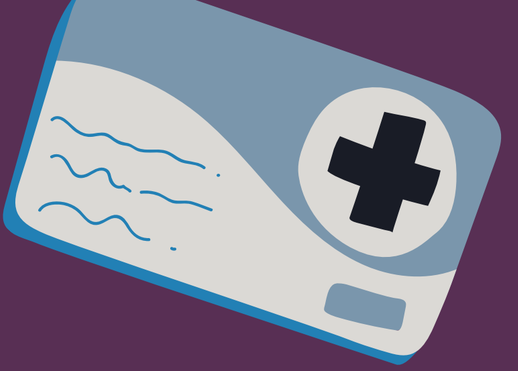
Blockchain, IOT and ML based P2P Health monitoring, record storage and sharing system

Cloud Computing

Sumanth Muni, Krishna Karthik Jonnala, Vasudev Awataramani

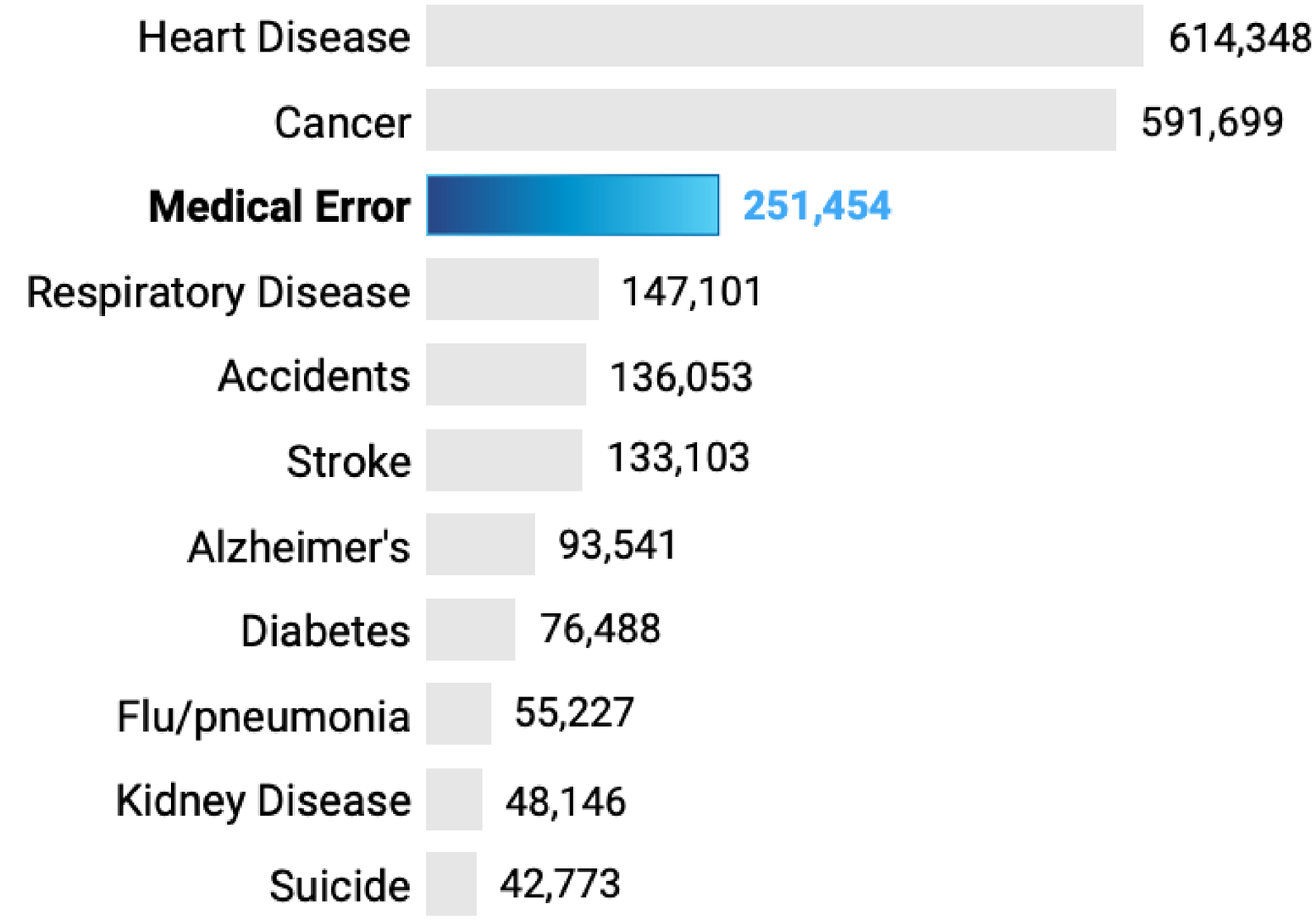
Healthcare in the 21st Century has serious data handling problems



