Name: Sumanth Reddy Muni, Krishna Karthik Jonnala, Vasudev Awatramani **Date:** May 17 2022
Section: CSI-GA.3033-026

Course Project - Part 2

Blockchain, IOT and ML based P2P Health monitoring, record storage and sharing system

Total in points (100 points total): _____

Professor's Comments:

Affirmation of my Independent Effort: Sumanth Reddy Muni, Krishna Karthik Jonnala, Vasudev Awatramani

(Sign here)

Blockchain, IOT and ML based P2P Health monitoring, record storage and sharing system

Final Code: https://github.com/sumanthreddym/project2final

As discussed with the professor, we have given access to the above repository which has all the code for project 2 to the github username "metacomp".

Motivation

As healthcare costs continue to rise, the industry is seeking methods to be more efficient and secure with patient data in a world where data breaches are all too frequent. To quote an instance[1], over 230K Indonesian COVID-19 Patients' Records Exposed on Darknet. To give an estimated impact of such a breach: patients who have recovered would be surprised at the calls they are getting for plasma donation, deep cleaning & sanitation service providers at homes as well as insurance companies charging a higher premium for health.

Data is scattered across numerous Health providers who employ outdated systems to store information, making data exchange across Health providers extremely insecure and challenging leading to medical errors, which is the third leading cause of mortality in the United States.

Analyzing medical death rate data over an eight-year period, Johns Hopkins patient safety experts have calculated that more than 250,000 deaths per year are due to medical error in the U.S. This makes medical errors i.e. patients died with preventable harm, one of the leading causes of death in the United States. However, policy makers such as CDC makers often ignore such studies due to obsolete data collection methods, resulting in non-optimal allocation of resources and research funds.

[1] https://cisomag.eccouncil.org/indonesian-patients-data-leak/

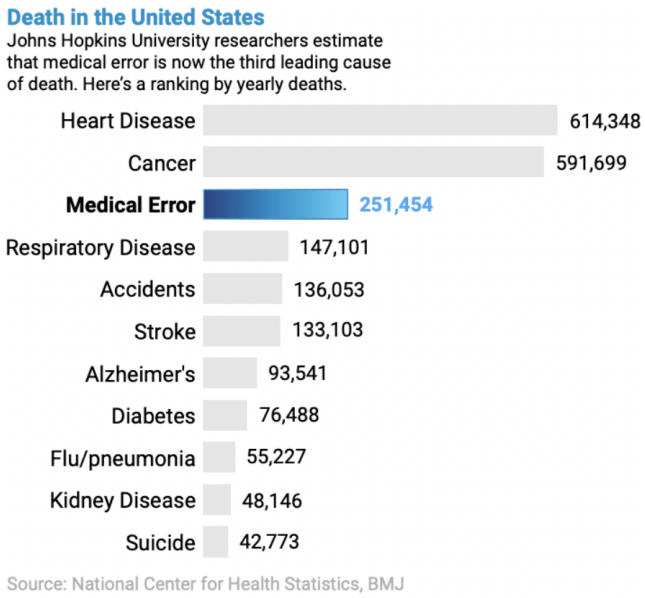


Image: Leading causes of Death for the Year 2013

On the other end of spectrum, there are 350 million people in the globe who are suffering with one of the 7000 incurable illnesses, but organizations conducting clinical trials do not have access to data from wearable devices or previous research - resulting in delays in drug development process. For Example, Wearable technology, such as smartwatches, create continuous streams of patient biomarkers and health vital. However, like many, such data is not open for research.

Proposed Solution

Data Storage not owned by single entity

Storage of electronic health records (EHR) and IoT data that isn't owned by any healthcare facility could help in information sharing among researchers, health care providers and speed up the drug creation process.

Make data sharing easy and accessible to everyone

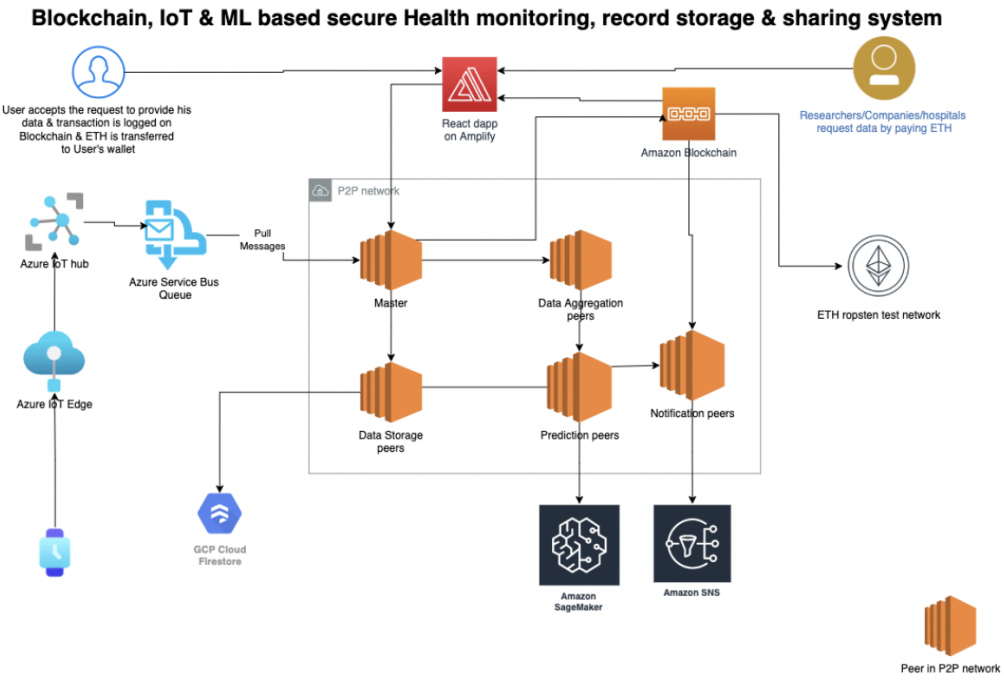
Electronic data of only a small percentage of the population who can afford exorbitant private treatment are stored in several African and Asian nations. In such regions, Healthcare could be made available on mobile devices in a similar way, utilizing a mix of mobile apps, the internet, and a decentralized and secure technology.

