# Décentralisation Stigmee

## A) Etat des lieux

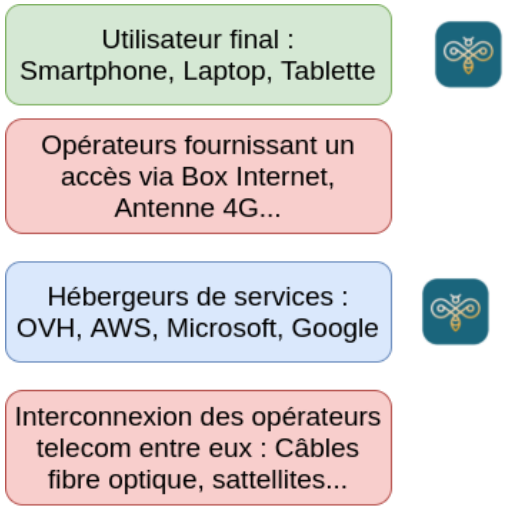
### 0. Un mot sur les Telecom et FAI.

Stigmee se voulant être un réseau décentralisé, le projet doit dès le départ être conçu pour ne pas être hébergé par une seule entité.

Partant de là, il nous faut définir qui héberge quoi. Un réseau composé uniquement d'appareils utilisateurs est pour le moment pas réalisable en raison des standards des opérateurs télécoms du monde entier.

Techniquement le réseau Internet (sur lequel a priori Stigmee s'appuiera) est taillé pur permettre la communication de n'importe quel appareil avec la totalité des membres du réseau, cependant, pour des raisons de sécurité et de manque d'IP que j'expliquerai plus bas, dans la réalité du monde réel, ça n'est pas le cas.

Pour comprendre où va se positionner Stigmee, voilà un résumé grossier de la façon dont fonctionne internet sous forme de couches :




**L'intérêt de ce schémas :**

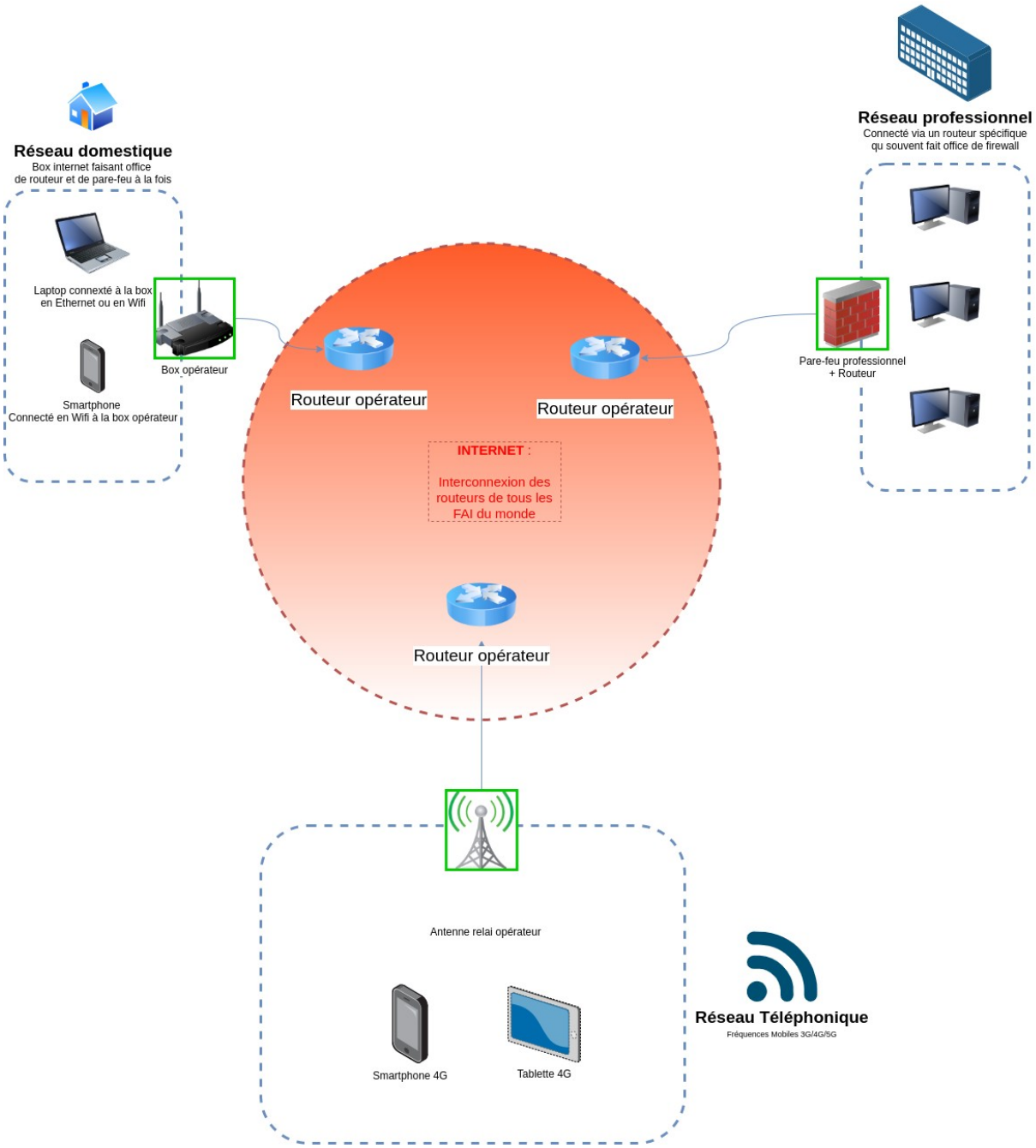
- En vert et bleu les segments sur lesquels nous avons une marge de manœuvre dans l'immédiat.
- En rouge les segments verrouillés sur lesquels stigmee ne peut pas agir (exemple : en ayant des routeurs stigmee dédiés).
- Mettre en relief les entités dont nous serons dépendants quoi qu'il arrive (Opérateurs téléphoniques).
- Quels intermédiaires entre les différentes parties du réseau (Les opérateurs interviennent à 2 endroits : Interconnexion entre eux, interconnexion des utilisateurs).

(à moins qu'un changement majeur mondial du fonctionnement d'internet arrive entre temps mais on va partir du principe que ça n'arrivera pas).

Compte tenu la façon dont internet fonctionne, Stigmee peut potentiellement exister à la fois au niveau que fournisseur de services en hébergeant une appli côté serveur, et en tant qu'utilisateurs, au travers. Si on élimine toute la partie routage, pare-feu, sur laquelle on ne peut pas agir, il y a grosso modo 2 types d'applications qu'on peut produire : Le serveur et le client.

C'est ainsi que fonctionnent la plupart des services des GAFAM aujourd'hui : Ils développent du code qui sera hébergé sur des serveurs, et développent des applications client téléchargés par les utilisateurs, programmées pour se connecter à ces serveurs uniquement.

**Frontières d'internet :**



A l’heure actuelle, il n’existe aucun réseau qui dépendent entièrement des terminaux des utilisateurs uniquement, et ce en raison de la façon dont sont implémentés les protocoles à la base de l’Internet par les Opérateurs / Fournisseurs d’accès à Internet.

En cause principalement : Les Firewall, et le NAT, qui placent systématiquement un intermédiaire puissant entre le terminal utilisateur, les autres terminaux, et les hébergeurs de services. Ce phénomène est mondial, et touche aussi bien les connexions mobiles que les box internet fixes ou StarLink de Tesla.