- As healthcare costs continue to rise, the industry is seeking for methods to be more efficient and secure with patient data in a world where data breaches are all too frequent.
- Data is scattered across numerous Health providers who employ outdated systems to store information, making data exchange across Health providers extremely challenging leading to medical errors, which is the third leading cause of mortality in the United States.
- There are 350 million people in the globe who are suffering with one of the 7000 incurable illnesses, but companies conducting clinical trials don't have access to data from wearable devices or previous research - delays drug development process.
- Wearable technology, such as smart watches, create continuous streams of patient health data - Not available for research

Death in the United States

Johns Hopkins University researchers estimate that medical error is now the third leading cause of death. Here's a ranking by yearly deaths.



Source: National Center for Health Statistics, BMJ

How can we address the issue?

Data Storage not owned by single entity

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

Make data sharing easy and accessible to everyone

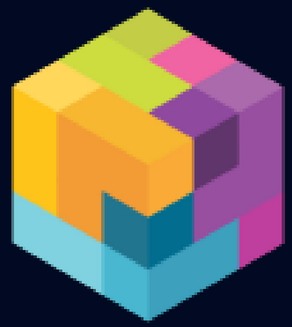
Electronic data of only a small percentage of the population who can afford exorbitant private treatment are stored in several African and Asian nations. In such regions, Healthcare could be made available on mobile devices in a similar way, utilizing a mix of mobile apps, the internet, and a decentralized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

Imrpove security and put patients in control of their informaiton

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

A modular peer-to-peer networking stack

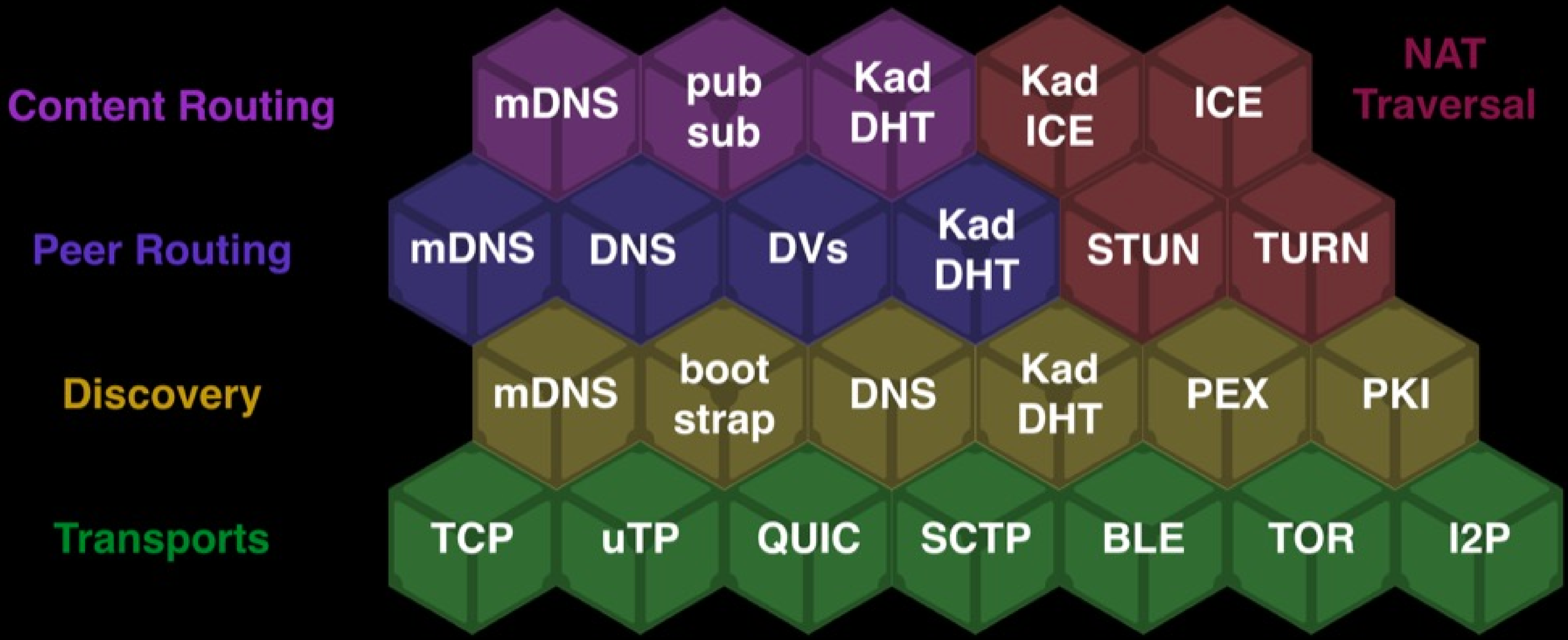
- Composable building blocks based on a shared core to assemble future-proof p2p networking layers.
- All you need to build peer-to-peer applications.
- Runs on many runtimes: browser, mobile, embedded.
- Implemented in 7+ languages
- Powers the IPFS, Ethereum 2, Filecoin and Polkadot network