Improve security and put patients in control of their information

Secure decentralized technology that can log access requests to user health information while putting information sharing decisions at the hands of the patients is essential.

Framework for scalable P2P networks with intelligent capabilities and IOT connectivity

We propose to integrate nodes of the P2P network with capabilities like Machine Learning and consuming data from edge devices like wearables.



Methodologies

1. P2P Network

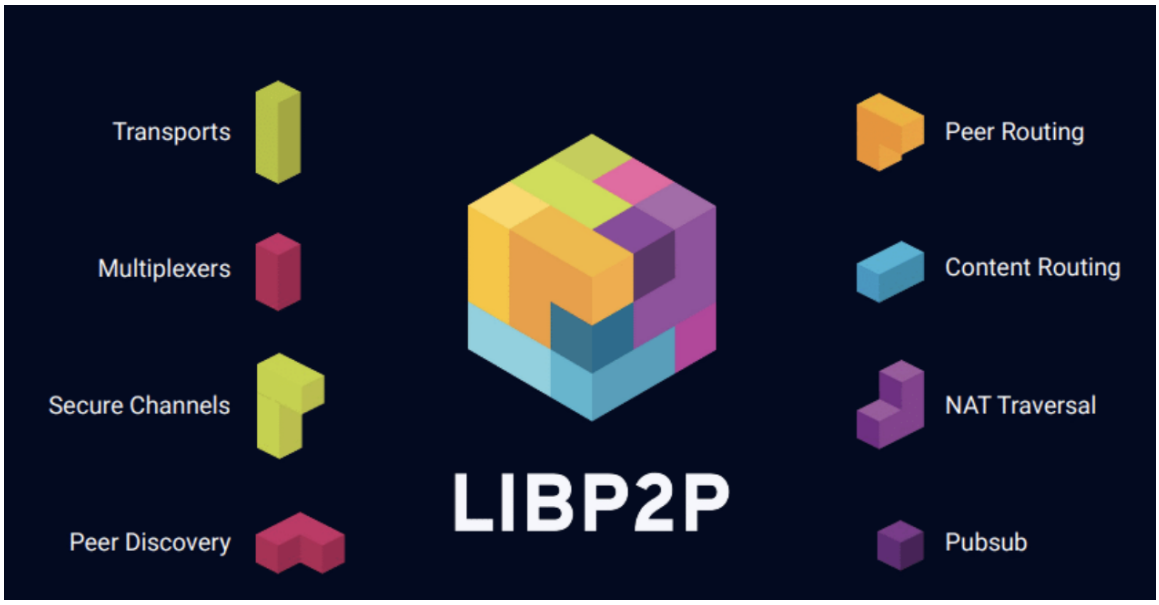
We develop a peer-to-peer network stack for building decentralized communication framework among various nodes having distinct decoupled functions. To accomplish this implementation we employ LIBP2P.

1.1 LIBP2P

LIB2P is a modular peer-to-peer network stack that allows building composable building blocks based on a shared core to assemble future-proof p2p networking layers. The

