Building an Application on ICP

Bachelor's Thesis

Enrico Mayor
mayore@ethz.ch

Group
Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors
Robin Fritsch
Prof. Dr. Roger Wattenhofer

June 2022

Contents

1 Introduction **3**

2 Technical Analysis **4**
- 2.1 Overview . . . 4
- 2.2 Layers . . . 5
  - 2.2.1 Peer-to-Peer Layer . . . 6
  - 2.2.2 Consensus Layer . . . 6
  - 2.2.3 Routing Layer . . . 6
  - 2.2.4 Execution Layer . . . 7
- 2.3 Network Nervous System . . . 7
- 2.4 Subnets . . . 8
- 2.5 Canisters . . . 9
  - 2.5.1 Query Calls . . . 9
  - 2.5.2 Update Calls . . . 10
- 2.6 Chain-Key Cryptography . . . 11
- 2.7 Consensus . . . 12
- 2.8 Internet Identity . . . 16
- 2.9 Motoko . . . 17
- 2.10 Conclusion . . . 17

3 Integrations **18**
- 3.1 Direct Integration with Ethereum . . . 18
  - 3.1.1 ICP smart contracts calling Ethereum smart contracts . . . 18
  - 3.1.2 Ethereum smart contracts calling ICP smart contracts . . . 18
- 3.2 Direct Integration with Bitcoin . . . 18

4 Tokenomics **19**
- 4.1 Locking . . . 19
- 4.2 Transforming . . . 19

5 Case Study **20**
- 5.1 Objective . . . 20
- 5.2 Visualization . . . 20
- 5.3 Development . . . 21
  - 5.3.1 Front-End . . . 22
  - 5.3.2 Back-End . . . 22
- 5.4 Application Overview . . . 25
- 5.5 Analysis . . . 28

6 Summary **30**

1 Introduction

According to its creator, the DFINITY foundation, a non-profit organization with headquarters in Zurich, the Internet Computer is the fastest and most scalable general-purpose blockchain which extends the Internet with computation. Decentralized applications (dapps) and smart contracts can run on the Internet Computer and serve their content directly to the user in a browser. Additionally, users can securely interact with and authenticate to dapps using Internet Identity, ICP's anonymous blockchain authentication framework. In other words, ICP seems to be an alternative to centralized cloud computing services like AWS or Google Cloud with the bonus that it is decentralized and, according to the DFINITY foundation, infinitely scalable. [1]

In this bachelor thesis, supervised by Prof. Dr. Roger Wattenhofer and Robin Fritsch, we will analyze the inner workings of the Internet Computer (ICP) and build a prototype project on top of it. We will try to find out how secure, fast, and scalable it is and if it is, as promised by the DFINITY foundation, the future of the Internet.

We will start with a technical analysis of ICP, where we deconstruct its inner workings. The literature on ICP is very one-sided. On the one hand, their marketing literature is reckoned by many as "over-the-top" and their developer literature is highly technical. In this thesis, we will try to strike the middle ground between the two by appealing to the broader audience, which lies somewhere between consumers and developers. After the technical analysis, we will look at the programming language Motoko, a new programming language for smart contracts designed to seamlessly support the programming model of ICP, followed by an outlook on what lies ahead for ICP, namely the Bitcoin and Ethereum integration. After having dug deep into the technical aspects of ICP, we will look at its tokenomics, which is the topic of understanding the supply and demand characteristics of a cryptocurrency. Having covered everything technical alongside the finance aspect of ICP, we will present the prototype developed on ICP for this bachelor thesis by defining its objective, showcasing the development and deployment process, and analyzing its speed and scalability. In the last chapter of this thesis, we will summarize everything we have learned and answer the question if ICP is really "the future of the internet".

[1] https://dfinity.org/howitworks/

2 Technical Analysis

2.1 Overview

The Internet Computer network consists of a hierarchy of network building blocks. Data centers on the first level (of which there eventually will be thousands) host the node hardware. These data centers host many nodes, which leads us to the second level. On the second level, there are nodes (of which there eventually will be millions) combined to a subnet, the third level. These subnets host canisters (of which there eventually will be billions), which are the compute-unit of ICP. These canisters are what developers upload their compiled web-assembly code on.

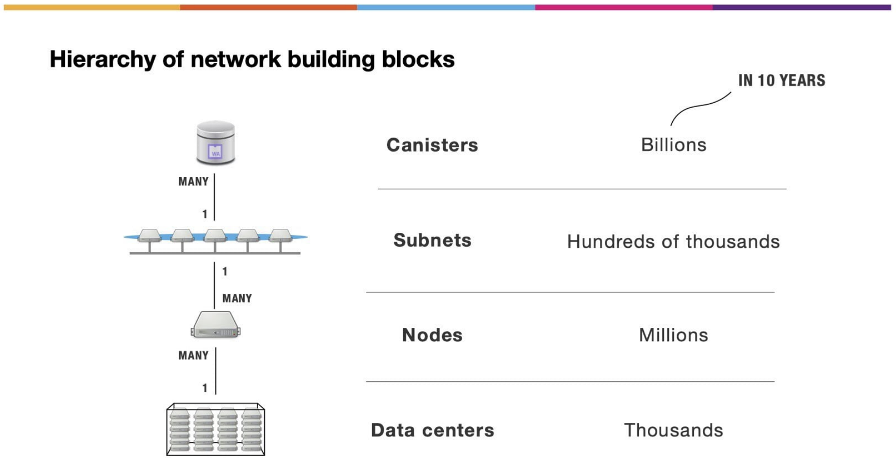


Figure 1: Hierarchy of network building blocks

The Network Nervous System (NNS) is the managing unit of the data centers. It plays a similar role to ICAN on the Internet and permits data centers to join.

Another vital concept are subnets. A subnet hosts a subset of the canisters. Subnets are created using nodes drawn from different data centers, which the NNS coordinates. All the nodes of a subnet replicate the content of one another. The NNS also can split or merge subnets to distribute the load (e.g., requests by other canisters or end-users) as efficiently as possible, which can happen without a service interruption.

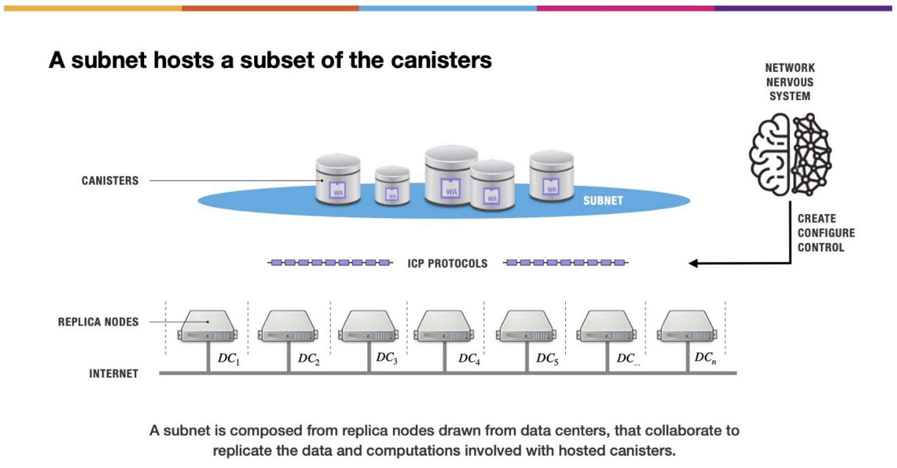

A subnet hosts a subset of the canisters
NETWORK NERVOUS SYSTEM
CANISTERS
WA
SUBNET
CREATE CONFIGURE CONTROL
ICP PROTOCOLS
REPLICA NODES
DC1
DC2
DC3
DC4
DC5
DC...
DCn
INTERNET
A subnet is composed from replica nodes drawn from data centers, that collaborate to replicate the data and computations involved with hosted canisters.

Figure 2: Subnets

There are different subnet types. A particular subnet hosts the NNS, which developers do not have access to. Developers can target a specific subnet type for their canister (e.g., make code for a specific subnet type) like a data subnet, system subnet, or fiduciary subnet, which allows the canister to have specific properties or capabilities.

As already mentioned, subnets are used to host canisters. Canisters are bundles of Web Assembly byte code and 4 KB memory pages. That Web Assembly byte code gets created by compiling the code of Motoko or Rust. Canisters can interact with each other via API calls. [2]

The current network status of the Internet Computer (as of June 1st 2022) looks as follows: [3]

- Node providers: 44
- Node machines: 518
- Subnets: 35
- Canisters: 75'930

2.2 Layers

The Internet Computer consists of four layers: [4]

- peer-to-peer layer

[2]https://medium.com/dfinity/a-technical-overview-of-the-internet-computer-f57c62abc20f
[3]https://dashboard.internetcomputer.org/
[4]https://dfinity.org/whitepaper.pdf

- consensus layer
- routing layer
- execution layer

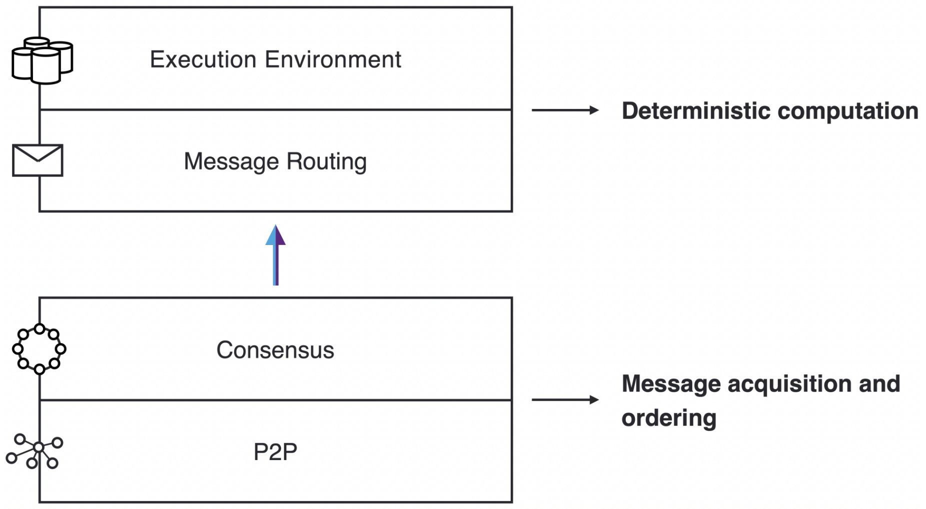


Figure 3: Layers

2.2.1 Peer-to-Peer Layer

The peer-to-peer layer's job is to transport messages between replicas in a subnet. The goal is the following: If an honest replica broadcasts a message, then that message will eventually be received by all honest replicas in the subnet, which is called a "best effort" broadcast channel.

