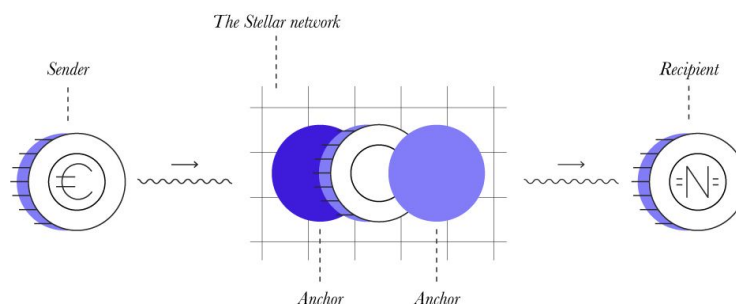


Figure 8. The anchor pathway. Taken from *Stellar for remittance* by Stellar, n.d.-f (<https://www.stellar.org/learn/stellar-for-remittances>).

In Figure 8 the anchor pathway is shown. When comparing Figure 7 and 8 you can see differences, namely shorter pathway and less institutions between the sender and the recipient.

### Anchor basics

There has already been a short introduction of Anchor basics, to read more about this [click here](#).

### Blockchain & Stellar

'Stellar uses a much more environmentally-friendly syncing mechanism than either platform and was designed specifically for remittances and payments, so it has "cashlike" delays between transactions (i.e. very short) and it's more or less free to use (transactions cost way less than a penny)'. Stellar says it distinguishes itself through connecting to real-world endpoints. By this they mean people can turn their digital representations of money into something they can actually spend (Stellar, n.d.-b).

### **Stellar Lumens**

There has already been a short introduction of Lumens in what it is in the previous text. The reason Lumens exists is because Stellar is too easy to use. Therefore there must be some costs or nominal barriers connected to the service of Stellar to prevent spam or nonsense. At this moment the minimum balance is 1 lumen on an account. The minimum fee per-transaction is 0.00001 lumen, This very low amount should ensure that people can use Stellar easily but that large scale abuse is discouraged. To use stellar you need Lumens, they are for sale on many exchanges, for example [Coinbase](#), [Binance](#) and Stellar's own exchange [Lobstr](#) (Stellar, n.d.-g).

#### **Pros and cons**

##### *Pros*

- Speed, transactions on Stellar are confirmed in 3-5 seconds.
- No pre-funding requirement, real-time transactions in digital currencies minimize working capital requirements.
- Currency support, transact in USD, EUR, NGN, MXN, BRL, ARS, and more ... or create new digital currencies for your specific needs by becoming an anchor.
- Compliant, built-in features allow unified KYC/AML via partners and travel rule compliance.
- Developer Tools, Stellar's open-source developer resources provide comprehensive documentation, SDKs, and tutorials.
- Global Scale, Stellar is a global network that can expand your company's reach to new geographies and users.
- Partnership with big names, Now that a tech behemoth such as IBM has begun implementing the Stellar payment protocol, the credibility of the company has increased even further (Shome, 2018).

- Relatively energy efficient

##### *Cons*

- Connected to a cryptocurrency (Lumen)
- High competition, there is great competition and therefore it is difficult to hold ground for Stellar. For example there is Ripple payment protocol (Shome, 2018).
- When using Stellar you must trust issuers of stablecoins TetherUSD or AnchorUSD
- Minimum balance required

### **Implementation**

To develop on Stellar you don't have to run your own Stellar Core node. There are organizations such as the Stellar Development Foundation that offer public-access API endpoints which allows you to submit transactions and query the ledger. Therefore you can first start with building your product before committing to setting up network infrastructure. To support developers there are [tutorials](#) with explanations, [SDKs](#) to make developing in your language of choice easier and a [list of operations](#) with programming rules, parameters and error codes explanation. There is also an [API reference documentation](#), which explains every resource, aggregation and error provided by Horizon, the Stellar API (Stellar, n.d.-d). Horizon is a RESTful HTTP API.

There are several channels which support developers of Stellar:

- [Community page](#) (work together with other developers)
- [Status page](#) (real-time info about network status)



- [Keybase Team](#) (place to chat with members of Stellar Development Foundation and other Stellar developers)
- [Stack exchange](#) (Knowledge base, FAQ and possibility of asking questions)
- [Developers Google Group](#) (dedicated to discussions about Core Advancement Proposals and Stellar's Ecosystem Proposal & notifications about upgrades and network-wide decisions)
- [Stellar Community Fund](#) (Crowd fund inside Stellar community)

Stellar has two key pieces of network software, namely Stellar core which tracks and adds transaction sets to the ledger and Horizon API that allows programmatic access to submit transactions and query network data. You can start developing in your language of choice by installing one of the Stellar SDKs. Stellar supports the following:

- Javascript
- Java
- Go
- Python
- C# .NET
- Ruby
- iOS
- Scala
- QT/C++
- Flutter

For more info and links to the documentation of each program [click here](#).

#### **Example of companies using it + a real use case**

There are several big companies who use the technology of Stellar, one of these companies is IBM. IBM does not use the Lumen coin but it uses the technology of Stellar in the [World Wire](#) payment platform. The goal of the World Wire project is to make it cheap and easy for financial institutions to send any currency anywhere in the world. [Read more here](#).

Also Deloitte started a project with Stellar to build a wide range of prototypes to service industries including insurance, employee management and cross-border payments. [Read more here](#).

#### **Conclusion**

Stellar is a non-profit open source network for currencies and payments between different valuta and cryptocurrencies. There is a lot of documentation about Stellar which helps you to implement and use it, but the quantity of the documentation can be confusing. Stellar is similar to Ripple, the big difference is that Stellar is a non-profit network. Stellar supports many different programming languages and is relatively easy to implement due to the Horizon API.

## 4.6 Raiden

<https://raiden.network/>

### Overview (Introduction)

The Raiden Network is an off-chain solution, enabling near-instant, low-fee and scalable payments. It works with any ERC20 compatible token. There are a lot of different ERC-20 Tokens available: <https://etherscan.io/tokens>. The Raiden Network is an infrastructure layer on top of the Ethereum blockchain. The Raiden Network is still in development. You can find the latest status of Raiden's development [here](#), and deploy the latest version of the software [here](#).

### Price

[The Raiden Network](#) has no transactions fees. A one-time on-chain deposit is needed to start a payment channel in the Raiden Network.

### Link to the official website and documentation

Official website: <https://raiden.network/>

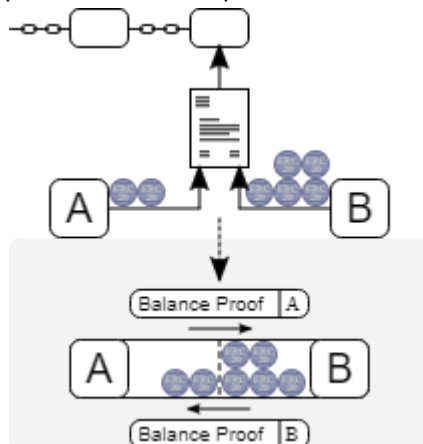
Documentation: <https://docs.raiden.network/>

General Github: <https://github.com/raiden-network/raiden>

### How it works

Blockchains don't scale well because there needs to be global consensus of all transfers. This makes processing micropayments a slow and expensive process. The [Raiden Network](#) uses a mix of mesh payment channels, deposits and cryptographic tricks to allow for secure token transfer off-chain. The blockchain is only used to settle claims which resulted from off-chain transfer activities. By doing most of the transfers off-chain the Raiden Network works cheap, fast and private.

[The Raiden Network](#) allows secure transfers of tokens between participants without the need for global consensus. This is achieved using digitally signed and hash-locked transfers, called balance proofs. This concept, illustrated in the picture below, is known as payment channel technology.



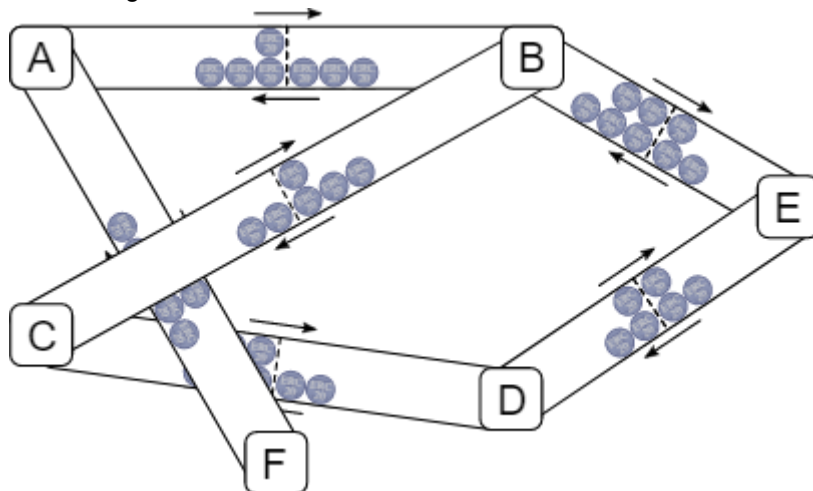
Payment channel technology.

Payment channels allow for practically unlimited bidirectional transfers between two participants, as long as the net sum of their transfers does not exceed the deposited tokens. These transfers can be performed instantaneously and without any involvement of the actual blockchain itself, except for an initial one-time on-chain creation and an eventual closing of the channel.

A Raiden balance proof is a binding agreement enforced by the Ethereum blockchain. Digital signatures make sure that neither party can back out of any of the value transfers contained therein,

as long as at least one of the participants decides to present it to the blockchain. Since nobody else other than the two participants has access to the tokens deposited in the payment channel's smart contract, a Raiden balance proof is as binding as an on-chain transaction.

The true strength of Raiden lies in its network protocol. Since opening and closing a payment channel between two peers still requires on-chain transactions, creating channels between all possible peers becomes infeasible. As it turns out, however, you do not need a direct payment channel between a payer and a payee if there exists at least one route through a network of channels that connects the two parties, as shown in the picture below. This network and its associated protocol for routing and interlocking channel transfers is called the Raiden Network.



Payment Channel Network.

In addition, payment channel transfers, in contrast to on-chain transactions, do not require any fees. Intermediaries within the greater network, however, will want to charge fees on a low percentage basis for providing their own channels to the network, leading to complex routing and a competitive channel fee market. The Raiden protocol aims to facilitate this market by using both protocol-level features and optional auxiliary services.

More information on how it works can be found here: <https://raiden.network/101.html>

### Pros and cons

The Raiden Network has provided the benefits and limitations of The Raiden Network on their website: <https://raiden.network/101.html>.

#### *Pros*

- Useful for small transfers.
- Instant transfers of off-chain Raiden transfers.
- Scalable. The Raiden Network scales linearly with the numbers of users.
- Secure via digitally signed and hash-locked transfers.

#### *Cons*

- On-chain transfers still depend on block time and the time it takes miners to pick the transaction from the pool of pending transactions.
- Raiden transfers require some of your tokens to be locked up in a smart contract for the lifetime of the payment channel. Although payment channels are expected to be comparatively small.

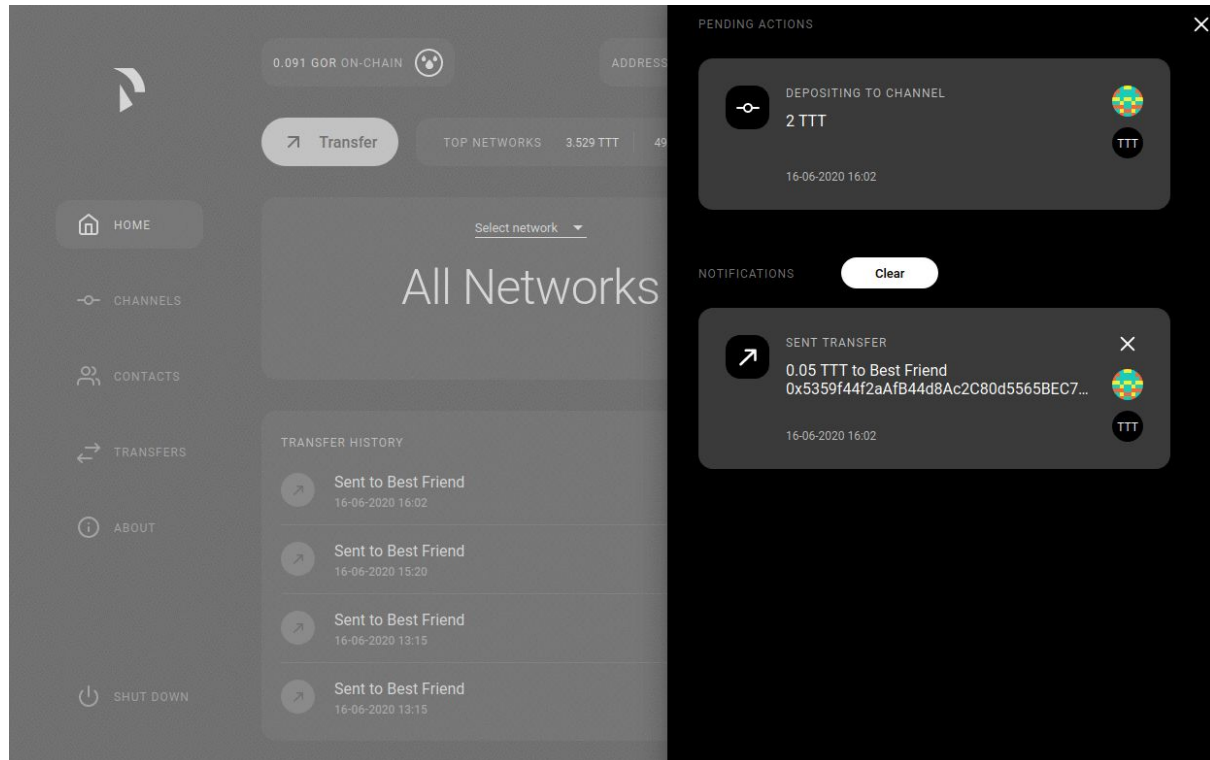
- Large value transfers should still be performed on the blockchain itself to save the extra cost of channel lifetime management and to avoid the need for routing through mostly inadequately equipped payment channels.
- A lot of the transfers are off-chain. You can't provide your clients with full blockchain-security that way.

## Implementation (Examples)

### *Raiden WebUI*

The WebUI allows users to easily access the main functionalities of the Raiden Network via a web interface.

See the code [here](#).



### *Paywall Infrastructure*

With Raiden it's easy to build an accountless pay-per-use paywall infrastructure.

See the code [here](#).