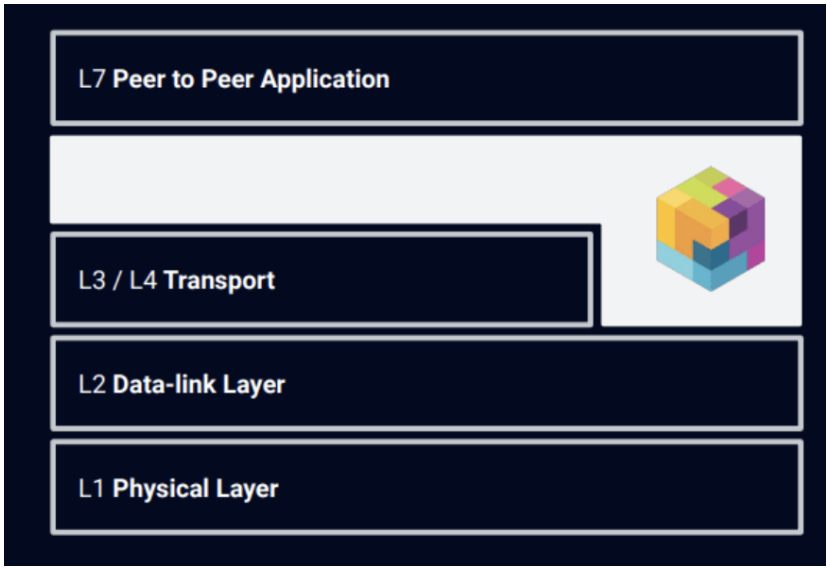
library currently powers IPFS, Ethereum 2, Filecoin and supports implementation in 7 languages. Furthermore, browser, mobile and embedded runtime is also supported.

In a broad view LIB2P is a collection of plug-and-play peer-to-peer protocols (see figure below). For our project we use Transports, Multiplexers, Peer Discovery, PubSub and Peer Routing. Apart from these modules, we also employ Decentralized Process Addressing for process interaction in the network.



From a network perspective, LIBP2P lives between layers 3 and layer 7 of Open Systems Interconnection Model.

In further sections we will outline briefly the P2P protocols we implement in our project.



1.2 P2P Modules

Transport

Transports are the core abstractions of LIBP2P enabling establishment of connections among peers through various protocols like TCP, WebRTC, Bluetooth etc using the same programming interface. Transports are defined in terms of two core operations, listening and dialing. Listening refers to the fact that a node can accept incoming connections from other peers, using whatever facility is provided by the transport implementation. All communication among peers occurs through the Transport module (Only TCP) in our application's P2P network.

Peer Identity

Peer Identity is a unique reference to a specific peer within the overall peer-to-peer network used to uniquely identify peers. As well as serving as a unique identifier for each peer, a PeerId is a verifiable link between a peer and its public cryptographic key. Each LIBP2P peer controls a private key, which it keeps secret from all other peers. Every private key has a corresponding public key, which is shared with other peers. Together, the public and private key allow peers to establish secure communication channels with each other. LIBP2P provides support for 4 key types: RSA, Ed25519, SEC256k1, ECDSA along with Multihash representation of the keys (Identify multihash in case public key is less than 48 bytes and SHA-256 multihash if public key is more than 48 bytes).

Multi Address

Flexible networks such as one we implement in our project need flexible addressing systems. A *multiaddress* is a convention for encoding multiple layers of addressing information into a single path structure. It defines human-readable and machine-optimized encodings of common transport and overlay protocols and allows many layers of addressing to be combined and used together. For example: /ip4/127.0.0.1/udp/1234 encodes two protocols along with their essential addressing information. The /ip4/127.0.0.1 informs us that we want the 127.0.0.1 loopback address of the IPv4 protocol, and /udp/1234 tells us we want to send UDP packets to port 1234.

Peer Routing and Discovery

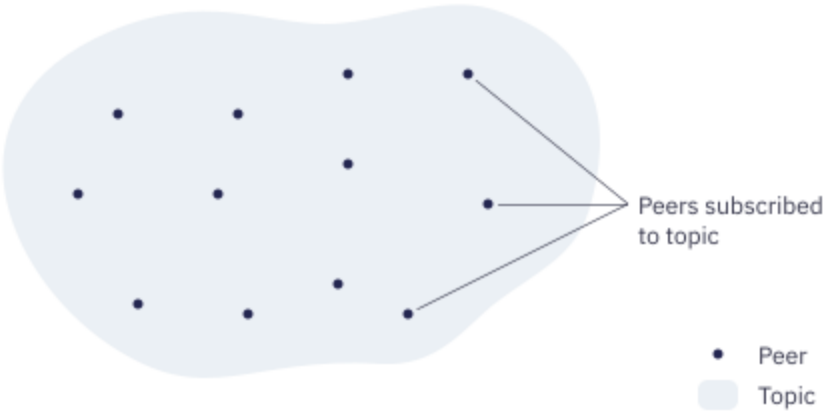
A peer in a P2P network needs to be able to communicate with the other peers in the network. In order to do so, it must first be able to find the other peers in the network. To find the other peers in the network, each peer has a routing table. The routing table contains references to a number of peers in the network. In a P2P network with millions of peers, each peer cannot hold a complete table of all other peers in the network. Such a table would take up a lot of resources on each peer, and would also be almost impossible to keep up to date, as peers join and leave the network. Instead a peer routing table contains references to a subset of peers in the network. Exactly how the routing table is organized depends on the P2P network algorithm. LIBP2P uses Kademlia that implements XOR as a distance function.

Multiplexing

Multiplexing allows multiple independent logical streams to all share a common underlying transport medium. The project has many open independent streams of communication between peers and several concurrent streams open at the same time with a given remote peer. Stream multiplexing allows us to amortize the overhead of establishing new transport connections across the lifetime of our interaction with a peer.

