Offline First & Distributed Web Tech

Introduction

Features of web applications are only available while the users have an active connection to the Internet. When they go offline these features are rendered useless. Offline web applications try to ensure these features are available even when the user is not connected to the Internet.

One of the design goals for offline first applications is that these ensure all/most of their features will work in a disconnected scenario. Once this capability has been ensured now the focus turns to replicate/synchronize the data created/consumed during the time the app was disconnected from the Internet.

Imagine a world where your phone, TV and computer could all communicate on a common platform. Imagine it was easy to add video chat and peer-to-peer data sharing to your web application.

Decentralized computing is a mega trend that is not getting nearly as much attention as it deserves; it will likely have an economic, social, and political impact larger than the desktop and cloud revolutions.

What is offline first?

> *"'Offline' is probably a wrong word for it. It is much more productive to speak about 'intermittent connectivity'"* [1]

"Web" and "online" are two closely associated terms, downright synonymous to many people. So why on earth would we talk about "offline" web technologies, and what does the term even mean?

Even assuming internet utopia, where everything is always online, offline technologies still serve a purpose. Why? Because it's faster that way. Offline technologies support caching and detailed control over caching process. Therefore, web apps can boot quickly and show data instantly. You might protest that these technologies should not be called "offline" if they are used by purely online apps. Well, the term is ambiguous. At one level, "offline technologies" could be defined as "technologies supporting offline apps", making the protest valid. But another definition is "technologies operating outside of the cloud", and that's the one used here. Of course, there's a

[1] http://swarmjs.github.io/articles/offline-is-async/

lot of overlap between the two definitions; offline apps certainly need to operate outside of the cloud.[2]

"Web apps used to be completely dependent on the server: it did all the work, and the client just displayed the result. Any disruption in your connection was a major problem: if you were offline, you couldn't use your app" [3]

With offline-first, we assume that, as a baseline, your app will not have a working network connection. Then, your app can be progressively enhanced to take advantage of network connectivity when available. This takes a shift in mindset to not assume that lack of connectivity is an error condition. Build your app to work without a connection. Get updated content, sync your data, or enable features that aren't practical to make work offline when the app is connected. Let users still interact with your app when there's no reliable connection available.[4]

What is the distributed web?

"Where two or more devices are gathered, there the Internet is also" [5]

The next wave of computing is going to be a massive shift away from cloud computing. There are two major problems with cloud computing: (a) users don't own their own data, and (b) remote servers are security holes. With a move away from cloud computing, decentralized systems like Bitcoin give explicit control of digital assets to end-users and remove the need to trust any third-party servers and infrastructure.

The ownership of data and the associated power to monetize that data will flip over from large companies to users. Cloud storage providers will get reduced to "dumb drives", used to store encrypted data for users. It'd become hard for cloud storage providers to differentiate themselves from each other. In the new model, they all provide just a basic storage utility.
Users will start owning (personal) cloud servers again; servers that can be online 24/7 and take actions on behalf of users when the users go offline. Running a secure, personal cloud server will become as easy as using current cloud services like Google Docs and Dropbox.

Publishing the source code for software will become almost a requirement for security reasons. Running closed-source "black box magic" software will be perceived as a security hazard.
Crypto tokens for protocols will become as ubiquitous as software licenses and terms-of-service agreements for cloud services: to use the software in decentralized computing you'll need the respective token.
Crypto economy will reshape the entire lifecycle of tech companies. Currently, they get their start

[2] https://www.html5rocks.com/en/tutorials/offline/whats-offline/
[3] https://alistapart.com/article/offline-first
[4] https://techbeacon.com/offline-first-web-mobile-apps-top-frameworks-components
[5] http://chadoh.com/online-offline-equivalence/#148

on Sand Hill Road and end up on Wall St. Soon, they'd start anywhere and end up on token exchanges.[6]