### *Instant M2M Payments*

Raiden can be used for making instant IoT machine-to-machine (M2M) micropayments.

See the code [here](#).

### *Raiden Wizard*

The Raiden Wizard provides a quick and easy way for Installing Raiden and running a Raiden node. It is a browser application, that assists with creating and funding an account for use with Raiden.

## Example of companies using it + a real use case

During ETHGlobal's ETHCapeTown Hackathon, the first Ethereum hackathon in Africa, the dTok team used Raiden software to build an application. The dTok team built a decentralized streaming and tipping platform with a Raiden BurnerWallet integration. More information on this project and how they built it, can be found [here](#).

**Conclusion**

The Raiden Network can be very useful for the SenneTTa project. Because it is secure, scalable and offers instant transfers with a very low fee. The usefulness of The Raiden Network depends largely on the payment structure of the SenneTTa project. If ICR3ATE wants the micropayment transfers to be made when data is bought, the Raiden Network offers a great solution. If ICR3ATE wants the micropayment transfers to be made only per month or per year, another solution might be better. The Raiden Network aims to use the above technology to provide near-instant, low-fee, scalable, and privacy-preserving payments based on Ethereum ERC20 tokens and to extend current on-chain limitations.

## 4.7 Nodle

### Overview (Introduction)

[Nodle](#) is a connectivity provider for the Internet of Things. The company has built a robust Bluetooth Low Energy-powered network to help companies and cities connect and collect data from their devices, sensors and tags. Whereas 95% of the mining power of the Bitcoin Network is now in the hands of [nine well-funded central actors](#) (Foreword), Nodle aims to counteract this oligopoly by letting anyone with a smartphone participate in the 'mining' of Nodle.

### Price

[Nodle](#) pays the fees in the Nodle Cash app. [Nodle Cash](#) has only been live for a short period of time and has not been listed on a cryptocurrency exchange. This makes it impossible to cash out the acquired Nodle Cash.

### Link to the official website and documentation

Official website: <https://nodle.io/>

SDK Documentation: <https://nodle.gitbook.io/nodle-sdk/>

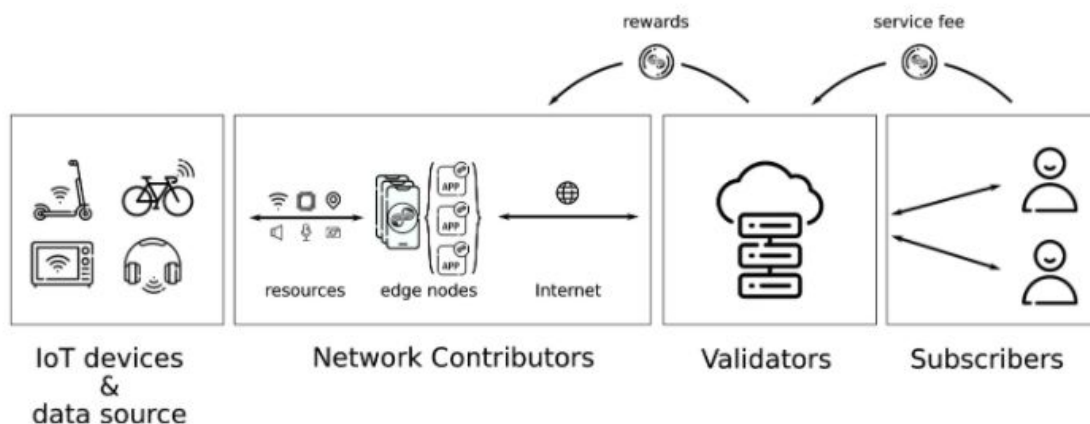
API Documentation:

<https://paper.dropbox.com/published/Nodle-Webhook-POST-Technical-Information-nXGI7o3BII10PjsjcKkk8c0>

### How it works

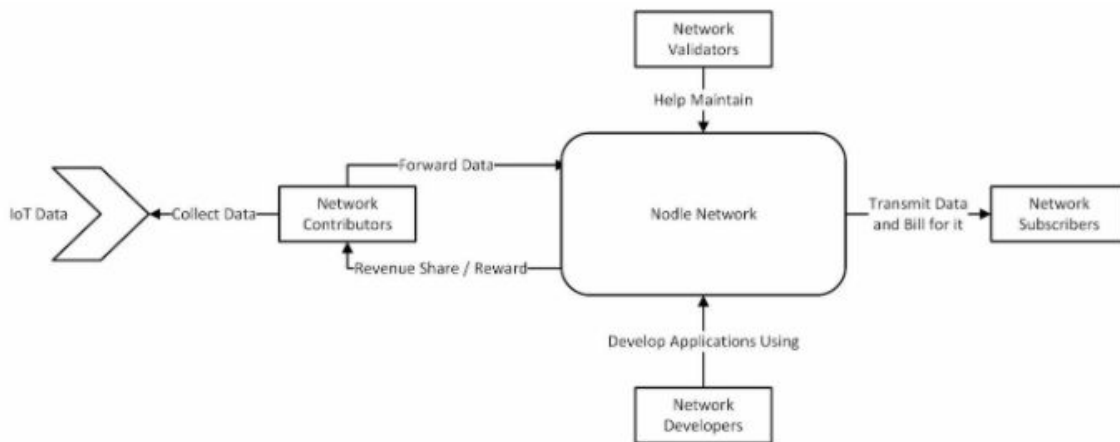
#### The Nodle Network

The Nodle Network is composed of many different elements interacting together in order to connect IoT devices to the cloud while generating revenues for [every contributor in the process](#) (Architecture).



Elements of The Nodle Network.

*The different elements of The Nodle Network explained*



### *Network Contributors*

The Network Contributors are living at the edge of the network and are comprised of all the devices that can interact with the IoT device and act as gateway for the Nodle Network, such devices are referred to as Edge Nodes. Most Edge Nodes are made of smartphones. To become a Network Contributor, one simply has to embed the Nodle SDK on devices that the contributor has access to. Contributors can be individuals, application developers, hardware manufacturers or autonomous IoT devices that are seeking ways to monetize their connectivity. Once a user authorises the Nodle SDK to run on their device, the device becomes an Edge Node by joining the Nodle Network and adding its local resources (such as BLE/Wi-Fi/Mobile interface, gps sensor, cpu & storage) to the resources of the global network. Network participation consumes little energy and requires minimal interaction.

### *Network Subscribers*

Network Subscribers are entities that utilize the services of the Nodle Network to affect communication or related services to targeted IoT devices for some purpose. Network Subscribers interact with the Nodle Network by paying a fee that is used to incentivize the Contributors.

### *Network Validators*

The Validators constitute the backend of the Nodle Network. Together they coordinate the work at the edge, settle the transaction on the Nodle Chain and deliver the services to the subscribers. The role of a Validator is two folds:

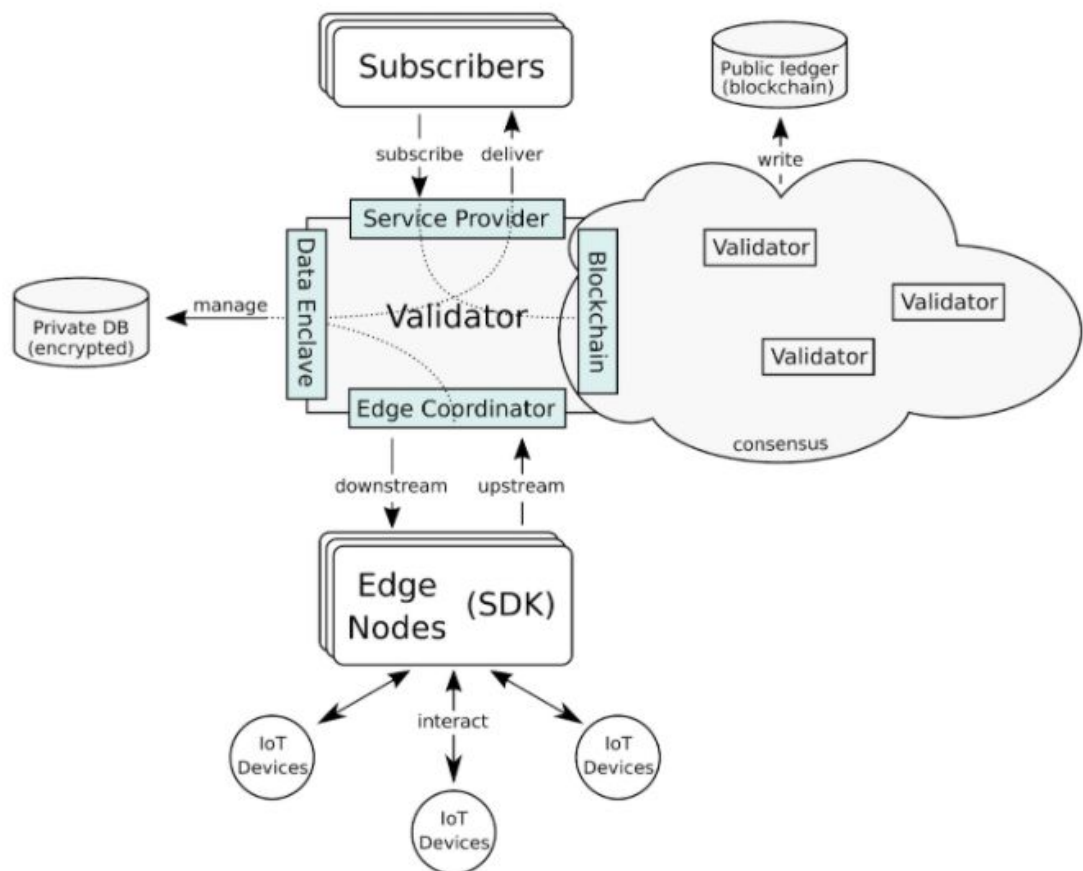
1. Verify and manage the data reported by Edge Nodes.
2. Maintain a distributed ledger (blockchain) called the Nodle Chain and used to handle settlements between the network members and operate the Nodle Cash token.

Validators are trusted parties, not everyone can become a Validator. In order to become a Validator, we propose the following requirements for a node:

1. Be vetted by Nodle to be a candidate to become a Validator. In order to guarantee the security and performance of the Nodle network in its infancy, Nodle will choose the nodes that are eligible to become a Validator.
2. Be selected by the community of Nodle Cash token holders. Even once Nodle vets potential applicants, the Nodle Cash token holders still have to decide and choose their own Validators. In order to do so token holders can “vote” for their preferred Validators by delegating some or all of their tokens to an applicant. The applicant with the most tokens delegated to their application will be chosen to be a Validator.

This system is somehow similar to traditional Validator selection mechanisms such a DPOS and its derivatives. Votes and Validator selection are handled through on-chain voting.

A Validator runs a software suite on its own cloud environment or datacenter including four interfaces:



Validator software suite.

### Network Developers

The Network Developers are technical users of the Nodle Network and its surrounding services. They develop custom applications that rely on the network or run on top of it.

### The Nodle Chain

The Nodle Chain is a core component of the Nodle Network, it is in charge of supporting the Nodle Cash token used for payments between every party along with other IoT centric features which will be described below.

### Payments

Nodle Chain has its own native token: the Nodle Cash token. Nodle Cash is used to pay for network fees and to serve as a currency between the different parties of the Nodle Network

### Certificates

Nodle Chain supports the management of cryptographic certificates through what's called a Public Key Infrastructure. It is envisioned for these certificates to be used inside IoT devices in order to considerably upgrade their level of security. These certificates could be used for authentication, tampering or access control and open new use cases for the whole industry.

### Smart Missions

In a future release of the Nodle Chain the creation of smart contracts made in Web Assembly will be supported. Smart contracts developed by developers will be able to be linked to some interactions that will have to be performed by Edge Nodes, we call these Smart Missions.



### *Path to decentralization*

The Nodle Network consists of two components: a decentralized ledger and a centralized oracle handling the interactions with the outside world. The Ledger handles most tasks related to security, authentication and tokenomics. The oracle has the role of exposing the IoT data to the decentralized world and also interface with traditional business customers. When IoT devices sign their data in a cryptographic way, the oracle is no longer needed.

### **Pros and cons**

#### *Pros*

- Uses Bluetooth connections, makes it possible for ICR3ATE to use cheap and low power consumption IoT devices.
- Open Source.
- More transactions per second possible due to Delegated Proof of Stake.
- Nodle Cash is used in the Network.
- Reached 1 million micropayments per day on February 10th, 2020:  
<https://medium.com/@nodle/nodle-cash-nodl-is-now-processing-as-many-micropayments-per-day-as-ripple-xrp-af09a1b24182>

#### *Cons*

- Security and performance is in the hands of Validators → Delegated Proof of Stake.
- Uses a centralized oracle to verify non cryptographic data.
- No examples which can be used for SenneTta.

### **Implementation (Example)**

#### *Tracking public furniture*

Nodle announced a [partnership](#) with the City of Paris to deploy 3.000 Bluetooth-enabled benches and other urban installations. The installations use Bluetooth Low Energy to transmit real-time data to city officials about bench locations and possible migration between parks over time. That way, the city officials can get an overview of the locations of the installations and use that information to reduce theft.

#### *Nodle M1*

Nodle has made the [Nodle M1](#), a smart wearable device that buzzes employees to help them social distance. This device securely records “smart interactions” between employees - measured in time, distance and duration. Information is saved on a secure server, while maintaining the utmost privacy and anonymity of employees.

Nodle M1 uses the [Whisper Tracing Protocol](#), an open source privacy-first framework.

### **Conclusion**

Nodle believes that a decentralized wireless network is inevitable. The Nodle Network proposes a solution to build the first layer of what could become an essential part of a Universal Basic Income (UBI). The amount of data generated by people and machines increases exponentially every year. Nodle enables anyone to benefit directly from capturing the value of this data. The Nodle Network could be very useful for the SenneTta project.

## 4.8 Dash

### Overview (Introduction)

Dash is a next-generation, p2p payments network and cryptocurrency where transactions are recorded on the blockchain: a decentralized, distributed, public ledger. It provides users with a digital currency that is fast, secure, portable, and fungible.

### Price table

The current fee schedule for Dash is as [follows](#):

Transaction type	Recommended fee	Per unit
Standard transaction	.00001 DASH	Per kB of transaction data
InstantSend autolock	.00001 DASH	Per kB of transaction data
InstantSend	.0001 DASH	Per transaction input
PrivateSend	.001 DASH	Per 10 rounds of mixing (average)

As an example, a standard and relatively simple transaction on the Dash network with one input, one output and a possible change address typically fits in the range of 200 - 400 bytes. Assuming a price of US\$100 per DASH, the fee falls in the range of \$0.0002 - \$0.0004, or 1/50th of a cent. Processing a simple transaction using InstantSend at the same price is free of charge, while more complex InstantSend transactions may cost around 1-2 cents per transaction, depending on the number of inputs. These fees apply regardless of the Dash or dollar value of the transaction itself.