Cette limitation ne concerne pas les fournisseurs de services, ou toute entité hébergeant des serveurs quelconque, qui appliquent la politique qui leur paraît appropriée leur permettant d’ouvrir directement leur service au monde entier (ceux qui reçoivent des tentatives de connexion d’IP chinoises à 3h du matin comprennent de quoi il s’agit).

En conséquence, toutes les initiatives visant à décentraliser le réseau n’ont pu faire reposer leur projet entièrement sur les terminaux des utilisateurs et ont donc eu recours chacun à leur manière à des membres spécifiques du réseau qui agissent en tant que pseudo-serveur : serveur relai, fournisseur partiel ou total de services connecté uniquement à d’autres serveurs.

Aujourd’hui, la façon la plus simple de rendre accessible un service au plus grand nombre est le protocole HTTPS. Le protocole est tellement répandu par la navigation web qu’il fait partie des rares protocoles à être autorisé quasiment partout par défaut sans besoin d’intervention utilisateur.

Ce protocole résous donc la question de faire communiquer un client avec un serveur, mais pas les clients entre eux. En effet, pour pouvoir communiquer grâce à ce protocole, il faut un programme qui « écoute » sur le port 443 en TCP, or, la quasi-totalité des fournisseur d’accès Internet mobile et fixe bloquent ce port par défaut sur les terminaux de leurs clients (nous).
Nos terminaux peuvent donc contacter des serveurs qui « écoutent » sur un port 443 en TCP, mais ne peuvent eux pas être contactés sur leur port 443 en TCP.

Tous les exemples utilisés plus bas partent du principe que le terminal utilisateur a au moins accès au port TCP 443 de la plupart des services en lignes, ce qui dans la pratique est le cas, et même lorsque certains services sont explicitement bloqués (sur demande gouvernementale ou autre) ce standard permet à l’utilisateur de passer par des proxy pour utiliser ces services.

**Objectif du document :**

Dans ce document je tâcherai de présenter les différents modèles de distribution des services qui existent, en partant du plus centralisé au moins centralisé. A noter que ce papier n’a pas pour but de décrire la totalité des fonctionnalités des plateformes présentées mais uniquement leurs infrastructure.
Au final l’idée sera de parvenir à un mix de toutes les solutions pour trouver la plus appropriée au projet Stigmee.

Nous ne nous attarderons donc pas sur la qualité et le nombre de fonctionnalités, le ou les moyens cryptographiques utilisés, le modèle économique de chaque exemple, etc. Seul compte l’architecture globale de la plateforme et sa matérialisation sur les serveurs et les terminaux (laptops, pc, smartphones, tablettes...)
Les exemple analysés proposent des services différents, mais dans le fond toute logique de répartition des ressource reste réutilisable pour d’autres services

# 1. L’exemple des GAFAM :

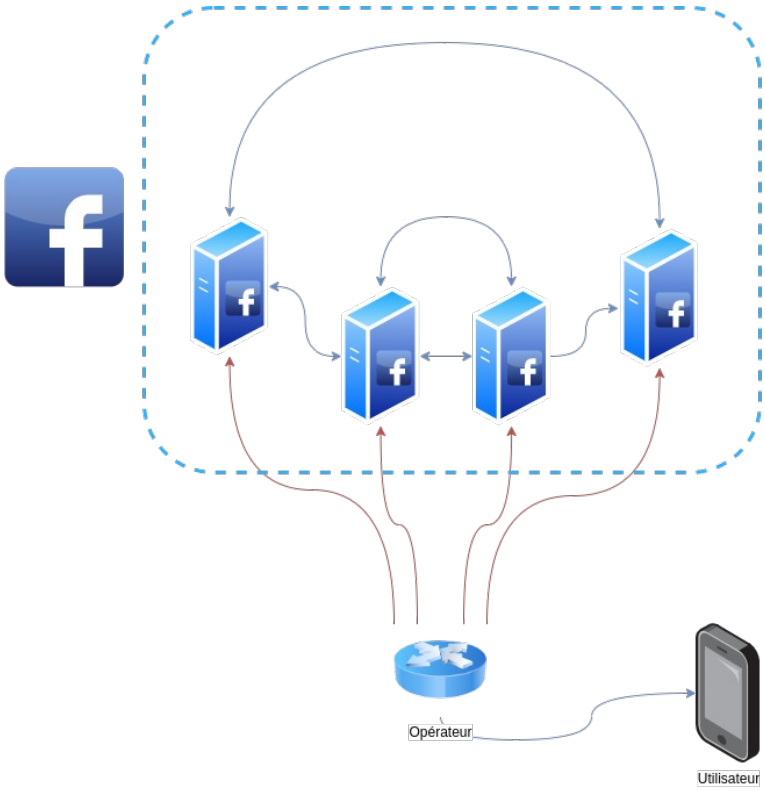
Centralisation humaine mais pas technique.

**Facebook est décentralisé ?**

En interne, l’architecture d’un GAFAM (prenons l’exemple du réseau social Facebook, indépendamment des autres services de Meta) n’est pas centralisé.
Facebook est une entreprise à but lucratif dont les services sont détenus et administrés par une entité hyérarchique totalement verticale. Mais cette centralisation est humaine et non technique.

En effet, physiquement parlant, il y a au contraire une grande distribution des ressources (bases de données, serveurs proxy, serveurs de cache, backups...) de façon a avoir une redondance et une distribution optimale des données.

Techniquement, Fracebook n’a donc pas 1 seul serveur sur lequel tous les utilisateurs se connectent, mais plusieurs Datacenters composés de milliers de serveurs qui se partagent la charge et les tâches, et que les opérateurs peuvent contacter à différents endroits de différentes manière, le tout de façon transparente pour l’utilisateur qui a la sensation d’accéder à un seul service :



La centralisation de facebook se matérialise donc surtout par son organisation humaine davantage que technique, sous la forme de ses noms de domaines, qui appartiennent à la même entitée.

Facebook peut très bien héberger une partie de ses serveurs chez AWS, OVH, ou dans ses propres datacenter, le résultat est le même :

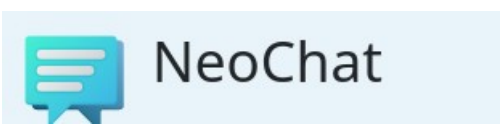
- Code source serveurs et clients fermé appartenant à une seule entité.
- Noms de domaines inséparables les uns des autres appartenant à une seule entité.
- Flou total sur l'architecture logicielle :( Bases de données ? Serveurs IA ? Serveur de Cache ? Serveurs Proxy ? Serveurs Web ?)

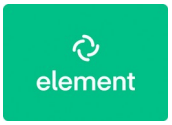
## 2. L'exemple Matrix :

Décentralisation des serveurs mais les clients très limités.


Matrix est un réseau de communication semblable à des messageries telles que Whatsapp, Telegram ou Signal (avec en ce moment une tendance à se rapprocher de discord) qui a pour particularité d'être décentralisée au niveau des serveurs.

