Monétisation à l'aide des chaines de blocs pour les marchés de données de l'Internet des Objets

par

Wiem BADREDDINE

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE
M. Sc. A.

MONTRÉAL, LE 09 JUILLET 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Chamseddine Talhi, directeur de mémoire
Département de génie logiciel et des TI, École de technologie tupérieure

M. Kaiwen Zhang, co-directeur
Département de génie logiciel et des TI, École de technologie supérieure

M. Georges Kaddoum, président du jury
Département de génie électrique, École de technologie supérieure

M. Julien Gascon-Samson, membre du jury
Département de génie logiciel logiciel et des TI, École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 02 JUILLET 2020

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à exprimer ma gratitude à tous ceux qui ont contribué à ma réussite et qui m'ont aidé à réaliser ce mémoire.

Tout d'abord, je voudrais remercier les membres du jury d'avoir accepté d'examiner mon mémoire et évaluer le travail que j'ai accompli au cours de ces deux dernières années.

J'adresse mes sincères remerciements à mon superviseur Mr. Chamseddine Talhi pour son soutien et ses conseils judicieux tout au long de mes travaux de recherche. J'ai beaucoup appris de son expérience et de ses directives.

Ma plus profonde gratitude va à mon codirecteur Mr. Kaiwen Zhang pour son suivi permanent et sa disponibilité. Ses directives et ses conseils ont été une véritable force motrice pour accomplir ce travail.

Je dédie ce mémoire à mes chers parents à qui je suis profondément redevable pour leur amour, leur confiance et leur sacrifice. Je leur suis reconnaissante pour tous leurs encouragements et leurs conseils à chaque phase de mes études. À mon frère et ma soeur, merci d'être toujours à mes côtés et de m'encourager à prendre de grandes décisions dans ma vie. Je remercie également mon fiancé de m'avoir soutenue dans les moments difficiles. Mes derniers mots de remerciement vont à tous mes amis et collègues du laboratoire de recherche Fusée Lab et Multimédia.

Monétisation à l'aide des chaines de blocs pour les marchés de données de l'Internet des Objets

Wiem BADREDDINE

RÉSUMÉ

Le nombre d'appareils de l'Internet des objets augmente de façon exponentielle, générant une énorme quantité de données qui devient un atout précieux pour les analystes de données. Cette tendance culmine avec la création des marchés des données de l'IdO, où des flux de données provenant de sources hétérogènes sont envoyés en temps réel à divers consommateurs de données et sont mesurés à des fins de monétisation. Les systèmes de publications/abonnements, tels que le MQTT (Message Queuing Telemetry Transport), sont des solutions prometteuses pour servir en tant que couche de transport pour les flux de données en temps réel de manière découplée et à grande échelle. Cependant, ces systèmes de publications/abonnements manquent de deux propriétés essentielles pour un marché de données IdO : (1) ils ne fournissent aucune logique de monétisation ; (2) ils supposent que les courtiers en Pub/Sub sont des entités de confiance, ce qui n'est pas le cas dans le cadre d'un marché décentralisé ou fédéré. À cet effet, nous abordons ces problèmes en utilisant un système de monétisation fiable et transparent basé sur la technologie de registres distribués (DLT) et des contrats intelligents. Nous proposons trois solutions de monétisation et nous démontrons le compromis entre les surcoûts liés au suivi des données IdO sur la chaîne de blocs, et la précision de la monétisation pour les producteurs et les consommateurs de données. Nous fournissons notamment une solution basée sur un filtre de Bloom pour une vérification efficace de l'échange de données. Nous mettons en œuvre notre système en utilisant Ethereum et Solidity et nous évaluons les performances de notre système par rapport au coût du gaz contractuel.

Mots-clés : IdO, Marché de données, Chaînes de blocs, Contrats intelligents

Monetization using Blockchains for IoT Data Marketplace

Wiem BADREDDINE

ABSTRACT

The number of Internet of Things devices is growing dramatically, generating a huge amount of data which is becoming a valuable asset for data analysts. This trend culminates towards the creation of IoT data marketplace, where streams of data from heterogeneous sources are sent in real time to various data consumers and are metered for monetization purposes. Publish/subscribe systems, such as Message Queuing Telemetry Transport (MQTT), are a promising solution to act as a transport layer for real-time data streams in a decoupled and large scale manner. However, pub/sub systems lack two key properties for an IoT data marketplace : (1) it does not provide any monetization logic ; (2) it assumes that the pub/sub brokers are trusted entities, which is not the case in a decentralized or federated marketplace setting. In this context, we address these issues using a reliable and transparent monetization system based on Distributed Ledger Technology (DLT) and smart contracts. We propose three monetization solutions and demonstrate the trade-off between the overhead of tracking IoT data on a blockchain vs. the accuracy of the monetization for data producers and consumers. In particular, we provide a Bloom filter-based solution for efficient verification of data exchange. We implement our system using Ethereum and Solidity and evaluate with respect to contract gas cost.

Keywords: IoT, DataMarketplace, Blockchain, Smart contracts

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

DLT	Distributed Ledger Technology
EVM	Ethereum Virtual Machine
HD	Hierarchical Deterministic
IdO	Internet des Objets
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
Pub/Sub	Publish/Subscribe
XML	Extensible Markup Language

INTRODUCTION

Avec la croissance exponentielle de l'Internet des Objets (IdO), et l'omniprésence de la connectivité et l'interaction entre les appareils IdO, une quantité volumineuse de données sensibles est continuellement générée. En effet, plus de 11 milliards d'appareils IdO sont connectés en 2018 selon Gartner, générant environ 52 zettaoctets en 2019 selon une estimation réalisée par les systèmes Cisco et IDC (Yusuf et al., 2020).

La majorité de ces données générées est inexploitée alors qu'elle peut produire une valeur financière importante. Ainsi, les flux de données IdO en temps réel sont désormais considérés comme un actif précieux créant des opportunités prometteuses pour construire des applications dans différents domaines tels que la recherche médicale (Albahri et al., 2018), le marketing ciblé (Lessmann, Haupt, Coussement & De Bock, 2019) et les villes intelligentes (Ahlgren, Hidell & Ngai, 2016). À cet effet, les propriétaires d'appareils intelligents cherchent à monétiser leurs flux de données et en faire une source de revenus. Cette tendance culmine avec la création des marchés de données de l'Internet des Objets. Les marchés de données ont de multiples approches centralisées et décentralisées. Les principaux problèmes sont la confiance et la fiabilité de l'échange de données ainsi que le manque d'automatisation et de l'évolutivité. Par ailleurs, le transport de flux de données IdO en temps réel peut être réalisé à l'aide de systèmes de publications/abonnements (Pub/Sub), tels que MQTT (Yassein, Shatnawi, Aljwarneh & Al-Hatmi, 2017). Les systèmes Pub/Sub permettent aux consommateurs de données (abonnés) d'exprimer leur intérêt par le biais d'abonnements, qui sont comparés au trafic entrant des publications provenant de producteurs de données (éditeurs). Ce processus de mise en correspondance et d'acheminement est assuré par des courtiers, qui peuvent réaliser ces tâches efficacement à grande échelle (Eugster, Felber, Guerraoui & Kermarrec, 2003). Les principaux systèmes de publications/abonnements utilisés dans la pratique incluent Facebook Wormhole (Sharma et al., 2015), Google Cloud Pub/Sub (Krishnan & Gonzalez, 2015), Redis (Redis), Apache Kafka (Kreps et al., 2011) et Yahoo ! Pulsar (Francis & Merli, 2016).

Les systèmes de publications/abonnements sont à l'origine destinés à être utilisés dans un environnement logiquement centralisé, où les informations sont échangées entre les producteurs de données et les consommateurs appartenant à la même entreprise. En conséquence, les systèmes Pub/Sub manquent deux propriétés clés lorsqu'ils sont utilisés dans le cadre d'un marché de données IdO. Premièrement, ils n'offrent aucune logique de monétisation. Bien que les courtiers gardent une trace des informations d'utilisation, comme le nombre de publications envoyées ou reçues par les différents clients, ou la quantité de bande passante utilisée, le système n'utilise pas ces informations de façon intuitive pour calculer des paiements vers les éditeurs et depuis les abonnés. Deuxièmement, les systèmes Pub/Sub sont conçus en partant du principe que les courtiers effectuent correctement les tâches de correspondance et de routage. Dans un marché de données IdO, les producteurs et les consommateurs de données devraient idéalement être en mesure de transférer des données de manière décentralisée (Draskovic & Saleh, 2017 ; Javaid et al., 2019 ; Özyilmaz, Doğan & Yurdakul, 2018). Ainsi, l'implication de tout tiers dans le processus de partage des données doit être transparente et exige un minimum de confiance.

Afin de traiter conjointement ces deux problèmes, nous optons pour l'utilisation de la technologie des registres distribués (DLT) qui constitue une nouvelle technologie émergente, présentant de nombreux avantages en termes de sécurité, de transparence et d'immuabilité. Nous utilisons la chaîne de blocs qui s'agit d'un registre distribué éliminant le risque de point de défaillance unique et de la confiance aux tiers. L'objectif principal de notre mémoire consiste à mettre en place un mécanisme de traçabilité et de commercialisation de données pour les systèmes Pub/Sub en assurant un échange transparent et une monétisation précise du marché des données IdO. Dans cette perspective, nous proposons un système de monétisation des données IdO basé sur les chaînes de blocs afin de fournir des enregistrements immuables et une traçabilité solide des activités de partage de données des éditeurs, des abonnés et des courtiers. En outre, nous considérons tirer avantage des contrats intelligents afin de fournir un mécanisme robuste de vérification de la cohérence des enregistrements sur la chaîne de blocs. Enfin, nous mettons

en avant une logique de monétisation échangeant des transactions de cryptomonnaies sur les chaînes de blocs.

Dans le cadre de ce mémoire, nos principales contributions sont les suivantes :

- nous proposons un système de monétisation de données IdO dans lequel les activités de Pub/Sub sont enregistrées et qui déclenche une exécution périodique des contrats intelligents pour vérifier et calculer les paiements de monétisation nécessaires (chapitre 3) ;
- nous introduisons trois solutions de traçabilité : Trace-MIN, Trace-MAX et Trace-BF, qui fournissent différents degrés de traçabilité et de surcoût (section 3.4). En particulier, la solution Trace-BF utilise le filtre de Bloom pour assurer un stockage efficace des données et une vérification plus rapide ;
- nous implémentons nos trois solutions avec MQTT en tant que notre protocole de messagerie Pub/Sub et nous mettons en œuvre les contrats intelligents écrits en Solidity et déployés sur Ethereum (chapitre 4). Notre évaluation démontre le compromis entre la précision de la monétisation et le coût du gaz contractuel.

Dans ce qui suit, nous organisons notre mémoire de la manière suivante : En premier lieu, nous introduisons le contexte de notre projet en décrivant les notions de base et nous passons en revue les principales technologies utilisées dans nos solutions. En second lieu, nous présentons les différents travaux de recherche que nous avons étudiés afin de cerner nos contributions et investiguer les limites des solutions existantes. Ensuite, nous décrivons dans le troisième chapitre les différents composants de notre architecture, nous définissons notre modèle de monétisation et nous détaillons nos solutions proposées. En effet, dans ce chapitre nous nous concentrons à répondre aux questions suivantes :

- Quelles sont les informations qui doivent être tracées dans chacune des solutions ?
- Quelle entité est responsable d'écrire ces informations sur la chaîne de blocs ?

- Comment vérifier la cohérence des informations échangées sur la chaîne de blocs ?
- Quelle est la métrique à utiliser pour monétiser l'échange de données ?
- Comment pénaliser les entités responsables en cas de détection d'incohérences ?

Dans le quatrième chapitre, nous présentons les résultats de l'évaluation de nos trois solutions. Nous présentons également les détails de l'implémentation de notre système. En dernier lieu, nous récapitulons notre parcourt durant ce mémoire ainsi que les défis rencontrés.

CHAPITRE 1

NOTIONS DE BASE

L'Internet des Objets représente des milliards d'appareils physiques dans le monde entier qui sont désormais connectés au réseau pour diffuser les informations collectées sur l'environnement via des capteurs, ou pour permettre à d'autres systèmes d'agir sur le monde physique par le biais d'actionneurs. L'incroyable croissance et diversité des systèmes et des applications IdO ont généré un énorme flux de données qui jouent un rôle essentiel dans les systèmes intelligents basés sur l'IdO. La valeur de cet échange réside dans la disposition des données en temps réel et pouvoir utiliser ces données pour une multiplicité d'objectifs. Dans ce chapitre, nous donnons un aperçu sur les systèmes de publications/abonnements qui permettent un échange en temps réel des messages de l'IdO. Ensuite, nous détaillons la technologie des registres distribués et des contrats intelligents. Enfin, nous définissons la technologie des filtres de Bloom que nous utilisons dans l'une de nos trois solutions.