### Masternodes

Running a masternode costs 1000 DASH, which [equals €86060](#). However, it is also possible to let a Dash community member run the masternode for you. [This](#) is a list of the different available hosting services. Costs can be 15% of masternode payments, from \$4,50 per month to 0.34 Dash (€29) per month.

### Link to the official website and documentation

Official website: [www.dash.org](http://www.dash.org)

Documentation: <https://docs.dash.org/en/stable/index.html>

### How it works

#### *What is Dash?*

Dash aims to be the most user-friendly and scalable payments-focused cryptocurrency in the world. The Dash network features instant transaction confirmation, double spend protection, optional privacy equal to that of physical cash, a self-governing, self-funding model driven by incentivized full nodes and a clear roadmap for on-chain scaling to up to 400MB blocks using custom-developed open source hardware. While Dash is based on Bitcoin and compatible with many key components of the Bitcoin ecosystem, its two-tier network structure offers significant improvements in transaction speed, privacy and governance.

#### *Safety*

Dash has a whole page dedicated to safety and makes people aware of scams which can be found [here](#).

The Dash Network improves security by using ChainLocks and a Decentralized Governance. ChainLocks are a feature provided by the Dash Network which provides certainty when accepting payments. This technology, particularly when used in parallel with InstantSend, creates an

environment in which payments can be accepted immediately and without the risk of “Blockchain Reorganization Events”. More information can be found [here](#).

Decentralized Governance by Blockchain, or DGBB, is Dash’s attempt to solve two important problems in cryptocurrency: governance and funding. Governance in a decentralized project is difficult, because by definition there are no central authorities to make decisions for the project. In Dash, such decisions are made by the network, that is, by the owners of masternodes. The DGBB system allows each masternode to vote once (yes/no/abstain) for each proposal. If a proposal passes, it can then be implemented (or not) by Dash’s developers. A key example is early in 2016, when Dash’s Core Team submitted a proposal to the network asking whether the blocksize should be increased to 2 MB. Within 24 hours, consensus had been reached to approve this change. Compare this to Bitcoin, where debate on the blocksize has been raging for nearly three years. More information can be found [here](#).

The risk of blockchain reorganization is typically addressed by requiring multiple “confirmations” before a transaction can be safely accepted as payment. This type of indirect security is effective, but at a cost of time and user experience. ChainLocks are a solution for this problem.

### *Masternodes*

Masternodes are used to power PrivateSend, InstantSend and for the governance and the treasury system. Users are rewarded for running Masternodes; 45% of the block reward is allocated to pay the masternode network.

Masternodes enable the following services:

- InstantSend allows for near-instant transactions. Dash InstantSend transactions are fully confirmed within two seconds.
- PrivateSend gives financial privacy through a decentralized implementation of CoinJoin.
- ChainLocks, which protects the blockchain against 51% mining attacks by signing blocks as they are mined.
- Governance and Treasury allows stakeholders in Dash to determine the direction of the project and devotes 10% of the block reward to development of the project and ecosystem.
- Dash Evolution will make using cryptocurrency as easy as using PayPal.

More information about Masternodes can be found [here](#).

### *The future*

The Dash Roadmap sets out delivery milestones for future releases of Dash and includes specific technical details describing how the development team plans to realise each challenge. The Dash Roadmap can be found [here](#).

### *Getting started*

Dash welcomes new merchants and supports integration through a standardised onboarding process. It’s easy to begin accepting payments in Dash and enjoy the following benefits:

- Settlement within seconds and clearance within minutes
  - Ability to accept payments from any market around the world
  - Irreversible transactions to prevent fraud
  - Advanced privacy for both customers and merchants
  - Lowest fees in the industry
- To get started with an integration in a sales system, simply select an online or point of sale payment solution from the list on this [page](#).

- - Any Dash received in payment can be automatically converted to the fiat currency of your choice using services such as Uphold. Simply select the card for the target currency and click Add funds -> With cryptocurrency. Any cryptocurrency deposited to this address will immediately be converted to the target fiat currency at the time of deposit.
  - 
  - *Integration overview*
  - 
  - **Dash Core** is a “fork” of **Bitcoin Core** and shares many common functionalities. Key differences relate to existing JSON-RPC commands which have been customized to support unique functionalities such as InstantSend.
  - **General Information:** Dash is a “Proof of Work” network and is similar to Bitcoin.
    - Block Time: ~2.6 Minutes per Block
    - Github Source: <https://github.com/dashpay/dash>
    - Latest Release: <https://github.com/dashpay/dash/releases>
  - **JSON-RPC Interface:** The majority of commands are unchanged from Bitcoin making integration into existing systems relatively straightforward. Note that the following commands have been modified to support InstantSend:
    - [getrawmempool](#)
    - [getmempoolancestors](#)
    - [getmempooldescendants](#)
    - [getmempoolentry](#)
    - [getrawtransaction](#)
    - [decoderawtransaction](#)
    - [gettransaction](#)
    - [listtransactions](#)
    - [listsinceblock](#)
2. **Block Hashing Algorithm:** Dash uses the “X11” algorithm in place of SHA256 used in Bitcoin. It’s important to note, however, that this only affects the hashing of the Block itself. All other internals utilize SHA256 hashes (transactions, merkle root, etc) which allows for most existing libraries to work in the Dash ecosystem.
  3. **Special Transactions:** Dash Core v0.13.x introduces the concept of “Special Transactions”. Please see the [Transaction Type Integration Guide](#) for more information.

### Special Transactions

This documentation is also available as a [PDF](#).

Dash 0.13.0 and higher implement [DIP002 Special Transactions](#), which form a basis for new transaction types that provide on-chain metadata to assist various consensus mechanisms. The following special transaction types exist:

Release	Version	Type	Payload Size	Payload	Payload JSON	Transaction Purpose
v0.13.0	3	0	n/a	n/a	n/a	Standard Transaction
v0.13.0	3	1	<variable int>	<hex>	proRegTx	<a href="#">Masternode Registration</a>
v0.13.0	3	2	<variable int>	<hex>	proUpServTx	<a href="#">Update Masternode Service</a>
v0.13.0	3	3	<variable int>	<hex>	proUpRegTx	<a href="#">Update Masternode Operator</a>
v0.13.0	3	4	<variable int>	<hex>	proUpRevTx	<a href="#">Masternode Revocation</a>
v0.13.0	3	5	<variable int>	<hex>	cbTx	<a href="#">Masternode List Merkle Proof</a>
v0.13.0	3	6	<variable int>	<hex>	qcTx	<a href="#">Quorum Commitment</a>

Integration notes:

1. [DIP002 Special Transactions](#) are a foundational component of Dash Core v0.13.0 and introduce a new Transaction Version and related “Payload” to the network.
2. Integrated Systems must be able to [serialize and deserialize](#) these new Transaction Types in order to accurately encode and decode Raw Transaction data.
3. From a [backwards compatibility](#) perspective, the 4 byte (32-bit) [version](#) field included in Legacy Transactions has been split into two fields: [version](#) and [type](#) (each consisting of 2 bytes).
4. Refer to the [Special Transactions](#) section of the dash developer reference for additional detail on these data types, e.g. <variable int>.
5. [InstantSend](#) status and Payload JSON (e.g. [proRegTx](#)) is included in the JSON-RPC response, please note that this data is not part of the calculated hash and is provided for convenience.

See the [v0.13.0 transaction types integration documentation \(PDF\)](#) for worked examples of each transaction type.

### InstantSend

This documentation is also available as a [PDF](#).

InstantSend is a feature provided by the Dash network that allows for zero-confirmation transactions to be safely accepted and re-spent. The network attempts to lock the inputs of every valid transaction when it is broadcast to the network. Every secured transaction is included in a following block in accordance with standard blockchain principles.

InstantSend is enabled by the Masternode Network which comprises approximately 5,000 masternode servers. These nodes are differentiated from standard nodes by having proven ownership of 1,000 Dash, making the network [highly resistant to Sybil attacks](#). Masternodes form [Long-Living Masternode Quorums \(LLMQs\)](#), which are responsible for providing near instant certainty to the transaction participants that the transaction inputs cannot be respent, and that the transaction will be included in a following block instead of a conflicting transaction.

This concept works as an extension to Nakamoto Consensus. InstantSend enables transacted funds to be immediately and securely respent by the recipient, even before the transaction is included in a block.

### Receiving InstantSend Transactions

Receiving an InstantSend Transaction introduces two requirements:

1. The ability to determine the “InstantSend Status” of a given transaction.
2. The ability to adjust “Confirmation Status” independently of block confirmation.

InstantSend Status is typically determined through direct connection with the dash daemon, [ZMQ notification](#), or through the usage of an external wallet notification script.

**Direct Connection:** InstantSend Status can be identified through direct connection with the Dash daemon using JSON-RPC protocol. The “instantlock” attribute of the JSON response reflects the status of the transaction and is included in the following commands:

- [getrawmempool](#)
- [getmempoolancestors](#)
- [getmempooldescendants](#)
- [getmempoolentry](#)
- [getrawtransaction](#)
- [decoderawtransaction](#)
- [gettransaction](#)
- [listtransactions](#)
- [listsinceblock](#)

**ZMQ Notification:** Whenever a transaction enters the mempool and whenever a transaction is locked in the mempool, ZMQ notifications can be broadcast by the node. A list of possible ZMQ notifications can be found [here](#).

The following notifications are relevant for recognizing transactions and their corresponding instantlocks:

- `zmqpubhashtx`
- `zmqpubhashtxlock`
- `zmqpubrawtx`
- `zmqpubrawtxlock`

**Wallet Notification:** The Dash Core Daemon can be configured to execute an external script whenever an InstantSend transaction relating to that wallet is observed. This is configured by adding the following line to the dash.conf file:

`instantsendnotify=/path/to/concurrent/safe/handler %s`

This is typically used with a wallet that has been populated with [watch-only](#) addresses.

### Broadcasting InstantSend Transactions

Since Dash v0.14.0 established LLMQs on the Dash network, quorums will now attempt to lock every valid transaction by default without any additional fee or action by the sending wallet or user. A transaction is eligible for InstantSend when each of its inputs is considered confirmed. This is the case when at least one of the following circumstances is true:

- the previous transaction referred to by the input is confirmed with 6 blocks
- the previous transaction is confirmed through an older InstantSend lock
- the block containing the previous transaction is [ChainLocked](#)

When checking the previous transaction for an InstantSend lock, it is important to also do this on mempool (non-mined) transactions. This allows chained InstantSend locking.

### Additional Resources

The following resources provide additional information about InstantSend and are intended to help provide a more complete understanding of the underlying technologies.

- [InstantSend Technical Information](#)
- [InstantSend Developer Documentation](#)
- [DIP0010: LLMQ InstantSend](#)
- [Product Brief: Dash Core v0.14 Release](#)

### API Services

This documentation is also available as a [PDF](#).

Several API services exist to facilitate quick and easy integration with the Dash network for services including:

- Transaction broadcasting
- Exchange rates
- Currency conversion
- Invoice generation

API Services are typically leveraged to eliminate that requirement of running your own infrastructure to support blockchain interactions. This includes mechanisms such as:



- Forming and Broadcasting a Transaction to the network.
- Address generation using HD Wallets.
- Payment Processing using WebHooks.

There are a variety of options for supporting these methods, with the key differentiator being the pricing model included and supported features. The following list of API Providers attempts to outline these key features/differentiators and also includes a link to related documentation.

## Pros and cons

### *Pros*

- Instant transactions using InstantSend
- Low fees (See Price Table)
- Secure and a well thought out decentralized governance
- Very well documented, a lot of information can be found online
- Self governed and self funded makes it independent.

### *Cons*

- The Masternodes make all the decisions of the Dash Network
- A lot of similarities with Bitcoin.

## Implementation (Example)

Several API services exist to facilitate quick and easy integration with the Dash network for services including:

- Transaction broadcasting
- Exchange rates
- Currency conversion
- Invoice generation

API Services are typically leveraged to eliminate that requirement of running your own infrastructure to support blockchain interactions. This includes mechanisms such as:

- Forming and Broadcasting a Transaction to the network.
- Address generation using HD Wallets.
- Payment Processing using WebHooks.

There are a variety of options for supporting these methods, with the key differentiator being the pricing model included and supported features.

One implementation is

A full list of all the available API services can be found [here](#).

## Conclusion

The Dash Network looks very promising for the SennetTa project. With very low fees, fast transactions and a well supported network the Dash Network might be very useful for ICR3ATE.

## 4.9 Hyperledger Quilt

### Overview (Introduction)

Hyperledger Quilt is a Java implementation of the Interledger protocol, enabling payments across any payment network — fiat or crypto. Quilt provides an implementation of all core primitives required for sending and receiving payments in a ledger-agnostic manner, allowing developers to write application payments logic once while gaining access to any other payment system that is Interledger-enabled.

### Price table

Transaction fee information is unknown.

### Link to the official website and documentation

Official website: <https://www.hyperledger.org/use/quilt>

Github code: <https://github.com/hyperledger/quilt>

Wiki: <https://wiki.hyperledger.org/display/quilt/Hyperledger+Quilt>

### How it works

#### *Key Characteristics*

By implementing the Interledger protocol, Quilt provides:

- Foundations: A set of protocols and primitives for enabling value interoperability across any fungible asset system, including crypto, fiat, in-game, and other token currencies.
- Identity: A standardized, globally unique identifier for payment accounts called Payment Pointers.
- Liquidity: A peer-to-peer packetized payments protocol specified under ILPv4.
- Sending & Receiving: A Java implementation of STREAM, which is a packetized payment protocol inspired by the QUIC Internet Transport Protocol. STREAM operates on top of core Interledger protocols by allowing a sender and receiver to coordinate a payment over any arbitrary Interledger topology.
- Extensibility: Quilt provides a framework for designing higher layer use-cases on Interledger to enable payment systems interoperability.

#### *Security*

The Quilt project uses two automated scans to improve the quality and security of their library. [Github Automated Dependency Scans](#) are used to scan for vulnerable dependencies which can damage the confidentiality, integrity or availability of the project.

[LGTM](#) or Semmle is used for continuous security analysis, which helps finding zero-days and preventing critical vulnerabilities.

#### *Payment Pointers*

[Payment Pointers](#) are a standardized identifier for payment accounts. In the same way that an email address provides an identifier for a mailbox in the email ecosystem a payment pointer is used by an account holder to share the details of their account with a counter-party.