Pub/Sub Interface

Publish/Subscribe is a system where peers congregate around topics they are interested in. Peers interested in a topic are said to be subscribed to that topic



Peers can send messages to topics. Each message gets delivered to all peers subscribed to the topic:

We use GossipSub[2] to find the other peers who are capable of performing a specific function. In short, we use it to find other available peers who can provide a service we need.

Content Routing

Kademlia[3] is a distributed hash table for decentralized peer-to-peer computer networks designed by Petar Maymounkov and David Mazières in 2002 (at New York University). It specifies the structure of the network and the exchange of information through node lookups. Kademlia nodes communicate among themselves using UDP. A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or node ID. The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values.

[2] https://arxiv.org/abs/2007.02754
[3] https://ipfs.io/ipfs/QmaVrnwZrnoG4YramcN75mbE5AUfCymiEErrHGXoQR968V

Basis of the above Distributed Hash Table, we use a lookup algorithm[4] for configuring content routes between various nodes of our network viz. Master to DataAggregation and DataStorage Node.

1.3 Peer Types

Master

- Responsible for bootstrapping the network.
- If a node needs to join the network, it needs to connect to the master.
- *Single Master Node = Single point of failure*. So, we **have other peers periodically checking if the master is alive**. If the master dies, they instantiate a new master so that newer nodes can join the network.
- Master periodically pulls IOT data from the Azure Service Bus. Then, it asks the Data Storage peers if anyone is available for writes. If a Data Storage node is available for write, master routes the data to that peer. Peer whose reply is received first is chosen by the master.
- Each message pulled from Queue has following format:

  {

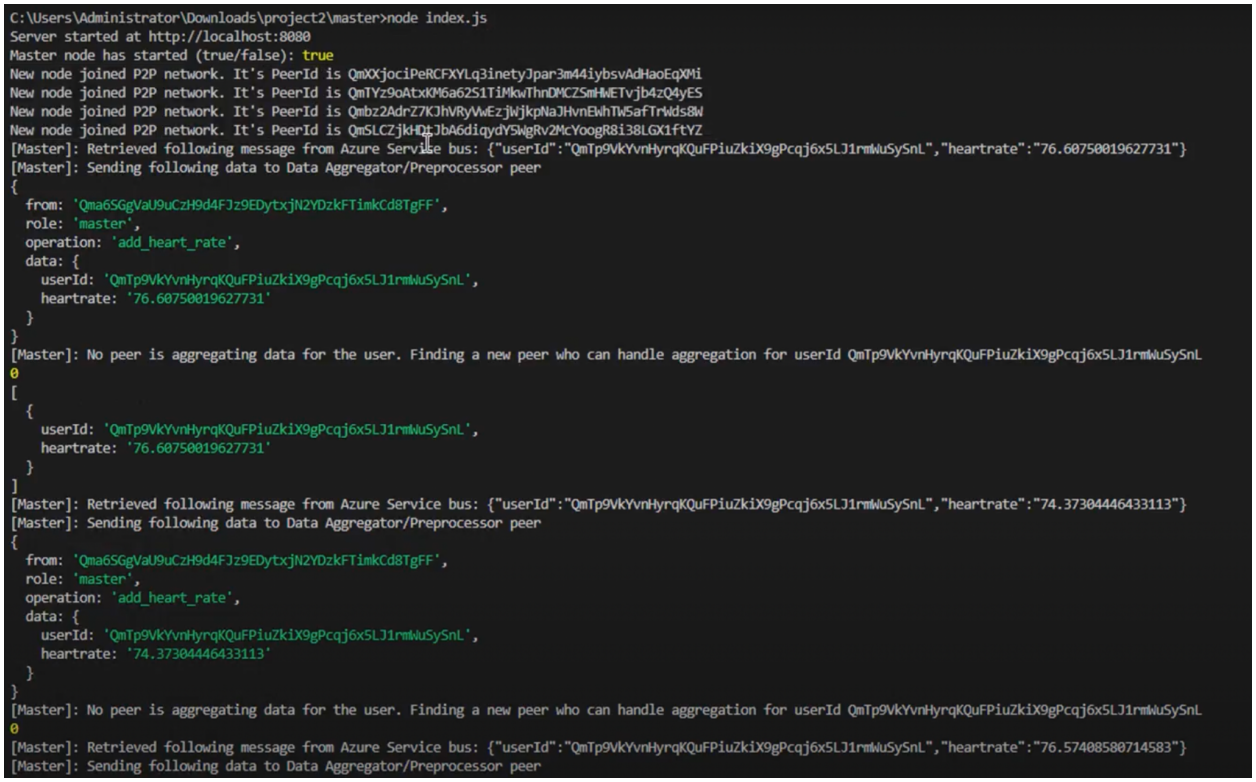
  "userId":"QmTp9VkYvnHyrqKQuFPiuZkiX9gPcqj6x5LJ1rmWuSySnL",

  "heartrate":"75.52868227718442"