1.1 Modèles de communication publications/abonnements

Le système de publications/abonnements (Pub/Sub : Publish/Subscribe) est un service de messagerie asynchrone, simple et puissant pour les communications entre machines. Pub/Sub permet aux consommateurs de données (*Subscribers*) de s'abonner à des canaux spécifiques, appelés *topics* auxquels les producteurs de données (*Publishers*) envoient des données générées en temps réel. Un courtier appelé *Broker* est responsable de faire correspondre les publications aux sujets et les acheminer aux abonnés. La structure que nous venons de détailler est illustrée dans le schéma de la figure 1.1 ci-dessous. Le modèle de publications/abonnements se caractérise par un support inhérent à la messagerie point à multipoint ce qui le rend utile pour les cas d'utilisation de l'IdO. De plus, il fournit une abstraction qui permet le découplage des producteurs de contenu (éditeurs) et des consommateurs de contenu (abonnés).

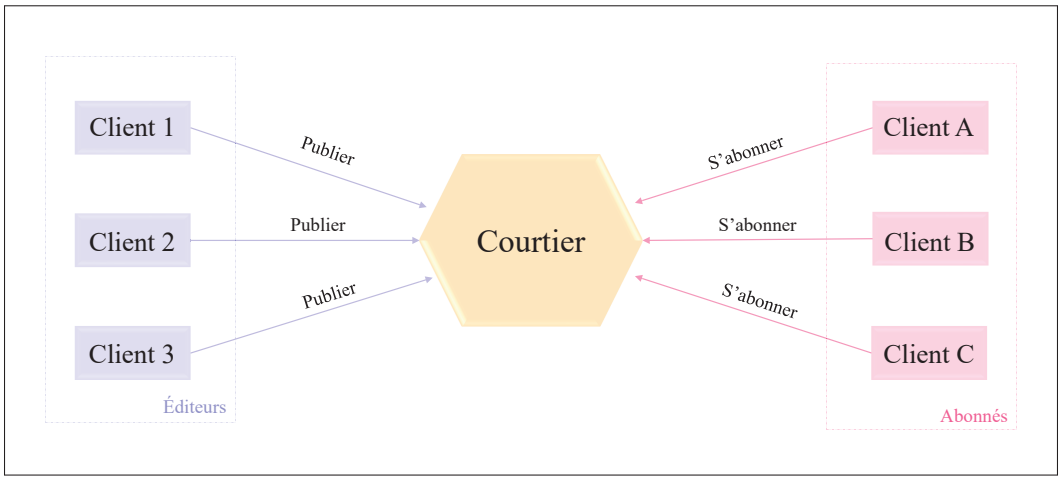

Figure 1.1 Exemple de modèle de communication Pub/Sub

1.1.1 MQTT : Message Queuing Telemetry Transport

MQTT est un protocole de messagerie de Pub/Sub largement utilisé et basé sur le protocole TCP/IP (MQTT). Il est spécialisé dans la communication de machine à machine avec une messagerie légère, un routage plusieurs à plusieurs et une compatibilité avec les appareils à ressources limitées et les réseaux à faible bande passante ou à forte latence ou peu fiables (Atmoko, Riantini & Hasin, 2017 ; Chooruang & Mangkalakeeree, 2016 ; Pooja, Uday, Nagesh & Talekar, 2017). Les principes de conception de MQTT visent à minimiser les besoins en bande passante du réseau et en ressources des appareils, tout en essayant de garantir la fiabilité et un certain degré d'assurance de livraison.

En effet, MQTT supporte trois niveaux de qualité de service (QoS) qui définit le principe de livraison des messages publiés par le protocole (Katsikeas et al., 2017). Les clients abonnés peuvent spécifier le niveau de QoS maximal qu'ils souhaitent recevoir :

- QoS 0 : Le message ne sera délivré qu'une seule fois,
- QoS 1 : Le message sera livré au moins une fois,
- QoS 2 : Le message sera livré au plus une fois.

1.1.2 XMPP : Extensible Messaging and Presence Protocol

Le protocole XMPP est un protocole de messagerie instantanée basé sur une architecture client/serveur. Le protocole permet un échange asynchrone des informations au format XML entre les entités au travers du serveur XMPP. Les entités peuvent créer des nœuds (topics) et publier des données sur ces nœuds. Ainsi, une notification d'évènement est diffusée pour informer toutes les entités qui se sont abonnées aux nœuds par la nouvelle publication. Ce protocole est caractérisé par son adressage des clients avec un identifiant unique (Saint-Andre, 2011).

1.1.3 AMQP : Advanced Message Queuing Protocol

Le protocole AMQP est un protocole de transfert binaire qui a été conçu principalement pour les applications d'entreprise et la communication de serveur à serveur (Vinoski, 2006). Il est basé sur le principe de producteur/consommateur qui est équivalent au publisher/subscriber de MQTT. En outre, le protocole AMQP permet l'acheminement d'un nouveau message d'un producteur vers plusieurs sujets selon certains critères du message tels que le contenu, l'entête ou les clés de routage. De ce fait, les différents consommateurs qui se sont abonnés sur divers sujets peuvent recevoir le même message.

1.1.4 Comparaison des protocoles de messagerie

On résume dans le tableau 1.1 les principales caractéristiques des protocoles de messagerie basé sur le modèle de communication de publications/abonnements décrits ci-dessous (Sennoun, 2018). Dans notre système Pub/Sub, nous intégrons le protocole MQTT en tant que couche de transport de messages pour le marché des données IdO. Nous considérons adapter nos solutions dans des travaux futurs à d'autres systèmes de communications Pub/Sub.

Tableau 1.1 Protocole de messagerie Pub/Sub

Protocole	**MQTT**	**XMPP**	**AMQP**
Type du protocole	Orienté Message	Orienté Message	Orienté Message
Transport	TCP/IP	TCP/IP	TCP/IP
Format	Binaire Texte	XML	Binaire Texte
QOS	Oui (3 modes)	Non intégré	Oui (3 modes)
Plate-forme	HiveMQ, Mosquitto, Eclipse Paho, Mosca	Jabber XMPPFramework	RabbitMQ StormMQ

1.2 Technologie des registres distribués

Un registre distribué est un registre de données enregistré et synchronisé sur un réseau d'ordinateurs. Lors de l'ajout de nouvelles données au registre, tous les nœuds du réseau doivent d'abord les valider. Une fois ajoutées, les informations ne peuvent jamais être modifiées ou supprimées.

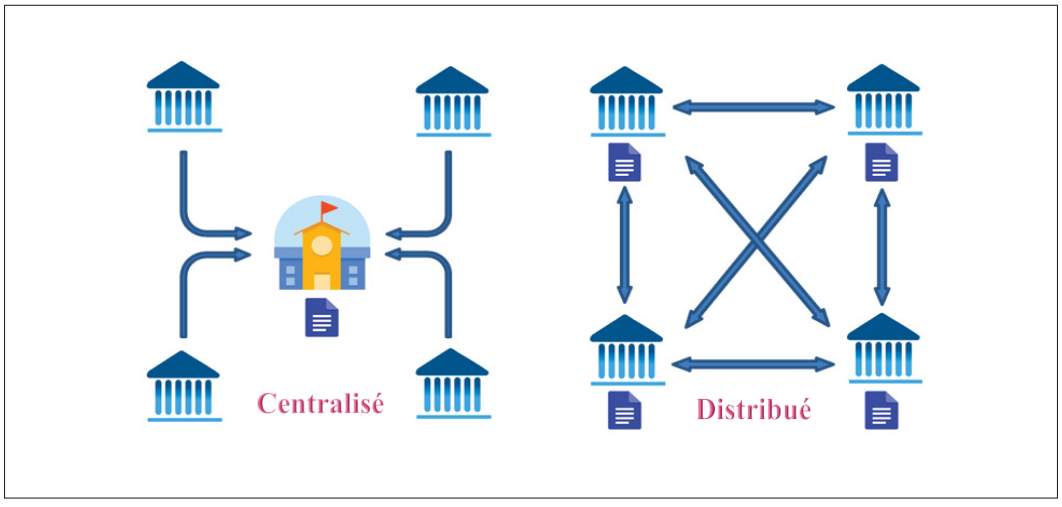

Figure 1.2 Systèmes de registres centralisés et distribués
(Chan & Williams, 2019)

Un registre distribué est décentralisé en vue d'éliminer le besoin d'une autorité centrale ou d'un intermédiaire pour traiter et valider les transactions ou autres types d'échanges de données tel

illustré dans la figure 1.2. Cette technologie fournit un historique vérifiable et auditable de toutes les informations stockées sur l'ensemble du réseau.

1.2.1 Chaînes de blocs

La chaîne de blocs est le type le plus connu de registre distribué. En effet, la chaîne de blocs est une base de données distribuée et sécurisée qui contient un enregistrement permanent et immuable sous forme de blocs liés les uns aux autres et formant une chaîne comme le montre la figure 1.3. Cet enregistrement constitue des transactions effectuées entre les utilisateurs du registre.

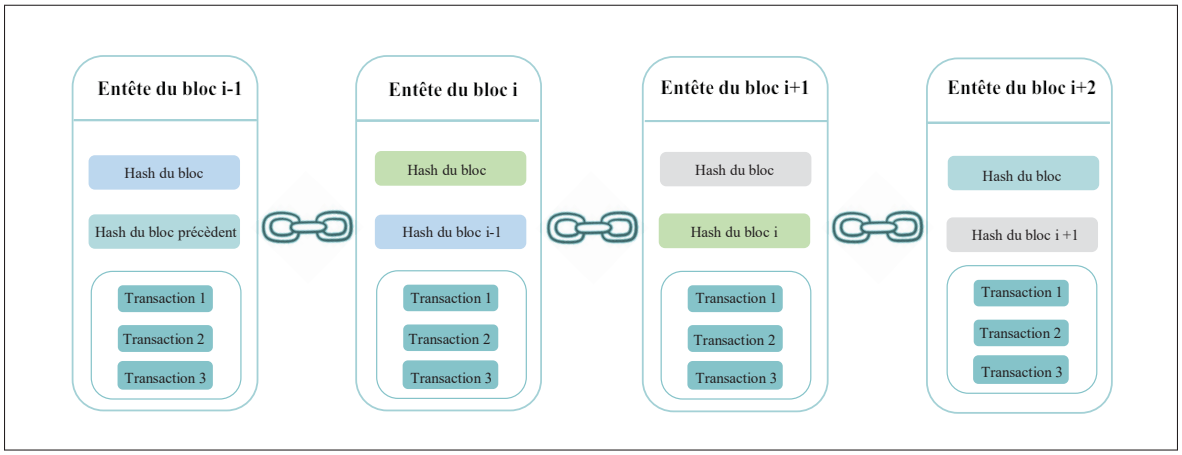

Figure 1.3 Structure des chaînes de blocs

Il existe deux types de chaînes de blocs :

- **Privée :** Une chaîne de blocs privée est définie dans le cadre d'un environnement restreint d'organisations qui veulent échanger des données de manière sécurisée entre les membres des organisations participantes. Seuls ceux de confiances et choisis, ont le droit de valider ou de refuser des blocs selon un ensemble de règles établies au préalable. Les transactions sont privées et ne sont accessibles que par les objets autorisés ;

- **Publique :** Une chaîne de blocs publique est un réseau totalement ouvert, sans autorité de gouvernance et n'importe qui peut le rejoindre. Tout membre du réseau peut lire ou écrire des données ainsi que créer un bloc et l'insérer dans la chaîne à condition qu'il satisfait le protocole de consensus du réseau. Les transactions ajoutées sur la chaîne sont également publiques.

1.2.1.1 Mécanismes de validation de transactions

Le mécanisme vise à assurer la validité des transactions d'un bloc ainsi que les blocs ajoutés sur une chaîne de blocs.

Preuve de travail : C'est un mécanisme de validation de transactions qui consiste à résoudre un défi mathématique complexe ou un problème cryptographique difficile à calculer. La difficulté du défi dépend du temps nécessaire pour valider un bloc. Ce mécanisme assure l'intégrité des transactions et des blocs étant donné que la chaîne de blocs est constituée par des liens cryptographiques et pour modifier un bloc il faut modifier tous les blocs qui le succèdent. Ensuite, il faut calculer une nouvelle preuve de travail pour chacun des blocs. Ce genre d'opération nécessite une énorme puissance de calcul et d'énergie.

Preuve d'enjeu : Dans ce mécanisme, les mineurs doivent fournir la preuve de possession d'une quantité donnée de cryptomonnaie afin de valider et ajouter un bloc à une chaîne. Ce qui signifie que plus le mineur dispose de monnaies, plus la chance de validation augmente. Le principal avantage du mécanisme de preuve d'enjeu c'est qu'il permet d'éviter la dépense élevée d'énergie du mécanisme de la preuve de travail.