#### *STREAM*

[This document](#) specifies the STREAM Interledger Transport protocol, which provides for reliably sending money and data over ILP. STREAM is designed to be used in applications that involve streaming payments or data, as well as those that require sending larger discrete payments and messages. A virtual connection is established between a "client" and a "server" and can be used to send authenticated ILP packets between them.

### Pros and cons

#### *Pros*

- A lot of documentation available
- STREAM

#### *Cons*

- Key exchange is not provided by STREAM

### **Implementation (Example)**

#### *Opening New Streams*

Streams are opened when either side sends a StreamMoney or StreamData frame with a previously unused stream ID.

Client streams MUST be odd-numbered starting with 1 and server-initiated streams MUST be even-numbered starting with 2. If an endpoint sends a packet for an unopened stream with the wrong number, the receiving endpoint MUST close the connection with a ProtocolViolationError.

### **Example of companies using it + a real use case**

#### *Coil*

Coil lets users and creators experience web monetized content in their browser while supporting the websites you [read, watch and listen](#). Coil has enabled a no advertising, improved privacy website for their visitors. It is made with the [Interledger protocol](#).

### **Conclusion**

HyperLedger Quilt is a Java implementation of the Interledger protocol and can be very useful for SennetTa. Further research is needed to choose which micropayment currency to use.

## 4.10 Paypal

### Overview

Paypal considers payments under 10\$ as a micropayment (Paypal, n.d.). Once you choose the micropayments plan, all payments will be charged with the micropayment rate. According to Villaester (2020) you will be charged 5% plus an additional fee of 0,05 EUR for every transaction. PayPal is centralized and does not use blockchain technology. One of the most important benefit of using PayPal is that it is a trusted payment provider.

### Price table with different intervals

PayPal charges 5% of the transaction plus an additional 0,05 EUR for every transaction. The volume or quantity of the payments does not change the transaction costs. The standard rate is 2,9% plus 0,30 cent.

Micropayments will be cheaper if the price of the data is below €11,70, which is the break even point. After that, Paypal's standard fee will be cheaper. A [comparison](#) of these can be found at Pressbin (n.d.). Due to it's huge pagina size a screenshot will not be provided in this report.

### Link to the official website and documentation

Official website: <https://www.paypal.com/us/home>  
Documentation: <https://developer.paypal.com/docs/>  
Github: <https://github.com/paypal>

### How it works

Paypal's micropayment system does not work any different than their normal money transferring method. It is important to know that once the Paypal account is configured for micropayments, the micropayment fee will be applied to all transactions.

The difference with PayPal and traditional banks is that with PayPal, users only need to provide an email adres.

### Pros and Cons of the solution

The following pro's and cons are based on reviews by Merchandmaverick (n.d.) and PCmag (n.d.)

#### *Pro's*

- Trusted by consumers
- Clear pricing
- Extensive integrations
- Good developer tools
- All-in-one payments system
- Excellent security track record

#### *Cons*

- Account stability issues
- Inconsistent customer support
- Not suitable for high-risk industries
- Cases of sudden closing of account
- Fixed fees will not be ideal if SennetTa has a high volume of transactions

### **Implementation**

#### *Enabling micropayments*

First, you'll need to enable the micropayment option. Otherwise, the regular fees will apply:

1. To implement Paypal micropayments, you'll need a free Paypal business account. Existing personal PayPal accounts can be converted to a business account. Otherwise, register a new one.
2. It is not possible to enable micropayment settings yourself. You'll have to contact PayPal to enable it for you which can be done [here](#) or, during business hours, through their [chat](#).

#### *Onboarding sellers*

If you want to onboard sellers, you'll need to be an approved partner. Detailed explanation can be found [here](#).

#### *Making payments*

With PayPal you'll be able to add a payment button which moves the money from the buyer to the seller. To do this, you must be an approved PayPal partner and users must complete onboarding before you can use this integration. [This guide](#) is to create a payment button with JavaScript to transfer money from the buyer to the seller.

### **Example of companies using it**

There are a lot of companies which use PayPal. Some big names are [Versa Products](#) and [Sourcebooks](#), [Uber](#) and [eBay](#). The amount of users in Q3 2020 is 361 million (Statista, 2020). According to Enlyft (n.d.) there are 600.053 companies which use PayPal. However, an example of a project close to SennetTa could not be found.

## 4.11 Mollie

### Overview

Mollie is a popular payment processor in the Netherlands. Its biggest benefit is that you are able to pay with up to 24 different payment methods. Transaction costs are 0,29 EUR, no matter the volume.

### Price table with different intervals

For payments under 1 cent, the Paysafecard is the only viable option since you will only pay 15% transaction costs, with no flat fee. However, Paysafecard requires a minimum transaction amount of €1,- (Mollie, n.d.).

If the transaction volume is higher, the most viable option with Mollie would be iDeal because of a €0,29 transaction cost, no matter the volume. iDeal is restricted to the Netherlands, so will only be useful if SenetTa is used within the Netherlands.

A quotation for bulk discount has been requested.  
Update 21-12: Still no quotation. Requested another one.

### Link to the official website and documentation

Official website: <https://www.mollie.com/en>  
Documentation: <https://docs.mollie.com/>  
Github: <https://github.com/mollie>

### How it works

Mollie offers a lot of payment methods. An overview of the available payment methods and their costs can be seen in Appendix 1. With Mollie, only the receiver needs a Mollie account. The buyer can select the preferred payment method and will be redirected. Like PayPal, Mollie does not use blockchain technology and uses a centralized database.

### Pros and Cons of the solution

#### *Pro's*

Most popular payment solution in the Netherlands  
Easy to use  
A lot of different payment options like iDeal, credit card, Paypal.  
Set up is easy  
Low transaction costs (€0,29 per transaction)

#### *Cons*

Does not have any information about implementing Mollie in a blockchain

### How to implement it

Mollie API is available for PHP, Ruby, Node.js and Python (Mollie, n.d.)

Mollie (n.d.)

Mollie uses a representational state transfer (REST) architecture. It breaks down to HTTP-methods get, patch, post and delete, matching the operations to read, update, create and delete. To reach the Mollie API you can reach <https://api.mollie.com/v2/> with the name of the resource you'd like to interact with (Mollie, n.d.)

### Example of companies using it

Companies that use Mollie are [Noppo](#) and [Boloo](#). There are no examples of Mollie being used in a blockchain, or with smart contracts.

## 4.12 Telefonica

### Overview

Telefonica is a company that wants to facilitate communication between people with secure and state of the art technology (Telefonica, n.d.) They claim to make a solution that will fit your needs. Knowledge about blockchain is not needed. TrustOS is their operating system, which is a set of tools that acts like an API which makes processes easier and you can make a direct connection with the service layer that is needed. You'll be able to implement them in your systems very easily. Telefonica's blockchain network is one of the biggest in Spain.

### Price table

There are no prices available. To get a quotation, one must schedule a demo. The button to schedule a demo can be found here:

<https://blockchain.telefonica.com/>

### Link to the official website and documentation

Official website: <https://blockchain.telefonica.com/>

Documentation: <https://trustos.readthedocs.io/en/latest/index.html>

Github: <https://github.com/Telefonica>

### How it works

TrustOS software is deployed on any Hyperledger Fabric based blockchain network. Support for Quorum/Ethereum and CORDA based networks will be added soon.

Once deployed, an HTTP API is published so that it can be used from any programming language. The API hides much of the complexity of working with a Blockchain, but its direct use still requires low-level knowledge. For that reason, TrustOS also includes some libraries to be consumed by applications and other Smart Contracts. Therefore, TrustOS commands can be invoked from outside Blockchain through its APIs or from other Smart Contracts inside the Blockchain itself.

Since the installation and deployment of TrustOS requires considerable efforts, TrustOS is perfectly designed to allow the majority of business logics and client systems take advantage of its benefits without having knowledge of Hyperledger Fabric neither deploying anything, just using simple APIs. Moreover, for those who are deepening into the field, doors will be open to deploy TrustOS software over its own network.

TrustOS also includes some administration features and interfaces that allow developers to add new modules to TrustOS.

APIs do not speak the Blockchain language (send, validate or verify a transaction, query a block, provision or obtain gas at an address, compile, deploy or execute contracts, manage and safeguard cryptographic material, etc.) but the language a customer understands (create an asset and update its status or position, add account activity, verify an identity, create or transfer a token, etc.).

Initially, TrustOS is available for deployment on any Hyperledger Fabric-based network and interacts with Ethereum (the public network). In the coming months support will be added for implementation in other technologies such as Ethereum/Quorum or CORDA.

Figure 3.12.1 shows the layer-based architecture of TrustOS.

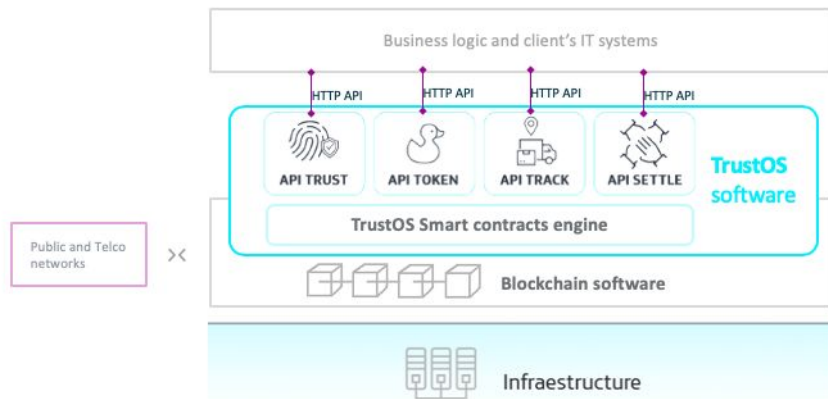


Figure 3.12.1 Layer-based architecture. Taken from *Architecture* by Trust OS, n.d (<https://trustos.readthedocs.io/en/latest/architecture.html>).

### Pros and cons

#### Pro's

- Software knowledge is useful, but not mandatory
- Implementing smart contracts is possible
- Implementing analytic and artificial intelligence is possible
- Telefonica is a big and trusted company in Spain.

#### Con's

- Telefonica is a spanish company, and so is their website. This may cause a language barrier. On the other side, TrustOS has an English documentation

### Implementation (Example)

#### Movistar

The aim of the project is to take advantage of the TrustOS platform to store the measurements that the IoT device collects at a certain frequency, so that they do not exceed a certain range. The combination of the IoT My Sensor device and the MQTT module makes it possible to safeguard human health in relation to harmful concentrations of certain gases such as radon.

Radon gas is a natural gas that in high concentrations can be harmful to health. Thanks to the execution of a chain code on the underlying Hyperledger Fabric network in TrustOS, these measures are monitored and it is possible to certify them at any time via the nature of the network itself without the intervention of trusted third parties. Thus, in the light of a legal or audit process, anyone can certify when the values exceeded the maximum threshold, if they have done so for a significant period of time or if, on the contrary, they have never reached it.

#### M2M technology and blockchain service to verify integrity

Thanks to the messaging service using the MQTT protocol, the devices can publish their measurements in the form of topics that any customer can subscribe to to monitor and manage the history of the entire data set. In addition, with every request to record a measurement, a device software check is performed to ensure that no one has tampered with it. An extra header is added to the HTTP request that contains a checksum of the device software that uniquely identifies it. With each call it is verified that said checksum is identical to the one stored in the first request of the device, thus ensuring the integrity of the software and the measurements sent. In case an anomaly is detected,



In addition to this, there are three videos of clients talking about their cooperation with Telefonica. However, since it is in Spanish and Google Translate is not helping much, only one is explained below:

Smart Protection offers protection services to companies to counteract counterfeits, piracy or abuse of trademarks. With Telefonica's blockchain technology, they can give their clients transparency of communication. Due to the immutability of data that is reached, Smart Protection clients' legal teams can use the data from the blockchain to take this as evidence if they decide to take legal action against offenders

# Chapter 5 Smart Contracts

In this research paper we look at different Smart Contract solutions. The following solutions will be discussed:

- DAML on top of Hyperledger Fabric
- Ethereum Smart Contracts
- Chaincode on top of Hyperledger Fabric

The different solutions will be looked at via a quick overview, links to the available documentation, how it works, how to implement it and what the pros and cons are.

## Smart Contract definition

The following definition of a smart contract is a literal copy of the explanation on the [Hyperledger Fabric website](#), as it provides a great explanation about a smart contract:

“Before businesses can transact with each other, they must define a common set of contracts covering common terms, data, rules, concept definitions, and processes. Taken together, these contracts lay out the business model that govern all of the interactions between transacting parties.

A smart contract defines the rules between different organizations in executable code. Applications invoke a smart contract to generate transactions that are recorded on the ledger.

Using a blockchain network, we can turn these contracts into executable programs – known in the industry as smart contracts – to open up a wide variety of new possibilities. That’s because a smart contract can implement the governance rules for any type of business object, so that they can be automatically enforced when the smart contract is executed. For example, a smart contract might ensure that a new car delivery is made within a specified timeframe, or that funds are released according to prearranged terms, improving the flow of goods or capital respectively. Most importantly however, the execution of a smart contract is much more efficient than a manual human business process.

In the diagram above we can see how two organizations, **ORG1** and **ORG2** have defined a **car** smart contract to **query**, **transfer** and **update** cars. Applications from these organizations invoke this smart contract to perform an agreed step in a business process, for example to transfer ownership of a specific car from **ORG1** to **ORG2**.”

## 5.1 Comparison table

We have made measurements from the information we have worked out below. We rank 1 to 5. 1 is worse, 5 is best. This comparison table is not objective.

Smart Contract Solution	Ease of implementation	Documentation	Documentation concerning Fabric	Possibility to update the Smart Contract	Remark	Average
-------------------------	------------------------	---------------	---------------------------------	--	--------	---------

DAML - Hyperledger Fabric	4	4	5	3	1 <sup>2</sup>	4
Smart Contract - Ethereum	3	4	1	N/A	2 <sup>2</sup>	2,7
Chaincode - Hyperledger Fabric	4	5	5	4	3 <sup>3</sup>	4,5

## 5.2 Conclusion

Chaincode and DAML on top of Hyperledger Fabric are the best solutions. Chaincode is part of Hyperledger Fabric, while DAML can be used on top of Hyperledger Fabric. Chaincode's API and SDK are available in Node.js. However, Chaincode's default language is Go, Their tutorials are in Go language. Newer updates will also be available sooner in Go language than in Node.js.

Below you can find the substantiation of the figure for each solution and a remark.