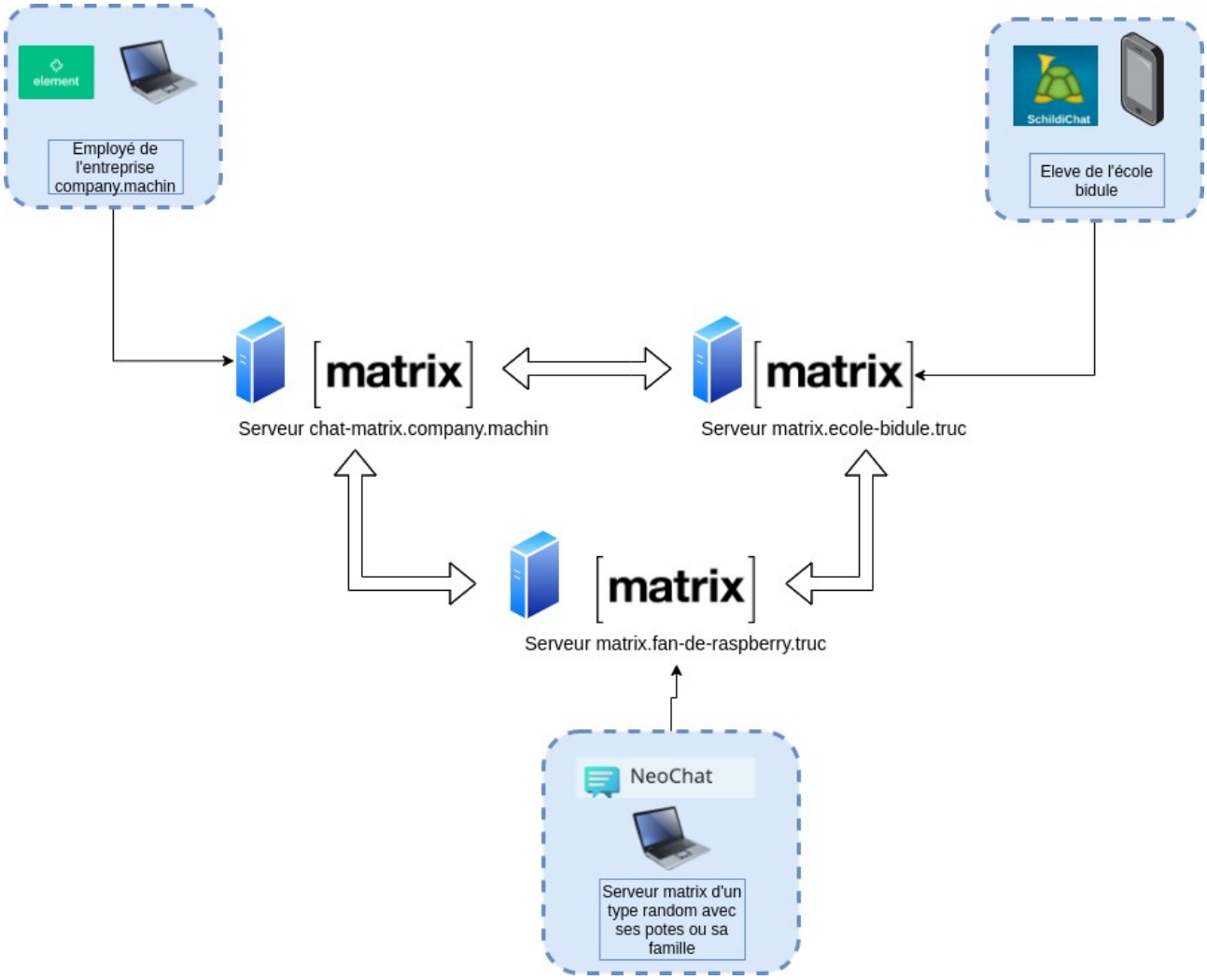
Matrix adopte donc toujours l'approche classique qui distingue clairement le serveur d'un client. D'ailleurs, il n'y a pas de client Matrix officiel, n'importe qui peut proposer son propre client qui fonctionnera correctement à condition qu'il respecte le protocole de communication tel qu'édicté par la fondation Matrix.
Un des clients les plus connus et grand-public étant element.io mais il en existe d'autres :




Conséquences de ces choix d’architecture :

- La décentralisation ne concernant QUE les serveurs Matrix et non les clients, qui communiquent entre eux pour transmettre les messages de leurs utilisateurs.
- Les utilisateurs d’un même serveur peuvent communiquer entre eux même si tous les autres serveurs du réseau Matrix sont off.
- Les clients ne sont pas indépendants des serveurs.
- Ils n’accèdent au réseau que par le biais d’un seul et unique serveur qui leur est attitré.
- Le contenu de leur compte est hébergé sur le serveur qui leur est attitré
- Ils ne peuvent pas s’en défaire à moins de créer un nouveau compte sur un autre serveur.




Le fonctionnement du réseau Matrix est très proche de celui des emails .

Un utilisateur a pour identifiant son pseudo précédé d'un @ et suivi par le serveur auquel il appartient, permettant ainsi à tous les utilisateurs de joindre un utilisateur sur un serveur différent du leur et communiquer dans un salon commun :

**@badro443:mestacloud.org**

**@samsam32:matrix.org**

**@idriss-a:stigmee.io**

Cette méthode d'organisation est très pertinent pour les cas comme les écoles, ou les organisations hiérarchiques ayant besoin d'avoir un certain contrôle sur leurs utilisateurs mais présente ses limites dans le cas d'un réseau se voulant le plus ouvert possible :

- Les données utilisateurs et l'ensemble de son compte bien que chiffrées et lisibles uniquement par les personnes autorisées sont stockés sur un serveur fixe, et tous les comptes sont perdus si le serveur disparaît.
- Le client n'a aucun autre moyen d'accès au réseau que l'unique serveur à partir duquel il a créé son compte.
- Si le serveur tombe (censure, coup de pelleteuse dans un cable optique, panne régionale...), l'utilisateur n'existe plus sur le réseau Matrix. Même si les terminaux (smartphone, pc…) de l'utilisateurs sont toujours actifs, il ne pourra utiliser son identité auprès d'un autre serveur.

La nature open-source du projet Matrix permet à tout le monde d'être membre du réseau de serveurs, avec les limitations quant aux clients qu'on a décrit plus haut.

# 3. Petit détour par BitTorrent :

Un protocole grillé, un principe intéressant.

Bittorrent est un protocole de partage de fichiers.  Il y a deux types d'entités sur un réseau bittorrent : Les Trackers, les seeders/leechers. Malgré la différence qui peut exister entre les seeders et leechers, leur existence logicielle est très similaire, donc pour synthétiser disons qu'ils ont le même rôle sur le réseau, désolé pour les puristes.

Dans le cadre d'une décentralisation, l'intérêt du protocole repose sur le fait qu'un réseau bittorrent ne s'appuie sur un serveur (appelé « tracker ») que pour avoir la liste des pairs qu'il pourra contacter pour télécharger le fichier demandé. Une fois la liste des pairs récupérés, le tracker n'est plus sollicités et les pairs discutent entre eux.

Le gros inconvénients du protocole bittorrent est qu'il est blacklisté par de nombreux fournisseurs d'accès, et très dépendant du bon vouloir des Firewall / NAT. Le protocole en lui-même n'est donc pas réutilisable bien qu'il soit open-source et un standard reconnu.

Le principe de faire appels à des serveurs uniquement pour mettre en lien les membres du réseau est intéressant et pourrait être réimplémenté dans un autre protocole qui ne sera pas limité par les casseroles de bittorrent.

Toutefois, il faudra contenir ce mécanisme dans une logique applicative, contrairement à bittorrent qui la traduit au niveau réseau.