2.2.2 Consensus Layer

The consensus layer's job is to set up a global ordering of all inputs such that all replicas in a subnet will process such inputs in the same order. The protocol used here is based on a blockchain. The root of the tree is called the genesis block, and each non-genesis block in the tree contains a payload consisting of a sequence of inputs and a hash of the block's parent in the tree. The inputs in the payloads of the blocks along the longest path are the ordered inputs processed by the execution layer of the Internet Computer.

2.2.3 Routing Layer

The routing layer's job is to take the payloads from the consensus layer and give it to the execution layer for processing, which updates the state of the canisters and generates an output that the routing layer processes.

2.2.4 Execution Layer

The execution layer's job is to process the input one at a time. Processing the input means updating the state of the canister and giving an output back to the routing layer.

2.3 Network Nervous System

The Network Nervous System (NNS), a tokenized open governance system, supervises and manages the Internet Computer. The NNS stores the information of which nodes belong to which subnet, alongside many more things. It also decides how to update that information (e.g., adding new nodes to subnets). A few unique canisters make up the NNS, which is what we will take a look at now.

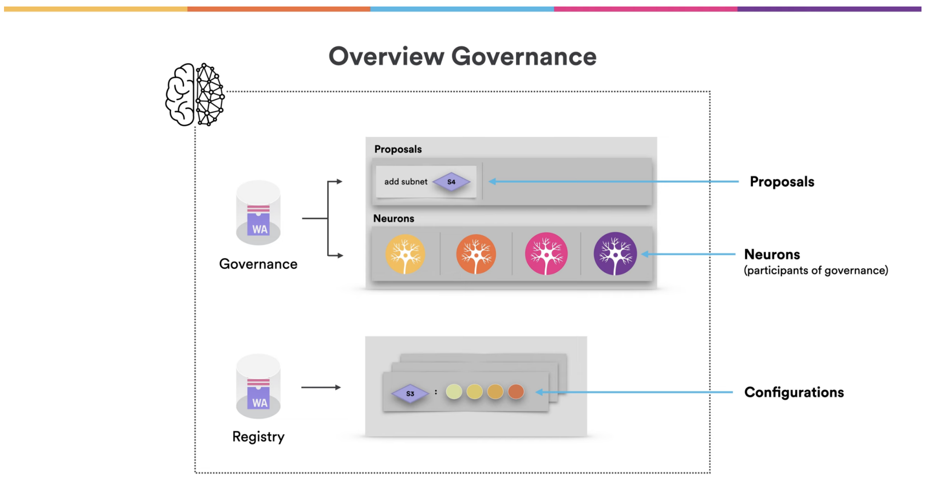


Figure 4: Governance and Registry Canisters

The governance canister stores proposals and neurons. Proposals are suggestions by the community on how ICP can or should be improved. These proposals can be voted on by people who stake ICP tokens (see tokenomics section). Neurons determine who is allowed to participate in governance.

The registry canister stores the configuration of the whole Internet Computer (e.g., which nodes belong to which subnets). For example, in the above figure, the registry canister stores the information that subnet S3 stores these four nodes.

Another paramount canister in the NNS is the ledger canister, which stores accounts and transactions. Accounts keep track of how many tokens a given entity has alongside the address of that account. Tokens can be sent from one account to another, which is recorded in transaction history. [5]

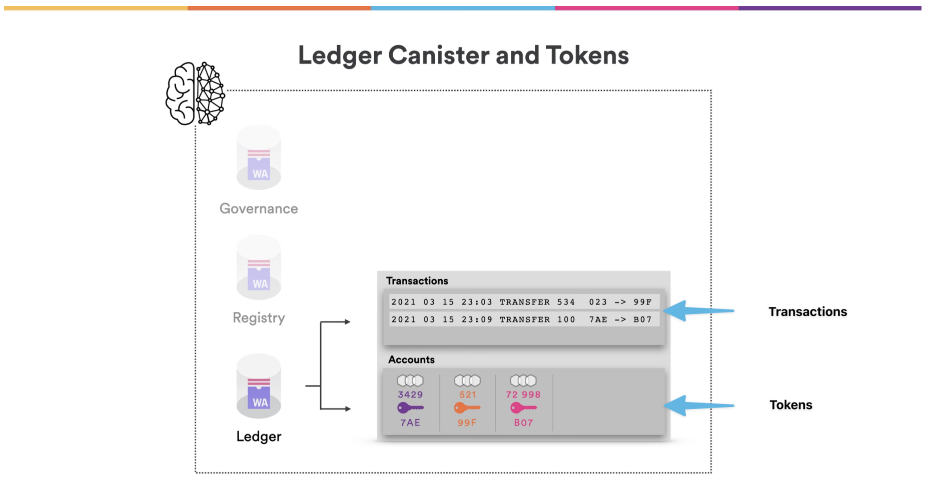

Ledger Canister and Tokens
Governance
Registry
Ledger
Transactions
2021 03 15 23:03 TRANSFER 534 023 -> 99F
2021 03 15 23:09 TRANSFER 100 7AE -> B07
Accounts
3429
7AE
521
99F
72 998
B07
Transactions
Tokens

Figure 5: Ledger Canister

2.4 Subnets

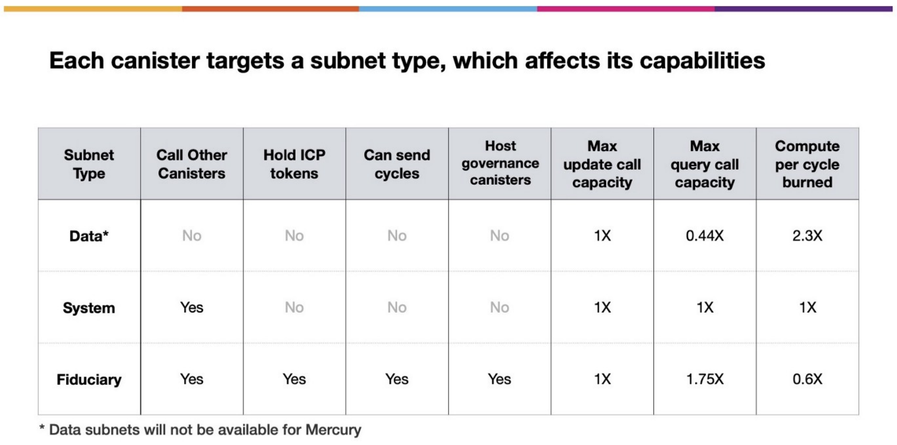
Subnets are the most crucial concept of ICP and are the fundamental building blocks of the system. In simple terms, a subnet is a subset of canisters that run on ICP. Node machines in a given subnet are drawn from independent data centers, ensuring security using Byzantine fault-tolerant technology and cryptography developed by cryptographers at DFINITY. Within a subnet, all canisters replicate each other's content and computations, which serve the purpose of load-balancing and security. One particular data center cannot simply say that the state of a canister is X if all other canisters in the subnet agree that it is, in reality, Y.

A feature of subnets is that they are transparent to users and software. The only thing that one needs to know (e.g., the user or canister software) is the identity of the canister in order to call the function that it shares. In other words, canisters can communicate with each other using function calls while only needing the canister ID to do so. This is quite similar to how the Internet works: On the Internet, the only thing a user needs to know is the IP address of the computer he wants to interact with alongside its TCP port that it listens to. ICP works pretty much the same where any software/canister given permission can call any other software/canister directly without knowing anything about the counter-party. The only thing that is needed for that is the canister ID.

[5]https://medium.com/dfinity/the-network-nervous-system-governing-the-internet-computer-1d176605d66a

In order to properly load balance, the NNS can split or merge subnets as needed, which further ensures the transparency of the subnets.

When someone contributes software to ICP, they have to specify a specific subnet type such as "data," "system," or "fiduciary."



Each canister targets a subnet type, which affects its capabilities

Subnet Type	Call Other Canisters	Hold ICP tokens	Can send cycles	Host governance canisters	Max update call capacity	Max query call capacity	Compute per cycle burned
Data*	No	No	No	No	1X	0.44X	2.3X
System	Yes	No	No	No	1X	1X	1X
Fiduciary	Yes	Yes	Yes	Yes	1X	1.75X	0.6X

* Data subnets will not be available for Mercury

Figure 6: Subnet Types

As seen in the figure above, each type gives specific capabilities and permissions. For example, if someone would like to call other canisters, one would need a "system" or "fiduciary" subnet since the "data" subnet does not allow this. [6]

2.5 Canisters

As previously discussed, subnets host canisters. A canister is WebAssembly byte code that can run on a WebAssembly virtual machine. That WebAssembly byte code gets created by compiling down a programming language compatible with ICP, like Motoko or Rust. Canisters can communicate via publicly specified API, as presented in the previous chapter. In simple terms, a canister is a bucket where one can upload his compiled code that then runs on ICP. We will dive deep into how to set up a canister on ICP in a later chapter. [7]

A function on a canister can be invoked in two ways. Either as a query call or an update call.