### **Substantiation of the figures**

#### 1. DAML - Hyperledger Fabric

##### *Ease of implementation: 4*

There are a lot of tools, support and documentation about implementing DAML into your project. Test environment.

##### *Documentation: 4*

Very extensive documentation, it can be confusing because of the quantity. They offer a [marketplace](#) with examples of products built with DAML.

##### *Documentation concerning Fabric: 5*

There is a special [Github page](#) about DAML and Hyperledger Fabric, also there is a report about the [Technical aspects](#) regarding DAML and Hyperledger Fabric.

##### *Possibility to modify Smart Contract: 3*

The Smart Contracts of DAML are immutable, you can archive the Smart Contract and create a new one to 'update' the contract.

#### 2. Smart Contract - Ethereum

##### *Ease of implementation: 3*

There is a lot of information about Ethereum's smart contracts. However, there is not a lot of information about the combination of Ethereum's smart contracts and Hyperledger Fabric.

##### *Documentation: 4*

---

<sup>2</sup> DAML, Much supporting tools, tutorials and documentation. Detailed information about DAML on top of Hyperledger Fabric. Test environment. Visual Studio Code is required. Interesting option.

<sup>2</sup> Ethereum, it looks like a promising solution when used on the Ethereum blockchain. However, using Ethereum's VM on Hyperledger Fabric might be difficult due to the lack of documentation about it.

<sup>3</sup> Chaincode, HF's Smart Contract program. A lot of information available in Node.js, tutorials in Go language, information about the processing of Smart Contracts. Looks promising.

There is a lot of information about the different programming languages in the documentation of Ethereum.

*Documentation concerning Fabric: 1*

There is no specific documentation about using Ethereum's Virtual Machine on Hyperledger Fabric in the documentation of Ethereum or Hyperledger Fabric.

*Possibility to modify the Smart Contract: N/A*

The solutions of Ethereum Smart Contract are just programming languages. It does not say anything about the possibility to modify the smart contract later. This depends on Hyperledger Fabric.

3. Chaincode - Hyperledger Fabric

*Ease of implementation: 4*

Because Chaincode is the Smart Contract language of Hyperledger Fabric it is the easiest to implement in Hyperledger Fabric. Chaincode's standard language is Go, but their documentation is also available in Node.js.

*Documentation: 5*

There's a lot of information available regarding Chaincode. The documentation about the contract API and the SDK is available in Node.js. All the documentation links can be found [here](#).

*Documentation concerning Fabric: 5*

Hyperledger Fabric has information available about Smart Contracts and Chaincode and the processing of Smart Contracts which can be found [here](#).

*Possibility to modify the Smart Contract: 4*

With Fabric v2.0 Chaincode it is possible to deploy a single chaincode package multiple times with different names on the [same channel or on different channels](#).

Upgrading transactions is only possible when it is approved by a sufficient number of organizations which takes away some risk for the other channel members.

## 5.3 DAML on top of our HyperLedger fabric

### Overview of it

'DAML is an open-source smart contract language for building future-proof distributed applications on a safe, privacy-aware runtime' (DAML, n.d.-c). DAML stands for Digital Asset Modeling Language and is part of the [Digital Asset Holding](#). DAML is built to run on blockchain platforms. DAML is easy to write and read, fast to build and deploy and provides high levels of privacy and abstraction (Rosic, n.d.).

### Link to the official website and documentation

Official website: <https://daml.com/>

Documentation: [https://docs.daml.com/index.html?\\_ga=2.177965408.416775879.1609076271-1757058918.1608031451](https://docs.daml.com/index.html?_ga=2.177965408.416775879.1609076271-1757058918.1608031451)

Github: <https://github.com/digital-asset/daml>

### How it works

DAML is compatible with Hyperledger Fabric, there is lots of [documentation](#) about it. As mentioned before DAML is an open-source smart contract language which is usable in a private execution environment. The information carried by the contracts written in DAML is accessible only to authorized parties (Goyal, 2019).

### Template structure DAML

In Figure 2 the DAML structure template is shown. All the different components of the template are explained [here](#).

```
template NameOfTemplate
with
  exampleParty : Party
  exampleParty2 : Party
  exampleParty3 : Party
  exampleParameter : Text
  -- more parameters here
where
  signatory exampleParty
  observer exampleParty2
  agreement
    -- some text
    ""
  ensure
    -- boolean condition
    True
  key (exampleParty, exampleParameter) : (Party, Text)
  maintainer (exampleFunction key)
  -- a choice goes here; see next section
```

Figure 2. DAML template structure. Taken from *App architecture* by DAML, n.d. (<https://docs.daml.com/daml/reference/structure.html>)

DAML launched [Project:DABL](#), DABL allows you to test and run your written smart contracts. DABL says the following thing on their website: 'DAML and DABL combine to create a framework designed for building composable, flexible applications in what was otherwise a stodgy, slow category. Only write the code that describes the behavior of your application, never to manage any of the infrastructures. When deployed in DABL, you will pay for what you use at the ledger update level instead of overprovisioning and underutilizing cloud infrastructure' (DABL, n.d.). DABL offers 3 subscriptions of which 2 are paid subscriptions, for details [click here](#).

### Architecture

The Architecture of DAML is divided into two sections, the backend and frontend, which is visualised in Figure 3.

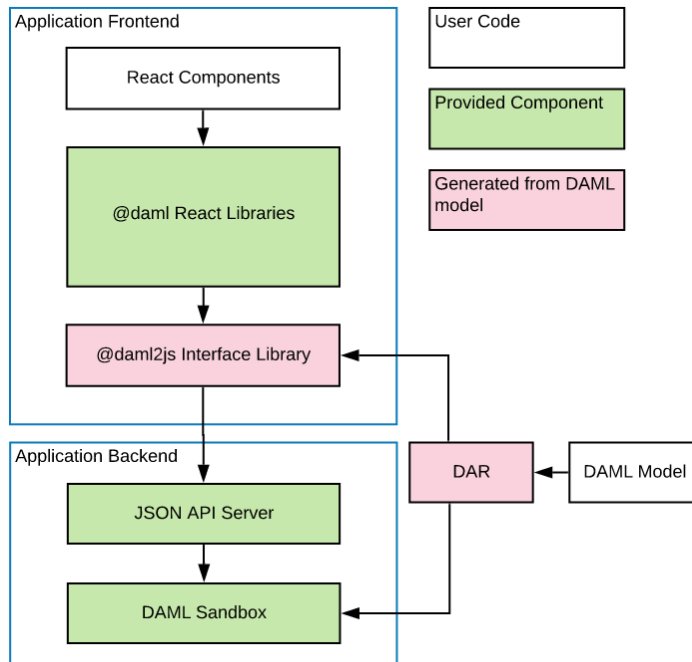


Figure 3. DAML app architecture. Taken from *App architecture* by DAML, n.d. (<https://docs.daml.com/getting-started/app-architecture.html>)

### Smart contracts

DAML contracts are immutable, this means they can't be changed. The only way to update contracts is to archive the old contract and create a new one (DAML, n.d.-a).

'Contract keys are an optional addition to templates. They let you specify a way of uniquely identifying contracts, using the parameters to the template - similar to a primary key for a database'. An example of such a contract key is shown in Figure 4.

```

type AccountKey = (Party, Text)

template Account with
  bank : Party
  number : Text
  owner : Party
  balance : Decimal
  observers : [Party]
where
  signatory [bank, owner]
  observer observers

  key (bank, number) : AccountKey
  maintainer key._1
  
```

Figure 4. Contract key. Taken from *Contract keys* by DAML, n.d. (<https://docs.daml.com/daml/reference/contract-keys.html>)

### DAML tools and ledgers

DAML provides different tools to support developing with DAML. [Sandbox](#) is such a tool, it is a simple ledger implementation that enables rapid application prototyping by simulating a DAM ledger.

The [Navigator](#) is a front-end that you can use to connect to any DAML Ledger and inspect and modify the ledger. You can use it during DAML development to explore the flow and implications of the DAML

models. With the Navigator you can view templates, view active and archived contracts, exercise choices on contracts and advanced time (Only when using Sandbox Ledger)

[DAML Studio](#) is an integrated development environment (IDE) for DAML. It is an extension on top of Visual Studio Code (VS Code), a cross-platform, open-source editor providing a rich code editing experience.

DAML created a [cheat-sheet](#) so you can always see easily what is what and what things mean. This is very useful when using, implementing and learning DAML.

At the DAML [Marketplace](#) you can discover products built with DAML. DAML is used in industries like aviation, banking, capital markets, CPG (consumer packaged goods), healthcare, retail and securitization.

## Installing

### 1. Install the dependencies

1. Install Visual Studio Code (Only compatible editor)
2. [JDK8](#) or greater

As a part of the installation you might need to set up JAVA\_HOME variables, for instructions [click here](#).

### 2. Install the SDK

[SDK for Windows 10](#)

MAC and Linux, open terminal and run following code:

```
curl -sSL https://get.daml.com/ | sh
```

You have to log out and log in to finish.

If you want to verify the SDK download for security reasons, see the [detailed instruction for manual download and installation](#)

### 3. Getting started

DAML created an extended and helpful [getting started guide](#) and lots of documentation. To make things even easier there is an [interactive learning environment](#). Here you can practise in an online editor with almost all the options DAML offers.

### How to implement it ( technical research)

There is already [documentation](#) about using DAML on top of Hyperledger Fabric, they call it DAML-on-Fabric. There is even a [technical document](#) about this written by Digital Asset.

‘DAML-on-Fabric allows you to build your DAML smart-contracts and deploy them to a local DAML Ledger using Hyperledger Fabric network as the infrastructure layer’(DAML, n.d.-b).

This document provides a lot of information about using DAML on top of Hyperledger Fabric, things they discuss in the document are shown in Figure 5. This document provides all the tools to use DAML on top of Hyperledger Fabric

2. Requirements Specification	5
2.1. Stakeholders	5
2.2. Functional Requirements	5
2.2.1. Ledger Bootstrapper	5
2.2.2. Ledger DAML Server	7
2.2.3. Ledger API Authorization	8
2.3. Non-Functional Requirements	9
2.3.1. Performance/Efficiency Requirements	9
2.3.1.1. Transaction Sizing Capacity	9
2.3.1.2. Transaction Hit Rate	9
2.3.1.3. Transaction Throughput	10
2.3.2. Reliability requirements	10
2.3.2.1. Recoverability	10
2.3.2.2. Maturity	10
2.3.3. Security Requirements	11
2.3.4. Usability Requirements	11
2.3.4.1. Learnability	11
2.3.4.2. Operability	11
2.3.4.3. User error protection	11
2.3.5. Maintainability Requirements	11
2.3.5.1. Modularity	11
2.3.5.2. Reusability	12
2.3.5.3. Analyzability	12
2.3.5.4. Modifiability	12
2.3.5.5. Testability	12
2.3.6. Portability Requirements	12
2.3.6.1. Adaptability	12
2.3.6.2. Installability	12
2.3.6. Documentation requirements	12
3. Architecture	13
3.1. System scope and context view	13
3.2. System containers view	14
3.2.1. Navigator & Application	15
3.2.2. DAML-on-Fabric	15
3.2.3. Hyperledger Fabric network	15
3.3. System components view	16
3.3.1. DAML Ledger API	17
3.3.2. DAML Ledger Server	17
3.3.2.1. DamlOnFabricServer	17
3.3.2.2. Fabric Adapter	17
4. Conclusions	18
4.1. Results	18
4.2. Recommendations	18

Figure 5. Table of content Technical document DAML-on-Fabric. Taken from *DAML-on-Fabric - Technical Document* by DAML, n.d.

([https://github.com/digital-asset/daml-on-fabric/blob/master/docs/DAML\\_on\\_Fabric\\_v2\\_Architecture.pdf](https://github.com/digital-asset/daml-on-fabric/blob/master/docs/DAML_on_Fabric_v2_Architecture.pdf))

### Pros and cons of the specific solution

#### Pros

- Easy to use with much and clear documentation
- Training and developer tools
- Part of big company Digital Asset
- Compatible with Hyperledger Fabric

#### Cons

- Very extensive product, which could make it confusing
- Must use Visual Studio Code

Pros and Cons by Mohammed:

#### Pros



- [Digital Assets](#) (the company behind DAML) has an impressive Ecosystem, backers and sponsors ( they raised around 150m in Serie C, includes high tech companies)
- Portability: DAML gives a portable solution, which can be used in different platforms
  - In Cloud Solution
  - Centralized Database such as PostgreSQL
  - Decentralized Databases ( Corda, Hyperledger Fabric,...)
- Interoperability: The ability for applications to communicate with each other even their underlying infrastructure is different (one on Cloud, other in Blockchain database for example)
- The launch of [DABL](#) to test and run the smart contracts locally
- They have a [marketplace](#) with different components that are ready to use (there are some paid ones).
- Easy to code Smart Contract, with a [good documentation](#) and a number of tutorials and sandbox to test and start your work.
- They provided drivers and example to integrate it with real code languages ( such as [node.js driver](#) and a [tutorial on how to use it](#))

#### Cons

- New concept, which makes it difficult at first to use it.
  - It takes time to understand and to create something useful with it.

## 5.4 Smart contract of Ethereum

### Overview of it

Since 2018, Hyperledger Fabric supports Ethereum Virtual Machine. Hyperledger Fabric made the decision to include EVM's Smart Contract because a lot of developers move from Ethereum to Hyperledger Fabric. This is not a program like the other two solutions in this research. We'll discuss Ethereum smart contracts best programming languages: Solidity & Vyper.

### Link to the official website and documentation

Official website Ethereum: <https://ethereum.org/en/>

Documentation Smart Contract Ethereum: <https://ethereum.org/en/developers/docs/smart-contracts/>

Documentation Solidity: <https://docs.soliditylang.org/en/latest/contracts.html>

Github Solidity: <https://github.com/ethereum/solidity>

Documentation Vyper:

<https://vyper.readthedocs.io/en/latest/vyper-by-example.html#simple-open-auction>

Github Vyper: <https://github.com/vyperlang/vyper>

### How it works

According to Entriken (2020) a smart contract is a program that runs on the Ethereum blockchain. It's a set of functions and data on the Ethereum blockchain. Transactions are not controlled by users. They will be done when the pre-defined rules of the smart contracts are met.

### Languages

Ethereum states that the programming languages of smart contracts are relatively developer-friendly. If you have experience with Python and/or JavaScript, you'll have no problem. The most active and maintained languages are Solidity (Which can be found in the example below) and Vyper. They also have an intermediate language for the Ethereum Virtual Machine named Yul and Yul+.