1.2.1.2 Exemples de chaînes de blocs

Depuis son introduction, la chaîne de blocs a suscité beaucoup d'attention en tant que système permettant la cryptomonnaie en raison de son faible coût de transaction, de son niveau de sécurité élevé et de la non-nécessité d'une autorité centrale. La technologie de chaîne de blocs va au-delà des cryptomonnaies avec l'introduction des contrats intelligents. Ces contrats sont des codes programmables stockés sur la chaîne de blocs et déclenchés via des appels de fonctions

effectués par les transactions. De ce fait, les plateformes de chaînes de blocs furent utilisées pour créer des applications décentralisées.

Bitcoin : Bitcoin est un système d'échange décentralisé permettant des transactions basé sur la cryptomonnaie sur un réseau de pair à pair. La monnaie numérique décentralisée de cette infrastructure est le bitcoin. Cette chaîne de blocs publique est la première à entrer en opération en 2008 et à provoquer une véritable révolution dans le monde des transactions financières (Nakamoto et al., 2008). Bitcoin est basé sur le mécanisme de consensus *preuve de travail* où toutes les transactions conclues sont répertoriées dans un nouveau bloc puis ajoutées à la chaîne. Les mineurs sont responsables de vérifier puis enregistrer chaque nouveau bloc afin de recevoir une rémunération en bitcoins pour chaque nouveau bloc créé et enregistré selon le protocole de rémunération.

Ethereum : Crée en 2014, Ethereum est une plateforme logicielle décentralisée ouverte qui permet le déploiement de contrats intelligents et des applications décentralisées sans aucune interférence de tiers. Les contrats intelligents d'Ethereum sont écrits dans un langage de programmation de haut niveau qui est ensuite converti en bytecode et exécutés à l'intérieur de la machine virtuelle Ethereum (EVM). Les transactions effectuées dans la chaîne de blocs sont publiques ce qui permet la vérification de la fiabilité de l'exécution du code. En outre, chaque nœud exécute le mécanisme de consensus de preuve de travail afin de vérifier les transactions et synchroniser les mises à jour de l'état de machine. L'équipe d'Ethereum a prévu une transition de la preuve de travail vers une preuve d'enjeu où chaque mineur à son tour peut proposer et voter sur le bloc suivant. Le poids du vote de chacun dépend de l'importance de sa participation en termes de quantité de cryptomonnaies détenues. Par ailleurs, Ethereum dispose de sa propre cryptomonnaie appelée Ether. Le réseau Ethereum est caractérisé par les notions suivantes :

- **Gaz :** La consommation du *gaz* représente le temps de travail nécessaire pour tous calculs programmables, y compris la création du contrat, l'exécution des opérations ainsi que les ressources de stockage associées à l'état du système. Ainsi, il existe une limite de gaz pour

chaque transaction définie par celui qui envoie la transaction, ce qui signifie que la transaction ne doit pas dépasser le nombre maximum de gaz autorisé.

- **Comptes Ethereum :** Le réseau d'Ethereum est basé sur le concept des comptes. Il existe deux types de comptes Ethereum : (1) *Comptes de contrats extraits du code*, et (2) *Comptes contrôlés en externe (ou) Comptes d'utilisateur*. Un compte utilisateur possède un solde d'Ether et une paire de clés publique et privée. Il peut transférer des Ethers ou déclencher l'exécution d'une fonction du smart contrat. De même, un compte de contrat a un solde d'Ether et en plus il possède un code associé.
- **Portefeuilles déterministes hiérarchiques :** ce sont des portefeuilles de cryptomonnaie structurés sous la forme d'une arborescente de clés publiques/privées qui part d'un nœud maître. Chaque nœud possède une clé privée et publique étendue, contrôlée par le nœud maître. Dans Ethereum, chaque adresse représente un compte. L'objectif d'un HD Wallet serait de gérer plusieurs comptes à partir d'une seule clé privée. Cette clé maître est générée à partir d'une phrase seed qui est une phrase avec 12 mots mnémoniques en se basant sur le standard BIP 39 (Palatinus, Rusnak, Voisine & Bowe, 2019). La phrase de départ doit être sauvegardée afin de pouvoir générer les clés et restaurer les données du portefeuille.

1.2.2 Autres registres distribués

IOTA : IOTA est un registre distribué publique qui permet d'enregistrer et exécuter des transactions entre les machines dans un environnement de l'Internet des Objets. IOTA est un réseau pair à pair qui n'utilise pas de chaîne de blocs. Par contre, le réseau est basé sur le système des nœuds Tangle, utilisé pour confirmer les transactions. Tangle est basé sur la structure de données DAG (Graphe Acyclique Dirigé) où chaque objet connecté joue le rôle d'un nœud et participe dans le consensus et la validation des transactions. Chaque nœud doit référencer deux transactions précédentes sur le réseau avant de confirmer une nouvelle transaction avec un puzzle de preuve du travail, facile à résoudre. Cela réduit le temps et la mémoire nécessaires pour confirmer une transaction. Pour comptabiliser les transactions, ce registre utilise une cryptomonnaie appelée mIOTA.

Corda : Corda est une chaîne de blocs d'entreprises privée où l'accès est réservé à certaines parties dont l'identité est connue. Le registre de Corda met l'accent sur la confidentialité des données. En effet, un nœud n'a accès qu'aux données pertinentes pour son activité. En revanche, l'ensemble des transactions est visible par un noeud particulier, appelé notaire, qui est chargé de la validation. Pour traiter une transaction, les nœuds concernés se coordonnent d'abord entre eux en exécutant le contrat et en signant le résultat de l'exécution. Une fois les signatures requises recueillies, le nœud initiateur envoi de la transaction au notaire pour une signature consensuelle. Ainsi, le notaire conserve un historique complet de toutes les transactions qui ont été soumises. Enfin, une fois approuvé, le résultat est engagé par toutes les parties.

1.2.3 Comparaison des registres distribués

Les deux registres distribués Ethereum et Bitcoin sont deux chaînes de blocs publiques basés sur le consensus de preuve de travail. Contrairement à Bitcoin qui dispose d'un langage de script très limité, Ethereum est conçu comme une chaîne de blocs programmables à usage général qui fonctionne sur une machine virtuelle. EVM est capable d'exécuter le code du contrat intelligent qui est arbitrairement complexe. Alors que le langage de script de Bitcoin est contraint à une simple évaluation des conditions de dépense. D'autre part, le temps de blocs dans Ethereum est considérablement réduit à 15 secondes en moyenne par rapport aux 10 minutes pour Bitcoin.

Quant au registre distribué IOTA, il a comme objectif de réaliser un registre distribué d'Internet d'Objets et ne se base pas sur la chaîne de blocs tel Ethereum et Bitcoin. Cependant, étant donné que le réseau est encore récent, IOTA subit plusieurs attaques entraînant des vols de cryptomonnaies mIOTA (Foundation, 2020 ; Heilman et al., 2019). De plus, le registre IOTA disposait d'un coordinateur pour protéger les noeuds contre les activités malveillantes et pour traiter les transactions qui étaient considérées comme point de défaillance unique. IOTA a récemment supprimer le coordinateur (Greve, 2019) ; Cependant, le réseau reste vulnérable à des problèmes de sécurité. En terminant, Corda est un registre privé, utilisé dans un contexte entrepreneurial qui permet la gestion des accords financiers entre deux ou plusieurs parties

identifiables. Toute participation à ce réseau nécessite une approbation et un accord préalable. D'autre part, Corda n'a pas de cryptomonnaie comparée aux autres registres distribués.

1.2.4 Discussion du choix de registre distribué

La valeur des chaînes de blocs provient de ses caractéristiques uniques permettant de créer des registres décentralisés, transparents, immuables et traçables. En effet, la chaîne de blocs publique est copiée et partagée sur l'ensemble du réseau ce qui permet à chaque participant d'accéder aux registres complets des transactions ainsi que les codes des contrats intelligents. En outre, cette duplication de données combinée avec le mécanisme de consensus de validation des transactions assure l'immuabilité et le suivi de toutes les opérations effectuées sur le registre. Effectivement, toute donnée ajoutée est permanente, horodatée et difficile à altérer ou supprimer.

Dans notre travail, nous tirons parti des contrats intelligents pour mettre en œuvre différentes logiques de monétisation des données de l'IdO et nous nous appuyons sur l'immuabilité des chaînes de blocs pour construire une traçabilité robuste de l'échange de données en temps réel. Ainsi, nous avons choisi de travailler sur la plateforme Ethereum qui bénéficie de toutes les propriétés de la technologie de chaîne de blocs afin de développer notre application décentralisée. La machine virtuelle d'Ethereum est Turing-complète qui fonctionne sur le réseau Ethereum et qui permet d'exécuter n'importe quel programme quelque soit le langage de programmation. Nous utilisons le langage de programmation Solidity pour développer nos contrats intelligents qui s'exécutent automatiquement sur l'EVM lorsque les conditions spécifiées sont remplies et sans autorité de confiance. De plus, la notion de comptes Ethereum qui est contrôlé par une clé privée et disposant d'un solde d'Ether nous permet d'identifier les utilisateurs de notre système, d'assurer la non-répudiation et de procéder à la monétisation en nous basant sur la cryptomonnaie. Enfin, nous nous concentrons dans notre évaluation sur la consommation de gaz comme indicateur de performance clé pour évaluer notre système et représenter les surcoûts d'exécution de nos contrats intelligents sur la chaîne de blocs. De ce fait, nous comparons nos trois modèles de monétisation et nous démontrerons qu'il existe un compromis entre le coût du gaz et la précision de la monétisation.

1.3 Les Filtres de Bloom

Dans cette section, nous présentons la technologie de filtre de Bloom afin de mieux comprendre notre solution adaptative (Trace-BF) basé sur ces filtres.
Un filtre de Bloom est une structure de données probabiliste utilisée pour déterminer si un élément est membre d'un ensemble ou non. Un élément est soit *probablement dans l'ensemble de données* ou *certainement pas dans l'ensemble de données*.
Les filtres de Bloom sont adoptés dans de nombreux cas d'utilisation tels que le routage (Quan, Xu, Guan, Zhang & Grieco, 2013), la sécurité du réseau (Geravand & Ahmadi, 2013) et la traçabilité (Dobreva & Albutiu, 2010) en raison de ces deux caractéristiques principales :

- efficacité spatiale : Quelque soit le nombre d'éléments insérés dans un ensemble, le filtre de Bloom utilise un nombre de bits fixe ;
- efficacité temporelle : Le temps utilisé pour ajouter un élément ou rechercher un élément à partir du filtre de Bloom est fixe et indépendant du nombre d'éléments contenus dans ce filtre.

1.3.1 Opérations sur le Filtre de Bloom

La structure des données de base d'un filtre de Bloom est un vecteur de bits initialement mis à zéro. Plusieurs opérations peuvent être appliquées sur un filtre de Blom :

Insertion d'élément dans un filtre de Bloom : Comme le montre la figure 1.4, chaque case du tableau représente un bit. Afin d'ajouter un élément au filtre de Bloom, il suffit de hacher l'élément plusieurs fois. Ensuite, il faut chercher les positions correspondantes aux hachages et enfin, mettre à un les bits de ces positions dans le vecteur binaire.

Chercher un élément dans un filtre de Bloom : Afin de chercher un élément et tester s'il appartient au filtre de Bloom, il faut reproduire les mêmes étapes d'insertion en hachant l'élément avec les mêmes fonctions de hachage, ensuite calculer les positions correspondantes et enfin vérifier si les valeurs obtenues sont définies dans le filtre.

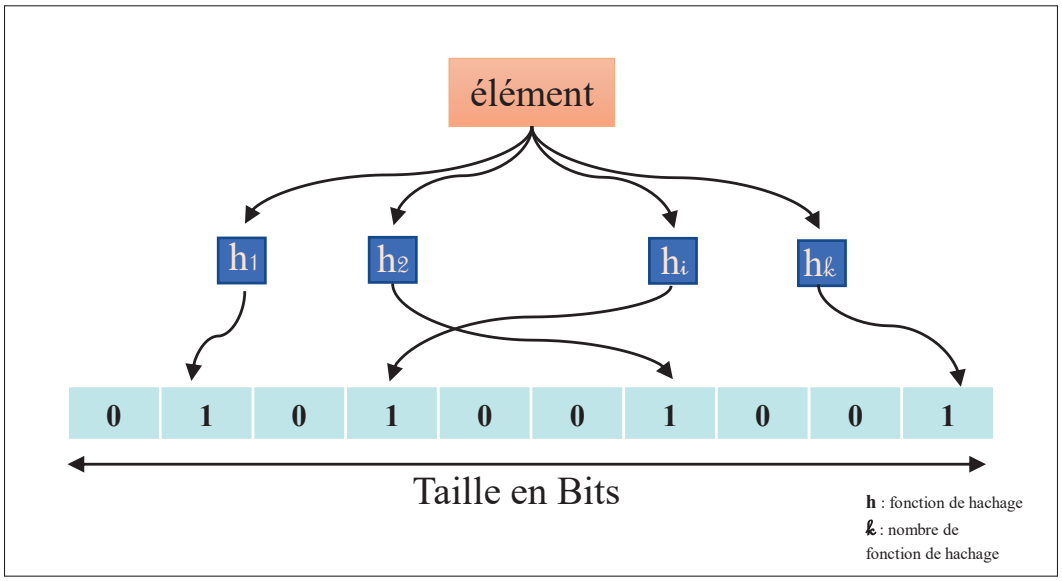

Figure 1.4 Structure du filtre de Bloom

Comparer deux filtres de Bloom : Étant donné deux filtres de Bloom, il est possible de déterminer si deux ensembles sont approximativement similaires en calculant le coefficient de dé (Randall, Ferrante, Boyd, Bauer & Semmens, 2014) :

$$\texttt{Coefficient} = \frac{2 \times \texttt{h}}{\texttt{a} + \texttt{b}} \tag{1.1}$$

Où `h` est le nombre de positions fixées à 1 dans les deux filtres de Bloom, `a` et `b` sont le nombre de positions définies sur 1 dans chaque filtre de Bloom.
La valeur du coefficient doit être comprise entre 0 et 1. Un nombre plus élevé indique une similitude plus élevée entre les deux filtres de Bloom.