2.5.1 Query Calls

Query calls do not make persistent changes to the state. Any changes they make to memory are discarded after running, making them performant and inexpensive. They do this by not running on every node of the subnet, which means a

[6] https://dfinity.org/whitepaper.pdf
[7] https://dfinity.org/whitepaper.pdf

lower security level. A use case for a query call could be a user requesting an update on his social media feed (read-only). [8]

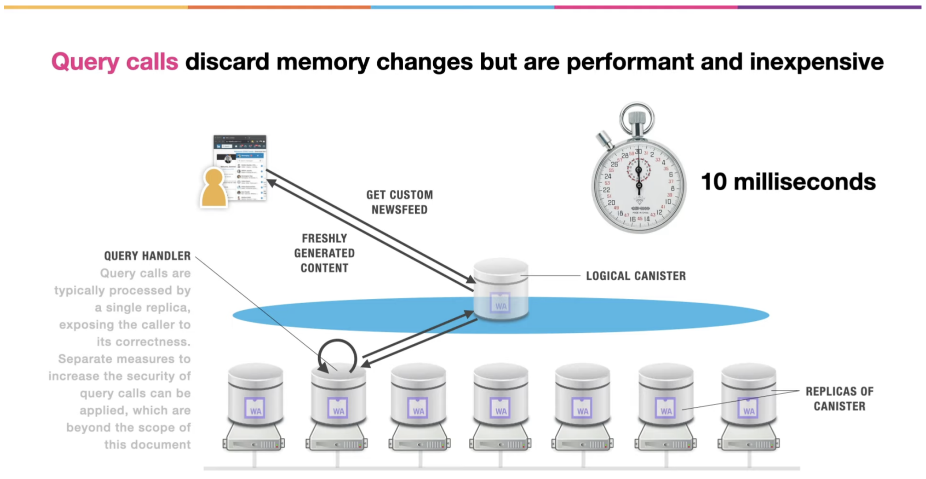

Query calls discard memory changes but are performant and inexpensive
10 milliseconds
GET CUSTOM NEWSFEED
FRESHLY GENERATED CONTENT
QUERY HANDLER
Query calls are typically processed by a single replica, exposing the caller to its correctness. Separate measures to increase the security of query calls can be applied, which are beyond the scope of this document
LOGICAL CANISTER
REPLICAS OF CANISTER
WA

Figure 7: Query Calls

The way a query call works is quite simple:

1. User makes call M to canister C, which the user's client sends to a boundary node of the subnet, which then sends M to a replica of the subnet that hosts canister C.

2. Canister M will do the computation on M and sends the result back to the user via the boundary node.

2.5.2 Update Calls

Update calls make persistent changes to the state and are tamperproof because they run on every subnet node. A use case for an update call could be a user submitting a new post to his social media account hosted in a canister. [9]

[8]https://dfinity.org/whitepaper.pdf
[9]https://dfinity.org/whitepaper.pdf

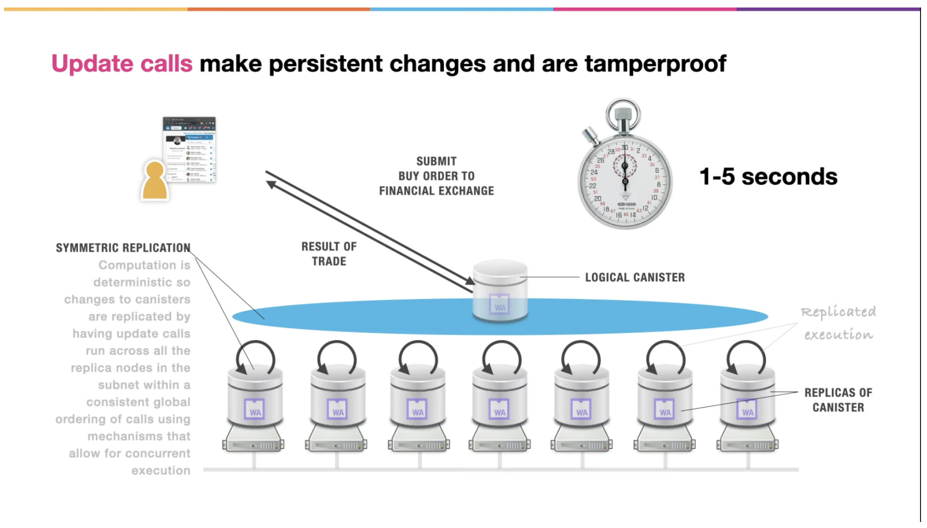

Update calls make persistent changes and are tamperproof
SUBMIT BUY ORDER TO FINANCIAL EXCHANGE
1-5 seconds
SYMMETRIC REPLICATION
Computation is deterministic so changes to canisters are replicated by having update calls run across all the replica nodes in the subnet within a consistent global ordering of calls using mechanisms that allow for concurrent execution
RESULT OF TRADE
LOGICAL CANISTER
Replicated execution
REPLICAS OF CANISTER
WA

Figure 8: Update Calls

The way an update call works is a bit more involved:

1. User makes a call M to canister C, which the user's client sends to a boundary node of the subnet, sending M to a replica of the subnet that hosts canister C.

2. That replica will broadcast M to all other replicates on that subnet (using the peer-to-peer layer).

3. The leader of the next round will bundle M together with other inputs to create the payload for block B proposed by the leader.

4. After block B is finalized, the payload is sent to the routing layer for processing.

5. The routing layer will place M in the input queue of canister C.

6. The execution layer will process M and update the internal state of canister C.

7. Canister C computes a response R to the request M which is then recorded in the ingress history data structure.

2.6 Chain-Key Cryptography

Unlike other blockchains, ICP does not use proof of work or proof of stake to process transactions. Instead, the subnets communicate with each other using chain-key cryptography. The Internet Computer will, down the road, run on millions of nodes at scale, and chain-key cryptography is what will enable that.

As we already discussed, to achieve the correctness of results, each canister runs on multiple nodes instead of just one, which removes the single point of failure possibility. In order to make this multi-node infrastructure work, messages need to be jointly signed by all the nodes hosting the canister that the user queries for a result, which is what chain-key cryptography enables. All nodes have a part of a secret key that, if put together, combines into the entire secret key, which enables them to sign a message with the requested result jointly. The signature that gets created this way can be verified using the public key of the Internet Computer, of which there exists precisely one. The significant advantage (for example, to the Ethereum ecosystem) is that even though ICP runs millions of nodes and thousands of subnets, all needed to validate any results from the subnets is one public key.

Now, for ICP to scale, the nodes are partitioned into subnets, and each subnet has its public key with which one can authenticate messages. All the nodes included in a subnet have a part of the secret key. If more than the required threshold of nodes agrees, they can use their part of the key to sign a message jointly using the so-called threshold signature scheme. The user can verify the message's authenticity using the subnet's public key.

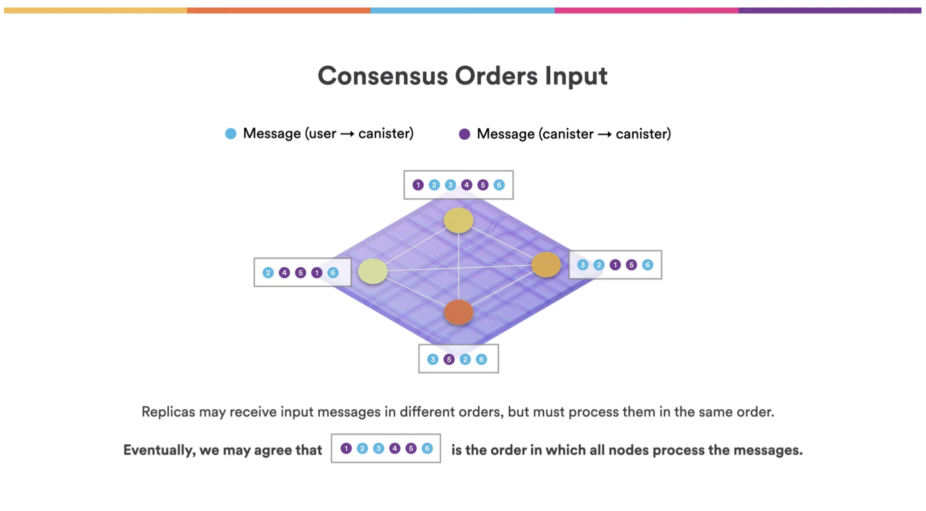
Putting all of that together, we can conclude that to verify a response from ICP, the only thing needed is a 48-byte public key. To validate a smart contract outcome on Ethereum, an Ethereum client needs to download more than 400 gigabytes, which will grow linearly with time. [10]

2.7 Consensus

The consensus problem in ICP boils down to the replicas in a subnet not always receiving the messages in the same order and then finding a global order per subnet in which to handle the messages. All the nodes within a subnet run the ICP consensus algorithm to agree on which messages to run in which order. [11]

[10] https://medium.com/dfinity/chain-key-technology-one-public-key-for-the-internet-computer-6a3644901e28

[11] https://medium.com/dfinity/achieving-consensus-on-the-internet-computer-ee9fbfbafcbc

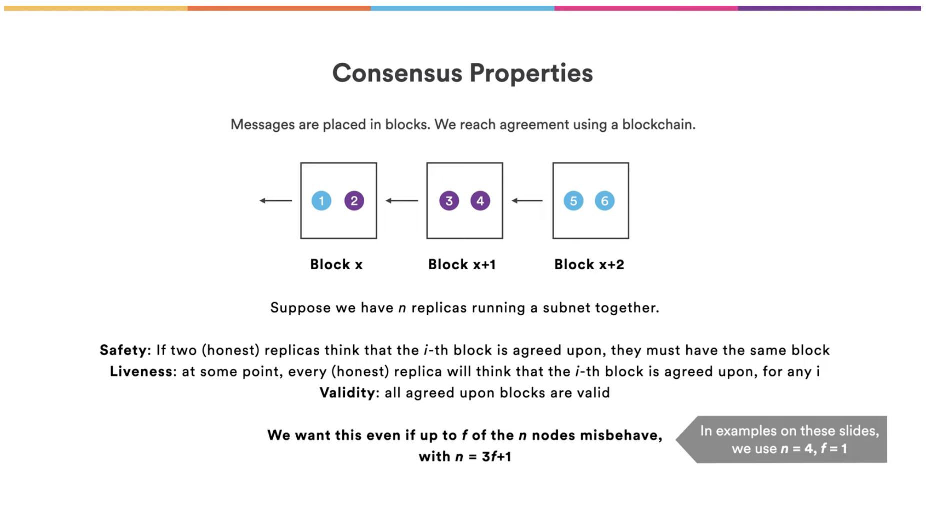

Consensus Orders Input

Replicas may receive input messages in different orders, but must process them in the same order.