Le problème de l’approche réseau de bittorrent est qu’elle encombre parfois rapidement les ports d’un routeur domestique et qu’elle s’étale sur de nombreux ports, la rendant assez « sale ».

On pourrait reprendre ce principe autrement, à voir plus tard...

# 4. L’exemple Briar :

Une décentralisation extrême qui estompes les frontières client/serveur.


**Présentation**

Briar est un service de messagerie dont la connexion entre les utilisateurs repose en partie sur le réseau Tor, mais pas uniquement. En effet, il est possible de choisir différents types de connexion, capacité introuvable dans l’éco-système des messageries instantannées. Nous allons d’abord nous concentrer sur la connexion Tor pour expliquer en quoi Briar diffère dans son approche de la gestion des identités.

Sa particularité première est qu’il stock LOCALEMENT le compte utilisateur. Là où tous les autres modèles maintiennent les informations utilisateur dans un ou plusieurs serveurs, avec Briar, le terminal de l’utilisateur est son propre serveur.
Pour réussir cet « exploit », Briar outrepasse les limitations firewall et NAT grâce au réseau TOR, qui est en soi un maillage de serveurs qui agit au niveau TCP (donc très bas dans l’abstraction des données), laissant une très grande liberté dans la variété des services proposées (HTTP, IMAP, SMTP...)

**Tor et Briar**

Tor n’est pas propre à Briar et joue dans notre cas le rôle que pourraient jouer simplement le réseau Internet s’il n’était pas aussi restreint et dépendant des politiques des Opérateurs Telecom / FAI.

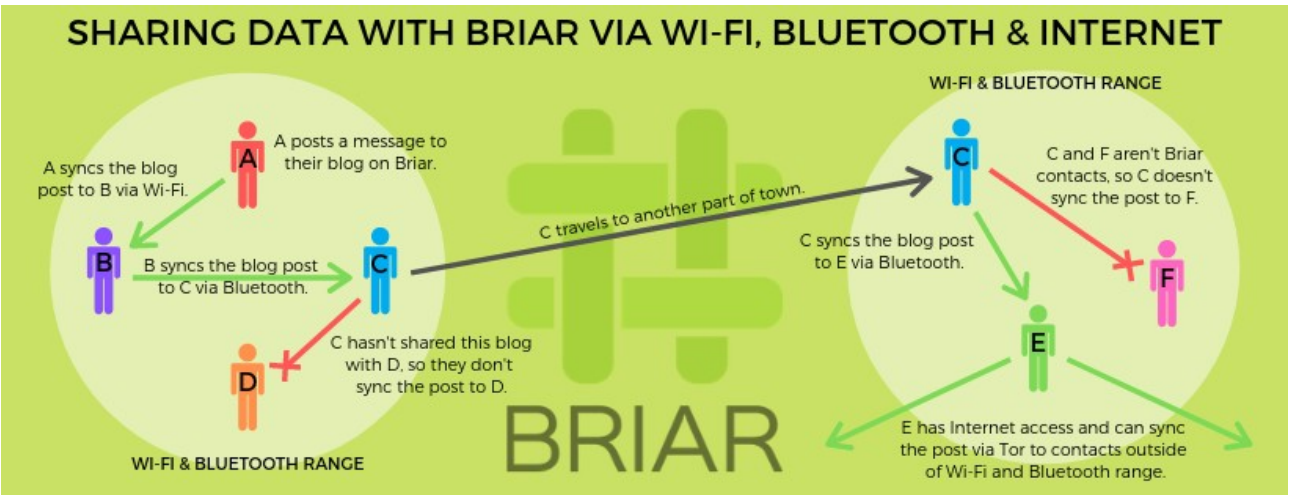
L’idée n’est pas ici de refaire une liste des fonctionnalités du réseau tor. Pour notre sujet, on peut retenir qu’en outrepassant les limites imposés par les NAT et Firewall, Tor recrée un réseau où tous les membres peuvent discuter entre eux sans restrictions, et où chaque demande peut être dirigée à son destinataire sans craindre d’être arrêtée par du NAT ou un Firewall.

Derrière cette prouesse : Les « Tor hidden services », qui permettent aux membres du réseau d’héberger leur propre serveur applicatif quelconque.
Briar embarque donc son propre « Tor hidden service » qui est en soi un genre de serveur à taille réduite, et stock dans la mémoire de l’appareil la *totalité* des informations utilisateurs. A tel point que si un utilisateur est hors-ligne, il n’existe tout simplement plus sur le réseau. L’avantage est que n’importe qui peut utiliser briar sans même savoir comment ça marche : Un QR Code à scanner, et tout marche tout seul.

La limite de cette approche est qu’il n’y a pas d’intermédiaire pour stocker le message en attendant qu’il soit délivré : il faut impérativement que l’expéditeur et le destinataire soient connectés au même moment (testé personnellement).

**Briar sans tor**

Cette limite a vocation a être contrebalancée à terme, en effet, Briar a vocation a être plus qu’un réseau pair-à-pair : un réseau « Meshé ».
C’est une façon de voir le réseau comme une grande toile faite d’interconnexions à la manière du réseau Internet tel qu’il existe dans sa dorsale (le routeurs qui composent le coeur d’internet). L’idée est que si durant une certaine période, deux parties A et B du réseau sont déconnectées, le simple fait qu’un utilisateur se déplace de la partie A à la partie B le transforme en genre de « postier » qui va livrer les messages qui n’ont pu l’être à cause de la rupture :




(Image provenant du site officiel de briar) https://briarproject.org/how-it-works/

Le fait est qu’actuellement cette fonctionnalité n’est pas opérationnelle ( je l’ai essayée avec plusieurs appareils en simulant la situation décrite ci-dessus, les messages ne sont pas livrés). Mais en théorie, le concept est valide.

Le point fort de cette fonctionnalité est qu'elle permet de se passer totalement de Tor et même d'internet ! Des réseaux briar éparpillés peuvent exister, se connecter, se déconnecter à souhait sans dépendre d'un serveur qui détient leur identitée et qui définit leur droit d'accès au réseau (comme dans le cas de Matrix).

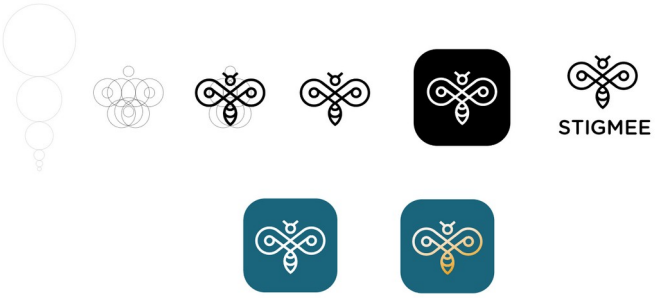
Sans s'étaler plus sur les fonctionnalités et le développement de Briar, l'idée principale est de faire de chaque utilisateur du réseau son propre serveur, et d'éliminer les coupures de communications possibles (NAT / Pare-feu) en utilisant le réseau Tor. Plus que ça, il a pour vocation d'être indépendant d'internet et de fonctionner localement à l'échelle d'une ville/village/pays si besoin.

Toutefois, cette décentralisation totale de l'identité et des données utilisateurs est rendue possible en grande partie par la légèreté des informations échangées. Chacun ne stock que ses propres messages qui ne contiennent ni photos ni fichiers ni autre donnée excessivement lourde.