1.3.2 Taux de faux positifs et fonctions de hachage

Le taux de faux positifs lors de la recherche d'élément dans un filtre Bloom est influencé par la taille du tableau en nombre de bits et le nombre de fonctions de hachage utilisées pour insérer un élément. L'avantage du filtre de Bloom est que pour une taille de bits fixe, il peut représenter un ensemble avec un nombre élevé d'éléments. Cependant, il faut savoir qu'un filtre de Bloom

plus grand aura moins de faux positifs qu'un filtre plus petit. En outre, afin de diminuer le taux de faux positifs, il faut calculer le nombre de fonctions de hachage à utiliser en se basant sur l'équation 1.2 :

$$k = \frac{m}{n} \times \ln 2 \tag{1.2}$$

Avec m la taille d'un filtre de Bloom en bits et n le nombre d'éléments à insérer.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les principales technologies utilisées dans notre travail. En résumé, nous avons détaillé le fonctionnement des modèles de communications de publications/abonnements et nous avons considéré le protocole MQTT comme protocole de messagerie dans l'échange des données de l'IdO. Ensuite, nous avons introduit la technologie des registres distribués et plus précisément la technologie des chaînes de blocs. Nous avons étudié les différentes plateformes de chaînes de blocs ainsi que les avantages de chacune. Enfin, nous avons expliqué les filtres de Bloom que nous utilisons dans l'une de nos solutions proposées (Trace-BF) en raison de son efficacité de stockage et de sa rentabilité. En effet, afin de détecter les incohérences dans les informations enregistrées sur la chaîne de blocs, nous nous servons de l'équation 1.1 du coefficient de dé pour calculer le degré de similitude entre les différents filtres de Bloom associés aux entités de notre système à savoir les éditeurs, les abonnés et le courtier.

CHAPITRE 2

REVUE DE LITTÉRATURE

Le nombre de capteurs et d'appareils intelligents connectés augmente de façon exponentielle en générant une énorme quantité de flux de données en temps réel. La vente de ces flux de données devient un domaine clé pour de nombreux industries, entreprises et gouvernements. Dans ce chapitre, nous présentons l'ensemble des recherches effectuées dans le domaine de monétisation de données. Tout d'abord, nous décrivons des travaux connexes pour les marchés des données IdO. Deuxièmement, nous nous concentrons sur les travaux liés à la monétisation des données. Enfin, nous présentons des articles discutant des contrats intelligents à des fins de traçabilité.

2.1 Marché de données de l'Internet des Objets

La commercialisation des données IdO est entrain de gagner en popularité durant ces dernières années. Ces marchés de données devraient atteindre 3 billion de dollars d'ici 2030 (Tang, Fujii-Hwang, Colwill, Arora, Shah, Mendizabal & Callejas, 2018). Plusieurs entreprises telles que BDEX (BDEX) et DAWEX (Dawex), offrent des marchés de données spécialisées qui permettent de vendre et d'acheter des flux de données IdO. Ces flux de données permettent aux entreprises de cibler les consommateurs et d'obtenir des statistiques représentant le comportement des utilisateurs dans le but de découvrir les défauts d'un produit spécifique ou de compléter des produits existants. De nombreuses études ont été menées pour concevoir des marchés de données pour l'Internet des Objets. Les auteurs (Bröring, Schmid, Schindhelm, Khelil, Käbisch, Kramer, Le Phuoc, Mitic, Anicic & Teniente, 2017) proposent un écosystème qui comble le problème d'interoperabilité des environnements IdO pour les fournisseurs des données et utilisent cinq plateformes d'interopérabilité offrant un modèle de marché de données indépendant de la plateforme avec une API commune.

Une autre solution proposée par (Mišura & Žagar, 2016) définit un modèle de marché de données où les propriétaires d'appareils enregistrent leurs capteurs avec toutes les informations pertinentes et les consommateurs de données interrogent le système en définissant le besoin et le

budget disponible pour se procurer ces données. Cependant, ces travaux se sont concentrés à élaborer des solutions de stockage de données où la gestion des données ainsi que l'échange ne se font pas en temps réel. Un exemple de marché de données IdO en temps réel est suggéré par I3 (Krishnamachari, Power, Kim & Shahabi, 2018) définissant un site Web unique qui récupère les informations sur tous les vendeurs et leurs produits et les affiche aux acheteurs selon les besoins. Les auteurs utilisent un courtier de publications/abonnements qui autorise des acheteurs à accéder aux données en fonction de leurs paiements. Les flux de données sont mesurés à des fins de facturation et désactivés automatiquement à la fin de la période de paiement. Les propriétaires de données peuvent décider à qui et quand partager leurs données. Cependant, ces marchés de données présentent un point d'échec central et un goulot d'étranglement des interactions qui suscitent un besoin de confiance important entre le serveur, les vendeurs et les consommateurs.

Ainsi, plusieurs recherches focalisent à proposer des alternatifs pour avoir des marchés de données décentralisés en utilisant la technologie de registres distribués. Datapace (Draskovic & Saleh, 2017) est une solution décentralisée de marché de données basées sur le registre distribué privé Hyperledger Fabric. L'application s'appuie sur les contrats intelligents pour définir les conditions dans lesquelles les données doivent être transférées. L'utilisation de ce registre garantit l'intégrité des données échangées. Dans (Özyilmaz *et al.*, 2018), les auteurs proposent IDMoB, une architecture décentralisée et distribuée de marché de données pour l'IdO. Ils utilisent Ethereum, la plateforme de contrat intelligent publique des chaînes de blocs, combinée avec le système de stockage décentralisé "Swarm" pour fournir un système de collaboration incitateur entre les fournisseurs de données IdO et les fournisseurs de solutions d'intelligence artificielle et d'apprentissage intelligent.

Contrairement aux travaux ci-dessus, notre solution est intégrée à un système de publications/abonnements afin de fournir des flux de données en temps réel pertinents pour les consommateurs intéressés, tout en surveillant en permanence les flux d'informations à des fins de monétisation.

2.2 Modèles de monétisation des données

Cette section présente les travaux antérieurs effectués sur les modèles de monétisation des données IdO. Les chercheurs dans (Javaid *et al.*, 2019), présentent une approche de monétisation basée sur la réputation des données IdO. Ils se basent sur le fait qu'un consommateur de données ne fait pas confiance au propriétaire de l'appareil et à la qualité des données générées. Ils proposent alors un modèle dans lequel le propriétaire de données crée un contrat intelligent Ethereum contenant des informations sur les sujets et l'adresse du courtier MQTT. Le client intéressé par ces données fait un dépôt au contrat intelligent pour recevoir les données. À la fin de la session, le client peut évaluer les données et laisser un avis. Une approche similaire est proposée dans (Suliman, Husain, Abououf, Alblooshi & Salah, 2018) qui illustre comment les contrats intelligents peuvent être utilisés automatiquement et monétiser systématiquement les données générées et collectées des appareils IdO. Plus précisément, les auteurs présentent un système exigeant au client d'effectuer un dépôt au contrat et décider de l'utilisation du solde sur les sujets d'abonnements. Dans (Bajoudah, Dong & Missier), les auteurs proposent une approche différente permettant aux parties commerciales de négocier une offre de données et de stocker l'accord dans le contrat intelligent. L'accord contient l'intervalle de temps de diffusion en continu des données, le prix total qui est basé sur le prix unitaire du message, le taux de diffusion en continu des messages et l'intervalle de temps de diffusion. Un accusé de réception est envoyé au contrat intelligent par le client comme confirmation après avoir reçu une quantité bien déterminée de données afin de minimiser les possibilités de fraude du client.

Ces recherches se concentrent sur l'utilisation d'un mécanisme de réputation pour évaluer la qualité des données reçues sur le marché. De plus, le point commun de ces travaux et de la plupart des modèles de monétisation des données de l'IdO est la nécessité de faire un dépôt comme garantie de paiement. Cela n'est pas préféré par les clients pour des raisons de confiance. Dans notre travail, nous n'envisageons pas d'utiliser un mécanisme de réputation et nous proposons un modèle de paiement périodique avec une facturation précise basée sur les données de Pub/Sub enregistrées sur la chaîne de blocs.

2.3 Traçabilité à l'aide de contrats intelligents

En raison de leur nature immuable, les chaînes de blocs sont largement utilisées pour assurer le suivi des opérations à des fins de traçabilité et d'audit. Plusieurs ouvrages discutent de leurs utilisations, notamment dans le domaine de la traçabilité de la chaîne d'approvisionnement (Behnke & Janssen, 2019)(Lin, Shen, Zhang & Chai, 2018). Dans (Dasaklis, Casino & Patsakis, 2019), les auteurs proposent une approche consistant à faire varier la granularité de la traçabilité en fonction des caractéristiques du produit, des processus de la chaîne d'approvisionnement et de l'engagement des parties prenantes. Une étude dans (Küsters, Rausch & Simon) montre que l'identification des parties responsables en cas de violation est importante afin d'avoir des garanties de sécurité raisonnables pour une chaîne de blocs. En effet, les auteurs mettent en avant le concept de responsabilité comme propriété autonome de sécurité. Si une propriété de sécurité, telle que la cohérence, n'est pas satisfaite, alors les parties qui se sont mal comportées peuvent être identifiées et en être tenues responsables. Ce travail s'intéresse aux contrats intelligents de la plateforme privée des chaînes de blocs Hyperledger où toutes les parties se connaissent et donc un mécanisme de gestion de responsabilité incite toutes les parties à se comporter honnêtement.

Dans notre recherche, nous mettons au point un mécanisme d'identification du responsable dans le cas d'incohérence des informations en considérant différents niveaux de granularité de traçabilité à l'aide des contrats intelligents d'Ethereum pour permettre une monétisation fidèle du marché des données de l'Internet des Objets.

2.4 Systèmes de publications et d'abonnements

Le système de Pub/Sub est un paradigme de communication intéressant pour les applications distribuées à grande échelle. Pub/Sub permet une diffusion d'informations basée sur des sujets spécifiques. Ainsi, le système est caractérisé par le découplage de la production et la consommation d'informations où le courtier est le seul responsable de communiquer avec différentes entités qui sont des éditeurs et des abonnés. Cependant, ce type de communications

centré sur le contenu suscite des préoccupations majeures en matière de confidentialité puisque le système Pub/Sub est naturellement utilisé dans des environnements centralisés et non fiables. Cela implique que les systèmes de Pub/Sub doivent être dotés des moyens adéquats pour protéger les données échangées, préserver leur comportement correct et faire face à d'éventuels scénarios d'attaque. L'étude de (Esposito & Ciampi, 2014) analyse les principaux défis de sécurité dans les sytèmes Pub/Sub. Selon (Onica, Felber, Mercier & Rivière, 2016), il existe plusieurs approches de préservation de la confidentialité pour les Pubs/Subs. Ces approches s'appuient sur des techniques de cryptographie sophistiquées telles que la cryptographie homomorphe. L'architecture centralisée de Pub/sub rend son modèle de messagerie susceptible de défaillances byzantines. Diverses études motivent le besoin d'avoir un courtier en Pub/Sub distribués avec une tolérance aux pannes byzantines. Les auteurs de (Chang & Meling, 2012) décrivent en détail les comportements byzantins des éditeurs, des abonnés et des courtiers et proposent une infrastructure informatique en nuage qui assure la sécurité du système en excluant les nœuds défectueux byzantins. Dans (Ramachandran, Wright, Zheng, Navaney, Naveed, Krishnamachari & Dhaliwal, 2019), la solution offre un nouveau courtier de publication et d'abonnement distribué avec une tolérance aux fautes byzantine et une immuabilité basée sur la chaîne de blocs. Trinity distribue les données publiées à l'un des courtiers du réseau à tous les courtiers du réseau et stocke les données dans un registre immuable grâce à la technologie de la chaîne de blocs. Les capacités de tolérance aux pannes de Trinity proviennent des nœuds de consensus qui exécutent un algorithme byzantin de consensus de tolérance aux pannes. Une autre tentative dans (Zupan, Zhang & Jacobsen, 2017) pour combiner le système de messagerie basée sur les évènements et les chaînes de blocs a été proposée en utilisant Hyperledger. Les auteurs proposent HyperPubSub, un service de publications/abonnements décentralisé pour un environnement multi-fédéré et autorisé utilisant des chaînes de blocs, qui fournit une messagerie sécurisée et préservant la vie privée.