Eventually, we may agree that 1 2 3 4 5 6 **is the order in which all nodes process the messages.**

Figure 9: Consensus Orders Input

In order to reach that consensus, ICP uses a blockchain. The messages that a subnet should process are grouped into blocks, and each block points to the next block, which forms a blockchain. The goal is then for all the replicas to agree on the blockchain, which gives the ordering.


Consensus Properties

Messages are placed in blocks. We reach agreement using a blockchain.

Suppose we have *n* replicas running a subnet together.

Safety: If two (honest) replicas think that the *i*-th block is agreed upon, they must have the same block
Liveness: at some point, every (honest) replica will think that the *i*-th block is agreed upon, for any i
Validity: all agreed upon blocks are valid

We want this even if up to *f* of the *n* nodes misbehave, with *n* = 3*f*+1

In examples on these slides, we use ***n* = 4, *f* = 1**

Figure 10: Blockchain

Essentially, four main components enable the consensus algorithm:

- Block making: creates candidate blocks with which the blockchain gets built

- Notarization: identifies valid blocks with which the valid blockchain gets built
- Random beacon: gives randomness to enhance the protocol further
- Finalization: an indicator for when an agreement is reached

Any node on the subnet can be the block maker, and this block maker will have some messages available that the subnet should process. The block maker then groups these messages and proposes a new block that could be appended to the blockchain. The catch is that we need more than one block maker because, otherwise, one block maker could be malicious and cause issues in the network.

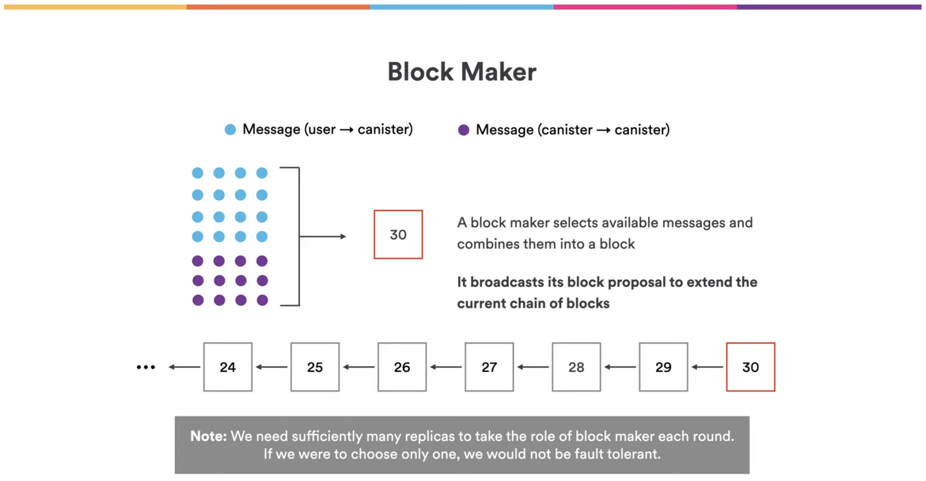

Block Maker
Message (user → canister)
Message (canister → canister)
30
A block maker selects available messages and combines them into a block
It broadcasts its block proposal to extend the current chain of blocks
...
24
25
26
27
28
29
30
Note: We need sufficiently many replicas to take the role of block maker each round. If we were to choose only one, we would not be fault tolerant.

Figure 11: Block Maker

Since any node could be a block maker, we need a process to find valid blocks. In other words, every block must be notarized, ensuring that a valid block proposal is published every round. Nodes can issue notarization shares if they agree that a block is valid, and if a certain threshold of notarization shares is hit, the new block will be notarized. It could be that a node signs multiple valid blocks in order to ensure that at least one block becomes fully notarized.

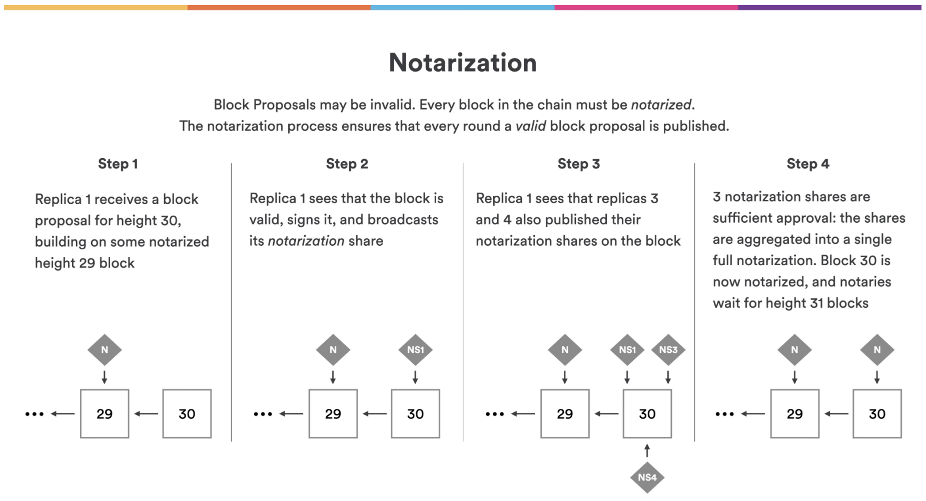

Notarization
Block Proposals may be invalid. Every block in the chain must be notarized.
The notarization process ensures that every round a valid block proposal is published.
Step 1
Replica 1 receives a block proposal for height 30, building on some notarized height 29 block
Step 2
Replica 1 sees that the block is valid, signs it, and broadcasts its notarization share
Step 3
Replica 1 sees that replicas 3 and 4 also published their notarization shares on the block
Step 4
3 notarization shares are sufficient approval: the shares are aggregated into a single full notarization. Block 30 is now notarized, and notaries wait for height 31 blocks
N
NS1
NS3
NS4
29
30

Figure 12: Notarization

In order to reduce the number of notarized blocks (which leads to less overall complexity in the system), we introduce the so-called random beacon, a random value shared by the replicas of the subnet. This random beacon is used to rank block makers, which gives an ordering on which valid block maker to choose, choosing a block of the respective block maker.

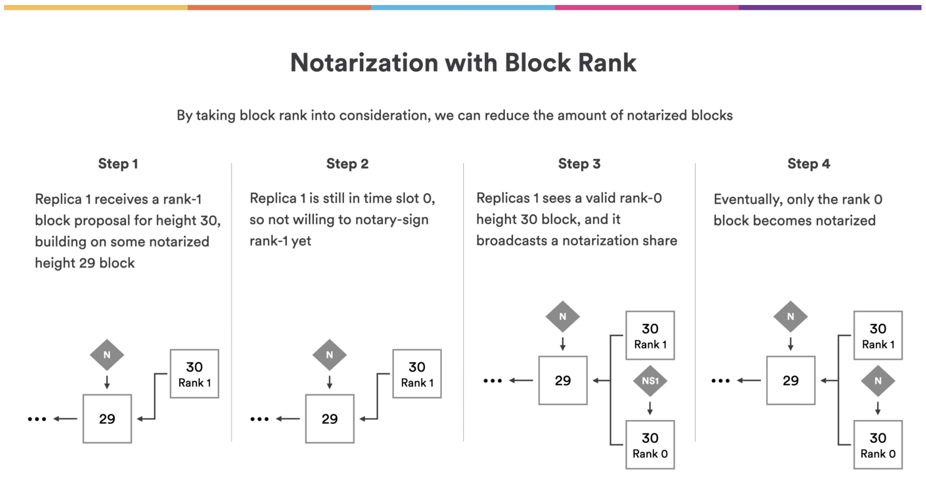

Notarization with Block Rank
By taking block rank into consideration, we can reduce the amount of notarized blocks
Step 1
Replica 1 receives a rank-1 block proposal for height 30, building on some notarized height 29 block
Step 2
Replica 1 is still in time slot 0, so not willing to notary-sign rank-1 yet
Step 3
Replicas 1 sees a valid rank-0 height 30 block, and it broadcasts a notarization share
Step 4
Eventually, only the rank 0 block becomes notarized
N
NS1
29
30 Rank 1
30 Rank 0

Figure 13: Random Beacon

Notaries create finalization shares on a block if they did not notary-sign any other block at that height. Whenever we see such finalization, we know the blockchain can be trusted up to that point because that is proof that no other finalized block at that height can exist.

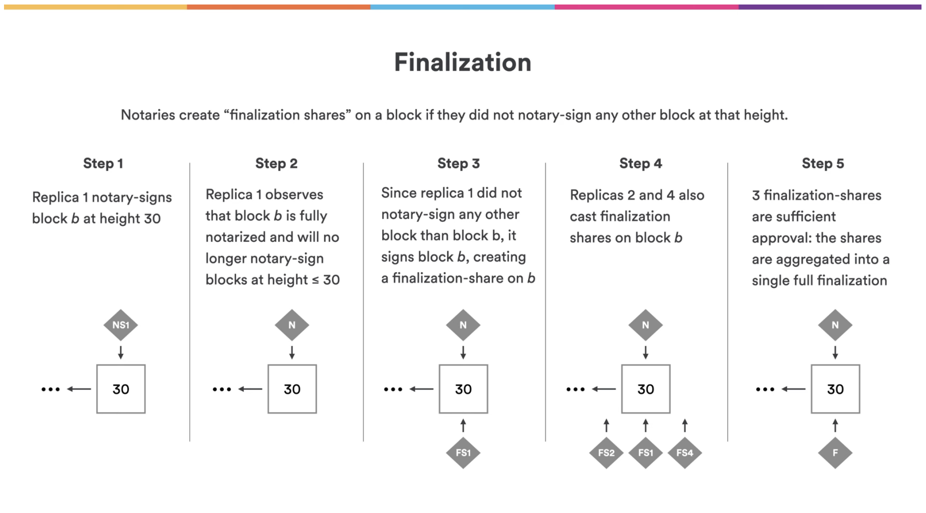

Finalization
Notaries create "finalization shares" on a block if they did not notary-sign any other block at that height.
Step 1
Replica 1 notary-signs block *b* at height 30
Step 2
Replica 1 observes that block *b* is fully notarized and will no longer notary-sign blocks at height ≤ 30
Step 3
Since replica 1 did not notary-sign any other block than block b, it signs block *b*, creating a finalization-share on *b*
Step 4
Replicas 2 and 4 also cast finalization shares on block *b*
Step 5
3 finalization-shares are sufficient approval: the shares are aggregated into a single full finalization
NS1
N
30
FS1
FS2
FS4
F

Figure 14: Finalization

2.8 Internet Identity

The Internet Identity lets users authenticate themselves anonymously to dapps on the Internet Computer. Instead of having a password for every website one visits, one has his Internet Identity. Essentially, it allows users to authenticate to dapps using one of their devices (e.g., face-id on iPhone or fingerprint check on laptops) instead of passwords, which is made possible by the already discussed Chain Key Cryptography. The Internet Identity runs on any device that uses the so-called WebAuthn standard. If a user runs on a device that does not support this standard, then that user can use an HSM device such as a YubiKey. The process of creating such an Internet Identity is quite simple. Simply go to identity.ic0.app (a page that runs on ICP) and follow the steps there.