LIBP2P

P2P network - built using LibP2P

libp2p a collection of peer-to-peer protocols



Where?

Where does libp2p live?

L7 **Peer to Peer Application**

L3 / L4 **Transport**

L2 **Data-link Layer**

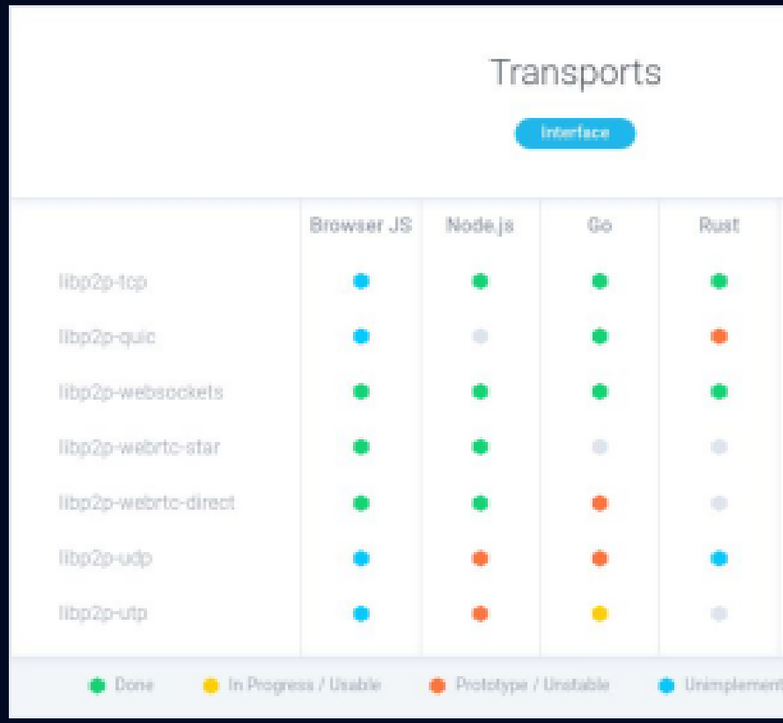
L1 **Physical Layer**


Transports
Multiplexers
Secure Channels
Peer Discovery
LIBP2P
Peer Routing
Content Routing
NAT Traversal
Pubsub

Transports

- Transports are core abstractions of libp2p
  - Enable connection establishment
  - Dialing and listening

- Current transports:
  - TCP, QUIC, WebSockets, WebRTC, Bluetooth, ...



Decentralized Process Addressing

Peer Identity

- Uniquely identifies peers
- IDs derived from their public key
    - Four key types: RSA, Ed25519, SEC256k1, ECDSA
    - Multihash representation
        - Identity multihash (i.e. no hash) if public key < 48 bytes
        - SHA-256 of multihash if > 48 bytes

Multiaddress

`/ip4/1.2.3.4/tcp/5001/p2p/<peer_id>`

- Network processes are named as filepaths.
- Specify available processes in a peer in a single address
- Used for dialing peer

Why?