Solidity:

- Influenced by C++, Python and JavaScript.
- Statically typed (the type of a variable is known at compile time).
- Supports:
  - Inheritance (you can extend other contracts).
  - Libraries (you can create reusable code that you can call from different contracts – like static functions in a static class in other object oriented programming languages).
  - Complex user-defined types.
  -

Solidity is great for beginners since there are many tutorials and guides. It has a big developer community so it's most likely someone will have an answer for your questions, if necessary. An example of Solidity contract syntax can be found in Appendix 2.

Vyper:

- Pythonic programming language
- Strong typing
- Small and understandable compiler code
- Deliberately has less features than Solidity with the aim of making contracts more secure and easier to audit. Vyper does not support:
  - Modifiers
  - Inheritance
  - Inline assembly

- Function overloading
- Operator overloading
- Recursive calling
- Infinite-length loops
- Binary fixed points

Vyper is great for Python developers who want to write Smart Contracts. It has less features, which makes it great for quick prototyping of ideas. It is easier to audit and tries to maximize human-readability.

An example of Vyper contract syntax can be found in Appendix 3.

A comparison between the Solidity and Vyper syntax can be found [here](#).

### *Composability*

Smart contracts are public on Ethereum and can be thought of as open APIs. That means you can call other smart contracts in your own smart contract to greatly extend what's possible. Contracts can even deploy other contracts.

### *Limitations*

Smart contracts alone cannot get information about "real-world" events because they can't send HTTP requests. This is by design as relying on external information could jeopardise consensus, which is important for security and decentralization. There are ways to get around this using [oracles](#).

## **How to implement it ( technical research)**

### **Step 1. Install the required tools**

You need to install the following tools:

- cURL: It's a tool for transferring files from or to a server. Install it following the instructions [here](#).
- Docker: Developers increasingly use "Docker" to create, deploy, and run applications using containers. Docker helps programmers to create a complete package for an app, including libraries and dependencies. They can then deploy it on their choice of platform. Read the [installation instructions on the Docker website](#). The installation process includes "Docker Compose", i.e., a tool to define and run multi-container apps.
- Programming languages: You are setting up Ethereum smart contracts on Fabric to use in your application. Developers typically write apps on Fabric using "Go" or Node.js. You need to install either of them. Check the [Go programming language website](#) for installation instructions. Fabric currently supports Node.js v8.9, and you can install it using [their instructions](#).

Read more in [this Fabric prerequisites web page](#).

### **Step 2. Install binaries, samples and Docker image**

Fabric documentation already includes sample applications, and developers can use these to expedite their project. Fabric also provides binaries and Docker images for these. You will find these useful to fast-track your projects. Read "[Install Samples, Binaries and Docker Images](#)" for more details.

### **Step 3. Install Fabric EVM chaincode**

Now, you need to install the Hyperledger Burrow-based EVM chaincode. This will enable you to interact with the Ethereum smart contracts you will write, using Fabric. Follow the detailed instructions in [this GitHub repository](#).

#### Step 4. use “Remix” IDE to write Solidity smart contracts

Remix is a web-based “Integrated Development Environment” (IDE) for Ethereum smart contract development. You don’t need to install anything, just start writing Solidity smart contracts on [Remix](#).

#### Step 5. Initial configuration of Fabric EVM chaincode

From this point onwards, I will use a guide that [Rahul Bairathi](#), a Fabric expert has written. The article explains how to code Ethereum smart contracts using Fabric. You can read it in “[Solidity smart contract on Hyperledger Fabric](#)”. I will refer to it as a “reference article”.

After you have cloned the sample projects from GitHub as I have mentioned above, you need to do the following configurations:

- Navigate to the “first-network” folder, and modify the Docker compose “Command Line Interface” (CLI) file. This is to map the folder where you have stored the code you had cloned from GitHub.
  - Start the network.
  - Navigate to the CLI Docker container and install the EVM chaincode.
  - Instantiate the EVM chaincode.
- Follow the commands in the “[reference article](#)”.

#### Step 6. Write Solidity/Vyper smart contracts

The next step involves writing Solidity smart contracts using the Remix IDE. If your team guidance, they can get check out [this Remix documentation](#).

#### Step 7. Compile and deploy smart contracts

You need to take the following steps:

- Compile the smart contract on Remix.
- Copy the Bytecode. You need it to deploy the smart contract. Read more about it in [this Ethereum stack exchange Q&A thread](#).
- Paste the bytecode in any text editor. This will be in JSON format.
- Deploy the smart contract using the “peer chaincode invoke” process. Your team can find the syntax of this command in [this Fabric documentation page](#).

Read the “[reference article](#)” for more details.

### Pros and cons of the specific solution

#### Pro's

- Ethereum has one of the biggest developers communities in the world. (Klemens, 2020)
- Good documentation

#### Cons

- For a lot of developers Solidity and Vyper are unfamiliar programming languages. Because of that, it is common for developers to leave vulnerabilities in their smart contract code (Klemens, 2020).

## 5.5 Hyperledger Fabric’s Chaincode

#### Overview of it

[Chaincode](#) is a program, written in Go, node.js, or Java that implements a prescribed interface.

Chaincode runs in a secured Docker container isolated from the endorsing peer process. A chaincode

typically handles business logic agreed to by members of the network, so it is similar to a “smart contract”. A chaincode can be invoked to update or query the ledger in a proposal transaction. Given the appropriate permission, a chaincode may invoke another chaincode, either in the same channel or in different channels, to access its state. Note that, if the called chaincode is on a different channel from the calling chaincode, only read query is allowed. That is, the called chaincode on a different channel is only a *Query*, which does not participate in state validation checks in subsequent commit phases.

### Link to the official website and documentation

Official website: <https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html>

Documentation: <https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode4ade.html>

Github: <https://github.com/hyperledger/fabric/tree/master/core/chaincode>

Node.js contract API: <https://github.com/hyperledger/fabric-chaincode-node>

Node.js contract API Documentation: <https://hyperledger.github.io/fabric-chaincode-node/>

Node.js SDK: <https://github.com/hyperledger/fabric-sdk-node>

Node.js SDK Documentation: <https://hyperledger.github.io/fabric-sdk-node/>

Smart Contracts and Chaincode:

<https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html>

Smart Contract Processing:

<https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/smartcontract.html>

### How it works

#### *Chaincode programs*

Every chaincode program must implement the *Chaincode interface* whose methods are called in response to received transactions. In particular the *Init* method is called when a chaincode receives an *instantiate* or *upgrade* transaction so that the chaincode may perform any necessary initialization, including initialization of application state. The *Invoke* method is called in response to receiving an *invoke* transaction to process transaction proposals.

The other interface in the chaincode “shim” APIs is this *one* which is used to access and modify the ledger and to make invocations between chaincodes.

### How to implement it (technical research)

#### *Go chaincode tutorial*

I’ve copied the whole chaincode tutorial using Go as found [here](#).

#### **Simple Asset Chaincode**

Our application is a basic sample chaincode to create assets (key-value pairs) on the ledger.

#### **Choosing a Location for the Code**

If you haven’t been doing programming in Go, you may want to make sure that you have Go Programming Language installed and your system properly configured.

Now, you will want to create a directory for your chaincode application as a child directory of *\$GOPATH/src/*.

To keep things simple, let's use the following command:

```
mkdir -p $GOPATH/src/sacc && cd $GOPATH/src/sacc
```

Now, let's create the source file that we'll fill in with code:

```
touch sacc.go
```

### Housekeeping

First, let's start with some housekeeping. As with every chaincode, it implements the `Chaincode` interface in particular, `Init` and `Invoke` functions. So, let's add the Go import statements for the necessary dependencies for our chaincode. We'll import the chaincode shim package and the `peer` protobuf package. Next, let's add a struct `SimpleAsset` as a receiver for Chaincode shim functions.

```
package main

import (
    "fmt"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}
```

### Initializing the Chaincode

Next, we'll implement the `Init` function.

```
// Init is called during chaincode instantiation to initialize any data.
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {

}
```

**Note:** Note that chaincode upgrade also calls this function. When writing a chaincode that will upgrade an existing one, make sure to modify the `Init` function appropriately. In particular, provide an empty "Init" method if there's no "migration" or nothing to be initialized as part of the upgrade.

Next, we'll retrieve the arguments to the **Init** call using the `ChaincodeStubInterface.GetStringArgs` function and check for validity. In our case, we are expecting a key-value pair.

```
// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data, so be careful to avoid a scenario where you
// inadvertently clobber your ledger's data!
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
    // Get the args from the transaction proposal
    args := stub.GetStringArgs()
    if len(args) != 2 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }
}
```

Next, now that we have established that the call is valid, we'll store the initial state in the ledger. To do this, we will call `ChaincodeStubInterface.PutState` with the key and value passed in as the arguments. Assuming all went well, return a `peer.Response` object that indicates the initialization was a success.

```
// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data, so be careful to avoid a scenario where you
// inadvertently clobber your ledger's data!
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
    // Get the args from the transaction proposal
    args := stub.GetStringArgs()
    if len(args) != 2 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }

    // Set up any variables or assets here by calling stub.PutState()

    // We store the key and the value on the ledger
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
    }
    return shim.Success(nil)
}
```

## Invoking the Chaincode

First, let's add the **Invoke** function's signature.

```
// Invoke is called per transaction on the chaincode. Each transaction is
```

```
// either a 'get' or a 'set' on the asset created by Init function. The 'set'
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {

}
```

As with the `Init` function above, we need to extract the arguments from the `ChaincodeStubInterface`. The `Invoke` function's arguments will be the name of the chaincode application function to invoke. In our case, our application will simply have two functions: `set` and `get`, that allow the value of an asset to be set or its current state to be retrieved. We first call `ChaincodeStubInterface.GetFunctionAndParameters` to extract the function name and the parameters to that chaincode application function.

```
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

}
```

Next, we'll validate the function name as being either `set` or `get`, and invoke those chaincode application functions, returning an appropriate response via the `shim.Success` or `shim.Error` functions that will serialize the response into a gRPC protobuf message.

```
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    var result string
    var err error
    if fn == "set" {
        result, err = set(stub, args)
    } else {
        result, err = get(stub, args)
    }
    if err != nil {
        return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}
```



```
}
```

### Implementing the Chaincode Application

As noted, our chaincode application implements two functions that can be invoked via the `Invoke` function. Let's implement those functions now. Note that as we mentioned above, to access the ledger's state, we will leverage the `ChaincodeStubInterface.PutState` and `ChaincodeStubInterface.GetState` functions of the chaincode shim API.

```
// Set stores the asset (both key and value) on the ledger. If the key exists,
// it will override the value with the new one
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}

// Get returns the value of the specified asset key
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
        return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
        return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}
```

### Pulling it All Together

Finally, we need to add the `main` function, which will call the `shim.Start` function. Here's the whole chaincode program source.

```
package main
```

```
import (
    "fmt"
```

```

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}

// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data.
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
    // Get the args from the transaction proposal
    args := stub.GetStringArgs()
    if len(args) != 2 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }

    // Set up any variables or assets here by calling stub.PutState()

    // We store the key and the value on the ledger
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
    }
    return shim.Success(nil)
}

// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    var result string
    var err error
    if fn == "set" {
        result, err = set(stub, args)
    } else { // assume 'get' even if fn is nil
        result, err = get(stub, args)
    }
    if err != nil {
        return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}

// Set stores the asset (both key and value) on the ledger. If the key exists,
// it will override the value with the new one

```

```

func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}

// Get returns the value of the specified asset key
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
        return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
        return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}

// main function starts up the chaincode in the container during instantiate
func main() {
    if err := shim.Start(new(SimpleAsset)); err != nil {
        fmt.Printf("Error starting SimpleAsset chaincode: %s", err)
    }
}

```

## Building Chaincode

Now let's compile your chaincode.

```

go get -u github.com/hyperledger/fabric/core/chaincode/shim
go build

```

Assuming there are no errors, now we can proceed to the next step, testing your chaincode.

## Testing Using dev mode

Normally chaincodes are started and maintained by peer. However in “dev mode”, chaincode is built and started by the user. This mode is useful during chaincode development phase for rapid code/build/run/debug cycle turnaround.

We start “dev mode” by leveraging pre-generated orderer and channel artifacts for a sample dev network. As such, the user can immediately jump into the process of compiling chaincode and driving calls.

### **Install Hyperledger Fabric Samples**

If you haven't already done so, please [Install Samples, Binaries and Docker Images](#).

Navigate to the `chaincode-docker-devmode` directory of the `fabric-samples` clone:

```
cd chaincode-docker-devmode
```

Now open three terminals and navigate to your `chaincode-docker-devmode` directory in each.

#### **Terminal 1 - Start the network**

```
docker-compose -f docker-compose-simple.yaml up
```

The above starts the network with the `SingleSampleMSPSolo` orderer profile and launches the peer in “dev mode”. It also launches two additional containers - one for the chaincode environment and a CLI to interact with the chaincode. The commands for create and join channel are embedded in the CLI container, so we can jump immediately to the chaincode calls.

#### **Terminal 2 - Build & start the chaincode**

```
docker exec -it chaincode bash
```

You should see the following:

```
root@d2629980e76b:/opt/gopath/src/chaincode#
```

Now, compile your chaincode:

```
cd sacc
go build
```

Now run the chaincode:

```
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./sacc
```

The chaincode is started with peer and chaincode logs indicating successful registration with the peer. Note that at this stage the chaincode is not associated with any channel. This is done in subsequent steps using the `instantiate` command.

### Terminal 3 - Use the chaincode

Even though you are in `--peer-chaincodedev` mode, you still have to install the chaincode so the life-cycle system chaincode can go through its checks normally. This requirement may be removed in future when in `--peer-chaincodedev` mode.

We'll leverage the CLI container to drive these calls.

```
docker exec -it cli bash
```

```
peer chaincode install -p chaincodedev/chaincode/sacc -n mycc -v 0
peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc
```

Now issue an invoke to change the value of "a" to "20".

```
peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc
```

Finally, query `a`. We should see a value of `20`.

```
peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc
```

### Testing new chaincode

By default, we mount only `sacc`. However, you can easily test different chaincodes by adding them to the `chaincode` subdirectory and relaunching your network. At this point they will be accessible in your `chaincode` container.

### Chaincode encryption

In certain scenarios, it may be useful to encrypt values associated with a key in their entirety or simply in part. For example, if a person's social security number or address was being written to the ledger, then you likely would not want this data to appear in plaintext. Chaincode encryption is achieved by leveraging the `entities extension` which is a BCCSP wrapper with commodity factories and functions to perform cryptographic operations such as encryption and elliptic curve digital signatures. For example, to encrypt, the invoker of a chaincode passes in a cryptographic key via the transient field. The same key may then be used for subsequent query operations, allowing for proper decryption of the encrypted state values.