When a user sets up his Internet Identity, the security chip on his device will generate a unique cryptographic key. The public key part of that will be stored on ICP together with the Identity Anchor. When a user loads a website on ICP (e.g., the front end of a given canister smart contract), that website shows a button with which the user can authenticate himself. This is similar to how it is handled in the Ethereum ecosystem. The user also has a button in dapps with which he can connect his wallet that practically pseudo-anonymously authenticates him towards that dapp. Returning to ICP, this button then launches the Internet Identity pop-up where the authentication takes place. In contrast to signing in via Single Sign-On (SSO), the Internet Identity authenticates on the user side rather than the server side, leading to less exposure to big tech firms and more security. [12]

2.9 Motoko

Motoko is the new open-source programming language designed for the Internet Computer. It is strongly typed, actor-based, and has built-in support for orthogonal persistence and asynchronous message passing. The exciting thing about Motoko is that a developer will not need the special knowledge usually needed for blockchain programming languages. It can be used as an ordinary programming language. The Motoko compiler takes the source code and does all the usual things (e.g., runs it through a parser, generates an abstract syntax tree, does type-checking) and then generates WebAssembly code. It does that because by doing so, any canister running on WebAssembly can communicate with any other canister also running WebAssembly without the source language needing to be necessarily Motoko. In fact, it could be any programming language that compiles to WebAssembly (e.g., Rust). [13]

2.10 Conclusion

In the previous chapters, we have learned that the Internet Computer consists of a hierarchy of network building blocks. The data centers at the bottom run many nodes, which are put together into a subnet that hosts canisters. All of that is being controlled by the NNS, the tokenized open governance system of ICP. Unlike other blockchains, ICP does not use proof of work or proof of stake to process transactions. Instead, the subnets communicate with each other using chain-key cryptography. All the nodes within a subnet run the consensus algorithm of ICP to agree on which messages to run in which order, for which ICP uses a blockchain.

[12]https://medium.com/dfinity/web-authentication-and-identity-on-the-internet-computer-a9bd5754c547

[13]https://medium.com/dfinity/motoko-a-programming-language-designed-for-the-internet-computer-is-now-open-source-8d85da4db735

3 Integrations

This section will introduce some new updates that are coming to the Internet Computer, namely the direct Bitcoin and Ethereum integration as a teaser of what lies ahead for ICP.

3.1 Direct Integration with Ethereum

This integration aims to enable Ethereum smart contracts to call into smart contracts directly (e.g., canisters) on the Internet Computer and vice versa. [14]

3.1.1 ICP smart contracts calling Ethereum smart contracts

In order to accommodate this feature, the Internet Computer introduces support for a threshold variant of ECDSA, which is the crypto scheme that secures Bitcoin and Ethereum balances and its smart contracts. This will enable canisters to create Ethereum transactions against public keys. The great thing for developers is that ICP introduces a so-called proxy smart contract (running on ICP) which makes creating a transaction on the Ethereum network as simple as calling a function.

3.1.2 Ethereum smart contracts calling ICP smart contracts

This feature cannot be solved using cryptography. However, the Internet Computer can copy new Ethereum blocks into the proxy contract (see the previous section) and that contract can detect when a block has been finalized. Then the proxy contract can scan the block for calls from Ethereum smart contracts to ICP smart contracts and the result of previous calls. The proxy contract will then communicate with the smart contract on ICP and return the respective values.

3.2 Direct Integration with Bitcoin

This integration aims to establish a connection to the Bitcoin ledger using Chain Key Cryptography. This enables developers to build canisters on ICP that directly communicate with the Bitcoin network without an intermediary. In essence, this lets canisters effectively become Bitcoin wallets that can receive and send BTC. This is huge because it brings smart contracts to the Bitcoin network, allowing the liquidity of Bitcoin to flow into a new hub for DeFi applications. In other words, we will probably soon have decentralized, Bitcoin-based lending and borrowing protocols similar to AAVE or Compound on Ethereum. [15]

[14]https://medium.com/dfinity/internet-computer-ethereum-integration-explained-6967456e35f9

[15]https://medium.com/dfinity/the-internet-computers-bitcoin-developer-preview-is-now-available-85ce1df6b17d

4 Tokenomics

Tokenomics is the topic of understanding a cryptocurrency's supply and demand characteristics. The Internet Computer has two native tokens that play the central role in the tokenomics of the Internet Computer ecosystem. The first token is called ICP, and it is a governance token, which means that it can be used to vote on decisions that influence the ICP ecosystem. The second token is called CYCLE, which within the ICP ecosystem is used as fuel for computation. Additionally, every canister can create its token if it wishes. Canisters can also hold token balances for any token used natively on the Internet Computer (e.g., ICP and CYCLE) and for tokens created by any canister. In addition to that, it is possible to send tokens from one canister to another using function calls. Now we will look at what ICP tokens can be used for. [16]

4.1 Locking

One can lock his ICP tokens inside the NNS and will get so-called Neurons in return. Neuron holders can vote on proposals and will earn voting rewards paid out in ICP. The Neuron must be dissolved to withdraw ICP tokens locked inside a Neuron. Dissolving takes time, but once the Neuron dissolution process has been completed, the locked ICP can be withdrawn. The longer the dissolving time of a Neuron is configured, which can be set by the user, the more voting power one has, and the more voting returns will be paid out in ICP. This is done to encourage the long-term holding of ICP, which generally leads to the best overall decisions (e.g., votes) for any network.

4.2 Transforming

When one wants to host a canister on ICP, he will need the CYCLE token, which can be obtained by transforming ICP tokens into CYCLE tokens. The CYCLE token is like fuel for the system and gets burned when computation power for a canister has been used. Since CYCLE tokens are created from ICP tokens, ICP tokens essentially get burned with every computation that happens on the Internet Computer, making it deflationary. In other words, as long as computations are being done on the Internet Computer, there must be someone on the market buying ICP to do these computations, which leads to ICP having a high monetary velocity. The conversion rate for transforming ICP tokens into CYCLE tokens is computationally set such that one trillion CYCLE tokens are roughly one Swiss Franc at the time of conversion.

[16]https://www.dfinitycommunity.com/beginners-guide-to-understanding-icps-tokenomics/

5 Case Study

5.1 Objective

The objective of this thesis was not only to give the reader a technical overview of ICP from a theoretical perspective but also a practical one. We wanted to analyze if what is being marketed is accurate and, most notably, how performant the ICP ecosystem is for hosting software. For that, we have partnered up with a shareholder services provider company that was interested in building a prototype for one of their services using a decentralized hosting provider. When someone buys a stock in Switzerland (e.g., Nestle, UBS, or Credit Suisse stock), this is recorded in the company's software. In addition to that, this company also organizes the general assemblies for their clients, which is where we come into play. Essentially, they wanted to enable general assemblies to be held online with unlimited computing power and scalability, which ICP can handle. In order to achieve that, we have built a working prototype of how that use case could be realized as a web application running on ICP.

5.2 Visualization

We kick-started this project by building a mockup, which visually represents how the application should look in the end. This process was realized together with the input of the client in order to visualize what the client wants this to look like. To that end, we started with three design directions that we thought could work. The first design direction we coined "futuristic, technology, mysterious, performance", the second design direction was "professional, minimalistic, visionary" and the last design direction we saw as "friendly, progressive, modern".

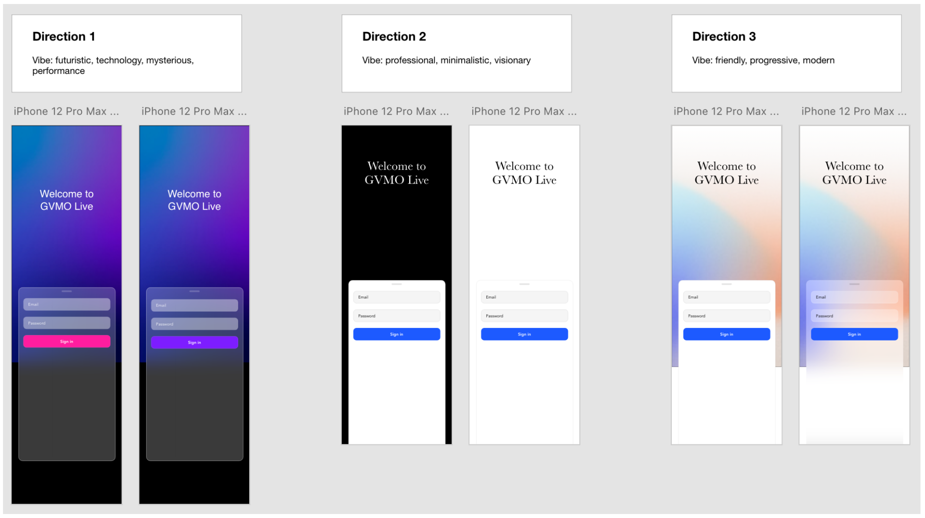

Direction 1
Vibe: futuristic, technology, mysterious, performance
Direction 2
Vibe: professional, minimalistic, visionary
Direction 3
Vibe: friendly, progressive, modern
iPhone 12 Pro Max ...
Welcome to GVMO Live
Email
Password
Sign in