2.5 Conclusion

Dans ce chapitre, nous avons étudié plusieurs travaux connexes liés aux marchés des données de l'Internet des Objets et la monétisation de ces données. Ensuite, nous avons introduit la notion de granularité de traçabilité en utilisant les contrats intelligents des chaînes de blocs. Finalement, nous avons présenté les principaux défis dans les systèmes de messagerie Pub/Sub. Contrairement aux travaux de recherches étudiés, nous nous concentrons plutôt sur le suivi des activités de Pub/Sub et les enregistrer sur la chaîne de blocs. Nous utilisons un contrat intelligent d'Ethereum pour vérifier la cohérence des traces collectées. Ainsi, nous proposons trois solutions de traçabilité et nous démontrons comment un contrat intelligent peut exploiter les informations enregistrées sur la chaîne de blocs pour calculer les résultats de la monétisation avec divers degrés de précision.

CHAPITRE 3

ARCHITECTURE ET SOLUTIONS PROPOSÉES

Dans ce chapitre, nous introduisons notre système de partage et de monétisation de données de l'Internet des Objets en temps réel. Nous utilisons le protocole MQTT combiné avec les contrats intelligents Ethereum. Tout d'abord, nous mettons en évidence les principales caractéristiques et composants de notre système. Ensuite, nous décrivons nos trois solutions proposées avec les différents niveaux de traçabilité pour répondre aux différentes exigences d'utilisation. Enfin, nous présentons une comparaison et une discussion de nos trois solutions.

3.1 Architecture du système

Notre système de monétisation utilise la plateforme de chaîne de blocs publique Ethereum à laquelle tout utilisateur peut accéder et effectuer des paiements en cryptomonnaies. Nous tirons parti de l'immuabilité des contrats intelligents pour fournir un environnement d'échange de données IdO transparent et éliminer la confiance à une tierce partie accordée au partage de données, au processus de comptabilité ainsi qu'aux paiements tout en garantissant la détection des fraudes. En plus du contrat intelligent, notre système comprend les entités participantes à l'échange de données suivantes :

Courtier : Un composant MQTT qui gère les connexions entre les éditeurs et les abonnés et achemine les messages reçus vers les destinations correspondantes. Dans notre contexte, le courtier est une entité non fiable qui doit soumettre régulièrement des informations de traçabilité à la chaîne de blocs ;

Éditeur : Un appareil IdO (p. ex. un capteur) qui génère et partage en continu des données sur un sujet spécifique, par le biais du courtier en utilisant le protocole de publications/abonnements. Chaque éditeur est payé selon le schéma de monétisation décrit dans la section suivante ;

Abonné : Un consommateur de données qui s'abonne à des sujets qu'il l'intéresse et reçoit les flux de données publiées sur MQTT. L'abonné paye avec la cryptomonnaie en fonction de la quantité de données reçues et de la durée d'abonnement active enregistrées sur la chaîne de

blocs.

Chaque utilisateur doit détenir une adresse Ethereum afin d'identifier son compte dans la chaîne de blocs Ethereum et interagir avec le smart contrat.

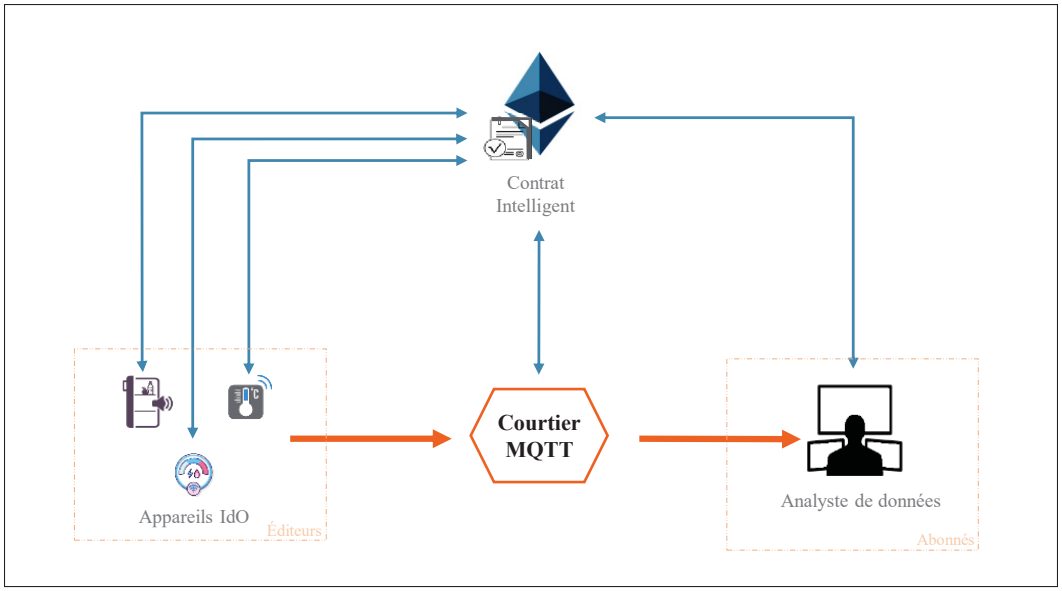

Figure 3.1 Architecture du système

Dans la figure 3.1 ci-dessus, nous illustrons l'architecture de notre système à l'aide d'un cas d'utilisation simple d'une maison intelligente : un propriétaire de maison intelligente souhaite utiliser notre système pour vendre les données générées en temps réel par ses appareils connectés. Un analyste de données désire traiter des données en temps réel dans le but d'étudier le comportement des utilisateurs et améliorer certains produits pour les maisons intelligentes. Tel le montre l'architecture du système, l'analyste de données correspond à l'abonné MQTT, les appareils IoT agissent en tant qu'éditeurs MQTT et le courtier diffuse ces données en respectant le protocole. Tous les composants interagissent avec le contrat intelligent déployé sur la chaîne de blocs Ethereum afin d'enregistrer l'activité de partage de données en fonction du modèle de traçabilité.

3.2 Processus d'enregistrement

Notre système comporte deux autres entités ayant un rôle important à l'initiation du processus de monétisation :

– un *gestionnaire de système*, qui détient le contrat intelligent et confirme l'enregistrement des appareils intelligents ;
– un *propriétaire des appareils intelligents*, qui est responsable d'attribuer les adresses Ethereum à ses objets et de les inscrire sur le système de monétisation.

Chaque éditeur ou abonné doit être préalablement enregistré au contrat intelligent avec son adresse Ethereum et son rôle ainsi que les sujets sur lesquels il agit en tant qu'un éditer ou abonné. Afin de gérer plusieurs comptes Ethereum, nous implémentons un mécanisme de génération de portefeuilles HD où nous offrons la possibilité d'affecter plusieurs adresses à des appareils intelligents appartenant à un même utilisateur. De ce fait, l'utilisateur sera capable de gérer et récupérer les informations de toutes les adresses de ces appareils avec une seule clé maître. De plus, le portefeuille HD offre un anonymat plus élevé étant donné qu'un utilisateur peut générer une nouvelle combinaison de clé publique/privé pour garder son activité globale anonyme.
En outre, nous procédons à hacher l'adresse Ethereum et écrire uniquement son hachage sur la chaîne de blocs comme identifiant du client afin de préserver l'identité du client. De là, une entité enregistrée peut initier une demande de connexion au contrat intelligent en envoyant un message signé avec la clé privée associée à son compte Ethereum. Le contrat intelligent vérifie son identité et notifie le courtier de la portée du client (c'est à dire les sujets sur lesquels il peut publier ou s'abonner). Ainsi, un éditeur ou un abonné est autorisé à communiquer avec le courtier MQTT et à commencer l'échange des données.

3.3 Schéma de monétisation

Notre système se base sur la chaîne de blocs pour garder les informations nécessaires à la comptabilisation des échanges de données provenant du protocole MQTT. Nous définissons dans notre contrat intelligent des tarifications pour tous les éditeurs et tous les abonnés en fonction du

nombre de messages envoyés et reçus respectivement et du volume de données envoyées ou reçues. Notre schéma de monétisation proposé est défini dans les équations 3.1 et 3.2. Nous calculons la facture d'un abonné et le revenu total d'un éditeur comme suit :

$$\texttt{Facture}_{\texttt{Abonne}} = \texttt{f}_1 \times \texttt{t} + \texttt{f}_2 \times \texttt{Nombre}_{\texttt{Publications}} + \texttt{f}_3 \times \texttt{Bytes}_{\texttt{Recus}} + \texttt{f}_4 \tag{3.1}$$

$$\texttt{Revenu}_{\texttt{Editeur}} = \texttt{f}'_1 \times \texttt{t} + \texttt{f}'_2 \times \texttt{Nombre}_{\texttt{Publications}} + \texttt{f}'_3 \times \texttt{Bytes}_{\texttt{Envoyes}} + \texttt{f}'_4 \tag{3.2}$$

Avec $Nombre_{Publications}$ est le nombre de publications reçues pour l'abonné et envoyées pour l'éditeur, $Bytes_{Recus}$ correspond au volume total d'octets reçus, $Bytes_{Envoyes}$ est le volume total d'octets envoyés, f_i représente les frais par unité et f'_i représente le revenu par unité ($i \in 1, 2, 3$). f_4 et f'_4 sont des frais fixes facturés pour l'inscription et les frais de vérification.
Ces frais sont fixés par l'opérateur du marché afin d'atteindre un objectif de profit cible basé sur l'équation 3.3 suivante :

$$\texttt{Profit} = \sum \texttt{Facture}_{\texttt{Abonne}} - \sum \texttt{Revenu}_{\texttt{Editeur}} - \sum \texttt{Gaz}_{\texttt{Utilise}} \tag{3.3}$$

Nous procédons à une monétisation périodique où le contrat intelligent calcule automatiquement les factures et les paies de chaque client. Ensuite, il informe chaque abonné par leurs factures. Par conséquent, les abonnées doivent déposer le montant requis sur le contrat en envoyant de la cryptomonnaie native d'Ethereum c'est-à-dire le Ether. Un autre évènement est déclenché contenant l'identifiant de chaque éditeur et ses revenus en vue d'initier le transfert de la valeur spécifiée. Enfin, suite à chaque virement ou dépôt, la facture du client correspondant est réinitialisée. En dernier lieu, les coûts d'enregistrement des publications, des livraisons et de la vérification varient selon le modèle de traçabilité choisie. Ainsi, chaque solution a ses propres prix et le client a le choix d'adopter le modèle qui répond à ses besoins et qui est adéquat en termes de frais essentiels et de traçabilité.

3.4 Modèles de traçabilité proposés

Dans cette section, nous présentons en détail les différents mécanismes proposés pour tracer et monétiser les activités de Pub/Sub sur la chaîne de blocs. En effet, afin de répondre à différents besoins en termes de traçabilité et de coûts, nous fournissons trois solutions : Trace-MAX, Trace-MIN et Trace-BF. Il est à noter que chaque solution dispose de son contrat intelligent qui définit le protocole d'échange, le protocole de vérification ainsi que le protocole de monétisation.

3.4.1 Trace-MAX : Traçabilité maximale

Trace-MAX assure une traçabilité maximale de tout le processus de transfert des données. Tous les participants au marché de données IdO devront écrire des informations détaillées de tout message échangé avec le protocole MQTT sur la chaîne de blocs. Une fois écrites, les informations sont immuables et les participants ne peuvent pas les modifier. Dans ce modèle, le contrat intelligent définit un ensemble de structures de données pour chaque éditeur, abonné et courtier afin de sauvegarder les informations liées aux publications et livraisons effectuées tel qu'il est illustré dans les tableaux 3.1 et 3.2. En effet, nous considérons les structures Client et Courtier suivantes :

Tableau 3.1 Structure de données Client dans Trace-MAX.