  }

  The userId is generated using the Content-Identifier (**peer-id** library) module in order to uniquely identify the flow of information among distributed peers in the network. The relay of data between Master, Data Storage and Data Aggregation is implemented through the Pub/Sub Interface (**GossipSub**).
- Then, Master node enquires Data Aggregation Peers which are currently aggregating the data for this particular user's content Id (userId).
- If one of the Data Aggregation Peers responds stating that they are currently aggregating data of a particular user, then data is forwarded to it.
- Else, master instantiates one of the Data Aggregation Peers to handle the new contentId.
- **KademliaDHT** is employed to carry out the above operation. Master adds each of the new Data Aggregation peers to the DHT. (See Content Routing)

[4] https://blog.ipfs.io/2020-07-20-dht-deep-dive/

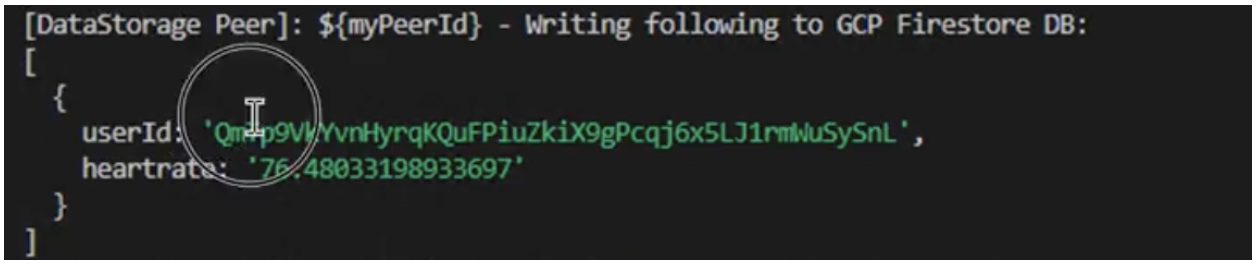


Then, Master node also asks which of the Data Aggregation Peers are currently aggregating the data for this particular user's content Id (userId).
If any of the Data Aggregation Peers respond saying that they are currently aggregating data of a particular user, then data is forwarded to it.
Else, master chooses one of the Data Aggregation Peers to handle the new contentId.
We use **KademliaDHT** to process this. Master adds each of the new Data Aggregation peers to the DHT.

Data Storage Peer

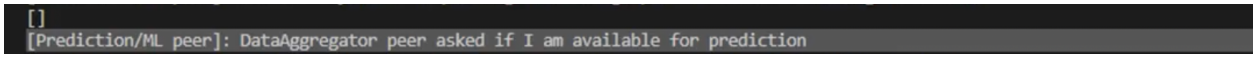
- DataStorage peers have the logic to perform CRUD operations on our NoSQL database on GCP called **Firestore**: a NoSQL document database that allows easy storage, synchronization, and querying of data with automatic scaling.
- There can be multiple such peers in the network.

- Each peer goes into a *busy* state once they start performing an action on the Database.



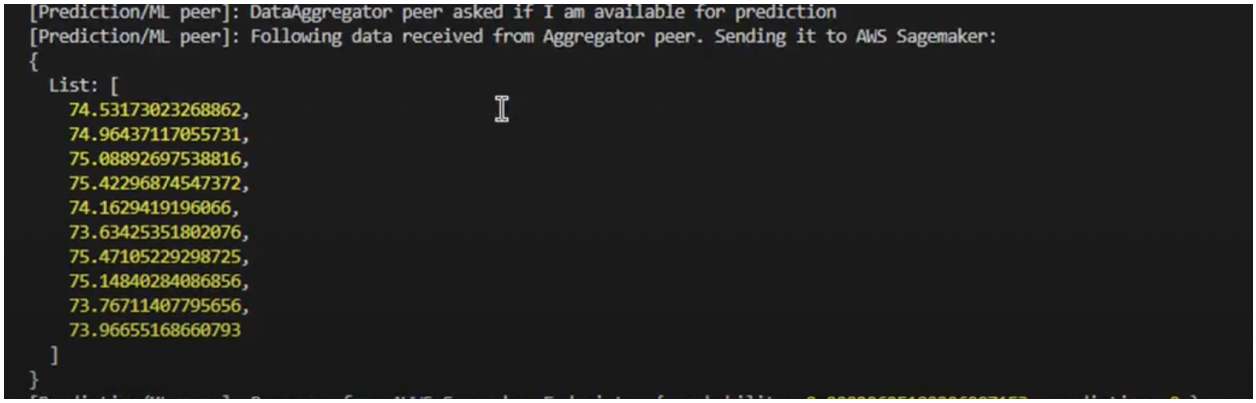
Data Aggregation Peer

- DataAggregation a.k.a Pre-processing peers aggregate the data for a particular user's content Id(userId).
- A user data is aggregated by a single node, however the same node can aggregate data of multiple users.
- To avoid data loss while aggregation is still in process, a distributed hash table which replicates the data in the P2P network is used to avoid failure of a single node.
- The reason for this aggregation is because our ML model performs inference over a window of 10 seconds of data of each user to make a prediction. Therefore, the DataAggregation peer keeps aggregating data of each user for 10 seconds and then sends the data to any of the available Prediction(ML) peers.
- This could be easily extended for any application. For instance, most ML/DL models expect data to be in a certain format. Eg- for transforming an image to current dimensions before handing it to an ML model.