[6] https://medium.com/@muneeb/the-next-wave-of-computing-743295b4bc73

The Context

Existing Tools

Offline client-side storage

- **localStorage**
  The read-only localStorage property allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions. localStorage is similar to sessionStorage, except that while data stored in localStorage has no expiration time, data stored in sessionStorage gets cleared when the page session ends — that is, when the page is closed.
- **IndexedDB**
  IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data.
  IndexedDB is a transactional database system, like an SQL-based RDBMS. However, unlike SQL-based RDBMSes, which use fixed-column tables, IndexedDB is a JavaScript-based object-oriented database. IndexedDB lets you store and retrieve objects that are indexed with a key; any objects supported by the structured clone algorithm can be stored. You need to specify the database schema, open a connection to your database, and then retrieve and update data within a series of transactions.
- **localForage**
  localForage is a JavaScript library that improves the offline experience of your web app by using an asynchronous data store with a simple, localStorage-like API. It allows developers to store many types of data instead of just strings. localForage includes a localStorage-backed fallback store for browsers with no IndexedDB or WebSQL support.
- **levelDB**
  LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
  Keys and values are arbitrary byte arrays.
  Data is stored sorted by key. The basic operations are Put(key,value), Get(key), Delete(key). Data is automatically compressed using the Snappy compression library.
  This is not a SQL database. It does not have a relational data model, it does not support SQL queries, and it has no support for indexes.

Synced databases (as a service)

- **Apache CouchDB**
  Seamless multi-master sync, that scales from Big Data to Mobile, with an Intuitive HTTP/JSON API and designed for Reliability. CouchDB lets you access your data where you need it by defining the Couch Replication Protocol that is implemented by a variety of

projects and products that span every imaginable computing environment from globally distributed server-clusters, over mobile phones to web browsers.

- **PouchDB**
  PouchDB is an open-source JavaScript database inspired by Apache CouchDB that is designed to run well within the browser. PouchDB was created to help web developers build applications that work as well offline as they do online. It enables applications to store data locally while offline, then synchronize it with CouchDB and compatible servers when the application is back online, keeping the user's data in sync no matter where they next login.
- **IBM Cloudant**
  IBM Cloudant is a fully managed JSON document DBaaS that's optimized for data availability, durability, and mobility - perfect for fast-growing mobile & web apps. What makes Cloudant unique is its advanced indexing and ability to push data to the network edge, across multiple data centers and devices, for faster access and greater fault tolerance. It allows users to access data anytime, anywhere.
- **SharedDB**
  ShareDB is a real-time database backend based on Operational Transformation (OT) of JSON documents. It is the realtime backend for the DerbyJS web application framework.
- **SyncedDB**
  SyncedDB makes it easy to write offline-first applications with real-time syncing and server-side persistence. SyncedDB makes web applications work beautifully both online and offline.
  You can write your client as if everything was stored offline! SyncedDB takes care of synchronizing the local database to other clients in real time.
  Since the widestream adoption of IndexedDB writing web applications with full offline support has been viable. But when storing data offline web applications typically lack the ability to seamlessly make a user's data available across devices. SyncedDB is a library that gives web applications the best of both worlds: a fully functional offline experience with real-time or on-demand synchronization of data when online.

Offline first + synced database frameworks

- **RxDB**
  A reactive Database for Progressive Web Apps.
  Multiplatform support for browsers, nodejs, electron, cordova, react-native and every other javascript-runtime.
  Replication between client and server-data, compatible with PouchDB, CouchDB and IBM Cloudant.
  Encryption of single data-fields to protect your users data.
  Multi-Window to synchronise data between different browser-windows or nodejs-processes.

- **SwarmJS**
  Swarm is a reactive data sync library and middleware. Swarm synchronizes your app's model automatically, in real time.
  Why do we make Swarm? Because new opportunities bring new challenges. Now, having all that laptops, smartphones and tablets on WiFi/3G, we need handover (aka continuity), real time sync and offline work. Those requirements stressed classic request-response HTTP architectures leading to fix-on-a-fix stacks that are far from perfection. We have built some apps like that.
  Our dream is to develop distributed applications like good old local MVC apps, by fully delegating the data caching/synchronization magic to a dedicated layer. We want to deal with the data uniformly, no matter where it resides. We believe, that CRDT is the only approach that allows to fully embrace the reality of distributed data.
  Swarm is a CRDT-based replicated model library (M of MVC) that keeps your data correctly cached and synchronized in real time using any storage and transport available.
- **Hoodie**
  A fast, simple and self-hosted backend as a service for your (web) apps, Open Source and free. No need to write server-side code or database schemas. Makes building offline-capable software a breeze.
  Hoodie is an Offline First, noBackend architecture:
  Hoodie’s API decouples the client and the server in the store process so it can get interrupted or stop at any stage without breaking. Thus, users can keep working without interruption and without having to fear data loss.
  noBackend doesn’t actually mean that there is no backend. It means that you don't have to deal with the backend because it is abstracted away and backend services can be accessed and used via the JavaScript API.
- **Firebase**
  Cloud-hosted NoSQL database that lets you store and sync data between your users in real-time.
  Real-time syncing makes it easy for your users to access their data from any device: web or mobile, and it helps your users collaborate with one another.
  Real-time Database ships with mobile and web SDKs so you can build apps without the need of servers. You can also execute backend code that responds to events triggered by your database using Cloud Function for Firebase.
- **DerbyJS**
  DerbyJS is a full-stack framework for writing modern web applications. Effortlessly sync data across clients and servers with automatic conflict resolution powered by ShareJS's operational transformation of JSON and text. The same templates can be rendered to HTML in the browser or on the server. This means you can have fast page loads, search engine support and even use the same templates to render emails.
- **RemoteStorage**
  An open protocol for per-user storage on the Web.
  Own your data: everything in one place – your place. Use a storage account with a provider you trust, or set up your own storage server. Move house whenever you want. It's