Figure 15: Mockup Exploration Phase

The client ended up choosing the third design direction, which is what we then build out as the full application. After optimizing the mockup together with the client, we ended up with something like this:

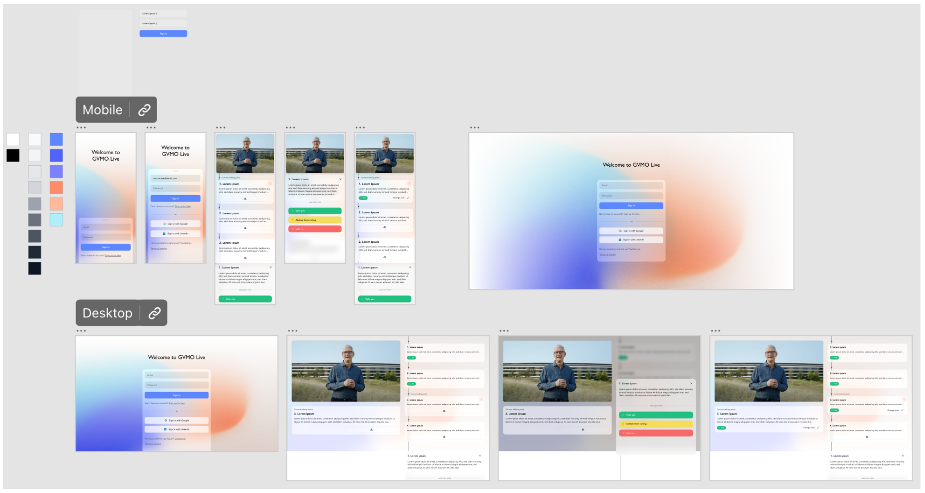


Figure 16: Final Version of Mockup

5.3 Development

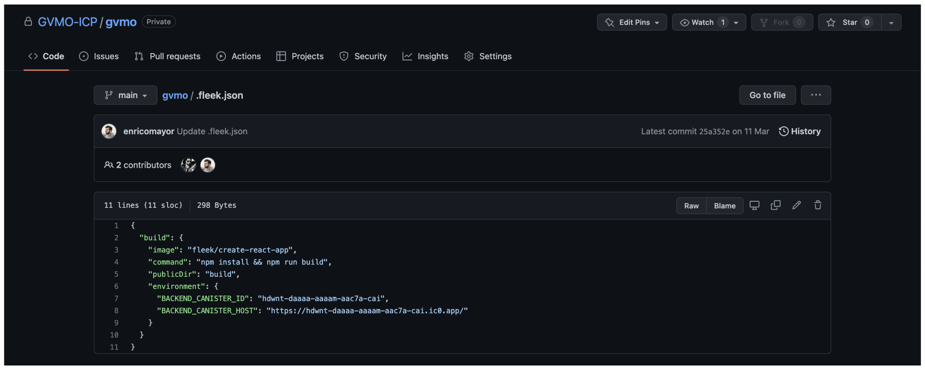
After finishing the mockup, we were ready to start the development process. We have separated our work into front- and back-end. In general, the work needed to build the application code-wise is the same for legacy cloud computing (e.g., AWS or Google Cloud) as ICP. The real difference becomes evident when it is time to deploy the application, which we will focus on in the upcoming sections.

5.3.1 Front-End

The front-end component of the prototype was built using React-JS, which runs on Fleek, which is like Netlify for the open web. Fleek lets us host our static front-ends on the Internet Computer without much work. The steps to get a front-end up and running on Fleek are remarkably simple:

1. Connect the GitHub front-end repository
2. Specify build settings in Fleek
3. Deploy front-end to ICP

That is it! The only thing left to do in Fleek is connecting the front-end with the back-end, which, again, is remarkably simple: Add the canister-id and the canister-host to the ".fleek.json" file and everything else is handled by Fleek. As the name suggests, the canister-id is simply the unique identifier of the canister. The canister-host is just the domain that gets created for accessing the back-end.



```
{
  "build": {
    "image": "fleek/create-react-app",
    "command": "npm install && npm run build",
    "publicDir": "build",
    "environment": {
      "BACKEND_CANISTER_ID": "hdwnt-daaaa-aaaam-aac7a-cai",
      "BACKEND_CANISTER_HOST": "https://hdwnt-daaaa-aaaam-aac7a-cai.ic0.app/"
    }
  }
}
```

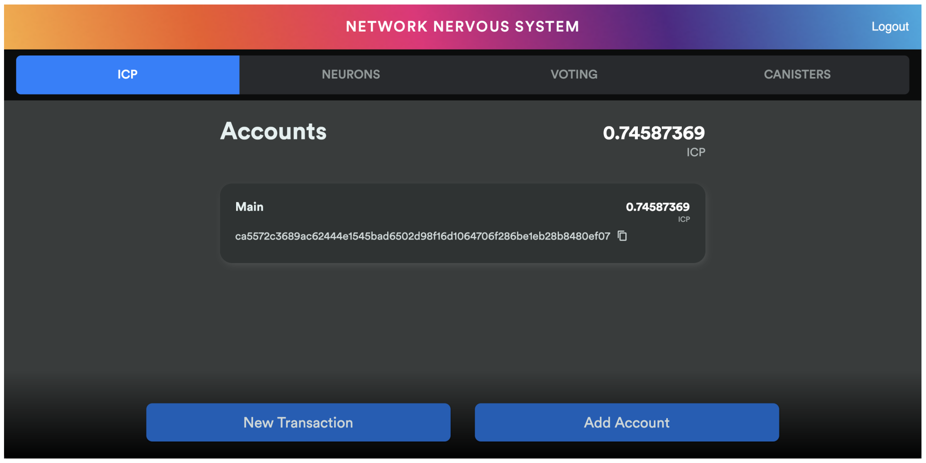
Figure 17: Fleek Setup

5.3.2 Back-End

The back-end component of the prototype was built using Motoko, which runs on a designated canister that we have set up. We had to do it this way because Fleek only allows for front-end canisters (at least for now). Deploying a back-end canister on ICP manually is more tricky than simply connecting the GitHub repository, as seen with Fleek before. It works as follows: [17]

[17] https://medium.com/dfinity/how-to-deploy-your-first-canister-using-the-nns-dapp-c8b75e01a05b

1. Go to NNS.
2. Create a new wallet or sign in.
3. Fund wallet with some ICP coins.

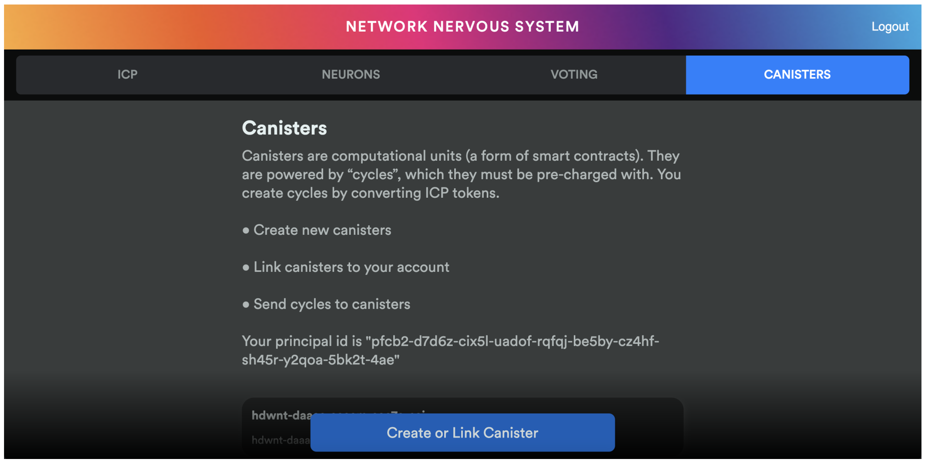


Accounts 0.74587369 ICP

Main 0.74587369 ICP
ca5572c3689ac62444e1545bad6502d98f16d1064706f286be1eb28b8480ef07

Figure 18: Wallet

4. Create a canister using the "Create or Link Canister" button.



Canisters

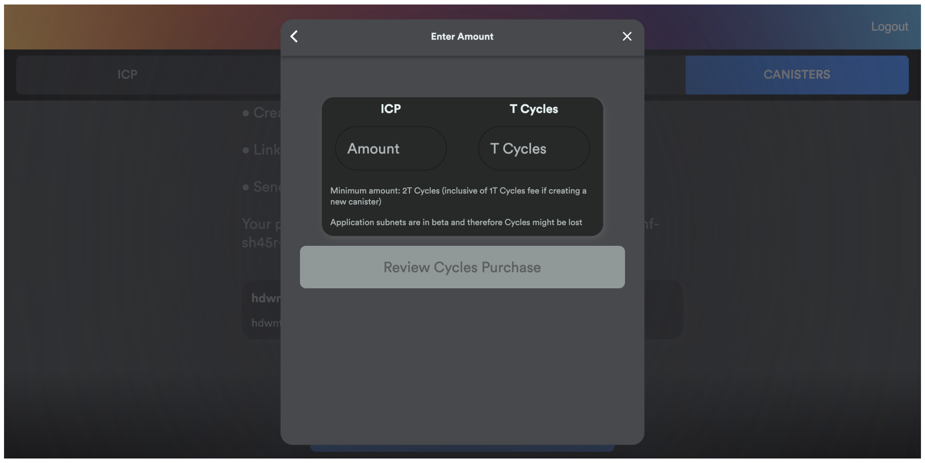
Canisters are computational units (a form of smart contracts). They are powered by “cycles”, which they must be pre-charged with. You create cycles by converting ICP tokens.