Structure Client	Description
Id_{client}	Identifiant unique du client qui est le hachage de son adresse Ethereum
$\mathrm{T}_{connexion}$	Temps à partir duquel le client a commencé à utiliser le système
$\mathrm{T}_{deconnexion}$	Temps à partir duquel le client a arrêté d'utiliser le système
Sujet->$\mathrm{Messages}_{Envoyes}$	Un mappage qui fait la correspondance entre chaque sujet et les messages envoyés sur ce sujet
Sujet->$\mathrm{Messages}_{Recus}$	Un mappage qui fait la correspondance entre chaque sujet et les messages reçus sur ce sujet

Dans Trace-MAX, chaque message échangé par le protocole MQTT est sauvegardé sous la forme décrite dans le tableau 3.3 ci-dessous.

Tableau 3.2 Structure de données Courtier dans Trace-MAX.

Structure Courtier	**Description**
$\text{Adresse}_{Courtier}$	Adresse du compte Ethereum du courtier
$\text{Max}_{ClientsConnectes}$	Nombre de clients connectés sur MQTT
$\text{Max}_{PublicationsRecus}$	Nombre de publications reçus sur MQTT
Id->Client	Un mappage qui fait la correspondance entre l'identifiant du client et la structure Client

Tableau 3.3 Structure de données Message dans Trace-MAX.

Structure Message	**Description**
$\text{Id}_{Message}$	Identifiant du Message défini par le protocole MQTT
Sujet	Nom du sujet sur lequel le message est envoyé/reçu
$\text{Hachage}_{donnees}$	Hachage des données envoyées/reçues sur le message
$\text{Taille}_{donnees}$	Taille des données envoyées/reçues sur le message en octets
$\text{Horodatage}_{blocs}$	Temps de création du bloc actuel horodaté en secondes

3.4.1.1 Protocole d'échange de données

Quand un éditeur publie un message sur un sujet spécifique par le biais du courtier MQTT, il écrit cette publication sur la chaîne de blocs. En appelant le contrat intelligent, l'éditeur envoie une transaction avec les informations du message nécessaire tel indiqué dans le tableau 3.3. Dès que le courtier reçoit la publication, il enregistre sur la chaîne de blocs que ce client a envoyé tel message et transmet la publication aux abonnés inscrits sur le sujet. Lorsque la publication est acheminée vers les abonnés au sujet, chaque abonné enregistre la publication reçue sur la chaîne de blocs et accuse réception de la publication au courtier. Ce dernier stocke cette livraison à son tour sur la chaîne de blocs avec les mêmes informations de journalisation écrites par les abonnées comme illustrée dans le diagramme 3.2 ci-dessous. En se basant sur cette logique, chaque étape est tracée sur la chaîne de blocs par chaque client ainsi que par le courtier pour chaque client et pour toute publication et toute livraison. Ce protocole d'échange garantit la fiabilité de la vérification et l'exactitude de la comptabilité.

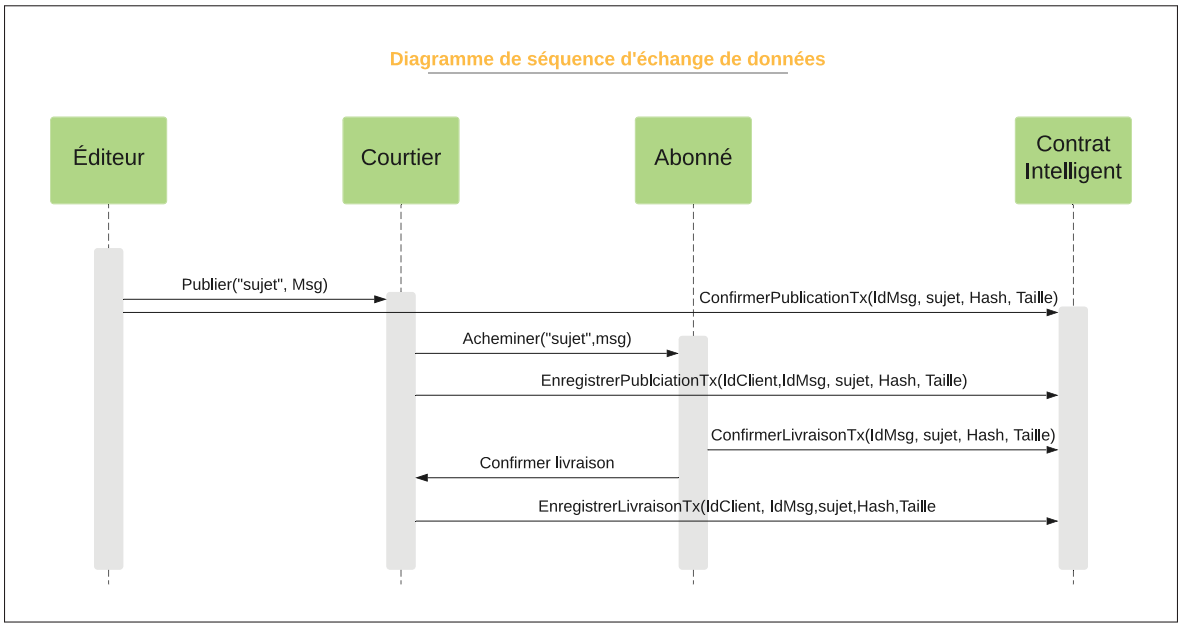

Figure 3.2 Diagramme de séquence du protocole d'échange de données Trace-MAX

3.4.1.2 Processus de vérification des données

Notre approche se base sur la collecte du maximum d'informations possible et l'enregistrer dans la chaîne de blocs afin de vérifier le processus d'échange et atténuer les risques de répudiation et de fraude. Ensuite, le système effectue périodiquement une vérification complète dans le but d'obtenir une majorité de deux sur trois par rapport à la cohérence des enregistrements. À cet effet, nous lançons un processus de vérification mensuel de la liste des clients enregistrés sur la chaîne de blocs où le contrat intelligent effectue un audit exhaustif pour chaque éditeur et chaque abonné. Le contrat intelligent vérifie la cohérence de ce que le client a écrit avec ce que le courtier a écrit, publication par publication, et livraison par livraison. Deux situations peuvent se produire. Prenons le cas de vérification des enregistrements d'un éditeur tel indiqué dans la figure 3.3 :

– si l'information enregistrée sur la chaîne de blocs est cohérente, le contrat intelligent comptabilise la publication pour la facturation ;

– dans le cas où le système détecte des informations incohérentes pour une certaine publication dans un sujet spécifique, le contrat intelligent effectue une deuxième vérification via les ressources enregistrées sur les livraisons du même sujet. Nous cherchons à déterminer si un des enregistrements des abonnés coïncide avec l'information incohérente trouvée ; c'est à dire ayant le même hachage de données, la même taille et dans l'intervalle de temps approprié.

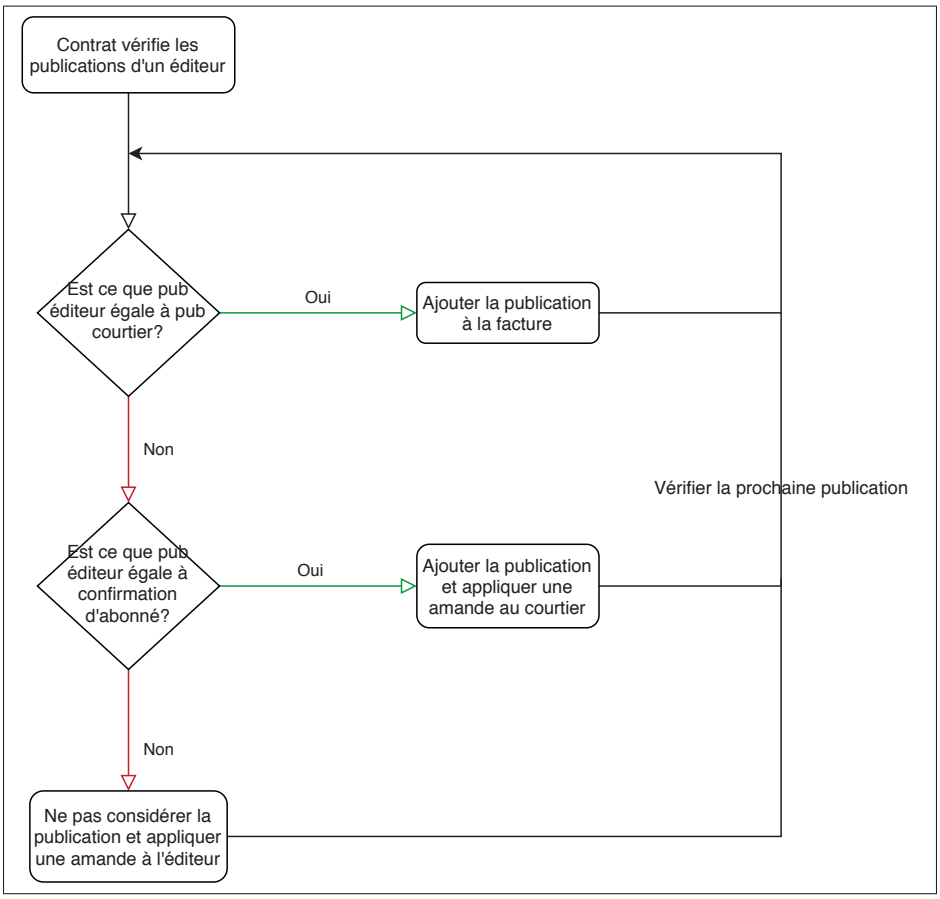

Figure 3.3 Diagramme de flux de vérification Trace-MAX

De même, le contrat intelligent effectue la vérification complète pour chaque abonné. Il compare toutes les confirmations envoyées par l'abonné à celles enregistrées par le courtier. En cas d'informations incohérentes pour certaines livraisons, une deuxième vérification est nécessaire pour identifier qui est responsable de ces enregistrements incorrects. Ainsi, le contrat cherche à trouver la publication correspondante enregistrée par un éditeur.

3.4.1.3 Protocole de monétisation des données

Notre système de marché de données IdO vise à permettre aux producteurs de données de gagner des revenus grâce aux données produites par leurs objets connectés. Pour ces motifs, notre système fournit une monétisation périodique, effectuée automatiquement suite à la fin du processus de vérification. Dans notre modèle Trace-MAX, nous offrons une monétisation précise et fiable grâce aux informations enregistrées dans la chaîne de blocs et au processus de vérification qui valide chacune de ces informations. Notre protocole se base sur l'équation 3.2 pour calculer le revenu de l'éditeur en fonction de la durée d'activité, et le nombre et le volume des messages envoyés. Ces informations sont comptabilisées lors de la vérification de chaque message. De même, le contrat calcule le montant que chaque abonné doit payer. En outre, notre contrat intelligent est capable de détecter les entités impliquées dans l'incohérence des informations. Cette incohérence coûte au système des opérations de vérification approfondies supplémentaires. De ce fait, une amende est infligée à l'entité fautive afin d'éviter l'écriture de données invalides dans la chaîne de blocs. Selon notre protocole de monétisation, l'entité responsable d'incohérence doit payer les frais de vérification pour chaque fausse information inscrite.

3.4.2 Trace-MIN : Traçabilité minimale

Nous proposons un deuxième modèle de monétisation qui est utilisé dans les cas où les clients ont besoin d'un marché de données offrant un minimum de traçabilité avec le moindre coût. Dans ce cas, la seule entité responsable d'écrire des informations sur la chaîne de blocs est le courtier. Les informations enregistrées sont également minimes et suffisantes afin de vérifier et monétiser l'échange de données. Dans ce modèle, notre contrat intelligent comprend une trace organisée par sujet. En effet, nous considérons la structure de données courtier décrite dans tableau 3.4. Nous détaillons aussi dans le tableau 3.5 les informations enregistrées par le courtier sur chaque sujet.

Tableau 3.4 Structure de données Courtier dans Trace-MIN.

Structure Courtier	Description
$\text{Adresse}_{Courtier}$	Adresse du compte Ethereum du courtier
Id_{Client}->Durée	Un mappage qui fait la correspondance entre l'identifiant du client et la durée d'utilisation du système
Nom_{Sujet} -> Info_{Sujet}	Un mappage qui fait la correspondance entre chaque sujet et les informations liés au publications et livraisons sur ce sujet (Tableau 3.5)
$\text{Max}_{PublicationsRecus}$	Nombre de publications reçus sur MQTT

Tableau 3.5 Structure de données Sujet dans Trace-MIN.

Structure sujet	Description
Id_{client}->OctetsRecus	Un mappage qui fait la correspondance entre l'identifiant de chaque client et le volume de données reçus sur le sujet
Id_{client}->OctetsEnvoyes	Un mappage qui fait la correspondance entre l'identifiant de chaque client et le volume de données envoyés sur le sujet
$\text{Volume}_{Publies}$	Volume de données total publiés sur le sujet

3.4.2.1 Protocole d'échange de données

Lorsque le courtier reçoit une nouvelle publication d'un éditeur sur un certain sujet, il envoie une transaction au contrat intelligent afin d'enregistrer l'identifiant de l'éditeur et la taille des données qu'il a publiées sur MQTT. Le contrat intelligent ajoute cet identifiant et la valeur correspondante du volume de données dans la structure de données du sujet tel expliqué ci-dessus. De plus, il augmente le nombre total d'octets publiés sur ce sujet ($\text{Volume}_{Publies}$). Par ailleurs, lorsque l'abonné reçoit la publication, il envoie une confirmation de livraison de données au courtier. Ainsi, le courtier enregistre l'identifiant de l'abonné et la taille des données reçues à la chaîne de blocs. Le contrat intelligent additionne la taille des données reçues au nombre d'octets reçus par l'abonné à ce sujet.