your data.
Stay in sync: remoteStorage-enabled apps automatically sync your data across all of your devices, from desktop to tablet to smartphone, and maybe even your TV.
Compatibility & choice: use the same data across different apps. Create a to-do list in one app, and track the time on your tasks in another one. Say goodbye to app-specific data silos.
Go offline: most remoteStorage-enabled apps come with first-class offline support. Use your apps offline on the go, and automatically sync when you're back online.

Peer to Peer distributed log stores

- **Hypercore**
  A secure, distributed append-only log.
  Sparse replication: only download the data you are interested in.
  Real-time: get the latest updates to the log fast and securely.
  Performant: uses a simple flat file structure to maximize I/O performance.
  Secure: uses signed merkle trees to verify log integrity in real time.
- **Scuttlebutt**
  A database of unforgeable append-only feeds, optimized for efficient replication for peer to peer protocols.
  Secure-scuttlebutt provides tools for dealing with unforgeable append-only message feeds. You can create a feed, post messages to that feed, verify a feed created by someone else, stream messages to and from feeds.
  "Unforgeable" means that only the owner of a feed can modify that feed, as enforced by digital signing. This property makes secure-scuttlebutt useful for peer-to-peer applications. Secure-scuttlebutt also makes it easy to encrypt messages.

Peer to Peer networks

- **IPFS**
  A peer-to-peer hypermedia protocol to make the web faster, safer, and more open.
  IPFS makes it possible to distribute high volumes of data with high efficiency. And zero duplication means savings in storage.
  IPFS provides historic versioning (like git) and makes it simple to set up resilient networks for mirroring of data.
  IPFS remains true to the original vision of the open and flat web, but delivers the technology which makes that vision a reality.
  IPFS powers the creation of diversely resilient networks which enable persistent availability with or without Internet backbone connectivity.
  IPFS aims to replace HTTP and build a better web for all of us.
- **DAT**
  DAT is the distributed data sharing tool. Use the desktop app, command line tool, and

javascript library.
Distributed Sync: DAT syncs and streams data directly between devices, putting you in control of where your data goes.
Efficient Storage: Data is deduplicated between versions, reducing bandwidth costs and improving speed.
Data Preservation: DAT uses Secure Registers with state of the art cryptography to ensure data is trusted, archived, and preserved.
Distributed Dataset Synchronization and Versioning paper[7]

- **Blockstack**
  Blockstack is a new decentralized internet where users own their data and apps run locally.
  Own Your Data: With Blockstack apps you truly own your data. It's kept on your device and encrypted before backed up in the cloud. This removes the need for blind trust in 3rd parties and makes it easier to keep your data safe.
  Own Your Apps: Blockstack apps put you in control of your software. Apps are loaded via a secure domain name system and live on your devices. Independence from 3rd parties makes you more safe.
  Own Your Identity: With Blockstack apps you own your identity. Your digital keys are seamlessly generated and kept on your device. This lets you move freely between apps and control your online experience.
- **ZeroNet**
  Open, free and uncensorable websites, using Bitcoin cryptography and BitTorrent network.
  Real-time updated sites.
  Password-less BIP32 based authorization: Your account is protected by the same cryptography as your Bitcoin wallet.
  Built-in SQL server with P2P data synchronization: Allows easier site development and faster page load times

[7] https://github.com/datproject/docs/blob/master/papers/dat-paper.pdf

The Goal

Thicket. A vine clone

Our goal is to create a Vine type app that works on the distributed web with offline first capabilities.