Decentralized Process Addressing

- Ability to locate, connect, authenticate, negotiate and interact efficiently with any process in the world
  - Independent of the runtime
  - In a seamless manner (NAT-traversal, heterogeneous networks, relays).
  - Fully operational even in a case of mobility, relocation, roam, etc.
  - Every peer is uniquely identified.
- Network- and topology-independent alternative to endpoint addressing (e.g. IP networks)

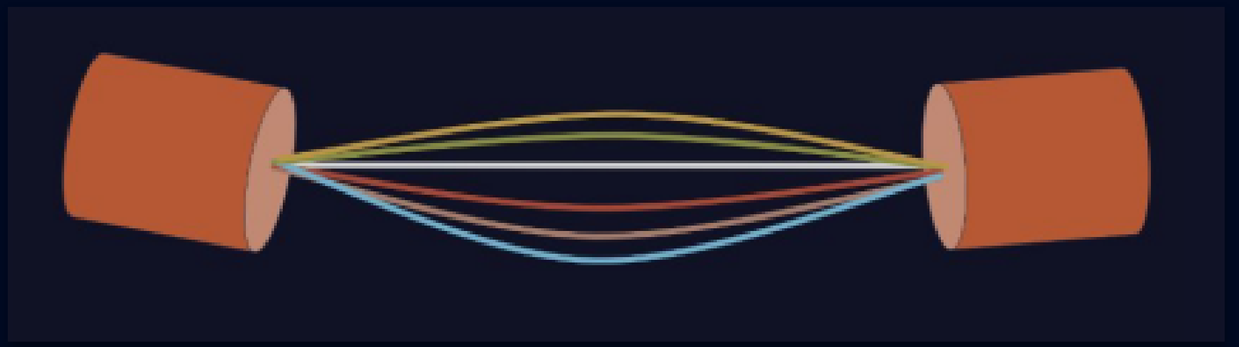
Peer Discovery / Routing

- Announce services to other peers
- Discover peers supporting certain services
- Find specific peers by peer ID

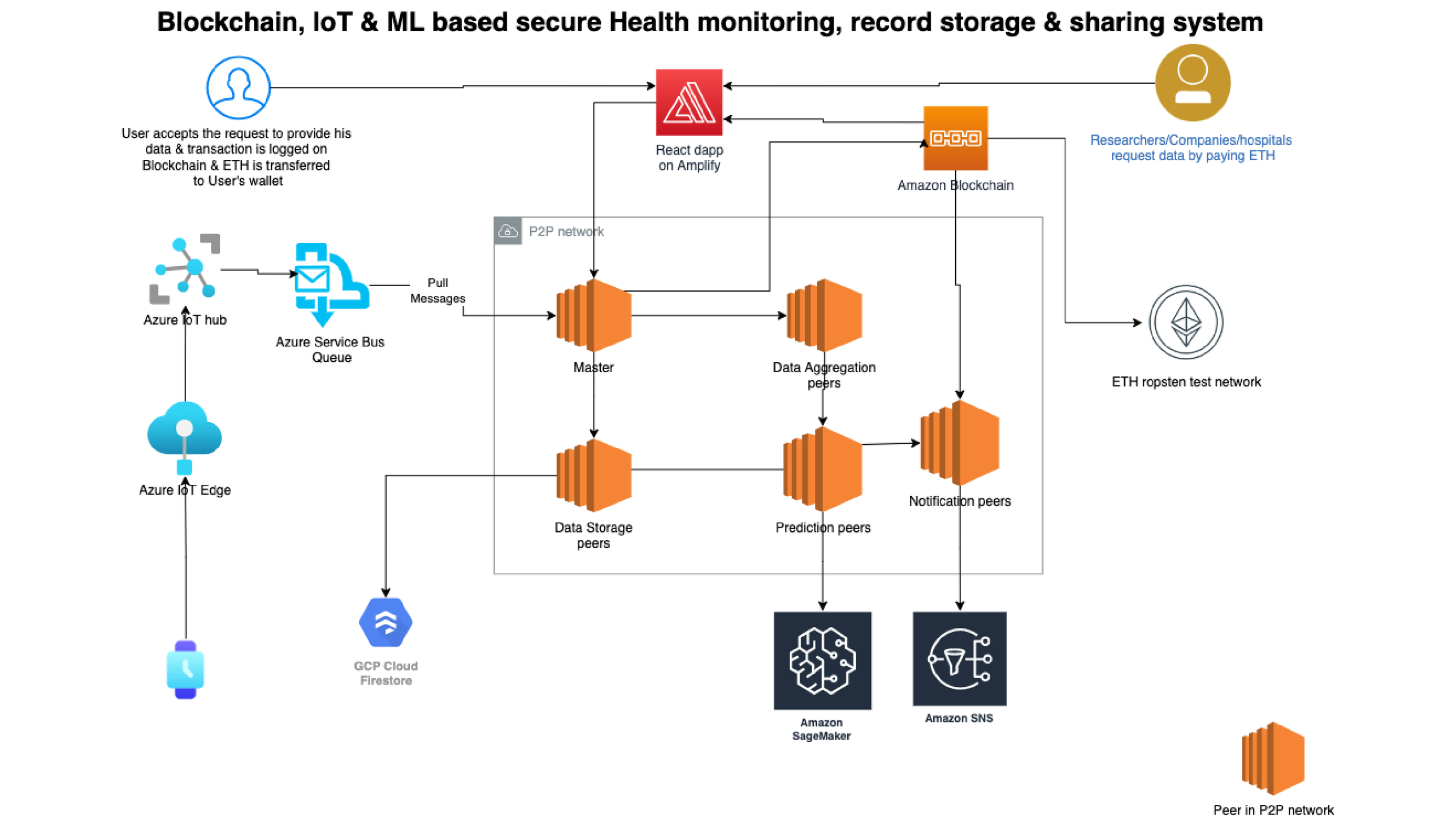
- Implementations
  - MDNS
  - Kademlia DHT