# B) Stigmee

## 1. L'architecture physique du réseau stigmee.

Quelle architecture choisir, à défaut d'une seule, quel mix ? Pourquoi ?




Maintenant qu'on a dégrossi très grossièrement (lol) les différentes architectures existantes voilà quelques proposition pour l'infrastructure de Stigmee.
On aurait pu parler aussi des résaux sociaux comme Mastodon (clone libre et décentralisé de Twitter) ou Diaspora (Clone libre et décentralisé de facebook) mais leurs principes sont déjà globalement décris dans les exemples cités plus hauts, aucun n'a foncièrement innové dans l'architecture matérielle et le stockage des éléments du réseau.

Il y a beaucoup d'aspects à aborder, je commence par la partie échange/communication : les différents membres du réseau.

Concrêtement, les membres existeront sous la forme de programmes (donc les livrables) différents qui auront chacun un rôle différent. Le but est de limiter le type de rôles possible au maximum pour garder le « Keep it simple stupid ».

Comme expliqué plus haut il n’est pas possible (à ma connaissance) de se passer entièrement de « serveurs » en raison des blocages telecoms expliqués plus haut entre autres (on parlera du stockage et du slow web plus tard qui sont des points aussi importants). Donc je pars du principe qu’il y aura de toute façon des membres qui serviront de « relai » sur le réseau Stigmee. Pour ne pas parler de serveurs (car notre architecture doit casser avec le modèle classique client/serveur si j’ai bien compris l’ambition), appelons ces points de concentrations des « ruches » ?

La question est maintenant de définir quel sera le rôle de la ruche, et surtout qui pourra l’héberger ?

Je propose un modèle basé sur deux membres différents : Les « stigmers », et les « ruches. »

# 2. Les stigmers, Qui sont-ils ? Que sont-ils ?

**Désapprendre la centralisation des utilisateurs.**

La centralisation est un paradigme tellement ancré que beaucoup de réflexes d’informaticiens sont à désapprendre pour pouvoir vraiment penser un modèle qui correspond à l’ambition du projet.

Jusqu’aujourd’hui, très peu d’applications hormis Briar ont remis en cause le principe de « compte » pour définir l’existence d’un membre sur le réseau. Or, cette notion est intimement liée à la centralisation des identités par les fournisseurs de services.

Les stigmers sont censé être un la manifestation abstraite sur le réseau de l’être humain possédant le terminal. La question est de savoir si l’on garde le modèle traditionnel de « compte » (Modèle Facebook, Matrix) et donc en rattachant une identité à un serveur, ou si le stigmer se matérialisera de façon différente ?

La notion de compte a de pratique le fait qu’elle décharge l’utilisateur de la responsabilité de préserver ses propres données et son identité sur le réseau.

Pour faire concret, si ma grand-mère casse son téléphone, elle pourra toujours se connecter à Facebook via son laptop ou un nouveau téléphone grâce à son identifiant et mot de passe, car c’est Facebook qui détient en réalité son identité sur le réseau. Idem pour matrix sauf que l’identité n’est détenue que par un seul serveur du réseau (comme vu plus haut).

Comme éléments définissant l’identité d’un utilisateur, on peut prendre par exemple :

- Pseudo/Nom
- Numéro de téléphone
- Clés de chiffrement
- Photo de profil
- Messages envoyés
- Post publiés
- Dernière date de connexion

- Adresse IP utilisée
- Messages reçus ou en attente
- Etc…

Donc, le modèle Briar est lourd de conséquence dans le sens où un utilisateur lambda est responsable de la préservation de sa propre identité. Si l'on veut déléguer cette responsabiliité à un ou des tiers, alors se pose la question du modèle de confiance.

**Proposition : Le stigmer décentralisé**

Le modèle de confiance sera le choix clé (comme dans n'importe quel réseau humain) qui déterminera les responsabilités de chacun. Pour éviter qu'un Stigmer ne soit livré qu'à lui-même, on peut par exemple créer une copie de son identité qui sera sauvegardée chez plusieurs autres stigmers de façon fragmentée.

Je vois d'ici ceux qui disent : pourquoi pas simplement un backup serveur ? Pourquoi pas, mais alors dans ce cas, pourquoi pas laisser à l'utilisateur le choix du serveur ou du/des stigmer chez qui il veut être sauvegardé ?
Car je rappelle qu'étant donné la nature décentralisée du projet, nous ne bénéficions pas en théorie de l'énorme datacenter de Google/AWS/Microsoft pour stocker des comptes à l'infini, et que de toute façon ça n'est pas la vocation des fondateurs.

Par contre ! Des « ruches » pourront proposer contre rémunération par exemple, un espace de stockage dédié à la conservation de l'identité d'un stigmer !

**La sauvegarde du stigmer et le multi-device :**

A la création de son compte, un stigmer peut donc choisir s'il désire être « sauvegardé » chez un ou plusieurs autres stigmers de confiance désignés par l'utilisateur (qui pourront se partager les données en blocs distincts pour distribuer la charge).

Bien évidemment, ces données seront illisibles et non modifiables par les « gardiens » de l'identité du stigmer. Les moyens cryptographiques ne manquent pas pour parvenir à cette fin. Toutefois, dans l'idée de pousser la décentralisation au maximum, je suggère de s'appuier sur le protocole OpenPGP, qui entre totalement dans cette philosophie.

Dans cette idée, le Stigmer, aura deux choses à sauvegarder : La copie compressée et cryptée (je sais on dit chiffré mais je suis l'école Hardisk à ce sujet franchement c'est plus clair) de ses données. Et Sa clé privée PGP permettant de décrypter cette sauvegarde.
Cette clé devra être conservée dans un endroit sûr par le stigmer, qui sans ce précieux sésame ne pourra plus accéder au réseau (prix de la véritable confidentialité des données, cf Protonmail par exemple).

Pour distribuer la sauvegarde entre plusieurs stigmers, et ruches, on peut couper la totalité de la sauvegarde en plusieurs parties qui seront identifiable via des alrgorithmes de hachage classiques (SHA256, SHA512, …)

Cette méthode peut être utilisée comme base pour permettre la synchronisation de l'identité du stigmer entre plusieurs de ses appareils (mac, pc, smartphone, laptop, tablette, raspberry pi pour les meilleurs)

**L'héritier :**

La clé GPG permettant de décrypter la sauvegarde est évidemment protégée par un mot de passe comme le permet le standard OpenPGP (a priori simple a implémenter vu l'existence de librairies dans de nombreux langages connus). On pourait donc pourquoi pas envisager un rôle de stigmer « héritier » qui sans accès au mot de passe, pourrait tout de même conserver une copie de la clé privée.

Le ou les héritiers seront des stigmers humains dans lesquels un stigmer aura placé sa confiance, et à qui il confiera sa clé privée pour lui laisser une chance si vraiment il venait à la perdre. (Optionnel, qu'en pensez-vous?).

Cette approche est contraire au standard actuel de la plupart sinon toutes les cryptomonnaies : quand tu perd le mot de passe de ton wallet, tu es coincé dehors point final.

Mais cette approche est une approche confiance zéro, à l'équipe de voir si c'est un modèle de confiance intéressant ou si on se sent de passer à autre chose (à ce sujet ça serait bien d'avoir l'avis de gens comme Anice Lajnef par exemple, qui travaille pas mal sur la confiance appliquée à l'économie).