- Create new canisters
- Link canisters to your account
- Send cycles to canisters

Your principal id is "pfcb2-d7d6z-cix5l-uadof-rqfqj-be5by-cz4hf-sh45r-y2qoa-5bk2t-4ae"

Figure 19: Canisters

5. Convert ICP tokens to CYCLE tokens in the pop-up that appears.

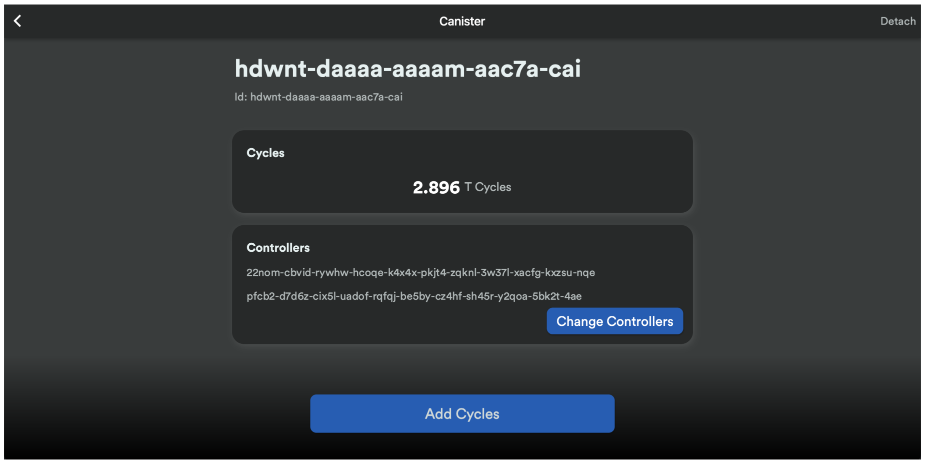


Minimum amount: 2T Cycles (inclusive of 1T Cycles fee if creating a new canister)

Application subnets are in beta and therefore Cycles might be lost

Figure 20: Cycles

6. Get the principal by running the "dfx identity get-principal" command on your local machine.
7. Go back to NNS, click on the "Change Controller" button, and add the principal from the previous step.



hdwnt-daaaa-aaaam-aac7a-cai

Id: hdwnt-daaaa-aaaam-aac7a-cai

Cycles

2.896 T Cycles

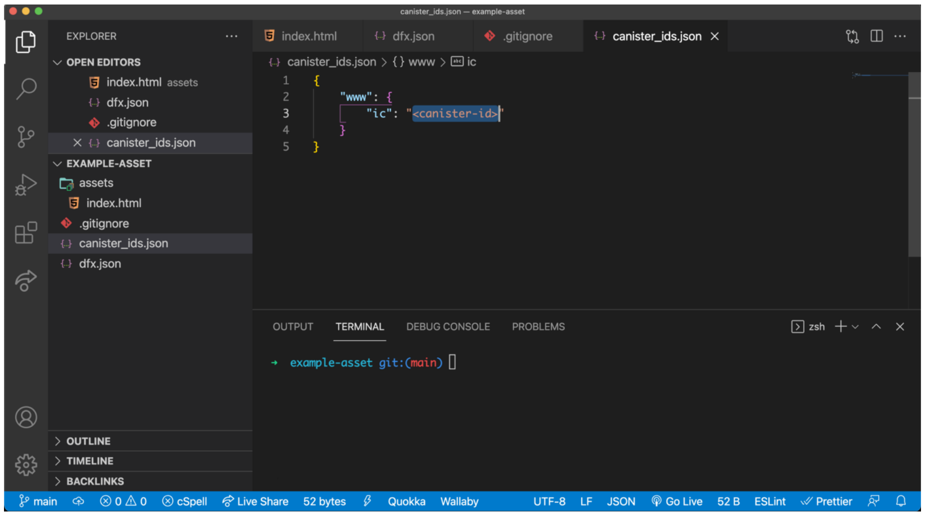
Controllers

22nom-cbvid-rywhw-hcoqe-k4x4x-pkjt4-zqknl-3w37l-xacfg-kxzsu-nqe

pfcb2-d7d6z-cix5l-uadof-rqfqj-be5by-cz4hf-sh45r-y2qoa-5bk2t-4ae

Figure 21: Controller

8. In the project's root directory, open the file "canister-ids.json" and add the back-end canister ID.



```json
{
    "www": {
        "ic": "<canister-id>"
    }
}
```

```
→ example-asset git:(main)
```

Figure 22: Add canister ID

9. Deploy the canister by running the "dfx deploy –network ic –no-wallet" command on your local machine.

That is already it. The canister is now up and running, and as long as we keep a CYCLE balance in the canister, it will run for as long as the Internet Computer is running with no third party being able to shut it down (thanks to ICP's decentralized approach).

5.4 Application Overview

After discussing the deployment process for both front- and back-end, let us finally look at the application itself. This chapter could also be omitted since the application content is not what ICP is about. It is much more about the infrastructure the software runs on (which is why we focused on the deployment process in such detail before). On ICP, developers can build anything from small company websites to colossal enterprise systems. What we have built is something in the middle, as we will discover now.

The purpose for this prototype is for the partner company to be able to take it on as a new project and further develop and fine-tune the current version. We start with the first screen where users can create an account or log in:

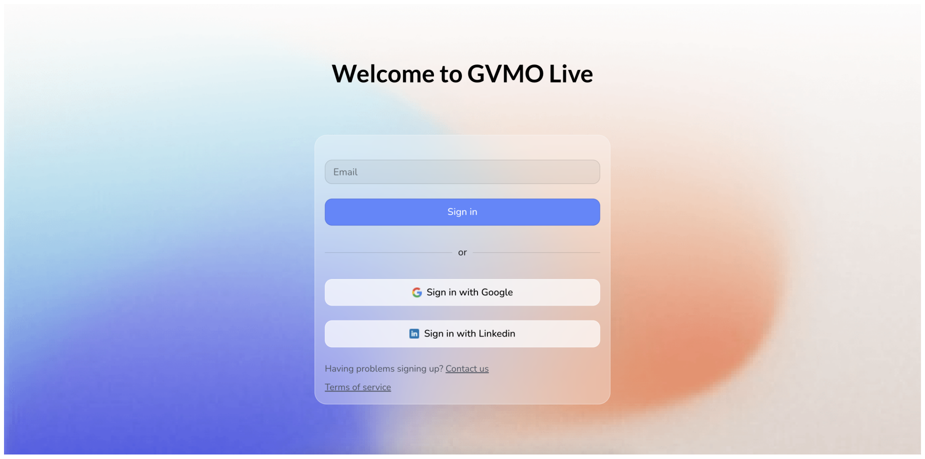
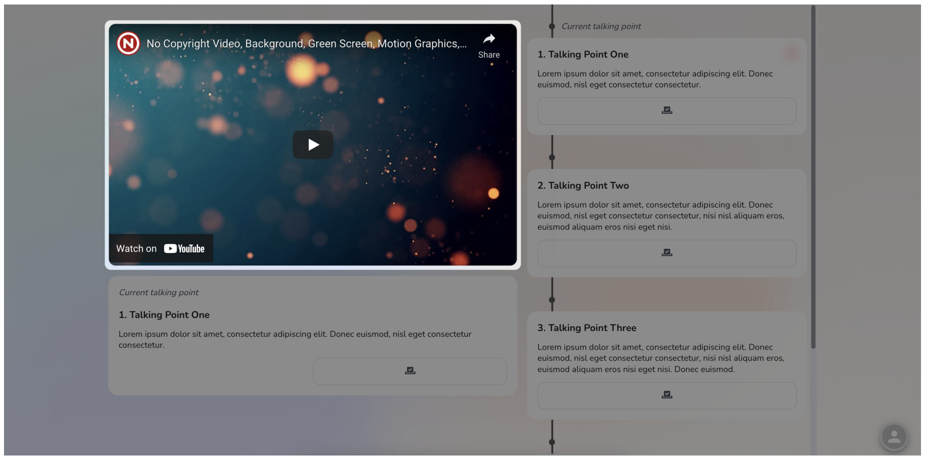


Figure 23: Login

After logging in, the user gets redirected to the application's main page, where he can see and do multiple things. There is a live stream of the general assembly on the left-hand side, which will be held when the user is on that page. In our prototype, we approximated this with a hard-coded YouTube video for now.


No Copyright Video, Background, Green Screen, Motion Graphics,...
Share
Watch on YouTube

Current talking point

1. Talking Point One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur.

Current talking point

1. Talking Point One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur.

2. Talking Point Two

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur, nisi nisl aliquam eros, euismod aliquam eros nisi eget nisi.

3. Talking Point Three

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur, nisi nisl aliquam eros, euismod aliquam eros nisi eget nisi. Donec euismod.

Figure 24: Livestream

On the right-hand side, there are the talking points of the general assembly which are things the shareholders can vote on (e.g., ”Should we appoint that new CEO?” or ”Should we accept the proposal of the board to increase the salary of department X?”).

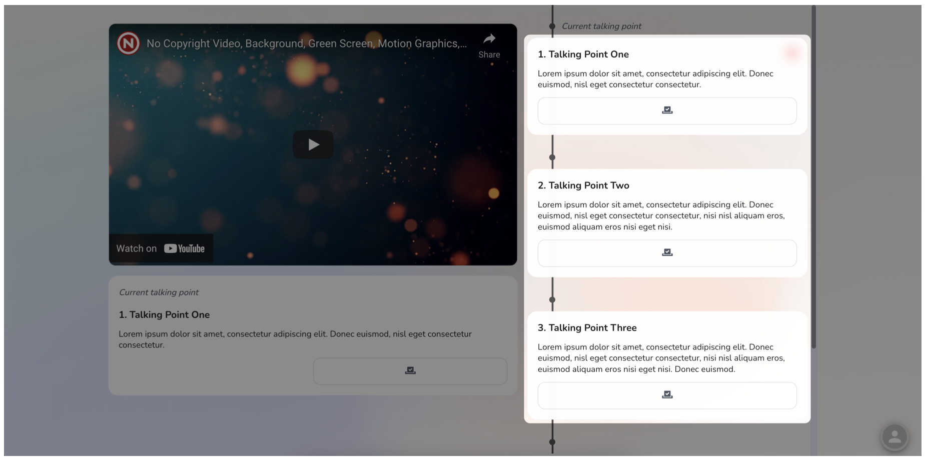


Figure 25: Talking Points

When clicking on any voting button, a pop-up appears where the user can cast his vote (e.g., yes, abstain, or no), triggering a call to the back-end canister that stores the vote.