Multiplexing

- Establishing a P2P connection may not be cheap or easy (e.g. hole punching, negotiation, handshake, etc.)
- Re-use established connections for several protocols.
  - Applications can leverage already established connections.
- Streams are uniquely identified based on its multiplexer specification.
- Several implementations of multiplexers available:
  - Language specific libraries for stream multiplex (Yamux, Mplex)
  - Transport protocol native multiplexing capabilities (QUIC)



Blockchain, IoT & ML based secure Health monitoring, record storage & sharing system



5 Peer types

- Master
- Data Storage Peer
- Data Aggregation Peer
- Prediction Peer
- Notification Peer

Master

- Responsible for bootstrapping 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 dead, they instantiate a new master so that newer nodes can join the network.
- Currently, Master also houses the API.

TAKEAWAY: Don't overcrowd the master with features. Let it perform only bootstrapping operation. Move other functionality to other peer types.

Master

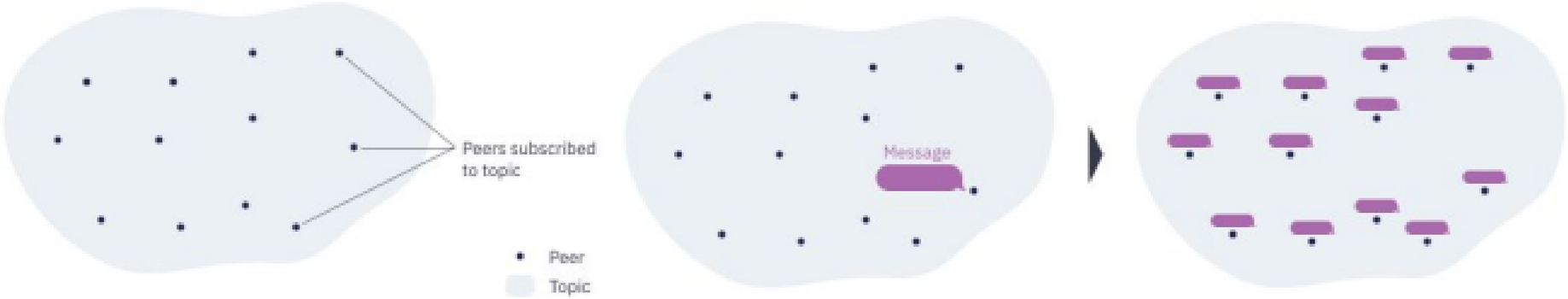
- Master periodically pulls IOT data from the Azure Service Bus. Then, it asks the **Data Storage peers** if anyone is available for writes. If a Data Storage node is available for write, master routes the data to that peer. Peer whose reply is received first is chosen by the master.
- Each message pulled from Queue has following format:

{"userId":"QmTp9VkYvnHyrqKQuFPiuZkiX9gPcqj6x5LJ1rmWuSySnL","heartrate":"75.52868227718442"}

- userId shown above(Content Identifier generated using peer-id library) is used to uniquely identify the flow(to locate) of information among distributed peers in the network.
- Gossipsub is used to build this feature.

PubSub Interface

- Decentralized P2P PubSub.
  - Brokerless, self-regulating, no global knowledge
  - Constructed as overlay networks collaborating for message deliverability
- Several available protocols
  - GossipSub
  - FloodSub
- Use cases: IPNS, content-addressing, blockchain consensus, message dissemination, etc.