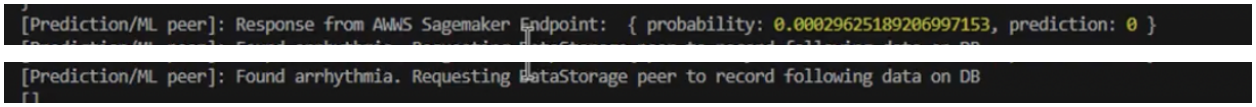


Prediction Peer

- Responsible for transmitting the current user's IoT data to the **AWS Sagemaker** ML endpoint. **Amazon SageMaker** is a fully-managed service that enables users to quickly and easily build, train, and deploy machine learning models at scale.
- A window of 10 seconds of data for each user is passed to the inference endpoint.

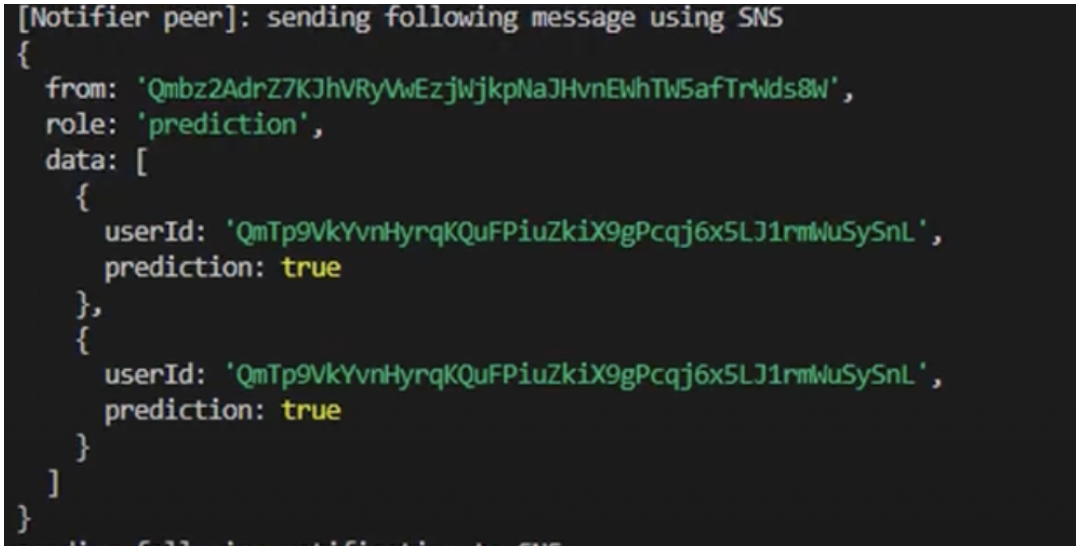


- If there are any predictions stating the user having heart arrhythmia, it communicates with one of the Notification peers to send an email to the user and their doctor. It also sends data to one of the DataStorage nodes so that it can be stored in the Database.



Notification Peer

- Responsible for sending notifications to the user and their doctor via Email using **Amazon SES**.
- Prediction peer and Blockchain dapp use this peer to send notifications.



IOT Data Streaming

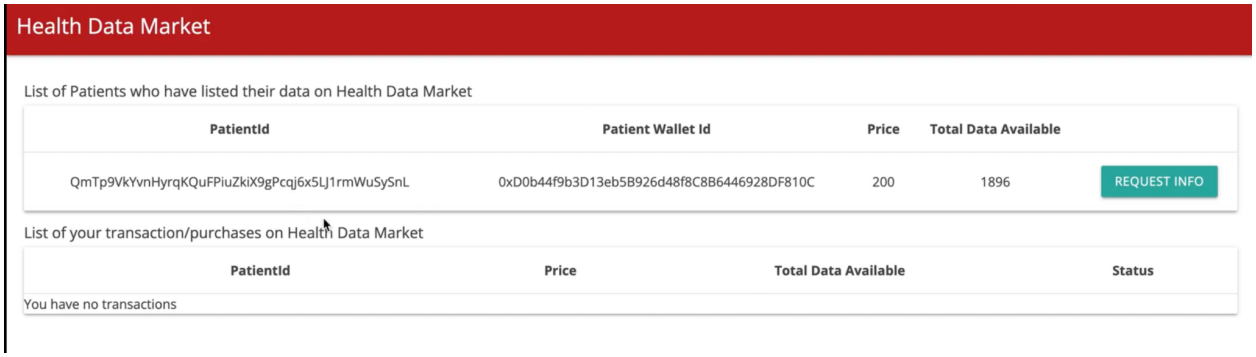
- We simulate multiple users with smart IOT devices that push wearable data, in this case heart beat measurements (QPRS wave measurement) to Azure IOT Edge.
- Azure IOT Edge is presently responsible for aggregating data for a specific user and transmitting it further to Azure IOT hub. **Azure IoT Hub** is a managed service hosted in the cloud that acts as a central message hub for communication between an IoT application and its attached devices
- Azure IOT hub forms emits the data to our network in time series fashion, using Azure Service Bus Queue. **Azure Service Bus** provides a set of cloud-based, message-oriented middleware technologies including reliable message queuing and durable publish/subscribe messaging. These brokered messaging capabilities can be thought of as decoupled messaging features that support publish-subscribe, temporal decoupling, and load-balancing scenarios using the Service Bus messaging workload.