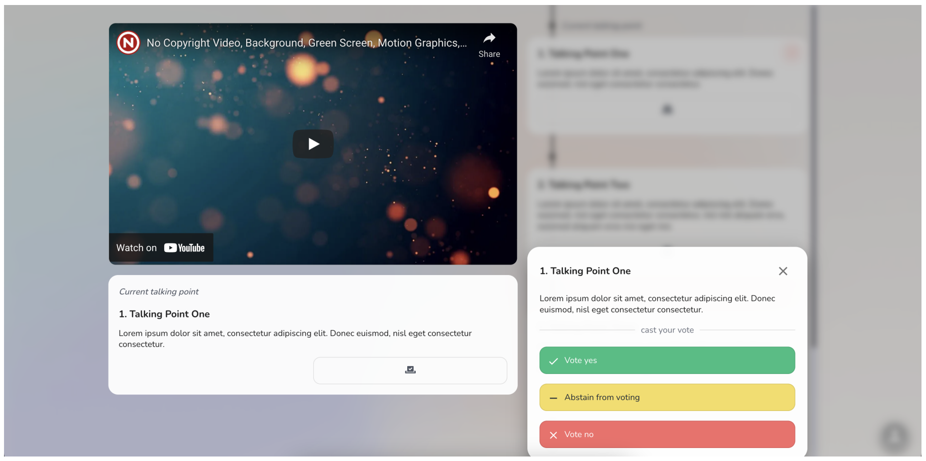
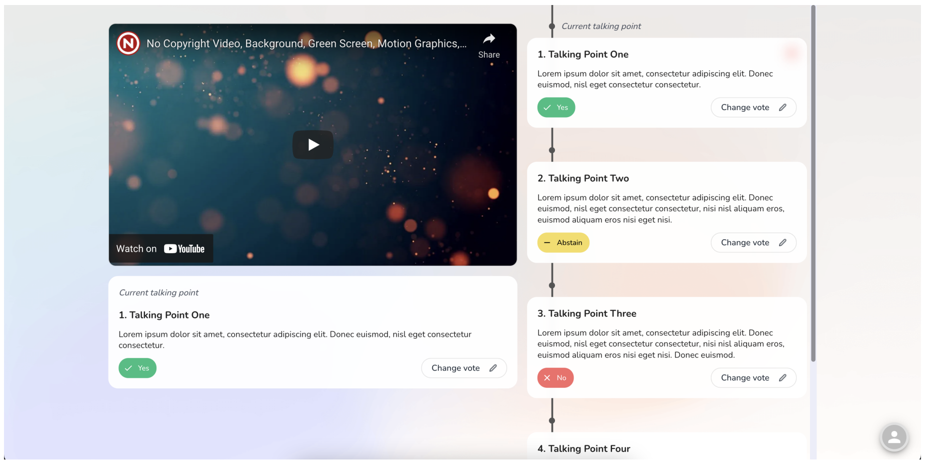


Figure 26: Voting

Finally, after the general assembly is finished, this is what the application will look like to a user. All talking points have been voted on and the final result of the votes is being computed on the partner company's server.


No Copyright Video, Background, Green Screen, Motion Graphics,...
Share
Watch on YouTube

Current talking point

1. Talking Point One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur.

Current talking point

1. Talking Point One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur.

2. Talking Point Two

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur, nisi nisl aliquam eros, euismod aliquam eros nisi eget nisi.

3. Talking Point Three

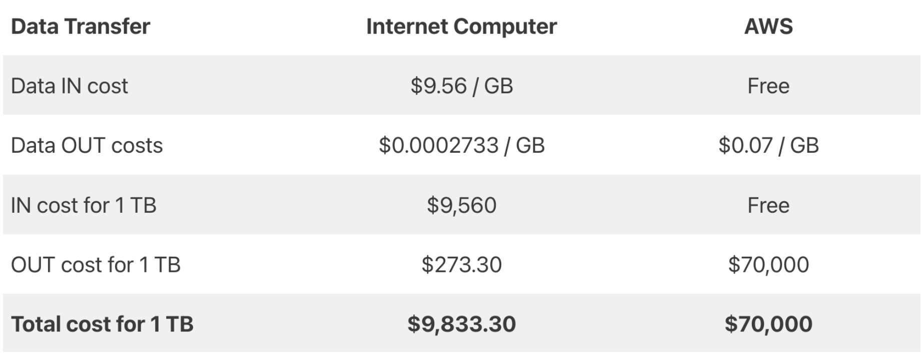
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget consectetur consectetur, nisi nisl aliquam eros, euismod aliquam eros nisi eget nisi. Donec euismod.

4. Talking Point Four

Figure 27: Final State

5.5 Analysis

The two KPIs we used to do our analysis were cost and speed. Multiple resources have done a cost analysis, and the conclusion was always the same or at least similar. Running software on ICP is cheaper than running it on a centralized cloud provider (e.g., AWS or Google Cloud) since AWS is using its monopolistic position in the market to charge whatever they want. To give a sense of how much cheaper it is, let us assume we put 1 TB of data in, and then we read it 1,000 times, approximating the usage of a typical web- or mobile application.

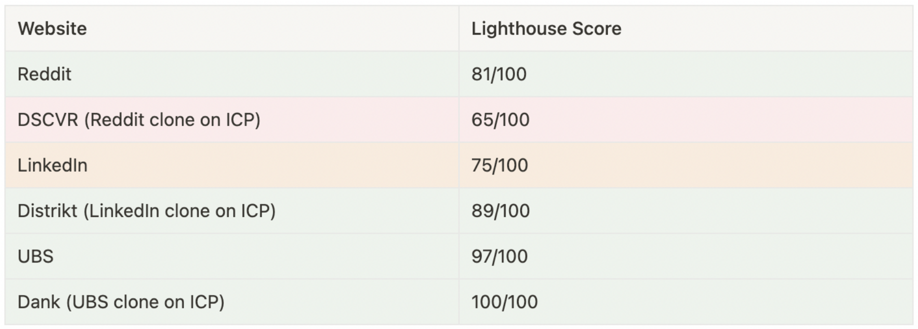


Data Transfer	Internet Computer	AWS
Data IN cost	$9.56 / GB	Free
Data OUT costs	$0.0002733 / GB	$0.07 / GB
IN cost for 1 TB	$9,560	Free
OUT cost for 1 TB	$273.30	$70,000
Total cost for 1 TB	**$9,833.30**	**$70,000**

Figure 28: Cost Comparison

As the above illustration shows, the cost for getting data into the system (e.g., uploading onto the servers) is higher on ICP than on AWS. That is because AWS is subsidizing this transaction from other revenues. Not only that, but this data in transaction is also a bit more complicated on a decentralized setup because on ICP all the nodes in the subnet need to reach consensus, unlike on AWS where everything is managed centrally. The cost for user requests (e.g., getting data out of the system) is much cheaper on ICP compared to AWS since, as already mentioned above, AWS is using its monopolistic position to charge whatever they want. Since the data out transaction (e.g., user request) is being done much more frequent than the data in transaction, ICP happens to be cheaper than AWS in a real-world setting. [18]

The Internet Computer promises web speed with software that runs on it. Therefore, instead of focusing on the numbers under the hood (e.g., how efficiently is the blockchain algorithm), we decided to focus on the numbers that an actual user would see (e.g., how responsive are the websites that run on ICP and how quickly do they load). We compared the page speed (using Lighthouse) of websites on the traditional web with their counter-part in the ICP ecosystem. Here are a couple of examples:



Website	Lighthouse Score
Reddit	81/100
DSCVR (Reddit clone on ICP)	65/100
LinkedIn	75/100
Distrikt (LinkedIn clone on ICP)	89/100
UBS	97/100
Dank (UBS clone on ICP)	100/100

Figure 29: Speed Comparison

The conclusion after our performance testing is that it does not matter where one hosts its software (e.g., ICP or AWS) since the speed is roughly the same. It is more about optimizing the website (e.g., compressing images) that positively plays into the performance metric. Hence, we conclude that ICP can deliver web-speed experiences as promised.

[18]https://icp.guide/costs-on-the-internet-computer/

6 Summary

In this bachelor thesis, supervised by Prof. Dr. Roger Wattenhofer and Robin Fritsch, we analyzed the inner workings of the Internet Computer (ICP) by building a prototype project on top of it. We tried to find out how secure, fast, and scalable it is and if it is, as promised by the DFINITY foundation, "the future of the internet".

The Internet Computer consists of a hierarchy of network building blocks. The data centers at the bottom run many nodes, which are put together into a subnet that hosts canisters. All of that is being controlled by the NNS, the tokenized open governance system of ICP. Unlike other blockchains, ICP does not use proof of work or proof of stake to process transactions. Instead, the subnets communicate with each other using chain-key cryptography. All the nodes within a subnet run the consensus algorithm of ICP to agree on which messages to run in which order, for which ICP uses a blockchain.

After building and deploying a working prototype on ICP in a joint partnership with a shareholder services provider company, we concluded that it is cheaper and every bit as performant to run software on ICP compared to centralized cloud computing services like AWS or Google Cloud. The Internet Computer ecosystem provides an exciting addition to the world of cloud-based hosting thanks to its decentralized nature. As of this moment, there are not many data centers running nodes. However, once the ICP network grows, it could very well be that ICP could become one of the many relevant players in software hosting since the underlying technology is well-made and its approach seems to be scalable. What is in that way is the mass adoption of the development community that has not yet built much software on top of ICP, which will most likely take some more time. However, having interviewed a handful of experienced developers exposed to ICP, the overall feedback was nothing but fantastic. Given enough time, this crypto project with headquarters in Switzerland will hopefully add value to the cloud computing landscape, which is good because it is about time that someone democratizes this stagnant, oligopolistic industry.