Vine (/ˈvaɪn/) was a short-form video hosting service where users could share six-second-long looping video clips. The service was founded in June 2012, and American microblogging website Twitter acquired it in October 2012, just before its official launch. Users' videos were published through Vine's social network and can be shared on other services such as Facebook and Twitter. Vine's app can also be used to browse through videos posted by other users, along with groups of videos by theme, and trending, or popular, videos. Vine competes with other social media services such as Instagram and Mobli. Launched on January 24, 2013, by December 2015 Vine had 200 million active users.[8]

For such we have the following high level requirements:

- Mobile app
- User authentication
- Social interaction (users follow other users)
    - Local storing of user's following list
- Creation of content
    - Local storing of user created content
- Creation of a personal feed of their own created content
    - Local storing of the personal feed
- Serving content
    - Sharing of local content via the decentralized web
    - Sharing of the personal feed via the decentralized web

The MVP

As a proof of concept for offline first and distributed web features we will develop an app where clients can connect to a decentralized network, create content and upload it to the network, find other users of the app and follow such users (read their content).

Client apps will share log stores (representing the user's content feed) amongst each other and similarly provide a way to serve/share content with to the network and thus to other client apps.

---

[8] https://en.wikipedia.org/wiki/Vine_(service)

The content will be posted to the decentralized network of choice and users will be able to request the files given the network's URL schemas.

The client apps will work in a p2p-swarm-like fashion being able to replicate user's logs to each other to make it easier for the network to maintain data synced in a distributed way.

The tools

How can we achieve **Thicket** with current technologies?

The following section describes how the current tools/frameworks/protocols can be used to achieve our requirements:

Mobile app

We will need to develop this.

User authentication

Scuttlebutt and Blockstack provide tools to authenticate users on their respective decentralized networks.

Social interaction (users follow other users)

Scuttlebutt provides tools to follow other users but we still need to develop/figure out a way for users to connect to each other, to connect to our app swarm or to connect to our network.

Creation of content and personal feed

We will need to develop this.

Local storing of user content & feed

We can use any of the offline client side storage tools described above for this.

Sharing the user feeds

Scuttlebutt and Hypercore provide tools to share and replicate (via p2p) log stores.

Serving content

IPFS, Scuttlebut, ZeroNet all provide tools to setup a server that responds to requests for content via their respective networks on the decentralized web.

Key Concepts, blockers with current tech & brainstorming

Best practices

Blockchains are not databases, these should store pointers (to data).
Move complexity and logic out of blockchains.

Serverless

Decentralized does not mean 100% serverless:

1. For clients/users to find each other and connect the need for servers still exists.
2. Users need to be able to *serve* their content.

The lack of a central place for the decentralized Internet

This is a conundrum. Since there is no one place where anyone can start contributing to the decentralized Internet does that mean we should a) develop our apps for all the decentralized platforms? b) create our own apps and provide our own discovery tools? c) create our own network/platform?

Some of the tools described above solve this issue in the following ways:

- Hypercore uses *signalhub* (https://github.com/mafintosh/signalhub)
- Scuttlebut uses *pubs* (https://scuttlebot.io/docs/config/create-a-pub.html)

There are some experiments trying to use *DHTs* (Distributed Hash Tables - this is how Torrent trackers work) in order to solve this issue[9] [10]:

- https://github.com/maxogden/discovery-channel
- https://github.com/mafintosh/discovery-swarm
- https://github.com/mappum/peer-exchange
- https://github.com/mafintosh/peer-network
- https://github.com/hallettj/bitable

Lastly, there is a proposal to attach information to any online resource that can be used to find entities online: https://webfinger.net/. This still relies on a central authority, maybe down the road this can be implemented into the decentralized web.

[9] http://libtorrent.org/dht_store.html
[10] https://github.com/webtorrent/webtorrent/issues/288

A server on your mobile phone?

In order to serve content IPFS, ZeroNet and Scuttlebut execute local servers that connect to their respective decentralized networks and respond to requests (with their own specific url schemas) to deliver content.
Currently these servers run on standalone/cli apps and might be overkill to try to run these on mobile devices.