**Le stigmer sur le réseau global :**

Pour donner un maximum de pouvoir aux stigmers et ne pas faire reposer la totalité du réseau entre les mains d'organisations, le livrable sur smartphone, tablette et pc qui sera le point d'entrée du stigmer sur le réseau, doit aussi être un membre du réseau.

Selon les capacités et les choix de l'utilisateur, il doit pouvoir :

- Héberger les sauvegardes d'autres stigmers qui l'auront autorisé à le faire.
- Héberger des maps selon sa capacité de stockage.
- Se déclarer auprès d'une ou plusieurs ruches comme « hébergeurs de playlists et autres services. » (Cette approche permet de contourner le NAT en allégeant le trafic des ruches qui ne serviront que de relai et qui ne fourniront pas directement le service demandé)
- Echanger des messages et autres types de données avec d'autres stigmers présents dans le réseau local sans passer par une ruche.

Le troisième point est crucial pour permettre à un utilisateur sans compétences particulières de pouvoir fournir des services s'il estime en avoir la capacité en terme de stockage et de puissance de calcul.

**Le stigmer en LAN (réseau local) :**

Dans ce qu'on appelle communément un LAN (Local Area Network) où tous les membres d'un réseau IP sont en mesure de s'échanger des données sans restrictions firewall ou NAT, les stigmers doivent pouvoir se connecter les uns aux autres sans passer par une ruche.

Pour  y parvenir, un protocole de broadcast est à prévoir, on pourrait par exemple se baser sur celui de syncthing, le Local Discovery Protocol :  (il faudra que je vous parle de vive voix de cette merveille qui est un bijou de décentralisation, je suis littéralement amoureux de syncthing).

Cette fonctionnalité cruciale permettra l'existence du réseau stigmee et l'échange de données en dehors d'internet et les plus débrouillards arriveront à connecter des

stigmers entre eux même si toutes les ruches sont désactivées, c'est un héritage du mode local de Briar comme on l'a décrit plus haut.

# 3. Les ruches, Qui sont-elles ? Que sont-elles ?

**Qui ?**

L'idée est que les ruches aient vocation à être hébergées par des entreprises, des associations ou autres organisations, qui trouveront un intérêt à héberger du contenu contre rémunération, tout en permettant aux autres stigmers d'utiliser leur précieux port TCP/443 pour discuter avec les autres stigmers.

Le protocole d'échange avec les ruches doit être taillé de façon à établir un réseau de routage avec les autres ruches permettant de mettre en relation les stigmers entre eux.
Un utilisateur peut ainsi se « déclarer » ou se « connecter » à plusieurs ruches, et sa communication avec un autre stigmer passera par le chemin le plus optimal selon des mécaniques de routages qui restent à définir mais qui ne devraient pas avoir à être réinventés compte tenue de l'offre de protocoles de routages existants.

**Quoi ?**

L'utilisation de « ruches » comme maillage du réseau a deux principaux objectifs :

- Répondre au bloquage posé par les NAT/Pare-feu
- Permettre des points de concentration de données et de services plus conséquents que les terminaux utilisateurs (Contre rémunération?).

Contrairement à la totalité des autres approches dont les serveurs sont décentralisés (Nextcloud, Mastodon, Matrix, Diaspora…) les ruches n'hébergent pas les comptes clients.
Comme on l'a vu plus haut, elles peuvent proposer des services de sauvegarde et d'autres qui seront à définir, mais les utilisateurs n'ont pas d'obligation de se référer à une ruche unique.

Les ruches peuvent comme les stigmers, à l'aide du même protocole, partager des maps et autres données entre elles. On pourrait par exemple imaginer qu'un stigmer « connecté » à plusieurs ruches ( connectées entre guillemets car les échanges qu'impliquent cette connexion restent à définir ) ait de ce fait accès à des maps différentes qui vont élargir sa map, son île, et ses abonnements, ses services, etc.

Le protocole d'échange entre ruches doit partager la même base que celui entre stigmers, de sorte d'abord à assurer une certaine robustesse et simplicité des échanges (keep it simple stupid ?).

La seule différence est qu'une ruche étant gérée par une organisation, cette organisation doit pouvoir avoir un droit de regard sur ce que les stigmers partagent sur sa ruche. De cette façon, on pourra voir naitre des « ruches à thèmes » et une certaine émulation qui (espérons?) donnera une concurrence saine…

En parallèle aux droits qu'auront les organisations, la nature du protocole d'échange les « forcera » à servir de maillage à la manière dont les nœuds Tor servent de

maillage aux terminaux Briar. L'idée sera de faire en sorte que les organisations trouvent un intérêt à héberger une ruche tout en permettant aux stigmers bloqués par des NAT/Pare-feu d'échanger des données avec d'autres stigmers.

Je sais qu'Idriss voulait que nos grands-mères puissent faire tourner un nœud du réseau sur leur smartphone, malheureusement leur opérateur telecom / FAI ne laissera personne (pour leur bien?) accéder à un des précieux port TCP de leur smartphone, chose indispensable pour rendre ça possible. Il nous faut donc de toute façon un intermédiaire, et les ruches peuvent jouer ce rôle.

Personnellement j'opte pour un protocole qui aura une base commune aux ruches et aux stigmers car il permettra de rendre le serveur non-indispensable tout en permettant aux organisation souhaitant offrir des fonctionnalités supplémentaires à leurs utilisateurs de passer par leur plateforme.

Points clés du protocole en résumé:

- Comme Briar, deux terminaux (un smartphone et un laptop par exemple) dans un même réseau local n'ont pas besoin de serveur pour communiquer (Mode Briar sans Tor).
- Si une ruche Stigmee est disponible et qu'un « contrat » est conclu entre le stigmer et la ruche, le stigmer via son (ou ses?) terminal(aux) a accès aux fonctionnalités proposées par le serveur (stockage supplémentaire, maps spéciales, ou autre…)
- Si un serveur Stigmee n'est pas disponible, le terminal peut partager en wifi/bluetooth/ethernet… des données avec les utilisateurs autour de lui, ou les utilisateurs connectés au même VPN que lui (le VPN est une façon robuste de créer des réseaux locaux « artificiellement » ).
- Un utilisateur peut à tout moment via un bouton de l'application activer le mode ruche s'il a les compétences nécessaires ou s'il a trouvé un moyen d'ouvrir son port 443 (Coucou les fans de raspberry pi branché à la freebox).
- Chaque organisation peut héberger sa propre ruche, mais l'utilisateur reste maître de son terminal, les éléments clés de son identités sont stockées sur son/ses terminaux (et ceux de ses « héritiers ») et il ne dépend pas de la ruche pour exister sur stigmee.

Ces points clés ont vocation à faire naître une dynamique qui se veut équilibrée entre les libertés individuelles, et la « vie » en communauté du réseau.

Point important : cela signifie que comme avec le réseau Matrix, il n'y aura pas une seule adresse stigmee mais autant d'adresses qu'il y a de ruches. Les stigmers étant des « électrons libres », ils n'ont pas vocation à être identifiés par un nom de domaine au sens DNS  (trop compliqué pour ma grand-mère), mais les ruches si.

Maintenant, pour ne pas que le réseau en patisse trop, si une ruche venait à disparaître, son importance doit être limitée de façon à pousser au maximum la distribution des données sur les terminaux utilisateurs.

# 4. Le PaaS (Platform as a service)

Promis je parle de la blockchain juste après.

Si j’ai bien compris le whitepaper et les discussions que j’ai lu à ce sujet sur le discord, la blockchain doit être basée sur un proof of work MAIS, un proof of work utile contrairement au bitcoin où l’énergie utilisée pour le minage est tout simplement gaspillée.