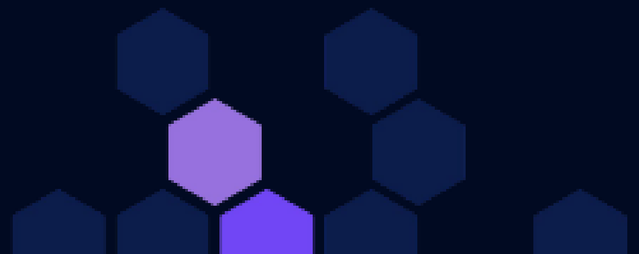


Master

- 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.

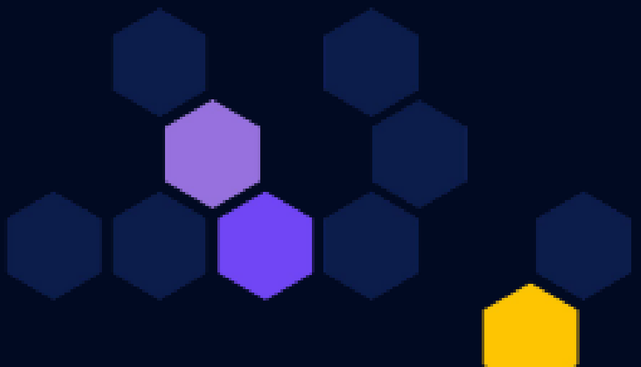
Content Routing

- Find and advertise Content-addressed chunks of data
- Implementations
  - Kademlia DHT
  - DNS
  - BitTorrent trackers
  - Any other providing subsystems



Kademlia DHT

- Distributed hash table
- Based on the Kademlia paper
- Operations:
  - FIND_NODE
  - GET_VALUE and PUT_VALUE
  - GET_PROVIDER and PUT_PROVIDER



Kademlia: A Peer-to-peer Information System Based on the XOR Metric

Petar Maymounkov and David Mazières
{petar,dm}@cs.nyu.edu
http://kademlia.scs.cs.nyu.edu

New York University

Abstract. We describe a peer-to-peer distributed hash table with provable consistency and performance in a fault-prone environment. Our system routes queries and locates nodes using a novel XOR-based metric topology that simplifies the algorithm and facilitates our proof. The topology has the property that every message exchanged conveys or reinforces useful contact information. The system exploits this information to send parallel, asynchronous query messages that tolerate node failures without imposing timeout delays on users.

https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf

DataStorage Peer

- DataStorage peers have the logic to perform CRUD operations on our NoSQL database on GCP called Firestore.
- There can be n peers in network.
- Each peer goes in busy state once they start performing an action on the DB.

DataAggregation(Pre-processing) Peer

- DataAggregation peers aggregate the data for a particular user's content Id(userId).
- One user's data aggregated by a single node.
- Multiple users data could be aggregated by the same node.
- To avoid data loss while aggregation is still in process, distributed hash table which replicates the data in P2P network is used to avoid failure of a single node.
- The reason for this aggregation is because our ML model needs 10 seconds of data of each user to make a prediction. Therefore, 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 certain format. Could be used for preprocessing an image before handing it to an ML model.

Prediction(ML) peer

- Has the logic to pass the current user's IoT data to the sagemaker ML endpoint.
- 10 seconds of data for each user is passed to the inference endpoint.
- If there are any odd predictions like a user having heart arrhythmia, it communicates with one of the Notification peers to send an email to user and their doctor. It also sends data to one of the Data Storage nodes so that it can be stored in the Database.

This could be easily replaced by any ML model by just changing the Ml/DL model deployment endpoint

Notification peer

- Has logic to send notifications to the user and their doctor via Email using Amazon SES.
- Prediction peer and Blockchain dapp use this peer to send notifications.

Issues faced : libp2p

- It was first time building a peer to peer network. It was difficult to start with but the library code itself is written elegantly that we started looking at their code in github instead of documentation which is not well written to figure out how the library works.
- Still some of the libraries are in beta phase due to which not all features are available in all languages.
- python library has the least amount of features due to which we had to use nodejs to build our p2p network even though python was our preferred language of choice.

Other components are all managed services from AWS, GCP and Azure.

- For **data storage**, we use Managed NoSQL database from GCP called Firestore. Since our application needs low latency and realtime access to data, we chose this service. They support using WebSockets to watch updates to particular JSON documents stored in the database. For instance, when a blockchain transaction is updated in our database, it will automatically reflect on our UI without the user having to refresh.
- It also provides high availability, redundancy and fault tolerence.