3.4.2.2 Processus de vérification des données

Dans ce modèle, nous envisageons également une vérification périodique. En se basant sur les informations stockées sur la chaîne de blocs, le contrat intelligent calcule la somme du volume

total des octets reçus par un abonné spécifique pour chaque sujet et compare cette somme à la somme du volume total de données publiées sur chaque sujet. Puisque nous effectuons la vérification dans un intervalle de temps capturé, le nombre total d'octets reçus par un abonné doit être égal à ce qui a été publié sur ce sujet au même intervalle de temps. Nous pouvons distinguer trois situations possibles :

- si le volume total de données reçues calculé dans le processus de vérification est inférieur à ce qui a été publié. Cela signifie que l'abonné a envoyé moins de confirmations de livraison par rapport à ce qu'il a reçu, et donc le courtier a effectué moins d'écriture sur la chaîne de blocs. Cela est déduit du fait que le courtier n'a aucun intérêt à rédiger moins d'informations car cela implique que l'abonné paye moins ;
- dans le cas où le nombre total d'octets reçus est supérieur à ce qui a été publié, le courtier peut être responsable de l'écriture de plus d'informations que ce qu'un abonné a réellement confirmé ;
- dans le cas où le volume total de données reçues est égal à ce qui a été publié, cela signifie qu'il y a une chance que tout soit enregistré correctement, mais cela peut également indiquer que le courtier écrit malhonnêtement plus de publications et de livraisons. Le système ne peut pas détecter ce comportement et il s'agit d'une limitation du modèle Trace-MIN.

3.4.2.3 Protocole de monétisation des données

Afin de monétiser l'échange des données et diminuer les coûts de calculs, nous avons défini un système de monétisation automatisé qui comptabilise les factures de chaque client lors de l'enregistrement des informations de l'échange sur la chaîne de blocs. En effet, nous mettons à jours les données de facturation au fur à mesure qu'on calcule les volumes reçus/envoyés par un client sur un sujet spécifique. Ainsi, une fois la vérification est terminée, nous procédons à informer chaque abonné du montant à payer et à chaque éditeur du montant qu'il va recevoir. Dans le modèle Trace-MIN, le contrat intelligent est capable de détecter qu'une incohérence s'est produite et suite à cette détection, il avise le problème d'incohérence repéré à toutes les parties concernées. De plus, lors de la facturation, le contrat prévient chaque abonné des données

calculées lors de la vérification à savoir, le volume de données publiées sur ses sujets, le volume de données reçues et le montant facturé.

3.4.3 Trace-BF : Traçabilité basée sur le filtre de Bloom

Dans ce modèle, nous proposons une solution d'échange de données qui combine la robustesse de la traçabilité, la fiabilité de la vérification et la modération du coût. Notre approche dans ce modèle consiste à assembler les avantages des deux modèles Trace-MAX et Trace-MIN en ayant plus d'informations sur la chaîne de blocs comme dans le modèle Trace-MAX et moins d'opérations de vérifications telle la solution Trace-MIN. A cet effet, nous nous servons de l'efficacité en espace et en temps des filtres de Bloom pour maintenir les hachages des données partagées et effectuer la vérification sur ces filtres avec moins d'opérations sur la chaîne de blocs.

Adaptation des filtres de Bloom : Afin d'utiliser les filtres de Bloom dans notre système, nous proposons notre propre implémentation avec le langage Solidity étant donné que les implémentations existantes des filtres Bloom supportent une taille maximale de 256 bits. Cette taille permet d'insérer jusqu'à 27 éléments avec une probabilité de faux positifs de 10^{-2} (Hurst). Cependant, cette configuration n'est pas adaptée à un environnement IdO qui génère une énorme quantité de données (par exemple, dix-mille évènements par jour). Afin de prendre en charge 10000 éléments avec une probabilité de faux positif de 10^{-2}, une implémentation de filtre de Bloom doit être composée de 96000 bits et doit utiliser un taux de hachage de 7. À cet effet, nous proposons une nouvelle conception du filtre de Bloom sous la forme d'un tableau de bytes32 de taille 375 ce qui correspond à 96000 bits.

Comme le montre la figure 3.4, l'élément que nous insérons dans un filtre de Bloom est représenté par la concaténation d'un message et de son sujet. L'élément est ensuite haché à l'aide de la fonction sha3. Enfin, nous déterminons la position en calculant le modulo 96000 du hachage. Cette opération est exécutée 7 fois pour finalement extraire 7 positions pour insérer un élément dans le filtre de Bloom.

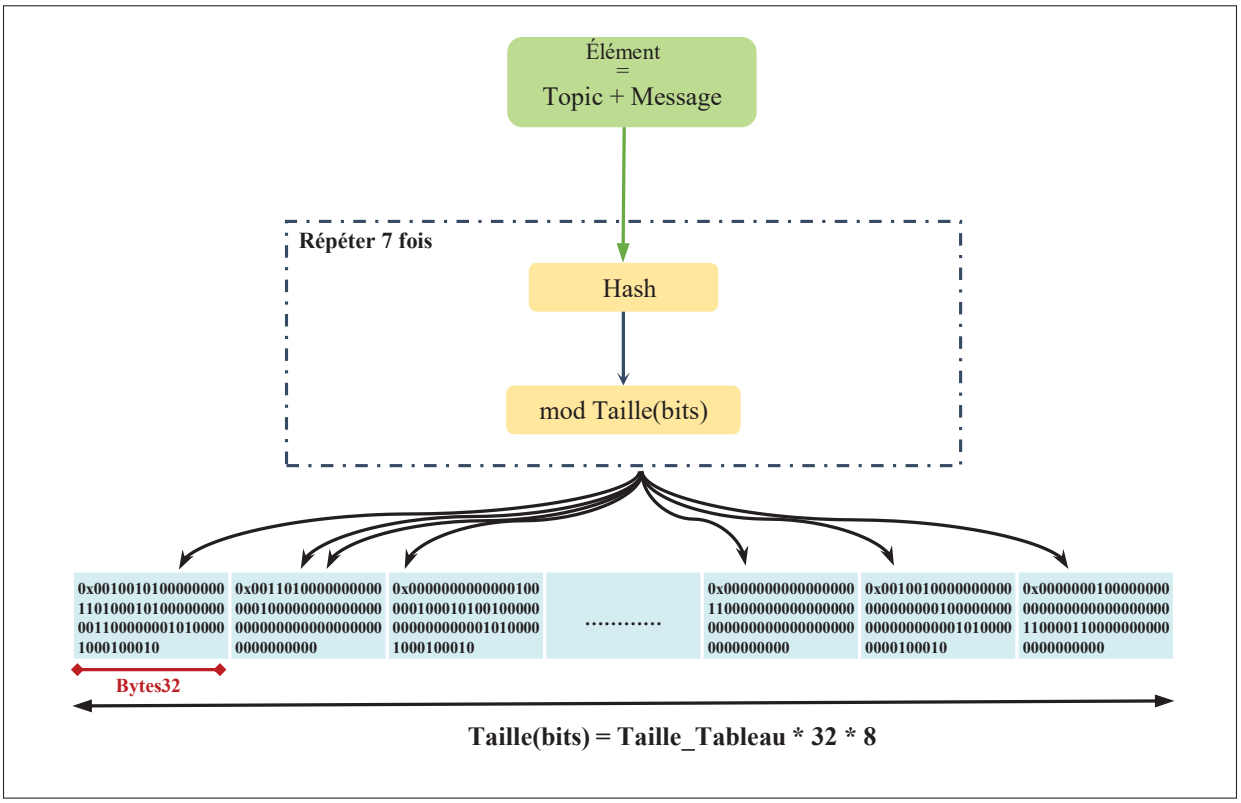

Figure 3.4 Filtre de Bloom proposée

Initialisation des filtres de Bloom : Lors du déploiement du contrat intelligent, nous initialisons trois filtres de Bloom de même taille : un filtre de Bloom pour les éditeurs afin d'écrire toutes leurs publications, un filtre pour le courtier pour garder une trace des publications envoyées, et un filtre aux abonnés pour insérer les confirmations de réception des publications. Tous les bits des filtres de Bloom sont initialisés à zéro. Le contrat intelligent définit une structure de données qui sauvegarde les paramètres d'un filtre Bloom comme le montre le tableau 3.6 :

Tableau 3.6 Structure de données Bloom dans Trace-BF.

Structure Bloom	**Description**
Nombre_{Hash}	Le nombre de fonction de hachage
$\text{Max}_{Elements}$	Le nombre d'élément maximale à insérer
$\text{BF}_{NombreElements}$	Le nombre d'éléments insérés dans le filtre de Bloom
Taille_{Bits}	La taille du filtre de Bloom en Bits
Taille_{Octets}	La taille du filtre de Bloom en Octets

Le contrat intelligent définit également une structure de données Courtier qui contient les éléments suivants (tableau 3.7).

Tableau 3.7 Structure de données Courtier dans Trace-BF.

Structure Courtier	Description
$\text{Adresse}_{Courtier}$	Adresse du compte Ethereum du courtier
Id_{Client}-> $\text{Nombre}_{PublicationRecues}$	Un mappage qui fait la correspondance entre l'identifiant de l'abonnée et le nombre de publication reçues
Id_{Client}-> $\text{Nombre}_{PublicationEnvoyees}$	Un mappage qui fait la correspondance entre l'identifiant de l'abonnée et le nombre de publication envoyées
$\text{Max}_{PublicationsRecus}$	Nombre de publications reçus sur MQTT

3.4.3.1 Protocole d'échange de données

Toutes les publications et les livraisons effectuées sur MQTT sont enregistrées sur la chaîne de blocs par tous les éditeurs et tous les abonnées ainsi que le courtier. Afin d'insérer chaque publication et livraison aux filtres de Bloom correspondant, nous considérons les étapes suivantes :

- la première étape consiste à hacher le nom du sujet concaténé avec les données ;
- ensuite, le hachage résultant est inséré dans chacune des k fonctions de hachage. Le modulo de ces hachages est calculé par rapport à la taille des bits du filtre de Bloom et le résultat correspond à k positions dans cet ensemble de bits ;
- notre filtre Bloom est un tableau de Bytes32. Par conséquent, nous devons trouver, à partir des positions calculées, la position correspondante dans les Bytes32 ainsi que la case dans le tableau. Pour ce faire, la position dans le tableau est déduite de la position calculée dans la deuxième étape. Nous divisons les k positions trouvées par 256 pour trouver la case du tableau qu'on doit modifier. Puis, la position du bit dans les Bytes32 de cette case correspond au reste de la division ;
- une fois calculés, les positions dans le tableau et dans les Bytes32 sont envoyées à la chaîne de blocs afin de changer les bits du filtre de Bloom dans ces positions à 1.

Tous les participants (le courtier, l'éditeur et l'abonné) doivent effectuer ces étapes afin de modifier leurs filtres de Bloom. Lorsque le contrat intelligent met à jour le filtre de Bloom du courtier avec une nouvelle publication sur un sujet spécifique, il met à jour le nombre de publications envoyées pour l'éditeur et le nombre de publications reçues pour tous les abonnés actifs inscrits sur le sujet défini dans la structure du tableau 3.7.

3.4.3.2 Processus de vérification des données

Dans ce modèle, nous considérons une vérification périodique qui dépend du nombre d'éléments dans les filtres de Bloom. Lorsque les trois filtres de Bloom atteignent le nombre maximum d'éléments à insérer, le contrat intelligent lance le processus de vérification. Tout d'abord, il calcule le degré de similitude entre les trois filtres de Bloom deux à deux selon l'équation 1.1. Nous obtenons trois scores représentant le coefficient de similitude : un score pour les filtres de Bloom des éditeurs et du courtier, un pour les filtres de Bloom des abonnés et du courtier et un score pour les filtres Bloom des éditeurs et des abonnés. Le score résultant doit être compris entre 0 et 1. Lorsque la valeur est plus proche de 1, elle correspond à une plus grande similitude des deux filtres de Bloom. À la fin de cette phase, le contrat intelligent envoie les scores de similitude obtenus et réinitialise les trois filtres de Bloom. En effet, au cours de cette phase, le contrat intelligent envoie un évènement à tous les participants pour les informer que toutes les communications doivent être suspendues afin de terminer la vérification et de réinitialiser les filtres de Bloom. Ce qui rend ce troisième modèle robuste, c'est que notre mécanisme rend toute action frauduleuse non rentable pour chaque entité. En effet, nous définissons un nombre d'insertion limité pour éviter la saturation des filtres de Bloom et afin de diminuer le taux de faux positifs. Cette limite est définie en fonction du nombre maximal d'éléments dans un filtre Bloom pour un taux de faux positifs spécifique. De plus, le contrat intelligent est responsable du maintien du nombre de publications et de livraisons pour tous les éditeurs et abonnés sur la base des insertions du courtier. Par conséquent, les actions malveillantes d'insertions incohérentes n'entraînent aucun profit à l'entité malveillante vu que cela engendre une non-similitude d'un filtre de Bloom par rapport aux deux autres mais ne modifie pas les informations de monétisation

maintenues. Dans le cas où le courtier insère plus de publications que ce qu'il reçoit réellement, son filtre de Bloom diffère des deux autres filtres Bloom.