Dans cette idée, encore une fois si j’ai bien compris, l’idée serait de mettre à disposition la puissance de calcul des terminaux (donc des stigmers et aussi éventuellement des ruches?) pour proposer des offres de cloud via Stigmee.

La référence dans le milieu étant AWS, je vais faire un rapide récap de ce que j’ai compris de l’infrastructure et de l’offre AWS. J’ai fait une formation d’une semaine sur leurs produits qui n’est clairement pas suffisante pour faire le tour de leurs services mais qui a le mérite de donner une idée de leur philosophie et ce qui fait leur attrait auprès des clients.

Sans etre expert sur le sujet, je me permet de vous livrer mon opinion car elle va fonder la suite de mon discours et si elle est fausse alors il faudra aussi revoir la suite :

Je pense que tout comme il est impossible et inutile de concurrencer google en reproduisant un moteur de recherche, il est inutile et impossible de concurrencer AWS directement sur leur terrain.

AWS répond à mon avis à une insatiabilité maladive de nos sociétés, dans lesquelles l’infobésité est la norme, et cette dynmique malsaine pousse à une gabegie énergétique monstrueuse. Ils mettent les moyens pour y répondre, et ils sont les meilleurs à le faire.

Voià quelques points qui me semblent être la force d’AWS que je vais détailler par la suite :

- Fiabilité : Robustesse, performance et redondance incroyable des services offrant une disponibilité parmi les meilleures.
- Confort : Une flexibilité et un pannel de services poussés à l’extrême qui offre un confort mêlant savamment enfermement propriétaire et efficacité maximale
- Confiance : Aussi curieux que ça puisse paraître, les client d’AWS font confiance à Amazon pour ne pas divulguer leurs données (très souvent des clients pro).

Avant d’aborder ces points en détails, je voudrai revenir sur la notion d’ubérisation d’AWS.

**Ubérisation d’AWS**

C’est un défi énorme qui est je pense possible de relever à condition de totalement changer de paradigme, parce que tout simplement : *AWS est déjà une ubérisation* du paradigme dans lequel l’informatique serveur est née.

Pour la petite histoire, jusqu’à il n’y a pas si longtemps et encore aujourd’hui dans pas mal d’entreprises, lorsqu’on avait besoin de services informatiques, on avait soit un service interne soit un prestataire (moi) qui venait dans vos locaux installer une salle serveurs, avec un contrôleur de domaine, un serveur de mails, de partage de fichiers, etc.

Cette approche traditionnelle (appelée on premise) est encore très présente mais tend de plus en plus à être remplacée par l'approche du tout cloud.
Les inconvéniants du « on-premise » sont les suivants :

- Consommation électrique permanente même lorsque les serveurs sont peu sollicités
- Gestion coûteuse d'une équipe dédiée à la maintenance de ces seuls serveurs en plus des éventuels développeurs travaillant sur la partie applicative.
- Manque cruel de flexibilité en cas de hausse ou baisse soudaine du trafic, ce qui rend des sites web indisponibles très facilement lors de période de soldes par exemple.

Ces points sont un résumé peu flatteur et très caricaturale de l'approche on-premise qui a pourtant ses avantages, mais on u reviendra une autre fois pour ne pas s'éparpiller. En attendant, voilà comment AWS a répondu à cette problématique :

- Venez chez nous et ne payer que ce que vous consommez (à la seconde)
- Venez chez nous et en cas de forte hausse de votre trafic on a assez de serveur instantannément disponible pour encaisser la hausse de votre trafic
- Comme on a plein de serveurs que tout le monde n'utilis epas en même temps, on mutualise les ressources
- Plutôt que de payer un énorme serveur pour lancer un script en python, utilisez notre superbe offre « AWS Lambda » pour ne payer que l'exécution d'un script (offre chirurgicale inédite à l'époque).

Bref en résumé, AWS a totalement ubérisé les providers cloud « traditionnels » qui louaient les serveurs facturés au mois de façon inflexible, et l'hébergement on premise et son lot d'inconvénients.

Maintenant penchons nous sur les trois forces d'AWS dont j'ai parlé plus haut : Fiabilité, confiance, confort :

**Fiabilité**

AWS répond à une demande toujours plus croissante d'instantannéité, et d'abondance de données. Que cette demande émane du grand public ou des professionnels.

Pour faire face à cette demande, Amazon a mis au point une ingénierie extrêmement poussée et fine qui va jusque dans les moindres détails techniques (ce qui nécessite une maîtrise totale de l'infrastrucutre physique), et des datacenters multipliés par autant qu'il le faut au travers d'« AZ «  (availability zones) pour garantir un basculement immédiat en cas de panne d'un point de leur infrastructure.

Le coût de cette approche est une infastructure tout simplement ahurissante, qui je pense dépasse encore aujourd'hui la puissance cumulée de tous nos terminaux personnels (a prouver évidemment…).

**Confort :**

On se sent tellement bien chez AWS qu'on ne peut plus bouger une fois qu'on y est (je parle en tant que sysadmin). Là où avant pour brancher un câble réseau il fallait lever ses fesse et aller en salle serveur, brancher le câble, paramétrer le switch ou le routeur, etc, on a toute cette infrastructure présentée de façon abstraite à l'écran.

En un clic, on branche un câble réseau. En un clic, on déploie un routeur. En un clic on a un serveur de l'OS de notre choix prêt à l'emploi. En un clic, on définit des légions de sous-réseaux pour notre entreprise qui nous aurait pris des jours de travail dans le monde « physique ».

Ce confort a un prix : Les services d'AWS ne sont absolument pas « interopérable ». Ce qui signifie que si vous voulez changer de fournisseur, il faut tout recommencer à zéro. Même et surtout pour partir chez un concurrent comme Microsoft Azure ou Google Cloud.
Toute vos infsrastructure, vos serveurs virtuels, vos segmentations réseaux, vos utilisateurs et politiques de sécurité, vos fonctions AWS *Lambda* ultra optimisée au poil, il faut tout refaire.

**Confiance :**

Malgré le cloud  Act, les « dérives » indénombrales de l'extra-territorialité du droit américain, AWS est un vecteur de confiance.
Les professionnels qui traitent des données client sensibles leur font confiance.
De même que ceux qui stockent leurs propres secrets professionnels, leurs échanges, leurs photos, leurs vidéos et autres.
C'est un point qui peut paraître banal, mais auquel il faudra penser si on veut décentraliser ce service. On peut garantir une très bonne sécurité pour le stockage, en utilisant des algos de cryptage (on dit chiffrement oui je sais) et donc fournir des services de stockae de données dans un cloud distribué…

Seulement voilà : cette logique ne peut s'appliquer qu'à de la donnée « morte », pas à des programmes qui sollicitent a mémoire vive et surtout qui ont besoin de faire transiter des données en clair pour êtres traitée par les processeurs de nos terminaux. On pourra donc concurrencer les différentes offres de stockage d'AWS, mais pas leurs offre de calcul.

Car il faut se dire qu'étant donné la nature totalement distribuée du réseau, des données sensibles vont potentiellement se retrouver chez n'importe qui n'importe quand, et c'est très difficile pour un professionnel d'accepter de gérer par exemple ses données clients dans un réseau distribué où il ne sait pas qui a accès à quoi.