Blockchain Marketplace

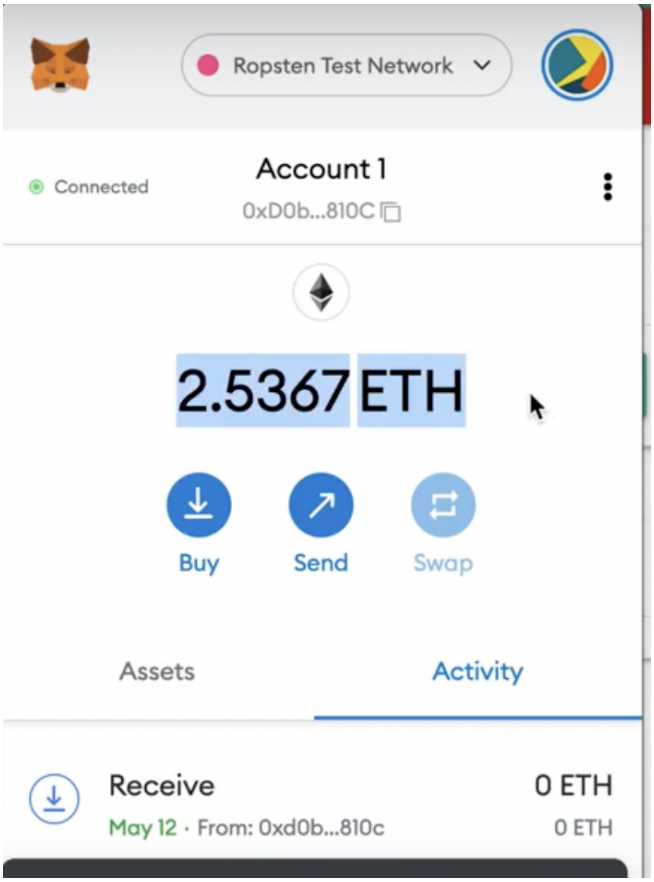
- For Blockchain, we use Managed Ethereum Blockchain node on AWS called Amazon Blockchain Service.
- All requests to the Ropsten Ethereum Blockchain public test network go through the Amazon Blockchain node. i.e., Amazon Blockchain node is used as the default provider instead of infura.io. Amazon Blockchain node in turn routes this request to Ethereum Ropsten Test network. This means that we won't be affected when there are high loads on infura.io provider endpoints.

- In our project, we use blockchain to store transactions made for data purchases and data access to patient records made by healthcare facilities or researchers or companies. Data itself is stored in GCP Firestore.
- This helps keep a ledger of all data access/purchase transactions.
- Moreover, we put the control of data sharing at the hands of the patients.

Following screenshots describe the flow of the project for the Blockchain based Marketplace:

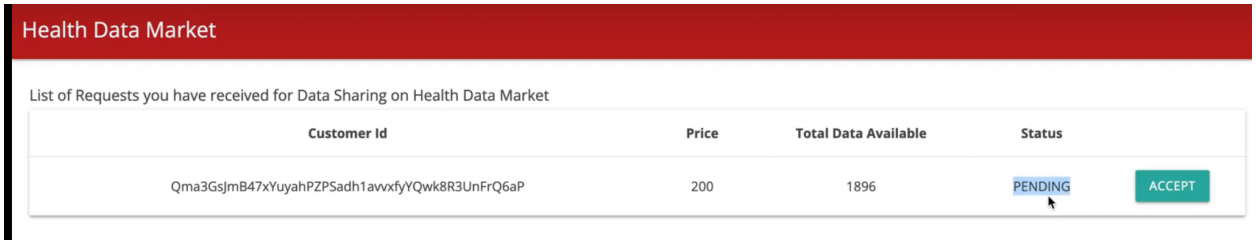


Market Screen for Organization to buy patient data.