For more information and samples, see the [Encc Example](#) within the [fabric/examples](#) directory. Pay specific attention to the [utils.go](#) helper program. This utility loads the chaincode shim APIs and Entities extension and builds a new class of functions (e.g. [encryptAndPutState](#) & [getStateAndDecrypt](#)) that the sample encryption chaincode then leverages. As such, the chaincode can now marry the basic shim APIs of [Get](#) and [Put](#) with the added functionality of [Encrypt](#) and [Decrypt](#).

### Managing external dependencies for chaincode written in Go

If your chaincode requires packages not provided by the Go standard library, you will need to include those packages with your chaincode. There are [many tools available](#) for managing (or “vendoring”) these dependencies. The following demonstrates how to use [govendor](#):

```
govendor init
govendor add +external // Add all external package, or
govendor add github.com/external/pkg // Add specific external package
```

This imports the external dependencies into a local [vendor](#) directory. [peer chaincode package](#) and [peer chaincode install](#) operations will then include code associated with the dependencies into the chaincode package.

### Pros and cons of the specific solution

#### Pros

- Compatible with Node.js.
- Hyperledger Fabric architecture enables transaction processing in [three phases](#). Fewer levels of trust and verification requirements across node types. Optimizes network scalability and performance.
- Chaincode is Hyperledger Fabric’s own Smart Contract and it helps to build [complex logic](#).
- Hyperledger Fabric v2.0 Chaincode [enables multiple organizations](#) to agree to the parameters of a chaincode before it becomes active on a channel.
- [Upgrade transactions](#) in chaincode need to be approved by a sufficient number of organizations which takes away some risk for the other channel members.
- Fabric chaincode makes [chaincode packages easily readable](#) in tar files. This makes it easier to inspect the chaincode package and coordinate installation across multiple organizations.
- With Fabric v2.0 Chaincode it is possible to deploy a single chaincode package multiple times with different names on the [same channel or on different channels](#).

#### Cons

- Go language is the default language. Tutorials are made with Go language. New features will be available in Go.

# Chapter 6 Business plan SennetTa

## Actoren

In de tabel hieronder wordt aangegeven wat de rollen zijn van de boeren, de overheid, ICR3ATE en derde partijen. Daarnaast wordt er gekeken naar wat zij wel of niet mogen binnen SennetTa.

Wie	Rol	Wat mogen ze wel	Wat mogen ze niet
Boeren	365 dagen p/j emissie meten door IoT-meetapparatuur.	Data leveren. Ijken meetapparatuur.	IoT-meetapparatuur verplaatsen of manipuleren.
Overheid	Data kopen. Data vergelijken met de norm. Wetten maken en handhaven	Boetes uitdelen, advies geven. Opstellen grenswaarde	Data verkopen.
ICR3ATE	Verschaffing IoT-meetapparatuur. Onderhouden SennetTa.	Betrouwbaarheid meetapparatuur waarborgen.	Privacy schenden.
Derde partijen	Data kopen.	Data inzien.	Data leveren.

## Te meten data

### Meet entiteit:

- Milligram
- Milligram/liter
- g/mol

### Te meten stoffen:

- Stikstof: N<sub>2</sub>
- Stikstofmonoxide: NO
- Stikstofdioxide NO<sub>2</sub>
- Koolstofdioxide: CO<sub>2</sub>
- Ammoniak NH<sub>3</sub>

## Ijken

Boeren dienen de meetapparatuur te ijken per kwartaal of per halfjaar.

## Camera

Er wordt een camera opgehangen in de stal om in de gaten te houden of de meetapparatuur niet gesaboteerd wordt (Persoonlijke communicatie, 2020).

## Beloning

Er dient vastgesteld te worden voor hoeveel euro stukjes data verkocht kunnen worden. Als je bepaalde periodes de meetdrempel haalt, krijg je een beloning.

<b>Onafgebroken meetdata verstrekken</b>	<b>Beloning</b>
Één maand	€2,50
Één kwartaal	€7,50
Één jaar	€50

### **SennetTa en Smart Contract**

Om SennetTa volledig efficiënt en effectief te laten werken dient SennetTa en het Smart Contract alle benodigde regels te omvatten.

Deze regels worden gekoppeld aan actoren.

Boeren:

- Dienen toestemming te geven voor plaatsing van IoT-meetapparatuur en een camera richting de IoT-meetapparatuur.
- Dienen z.s.m. ICR3ATE in te lichten wanneer de apparatuur niet meer werkt.
- Worden beloond voor het verstrekken van data.

Overheid:

- Opstellen grenswaarden, regels en limieten.
- Grenswaarden communiceren naar ICR3ATE en boeren.

ICR3ATE:

- Zorgt voor een werkend systeem: SennetTa.
- Automatische beloningen programmeren.
- Automatische boetes programmeren.
- Data betrouwbaar meten en omzetten in begrijpbare data.
- Pleegt onderhoud aan SennetTa.
- Updatet de wetgeving vanuit de overheid.
- Meetapparatuur ijkten per kwartaal of halfjaar.
- Stelt Smart Contract op.



# Chapter 7 Conclusion

## Conclusie subvragen

De richting van het onderzoek is gedurende het onderzoek veranderd. Dit komt doordat er vanuit ICR3ATE ook onderzoek wordt gedaan naar SennetTa. Hierdoor zijn er in de uitwerkingen van dit onderzoek sommige subvragen niet beantwoord. Ook zijn er nieuwe subvragen toegevoegd. In deze conclusie worden eerst de, nog relevante, oorspronkelijke subvragen beantwoord. Vervolgens worden de nieuwe subvragen beantwoord. Tot slot wordt er antwoord gegeven op de hoofdvraag.

### Subvragen

*Op welke manier kan er economische waarde toegevoegd worden voor de boeren?*

ICR3ATE stelt dat door IoT sensoren in een stal te hangen er data kan worden verzameld die gekocht kan worden door overheden, onderzoeksinstellingen en andere partijen. Doordat er betaald wordt voor de data kan de boer hier economisch van profiteren.

*Hoe kunnen micropayments worden ingeschakeld om data te verwaarden?*

Door gebruik te maken van micropayments is het mogelijk om hele kleine betalingen toch winstgevend te maken. Doordat de transactiekosten bij een micropayment systeem zeer laag zijn kunnen transacties van 1 cent toch geld opleveren in plaats van kosten.

*Op welke manier kan er hogere sensor betrouwbaarheid worden gerealiseerd?*

Ondanks dat er met de gemeten data niet geknoeid kan worden omdat deze op een veilige en onweerlegbare manier verwerkt en opgeslagen wordt, kan er wel geknoeid worden met de manier waarop de sensoren de gegevens verzamelen. Boeren kunnen een zak om de sensoren hangen waardoor de gemeten waardes niet overeenkomen met de daadwerkelijke waardes in de stal. Tijdens het onderzoek is er vastgesteld dat een camera die gericht wordt op de sensoren ervoor zorgt dat er niet geknoeid kan worden met de sensoren. Om verval van sensoren tegen te gaan wordt er ook geregistreerd of er periodiek wordt geijkt.

### Nieuwe subvragen

*Welke micro payment provider is het meest geschikt voor SennetTa?*

Nadat er onderzoek is gedaan naar 10 micropayment systemen waren de volgende systemen het beste bevonden: Dash, Hyperledger Quilt, Interledger en Stellar. Dash en Stellar zijn afgefallen omdat deze systemen gebruik maken van cryptocurrencies, er is besloten dat de voorkeur uitgaat naar systemen die hier geen gebruik van maken. Dus zijn de aanbevelingen Hyperledger Quilt en Interledger geworden. Beide systemen zijn relatief eenvoudig te gebruiken met Hyperledger Fabric, de uiteindelijke keuze zal afhangen van ICR3ATE.

*Welk smart contract oplossing is het meest geschikt voor SennetTa?*

Tijdens het onderzoek naar smart contracts is er gekeken naar DAML, Chaincode en Ethereum. Deze drie opties zijn vergeleken op de volgende punten: implementatie, documentatie, documentatie omtrent Hyperledger Fabric en de mogelijkheid om contracten aan te passen of te updaten. Uit dit onderzoek en de vergelijking is gekomen dat Chaincode het beste gebruikt kan worden in combinatie met Hyperledger Fabric. Dit komt omdat Chaincode de smart contract taal van Hyperledger Fabric is en hierdoor gemakkelijker te implementeren is.

**Conclusie hoofdvraag**

*Welke technieken kan ICR3ATE gebruiken om doelgericht vergunnen vertrouwelijk en kosten efficiënter te maken en hier economische waarde mee toe voegen?*

Om het doelgericht vergunnen vertrouwelijk en kosten efficiënter te maken zal er gebruik worden gemaakt van IoT sensoren, Hyperledger Fabric, Hyperledger Quilt of Interledger en Chaincode.

**Doelgericht vergunnen vertrouwelijk**

Door gemeten data van IoT sensoren op te slaan in Hyperledger Fabric kan deze data niet meer aangepast worden. Hierdoor kan deze data met meer vertrouwen gepresenteerd worden aan geïnteresseerden.

**Kosten efficiënter**

Door in een smart contract de regels omtrent grenswaarden, vergunningen, boetes, beloningen en data zendingen op te nemen worden deze automatisch uitgevoerd wanneer daartoe aanleidingen wordt gegeven. Overschrijdt de boer de limieten, krijgt hij automatisch een boete. Wordt er data afgenomen door een partij, krijgt de boer automatisch een geldelijke beloning d.m.v. een micropayment.

Door dat een smart contract de administratieve taken overneemt, dalen de kosten. ICR3ATE zal Chaincode gebruiken om het smart contract te programmeren.

**Economische waarde**

Door gemeten data te verkopen m.b.v. een smart contract en micropayments voegt SennetTa economische waarde toe. Met behulp van Hyperledger Quilt of Interledger kan ICR3ATE de micropayments uitvoeren.

# Literature list

BigchainDB blog. (2018, 27 March). BigchainDB 2.0 is Byzantine Fault Tolerant. Retrieved October 30, 2020, from: <https://blog.bigchaindb.com/bigchaindb-2-0-is-byzantine-fault-tolerant-5ffdac96bc44>

Bigchaindb. (n.d.). How BigchainDB is Immutable / Tamper-Resistant. Retrieved November 14, 2020, from <https://bigchaindb-server.readthedocs.io/en/v0.5.1/topic-guides/immutable.html>

Buchman, E. (2016, June). Tendermint: Byzantine Fault Tolerance in the Age of Blockchains. Retrieved November 20, 2020, from: [https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman\\_Ethan\\_201606\\_MAsc.pdf](https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf)

Chiesa, A., Green, M., Liu, J., Miao, P., Miers, I. & Mishra, P. (2017, April 1). *Decentralized anonymous micropayments*. Retrieved October 27, 2020, from [https://link.springer.com/chapter/10.1007/978-3-319-56614-6\\_21](https://link.springer.com/chapter/10.1007/978-3-319-56614-6_21)

Chiesa, A., Green, M., Liu, J., Miao, P., Miers, I. & Mishra, P. (2017, April 1). *Decentralized anonymous micropayments*. Retrieved October 27, 2020, from [https://link.springer.com/chapter/10.1007/978-3-319-56614-6\\_21](https://link.springer.com/chapter/10.1007/978-3-319-56614-6_21)

Christidis, K. & Devetsikiotis, M. (2016). *Blockchain and Smart Contracts for the Internet of Things*. Retrieved November 25, 2020, from <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7467408>

Cosmos. (n.d.). Introduction. Retrieved 20 November, 2020, from <https://cosmos.network/intro>

DABL. (n.d.). *DABL product knowledge base*. Retrieved December 29, 2020, from <https://projectdabl.com/use-cases/>

DAML. (n.d.-a). App architecture. Retrieved December 28, 2020, from <https://docs.daml.com/getting-started/app-architecture.html>

DAML. (n.d.-b). *DAML-on-Fabric - Technical document*. Retrieved December 29, 2020, from [https://github.com/digital-asset/daml-on-fabric/blob/master/docs/DAML\\_on\\_Fabric\\_v2\\_Architecture.pdf](https://github.com/digital-asset/daml-on-fabric/blob/master/docs/DAML_on_Fabric_v2_Architecture.pdf)

DAML. (n.d.-c). *DAML Readme*. Github. Retrieved December 27, 2020, from <https://github.com/digital-asset/daml/blob/main/README.md>

Doubleday, K. (2018, November 2). *Why Blockchain Immutability Matters*. Retrieved November 12, 2020, from <https://hackernoon.com/why-blockchain-immutability-matters-8ce86603914e>

Entriken, W. (2020, November 30) *Introduction to smart contracts*. Retrieved on December 27, 2020, from <https://ethereum.org/en/developers/docs/smart-contracts/>

FINN. (2020, 17 March -g). *Research report: Forrester's 'The dawn of autonomous finance'*. Retrieved November 18, 2020, from <https://makethingsfinn.com/resources/blog/research-forrester-autonomous-payment-finance>

FINN. (2020, 17 March -h). *Research report: Forrester's 'The dawn of autonomous finance'*. Retrieved November 18, 2020, from <https://makethingsfinn.com/resources/blog/research-forrester-autonomous-payment-finance>

FINN. (n.d.-a). *Autonomous payments*. Retrieved November 11, 2020, from <https://makethingsfinn.com/product/business/autonomous-payments>

FINN. (n.d.-a). *Autonomous payments*. Retrieved November 11, 2020, from <https://makethingsfinn.com/product/business/autonomous-payments>

FINN. (n.d.-b). *Future business created today*. Retrieved November 11, 2020, from [https://makethingsfinn.com/?gclid=CjwKCAiAtK79BRAIEiwA4OskBgtcD5ifknbuinSqbeZm4zTV6pqUiY-bsHomZBZJe1rK-y2Hop2RIBoCXYcQAvD\\_BwE](https://makethingsfinn.com/?gclid=CjwKCAiAtK79BRAIEiwA4OskBgtcD5ifknbuinSqbeZm4zTV6pqUiY-bsHomZBZJe1rK-y2Hop2RIBoCXYcQAvD_BwE)

FINN. (n.d.-b). *Future business created today*. Retrieved November 11, 2020, from [https://makethingsfinn.com/?gclid=CjwKCAiAtK79BRAIEiwA4OskBgtcD5ifknbuinSqbeZm4zTV6pqUiY-bsHomZBZJe1rK-y2Hop2RIBoCXYcQAvD\\_BwE](https://makethingsfinn.com/?gclid=CjwKCAiAtK79BRAIEiwA4OskBgtcD5ifknbuinSqbeZm4zTV6pqUiY-bsHomZBZJe1rK-y2Hop2RIBoCXYcQAvD_BwE)