Notre modèle de confiance actuel donne une place importantes aux « gros machins » dans tous les domaines malgré les innombrables bavures : Banques, Industrie agro-alimentaire, GAFAM pour la data, sécurité pour les Etats…
C'est un problème de mentalité qui pourrait changer si la confiance qu'on a les uns envers les autres change (Voir Anice Lajnef qui parle beaucoup de ce sujet concernant l'économie et les banques) mais en attendant le fait est qu'on ne peut pas se permettre de faire transiter en clair des données chez n'importe qui.

IBM fait partie des leaders d'un nouveau secteur : la « homomorphic encryption » (cryptographie homomorphique) qui permettrait d'utiliser la puissance de calcul de certains processeurs pour effectuer des calculs sur des données cryptées et donc préserver leur confidentialité, mais c'est encore très balbutiant et très peu performant. Mais la piste est lancée…

**Lien avec le Great Internet Mersenne Prime project**

L'idée de s'inspirer du Great Internet Mersenne Prime project est intéressante, mais la différence avec ce projet est de taille car l'approche est inversée. En effet, dans le cas deu projet GIMPS, le demandeur arrive avec un programme prêt et surtout un gage de

confiance : Le programme qui tourne sur la machine utilisateur est entièrement open-source.

Ce détail a son importance car exécuter un programme de source inconnue sur son ordinateur est un risque énorme surtout en 2021. GIMPS, en ouvrant le code source de son programme, garantit à tous qu'il ne sera pas utilisé à des fins d'espionnage.
Or, uberiser cette fonctionnalité reviendrait à offrir la puissance de calcul de machines inconnus à des programmes inconnus. La confiance n'est pas là pour permettre à un tel mécanisme de fonctionner.

**Solution possible : conteneuriser le calcul**

Comme solution, on pourrait imaginer des calculs très spécifiques aux données non-sensibles cloisonnés dans des environnements type conteneurs maîtrisés. Il suffirait par exemple de produire un conteneur embarquant un environnement de développement miniaturisé et étanche vis à vis du reste de l'OS du stigmer acueillant le calcul (avec des compilateurs de langages intégrés). Cela résoud la question de la confiance pour le stigmer.

Pour que cette unité de calcul cloisonnée trouve son marché, il faudrait qu'elle soit suffisamment ouverte au travers d'un standard de calcul qui rendrait possible l'exécution de calculs distribués. Actuellement les projets de recherches ont chacun le leur, aucun standard n'existe : il faut le créer.
On pourrait imaginer par exemple un standard d'interpréteur python distribué, un compilateur rust, C, C++, Go ou autre intégré, de façon à supporter les langages les plus communs et attirer un maximum de monde.

Concernant la conteneurisation, il faudra également inventer un standard. Il en existe actuellement de nombreux (Docker, LXC, OCI...), mais ils sont orientés serveurs et non déployables en un bouton pour ma grand-mère. Il faudra donc un standard de conteneurs disponible sur les grandes familles d'OS, qui grâce à une couche d'abstraction proposera l'exécution de calculs dans des langages préalablement inclus dans le standard (qui pourra évoluer au fur et à mesure que des langages seront supportés).

Sur linux, MacOS, iOS, et Android (qui est aussi un linux), la même base pourrait servir comme standard pour le conteneur étant donné la proximité (famille UNIX). En revanche pour windows c'est une autre histoire…
Je n'ai pas les compétences pour aller plus loin sur ce sujet et il faudra de toute façon sans doute des programmeurs bas niveau pour rendre une telle conteneurisation possible.

# 5. La blockchain

Enfin ? Vous allez être déçus !

Une fois qu'on a parlé de toute la partie architecture physique du réseau, on peut commencer à voir où caser la blockchain dans tout ce bazar.

La réponse courte est : Je sais pas.

La réponse longue :

On pourrait imaginer un système qui hash les « strands » ou les maps produits par les stigmers, ce qui constituerait le token de base du « Stigmee Solid » associé à chaque stigmer qui aura le premier découvert cette strand.
Cette partie là ne me paraît pas compliqué mais c'est surtout à voir avec les développeurs blockchain car pero j'ai aucune compétence là dedans.

Par contre là où ça me semble plus compliqué, c'est lorsqu'il faudra rémunérer les utilisateurs pour les calculs effectués sur leurs machines, et là les défis sont de taille. Je vais en lister quelques uns et ensuite je passe la balle aux développeurs de la blockchain parce que mes connaissances actuelles ne me permettent pas d'aller plus loin :

- Imaginer quels types de calculs possibles on pourrait distribuer sur le réseau en tenant compte des contraintes de confidentialité.

- S'assurer que le calcul effectué est « vrai » et pas du vent visant à stimuler le réseau pour miner comme un porc.
- Anticiper les déconnexion longues de certains membres de la blockchain (terminaux utilisateurs comme smartphones ou PC par exemple) qui du coup ne pourront plus valider de blocks en raison de leur déconnexion

Par rapport au dernier point cité : Si j'ai bien compris la blockchain bitcoin tous les nœuds sont connectés en permanence et si un nœud est offline, il devra rattraper son retard lorsqu'il revient à nouveau sur le réseau pour pouvoir valider des blocks. Cette architecture est extrêmement lourde et à priori intenable sur des terminaux comme des laptops ou des smartphones non ? Qu'en pensez-vous ?

Il nous faut soit réinventer une blockchain qui assume son côté plus « lent » (voir les militants du slow web qui seront peut-être intéressés?) soit se pluger sur une blockchain existante comme l'ETH ? A voir…

Ou alors, envisager de ne pas considérer tous les nœuds du réseau au même niveau et par exemple instaurer des relations de confiance entre certains stigmers et certaines ruches, de façon à ce qu'un terminal qui rejoint le réseau après une longue déconnexion (24h c'est long en temps internet) ne porte pas la responsabilité de valider tous les hashs minés en son absence.

# 6. Conclusion

C'est un déffrichage très grossier, j'ai essayé de faire le tour de toutes les problématiques pour ne rien oublier même sans rentrer dans les détails, mais il y a certainement encore beaucoup de choses à aborder.

En tout cas si il y a un point central pour moi c'est celui de la confiance. Les GAFAM sont construit sur les mêmes mécanismes de confiance que les grosses Banques ou les multi-nationales : Personne ne les aime mais tout le monde les utilise.

Avant de parler d'architecture technique, il faut je pense déjà parler d'architecture humaine car comme j'ai essayé d'expliquer plus haut, en réalité la véritable

décentralisation est davantage humaine que technique. Techniquement parlant les réseaux d'AWS, Google, Facebook et autres sont très décentralisés.

Qui fait confiance à qui comment ? Sur quel « contrat social » baser cette confiance ? Je n'ai pas la réponse à ces questions, c'est a réfléchir collectivement (invitez Anice Lajnef lol).

J'espère que ce document aura été utile. Beaucoup de points sont sujectifs mais évidemment tout est contestable, modifiable, discutable, mérite approfondissement ou remise en cause complète.

Ca fait très longtemps que j'attend un véritable challenger pour bousculer les GAFAM et qu'à mon niveau j'héberge au maximum mes propres servies cloud (le petit raspberry pi familial), je ne sais pas si j'ai les compétences requises pour aller très loin dans ce projet mais j'espère vraiment qu'une vrai plateforme décentralisée et puissante verra le jour, basant sa confiance sur un contrat plus horizontal que vertical.

Bonne continuation à tous, à bientôt !

16/11/2021 – v1.1 badro