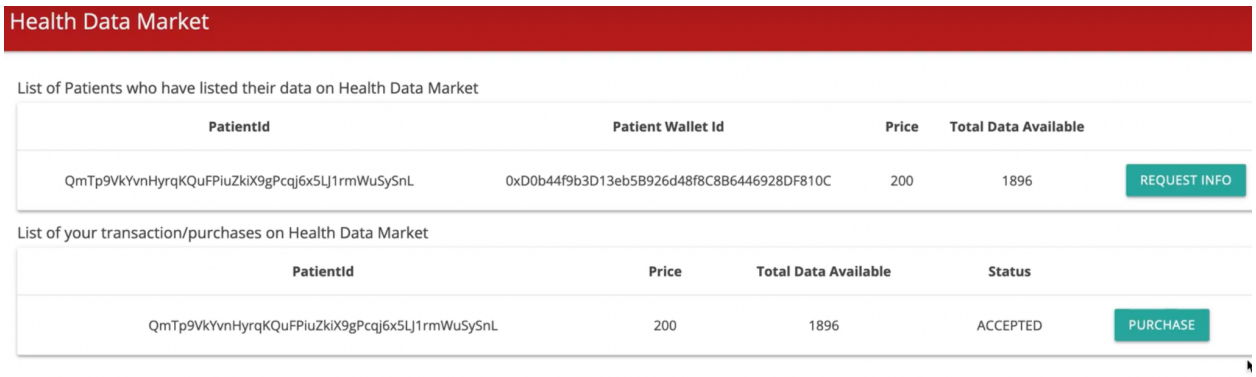
Metmask outline account of Buyer

Once a buyer requests permission from a patient to purchase their data. Correspondingly Patient screen is also updated

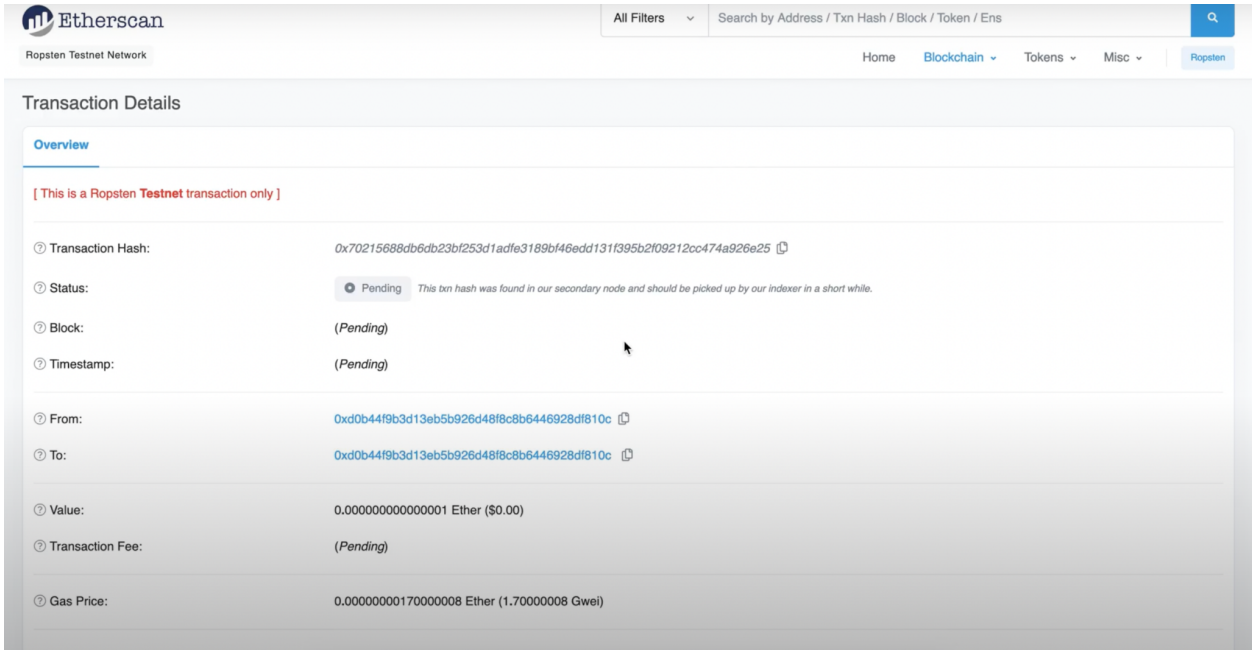


Patient Screen having describing a list of potential buyer of data

Patient Screen after they accept Buyer request



Updated Buyer screen after Patient accepts to sell their data.



Once Buyer hits purchase, an Ethereum transaction is initiated which can be verified through Etherscan

Issues Addressed

- LIBP2P does not offer detailed documentation for NodeJS implementation (preferred was Python which was substantially underdeveloped to alternatives like Rust). However, the code implementation is relatively simple and easy to pick up.
- Some of the libraries are in beta phase due to which not all features are available in all languages.
- AWS Sagemaker deployment is costly as both training and inference requires significant computational resources. Alternatively, AWS Lambda is a good option for endpoint inference.

Room for Improvement

- Our architecture overcrowds functionality of Master leading it to be a single point of failure. Alternatively, Master should be only restricted to orchestrate flow of information in the application and functions should be pushed to peers.
- Azure IOT Edge can host a lightweight ML model itself to help to provide faster inference with more privacy protection.

Future Scope

- We plan to make the project as a generalisable template for allowing patients and researchers to share data in a decentralized manner, allowing training of ML models in a more secure and privacy-oriented environment.
- The application can be extended to open research tasks like Kaggle Competitions where the burden of researching model architecture is shifted to participants.