Blockchain

- For Blockchain, we use Managed Ethereum Blockchain node on AWS called Amazon Blockchain Service.
- All requests to Ropsten Ethereum Blockchain public test network go through 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.

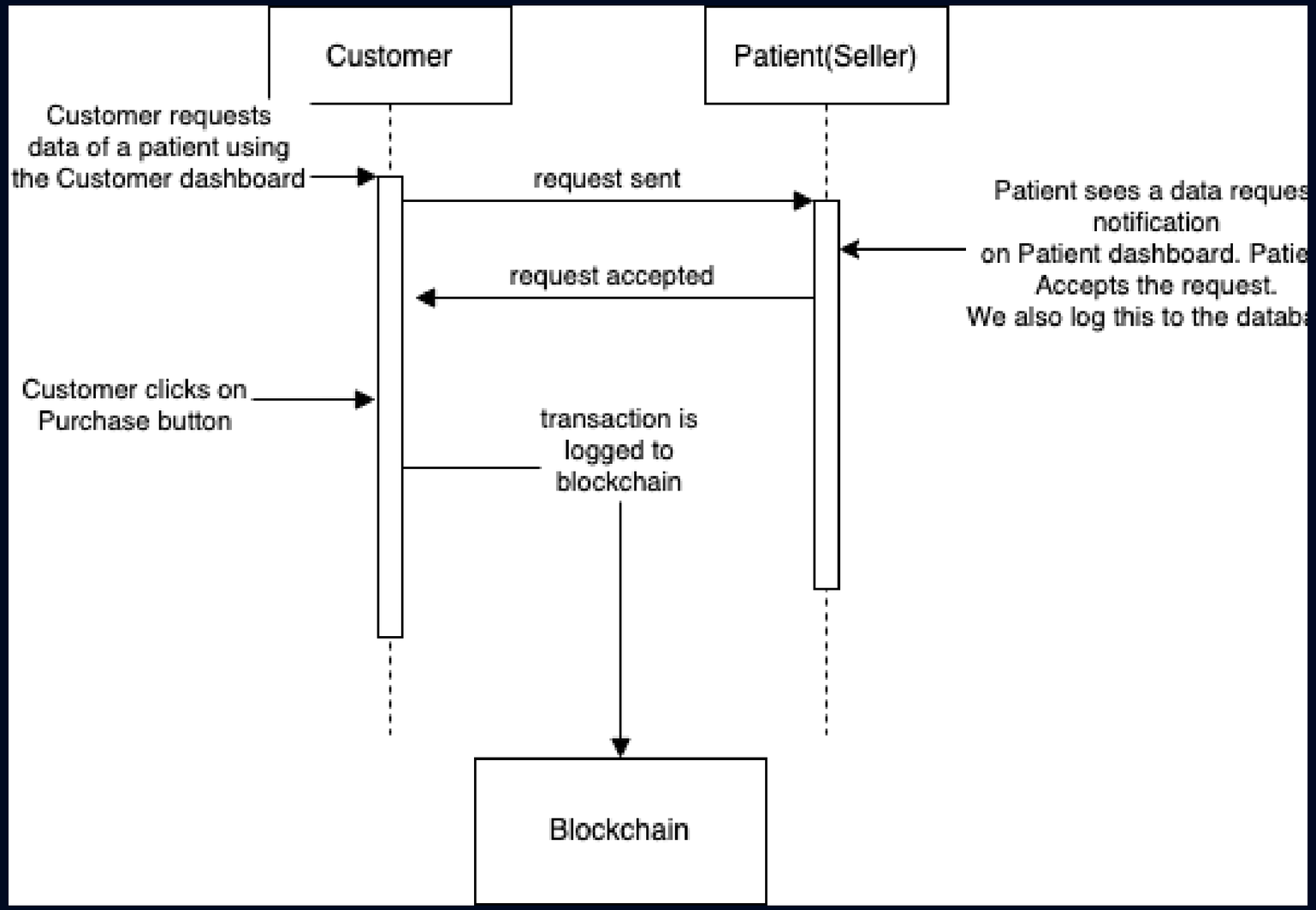
Why use blockchain?

- 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 keeping a ledger of all data access/purchase transactions.
- Moreover, we put the control of data sharing at the hands of the patients.

What is the frontend for Blockchain Dapp?

- We used React to build the frontend of the Blockchain Dapp.
- For styling UI, we used MaterializeCSS library.