FINN. (n.d.-c). *Getting started*. Retrieved November 18, 2020, from <https://docs.bankingofthings.io/getting-started>

FINN. (n.d.-c). *Getting started*. Retrieved November 18, 2020, from <https://docs.bankingofthings.io/getting-started>

FINN. (n.d.-d). *Get going with our IoT payment platform*. Retrieved 30 November, 2020, from <https://makethingsfinn.com/pricing>

FINN. (n.d.-d). *IoT SDK's*. Retrieved on November 18, 2020, from <https://docs.bankingofthings.io/iot-sdk/overview>

FINN. (n.d.-e). *IoT SDK's*. Retrieved on November 18, 2020, from <https://docs.bankingofthings.io/iot-sdk/overview>

FINN. (n.d.-e). *Our business solutions*. Retrieved November 11, 2020, from <https://makethingsfinn.com/product/business/overview>

FINN. (n.d.-f). *Our business solutions*. Retrieved November 11, 2020, from <https://makethingsfinn.com/product/business/overview>

FINN. (n.d.-f). *Our toolkit gets you going*. Retrieved November 14, 2020, from <https://makethingsfinn.com/product/toolkit>

FINN. (n.d.-g). *Our toolkit gets you going*. Retrieved November 14, 2020, from <https://makethingsfinn.com/product/toolkit>

FINN. (n.d.-i) *Switching to usage-based business models*. Retrieved November 30, 2020, from <https://makethingsfinn.com/product/business/pay-per-use>

Frankenfield, J. (2020, 23 September). *Interledger Protocol*. Retrieved November 17, 2020, from <https://www.investopedia.com/terms/i/interledger-protocol.asp>

Github. (n.d.). *Finn - Banking of Things*. Retrieved November 23, 2020, from <https://github.com/BankingofThings/>

Github. (n.d.). *Finn - Banking of Things*. Retrieved November 23, 2020, from <https://github.com/BankingofThings/>

Goyal, S. (2019, May 12). *DAML- The open-source language for smart contracts originated by Digital Asset*. 101 Blockchains. <https://101blockchains.com/daml/>

Interledger. (n.d.-a). *Interledger Architecture*. Retrieved December 16, 2020, from <https://interledger.org/rfcs/0001-interledger-architecture/>

Interledger. (n.d.-b) *Interledger is an open protocol suite for sending payments across different ledgers*. Retrieved December 16, 2020, from <https://interledger.org/>

Interledger. (n.d.-c). *Interledger overview*. Retrieved December 16, 2020, from <https://interledger.org/overview.html>

Interledger. (n.d.-d). *Interledger Protocol V4*. Retrieved December 16, 2020, from <https://interledger.org/rfcs/0027-interledger-protocol-4/>

Klemens, S. (2020, October 7) *Ethereum Review: Ethereum Use Cases, Advantages & Disadvantages*. Retrieved on December 28, 2020, from <https://www.exodus.io/blog/what-is-ethereum/>

Lumen (n.d.). *Immutability and Why It Matters*. Retrieved November 12, 2020, from <https://www.ctl.io/developers/blog/post/immutability#:~:text=Immutability%20is%20the%20idea%20that,Haskell%2C%20Erlang%2C%20and%20Clojure>

Merchantmaverick (n.d.). *PayPal Review (For Businesses)*. Retrieved on December 17, 2020, from <https://www.merchantmaverick.com/reviews/paypal-review/>

Mollie (n.d.) *Wat is het minimum- en maximumbedrag per betaalmethode?* Retrieved December 5, 2020, from <https://help.mollie.com/hc/nl/articles/115000667365-Wat-is-het-minimum-en-maximumbedrag-per-betaalmethode->

Mollie (n.d.-a) *Mollie API: Simple & powerful*. Retrieved December 21, 2020, from <https://docs.mollie.com/>

Párhonyi, R., Nieuwenhuis, L.J.M. & Pras, A. (n.d.). *The fall and rise of micropayment systems*. Retrieved October 27, 2020, from [https://ris.utwente.nl/ws/files/5350260/parhonyi\\_fall\\_rise\\_micropayment\\_systems.pdf](https://ris.utwente.nl/ws/files/5350260/parhonyi_fall_rise_micropayment_systems.pdf)

Párhonyi, R., Nieuwenhuis, L.J.M. & Pras, A. (n.d.). *The fall and rise of micropayment systems*. Retrieved October 27, 2020, from [https://ris.utwente.nl/ws/files/5350260/parhonyi\\_fall\\_rise\\_micropayment\\_systems.pdf](https://ris.utwente.nl/ws/files/5350260/parhonyi_fall_rise_micropayment_systems.pdf)

Paypal (n.d.) *What are micropayments?* Retrieved November 22, 2020, from <https://www.paypal.com/us/smarthelp/article/what-are-micropayments-faq664>

PCMag (n.d.). *PayPal Review*. Retrieved on December 17, 2020, from <https://www.pcmag.com/reviews/paypal>

Pressbin (n.d.) *Transaction Costs Table for Paypal Micropayments*. Retrieved December 17, 2020, from [http://pressbin.com/tools/paypal\\_micropayments/](http://pressbin.com/tools/paypal_micropayments/)

Rijkswaterstaat (z.d.) *Toestemming milieu*. Geraadpleegd op 20 oktober 2020 via: <https://www.infomil.nl/onderwerpen/integrale/wet-algemene/toestemmingen-wabo/toestemming-milieu-0/>

Rijkswaterstaat (z.d.-b) *Omgevingsvergunning milieu bij veehouderijen*. Geraadpleegd op 20 oktober 2020 via: <https://www.infomil.nl/onderwerpen/landbouw/systematiek/omgevingsvergunning/>

Rosic, A. (n.d.). *DAML - An open-source ecosystem for building smart contract based distributed applications*. Blockgeeks. Retrieved December 27, 2020, from [https://blockgeeks.com/guides/daml-an-open-source-ecosystem-for-building-smart-contract-based-distributed-applications/#What\\_is\\_DAML](https://blockgeeks.com/guides/daml-an-open-source-ecosystem-for-building-smart-contract-based-distributed-applications/#What_is_DAML)

Scriptium (2018, 25 april) *Onderzoeksvormen op een rijtje*. Geraadpleegd op 20 oktober 2020 via <https://www.scriptium.nl/deskresearch/>

Shome, R. (2018, November 26). *Pros and Cons of Stellar*. Retrieved December 21, 2020, from <https://www.btcwires.com/block-o-pedia/pros-and-cons-of-stellar/>

Statista (2020) *Number of PayPal's total active user accounts from 1st quarter 2010 to 3rd quarter 2020*. Retrieved December 21, 2020, from <https://www.statista.com/statistics/218493/paypals-total-active-registered-accounts-from-2010/>

Stellar (n.d.-a). *Anchor basics*. Retrieved December 21, 2020, from <https://www.stellar.org/learn/anchor-basics>

Stellar (n.d.-b). *Blockchain basics*. Retrieved December 21, 2020, from <https://www.stellar.org/learn/blockchain-basics>

Stellar (n.d.-c). *Fees*. Retrieved December 21, 2020, from <https://developers.stellar.org/docs/glossary/fees/>

Stellar (n.d.-d) *Introduction*. Retrieved December 21, 2020, from <https://developers.stellar.org/docs/start/introduction/>

Stellar (n.d.-e). *Intro to Stellar*. Retrieved December 20, 2020, from <https://www.stellar.org/learn/intro-to-stellar>

Stellar (n.d.-f). *Stellar for remittance*. Retrieved December 21, 2020, from <https://www.stellar.org/learn/stellar-for-remittances>

Stellar (n.d.-g). *Stellar Lumens*. Retrieved December 21, 2020, from <https://www.stellar.org/lumens>

Stellar (n.d.-h). *The power of Stellar*. Retrieved December 21, 2020, from <https://www.stellar.org/learn/the-power-of-stellar>

Taçoğlu, C. (n.d.). *Immutability*. Retrieved November 11, 2020, from <https://academy.binance.com/en/glossary/immutability>

Telefonica (n.d.) *In brief*. Retrieved December 21, 2020, from [https://www.telefonica.com/en/web/about\\_telefonica/in-brief](https://www.telefonica.com/en/web/about_telefonica/in-brief)

TrustOS (n.d.) *Architecture*. Retrieved December 21, 2020, from <https://trustos.readthedocs.io/en/latest/architecture.html>

Wackerow, P. (2020, December 6). *Smart contract languages*. Retrieved on December 28, 2020, from <https://ethereum.org/en/developers/docs/smart-contracts/languages/>

Wong, K.S. & Kim, M.H. (2016, November). *Towards autonomous payments for internet of things*. Retrieved November 11, 2020, from [https://www.researchgate.net/publication/322140226\\_Towards\\_autonomous\\_payments\\_for\\_internet\\_of\\_things](https://www.researchgate.net/publication/322140226_Towards_autonomous_payments_for_internet_of_things)

Wong, K.S. & Kim, M.H. (2016, November). *Towards autonomous payments for internet of things*. Retrieved November 11, 2020, from [https://www.researchgate.net/publication/322140226\\_Towards\\_autonomous\\_payments\\_for\\_internet\\_of\\_things](https://www.researchgate.net/publication/322140226_Towards_autonomous_payments_for_internet_of_things)

# Appendix 1

























Betaalmethode	Markt	Per transactie	
 iDEAL	Nederland	€ 0,29	
 Mastercard	Persoonlijke kaarten (Europese Unie)	€ 0,25	+ 1,8%
	Zakelijke & buiten-EU kaarten	€ 0,25	+ 2,8%
 Visa	Persoonlijke kaarten (Europese Unie)	€ 0,25	+ 1,8%
	Zakelijke & buiten-EU kaarten	€ 0,25	+ 2,8%
 American Express	Wereldwijd	€ 0,25	+ 2,8%
 Maestro	Nederland	€ 0,39	
 Apple Pay	Wereldwijd	afhankelijk van kaart	
 Bancontact	België	€ 0,39	
 Maaltijd-, eco- en cadeaucheque	België	€ 0,50	+ 0,5%
 Klarna: Achteraf betalen	Oostenrijk	€ 0,35	+ 2,99%
	Finland	€ 0,59	+ 2,79%
	Duitsland	€ 0,35	+ 2,99%
	Nederland	€ 1,00	+ 2,99%
	België	€ 1,00	+ 2,99%
 Klarna: Gespreid betalen	Finland	€ 0,59	+ 0,99%
	Duitsland	2,99%	
	Oostenrijk	2,99%	
 PayPal	Wereldwijd	€ 0,10	+ PayPal
 SEPA-incasso	Europa	€ 0,25	
 SEPA-overboeking	Europa	€ 0,25	
 SOFORT Banking	Europa	€ 0,25	+ 0,9%
 EPS	Oostenrijk	€ 0,25	+ 1,5%
 Cartes Bancaires	Frankrijk	€ 0,25	+ 1,8%
 Giropay	Duitsland	€ 0,25	+ 1,5%
 KBC/CBC-betaalknop	België	€ 0,25	+ 0,9%
 Belfius betaalknop	België	€ 0,25	+ 0,9%
 ING Home'Pay	België	€ 0,25	+ 0,9%
 Postepay	Italië	€ 0,25	+ 1,8%
 Cadeaukaarten	Nederland	€ 0,25	+ Brand
 paysafecard	Wereldwijd	15%	
 Przelewy24	Polen	€ 0,25	+ 2,2%



Figure 1.1 Payment methods Mollie. Retrieved from Mollie, n.d.  
(<https://help.mollie.com/hc/nl/articles/115000667365-Wat-is-het-minimum-en-maximumbedrag-per-betaalmethode>). Copyright © 2020 Mollie B.V.

## Appendix 2

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Figure 2.1 Solidity example code. Taken from *Smart contract languages* by Wackerow, n.d. (<https://ethereum.org/en/developers/docs/smart-contracts/languages/>)

## Appendix 3

```

# Open Auction

# Auction params
# Beneficiary receives money from the highest bidder
beneficiary: public(address)
auctionStart: public(uint256)
auctionEnd: public(uint256)

# Current state of auction
highestBidder: public(address)
highestBid: public(uint256)

# Set to true at the end, disallows any change
ended: public(bool)

# Keep track of refunded bids so we can follow the withdraw pattern
pendingReturns: public(HashMap[address, uint256])

# Create a simple auction with `_bidding_time`
# seconds bidding time on behalf of the
# beneficiary address `_beneficiary`.
@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time

# Bid on the auction with the value sent
# together with this transaction.
# The value will only be refunded if the
# auction is not won.
@external
@payable
def bid():
    # Check if bidding period is over.
    assert block.timestamp < self.auctionEnd
    # Check if bid is high enough
    assert msg.value > self.highestBid
    # Track the refund for the previous high bidder
    self.pendingReturns[self.highestBidder] += self.highestBid
    # Track new high bid
    self.highestBidder = msg.sender
    self.highestBid = msg.value

# Withdraw a previously refunded bid. The withdraw pattern is
# used here to avoid a security issue. If refunds were directly
# sent as part of bid(), a malicious bidding contract could block
# those refunds and thus block new higher bids from coming in.
@external
def withdraw():
    pending_amount: uint256 = self.pendingReturns[msg.sender]
    self.pendingReturns[msg.sender] = 0
    send(msg.sender, pending_amount)

# End the auction and send the highest bid
# to the beneficiary.
```

Figure 3.1. Vyper example code. Taken from *Smart contract languages* by Wackerow, n.d. (<https://ethereum.org/en/developers/docs/smart-contracts/languages/>)

See next page

```

@external
def endAuction():
    # It is a good guideline to structure functions that interact
    # with other contracts (i.e. they call functions or send Ether)
    # into three phases:
    # 1. checking conditions
    # 2. performing actions (potentially changing conditions)
    # 3. interacting with other contracts
    # If these phases are mixed up, the other contract could call
    # back into the current contract and modify the state or cause
    # effects (ether payout) to be performed multiple times.
    # If functions called internally include interaction with external
    # contracts, they also have to be considered interaction with
    # external contracts.

    # 1. Conditions
    # Check if auction endtime has been reached
    assert block.timestamp >= self.auctionEnd
    # Check if this function has already been called
    assert not self.ended

    # 2. Effects
    self.ended = True

    # 3. Interaction
    send(self.beneficiary, self.highestBid)

```

Figure 3.2. Vyper example code. Taken from *Smart contract languages* by Wackerow, n.d. (<https://ethereum.org/en/developers/docs/smart-contracts/languages/>)