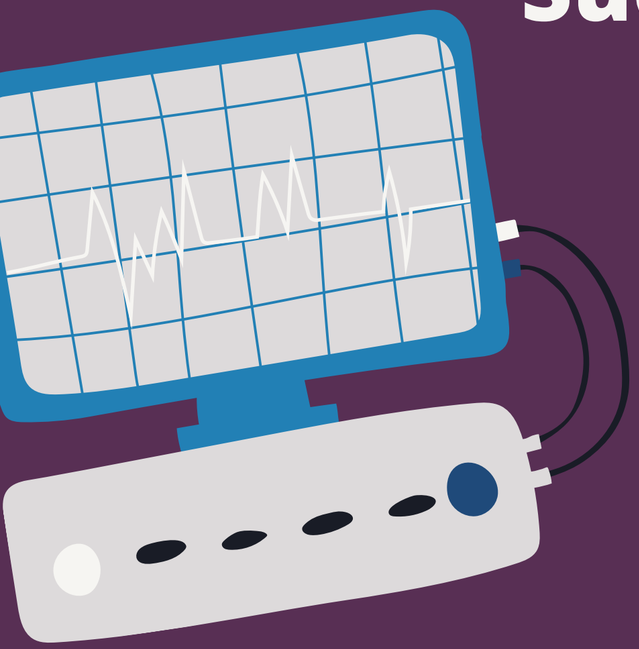
How does interaction happen between the users on Blockchain



How does interaction happen between the users on Blockchain

- Customers who are in need of data send a request to the user. This request is send to the database as well as the patient. Patient then needs to accept the request.
- Once, the patient accepts the request, customers will get a button in the UI to purchase user's data.
- **When the user clicks on Purchase button, smart contract transaction is made using the smart contract deployed on Ethereum Ropsten test network. A certain price is set by each patient for sharing their information. ETH equal to the price is deducted from customer's wallet and sent to the patient.**
- **Then, the user gets a button to access the data. When user accesses the data another contract method is invoked which creates a transaction on the blockchain.**

Success Story - Blockchain in Healthcare



- The Estonian government was seeking for new and creative ways to protect the health information of its 1.3 million citizens in 2016.
- It turned to Blockchain technology, which proved to be a huge success for Estonia's e-savvy government, which has now migrated most government operations to Blockchain, including coronavirus vaccination monitoring, digital court system and property registry.
- Most governments now wait an average of seven months to identify data breaches. These breaches, on the other hand, may be identified instantaneously using Estonian Blockchain technology.

ESTONIA

Issues faced: Blockchain

- It takes about 30 mins to 2 hours for the Blochchain node to be available on AWS since it needs to sync the current state of the Ethereum Blockchain node on AWS.
- AWS doesn't provide a library/SDK to make blockchain transaction from the Browser since they require that each of the transactions be signed using AWS Signature V4 mechanism. So, we had to write our own library which signs all the Blockchain transactions going to AWS Blockchain node. We use eth_sendRawTransaction to do this. AWS doesn't document well about this limitation due to which we had to spend a lot of time to figure out how to use their JS SDK on the browser.
- Also, when we checked their code, we found that it uses a very old JS mechanism to send requests using XMLHttpRequests. Hence, even though we modified it, we ended up getting CORS errors for all API requests since they have explicitly turned off CORS response headers on the server side.

IoT part of the project

Our IoT part of the project uses the following Azure services:

- Azure IoT Edge: acts as an entrypoint for our script that simulates receiving the heartrate data of 2 users. Each user's heartrate reading is sent every second to IoT edge module.
- Azure IoT Hub: We use this service to manage user devices. We also use it to receive data pushed by IoT edge.
- Azure Servicebus Queue: IoT hub pushes data into Azure Service bus Queue. This helps us regulate the rate of data flow between sender and receiver(master in P2P) so as to not overwhelm the receiver in cases where the receiver is down or slow.
- Master peer pulls data from Queue

Machine Learning part of the project

- For ML part of the project, we used AWS Sagemaker to deploy the model.
- Modeling itself was done using AWS Sagemaker notebooks.
- Endpoint is invoked by prediction peer.
- Heartrate data used for ML modeling
- Sagemaker endpoint returns whether the user has heart arrhythmia.

This could be extended to any ML model deployed on Sagemaker

References

- https://medium.com/encode-club/encode-filecoin-club-intro-to-libp2p-video-slides-2219792c45f8
- https://docs.libp2p.io/

Questions or comments?

Thank you