3.4.3.3 Protocole de monétisation des données

En fonction des coefficients de similitude calculés par le contrat intelligent, nous basons notre approche de façon à avoir au moins deux filtres de Bloom similaires, c'est-à-dire ayant un coefficient de similitude égal à 1. Ainsi, nous considérons deux possibilités :

- dans le cas où le filtre de Bloom du courtier est égal à l'un des deux autres filtres Bloom, le contrat procède au calcul de la facture des abonnés et de la rémunération des éditeurs en fonction du nombre maintenu de publications envoyées/reçues pour chaque client lors des insertions ;
- dans le cas où le filtre de Bloom des éditeurs et des abonnés a un degré de similitude égal à 1 et que le score du filtre Bloom du courtier est plus proche de 0, cela indique que le filtre de Bloom du courtier contient des informations incorrectes. Dans cette situation, le courtier est responsable du paiement des éditeurs.

Cette règle de monétisation persuade le courtier à détenir un filtre de Bloom correct qui correspond à au moins un des autres filtres Bloom.

3.5 Comparaison et discussion

Les trois modèles que nous proposons pour monétiser les données IdO en temps réel respectent une caractéristique essentielle de MQTT, à savoir le découplage des clients qui envoient les données des clients qui reçoivent et achètent les données. Nous concevons nos solutions en garantissant l'anonymat des clients. Les éditeurs et les abonnés ne pourront pas s'identifier mutuellement grâce aux technologies de chiffrement appliquées à tous les identifiants clients et au fait qu'aucune interaction n'a lieu entre les deux parties. En outre, nous proposons trois modèles avec différents niveaux et coûts de traçabilité. Les trois modèles effectuent des enregistrements sur la chaîne de blocs pour tracer l'échange de données sans affecter la latence du système étant

donné que les entités participantes n'ont pas besoin d'attendre une confirmation de transactions envoyées.

Notre modèle Trace-MAX fournit une traçabilité précise des échanges de données en stockant toutes les informations de chaque interaction. Le modèle assure également une vérification solide et offre une détection des incohérences dans toutes les situations. Cependant, garder une trace complète de l'échange, nécessite d'enregistrer toutes les publications et les livraisons sur la chaîne de blocs avec toutes les données chiffrées et d'effectuer des opérations complexes pour un audit exhaustif. D'autre part, le modèle Trace-MIN garantit un faible coût étant donné que les clients n'écrivent pas sur la chaîne de blocs et que le processus de vérification appelle simplement la chaîne de blocs sans changer l'état de la chaîne. Avec ce modèle, les données sont monétisées tout en traçant le volume de données échangées mais aucun autre audit n'est nécessaire. Enfin, notre modèle Trace-BF surmonte les limitations mentionnées dans les deux autres modèles. Il permet d'optimiser l'enregistrement des données échangées avec un coût réduit par rapport à la solution Trace-MAX. En outre, le modèle Trace-BF effectue une vérification en comparant les filtres de Bloom et non pas toutes les publications ce qui réduit le coût et le temps de calcul. Cependant, l'exécution du processus de vérification exige la suspension de toutes les communications qui pourraient affecter la disponibilité du système pendant ce processus. Nous résumons les propriétés de nos trois modèles dans le tableau 3.8 ci-dessous.

Tableau 3.8 Comparaison des modèles de traçabilité

Modèles / Propriétés	**Trace-MAX**	**Trace-MIN**	**Trace-BF**
Traçabilité	Enregistrement rigoureux	Enregistrement minimal	Sauvegarde représentative
Informations enregistrées	Hachage et volume des données	Volume de données échangés	Positions des hachage dans le filtre de Bloom
Entités participant à l'enregistrement	Courtier, Éditeurs, Abonnés	Courtier	Courtier, Éditeurs, Abonnés
Fraude	Détection et prévention	Détection peu fiable	Détection et prévention
Coût	Élevé	Faible	Modéré

3.6 Conclusion

Dans ce chapitre, nous avons présenté notre système d'échange de données en temps réel ainsi que notre schéma de monétisation considéré. Nous avons expliqué le fonctionnement du système ainsi que les détails des trois solutions de traçabilités et de monétisation proposées. Pour conclure, nos trois modèles de traçabilité fournissent un enregistrement transparent du marché des données et une monétisation complète des données. Cependant, ils ne suivent pas les échanges hors chaîne. En fait, les composants du système peuvent vendre les données hors chaîne. Pourtant, ils ne peuvent pas prouver la provenance et la légitimité des données qu'ils vendent.

CHAPITRE 4

ÉVALUATION ET DISCUSSION DES RÉSULTATS

Dans ce chapitre, nous détaillons l'environnement de notre marché de données ainsi que l'implémentation de notre système de monétisation de données. Ensuite, nous présentons nos différents scénarios ainsi que les résultats obtenus dans chaque scénario. Enfin, nous comparons les performances de nos trois solutions en termes de consommation de gaz pour notre système dans sa globalité et en particulier pour le processus de vérification de chaque modèle.

4.1 Environnement de travail

Avant de mettre en œuvre notre marché de données basé sur la chaîne de blocs, nous avons commencé par nous familiariser avec un environnement d'échange de données de l'IdO.

4.1.1 Environnement d'échange de données

Dans cette partie, nous avons utilisé deux appareils IdO composés chacun d'une Raspberry Pi 3 qui est un nano ordinateur monocarte à processeur ARM. Sur l'une de ces cartes, nous connectons un capteur de température et sur l'autre carte nous insérons des LEDs en spirale. Nous connectons ensuite les deux cartes aux réseaux Internet. Pour implémenter un environnement de partage Pub/Sub, nous déployons la bibliothèque du client MQTT où le capteur de température est notre éditeur MQTT et l'actionneur LEDs est notre abonné. Le courtier Pub/Sub est intégré dans notre application en utilisons la librairie Mosca afin de collecter les données et les acheminer vers l'abonné comme illustré dans la figure 4.1.

Dans le courtier Mosca, nous définissons trois méthodes pour restreindre l'accès aux sujets à la clientèle de notre marché de données (Moscajs) :
Authentifier : Pendant la phase de connexion, nous vérifions si le client s'est enregistré au contrat intelligent pour que le courtier accepte sa connexion.
AuthoriserPublication : Nous vérifions si l'utilisateur peut publier au sujet en inspectant si le

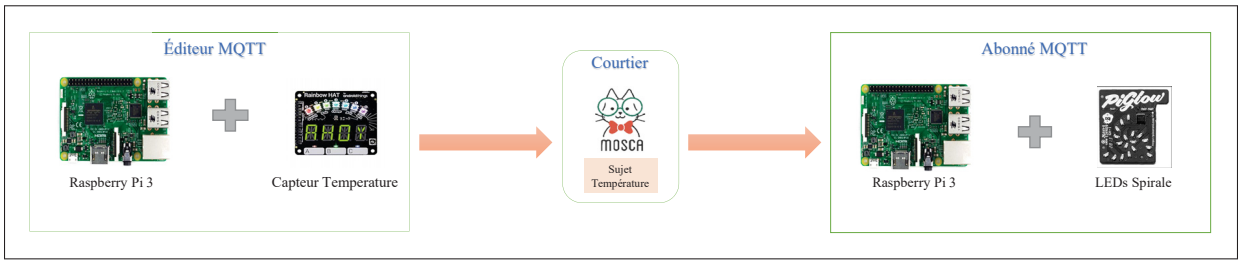

Figure 4.1 Environnement Pub/Sub

nom du sujet coïncide avec celui envoyé par le contrat dans la portée de cet utilisateur.
AutoriserAbonnement : Nous vérifions si l'utilisateur peut s'abonner au sujet en inspectant si le nom du sujet existe dans la liste des sujets sur lesquelles le client peut s'abonner.

4.1.2 Environnement de chaînes de blocs

Pour mettre en place notre environnement de chaînes de blocs basés sur les contrats intelligents d'Ethereum et développer notre système de monétisation distribué, nous utilisons :

- **Truffle :** un environnement de développement basé sur la chaîne de blocs Ethereum, permettant de compiler et déployer les contrats intelligents ;
- **Ganache :** un réseau de chaîne de blocs Ethereum personnel que nous utilisons pour tester les contrats et exécuter des transactions.

En outre, nous utilisons Solidity pour mettre en oeuvre les contrats intelligents ; Solidity est un langage de développement haut niveau, orienté objet. Nous définissons 10 Gwei comme étant notre prix de gaz. Gwei est une dénomination de la cryptomonaie Ether. Un gaz coûte 1 gwei signifie qu'il coûte 0,000000001 Ether. Notre choix constitue un prix adéquat pour un temps de confirmation moyen de 31 secondes selon *Eth gas station* (ETHGasStation).
Enfin, nous déployons une page web en utilisant le langage de programmation HTML afin de fournir un formulaire d'enregistrement où chaque client doit insérer son adresse Ethereum, son rôle dans le système à savoir un éditeur ou un abonné et le sujet sur lequel il va publier ou

recevoir des données. Ce formulaire appelle le contrat intelligent pour enregistrer les données sur la chaîne de blocs tel illustré sur la figure 4.2.

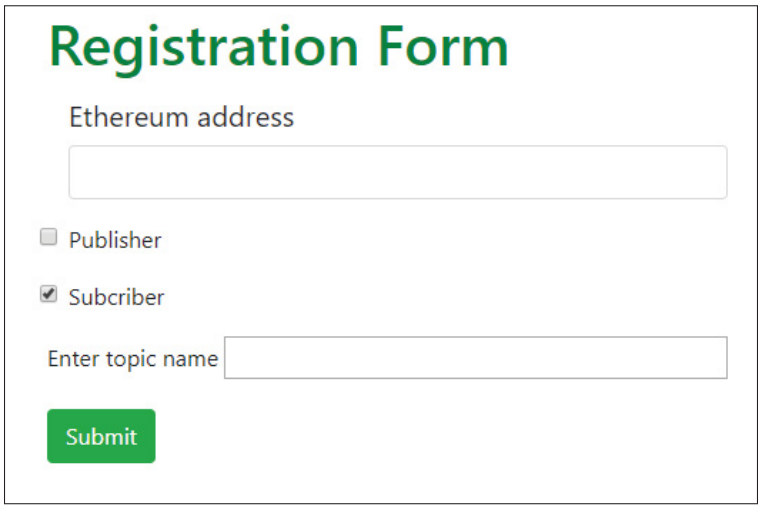

Figure 4.2 Formulaire d'enregistrement

Dans ce qui suit, nous exécutons notre système sur un ordinateur portable équipé d'un processeur Intel (R) Core (TM) i7 2,00 GHz fonctionnant sur Ubuntu 18.04. Nous y implémentons Truffle et Ganache ainsi que la bibliothèque $web3.js$ qui permet d'interagir avec les noeuds d'Ethereum en utilisant une connexion Websocket. Nous utilisons $node.js$ pour développer notre application distribuée avec les clients MQTT et le courtier Mosca. Nous varions le nombre d'éditeurs et d'abonnés en fonction du scénario de test.

4.2 Évaluation du Coût global d'exécution

Le but de cette expérimentation est d'évaluer la consommation totale de gaz pour nos trois modèles de traçabilité. Ainsi, nous exécutons différents scénarios de test pendant une heure et nous calculons la consommation moyenne de gaz par minute.

4.2.1 Scénario 1 : Effet de la fréquence de publication

Pour évaluer l'impact de la fréquence des publications sur la consommation globale de gaz de notre système, nous considérons un éditeur MQTT qui génère des messages et les envoie sur un sujet nommé *Temprature* et un abonné inscrit pour recevoir des publications sur ce sujet. Pour chaque itération, nous augmentons la fréquence des messages envoyés par l'éditeur par minute. Ensuite, nous mesurons la consommation de gaz qui englobe toutes les transactions envoyées au contrat intelligent depuis tous les composants du système (i.e. courtier, abonné, éditeur). Pour chacune des solutions proposées, nous obtenons les résultats illustrés dans la figure 4.3. La

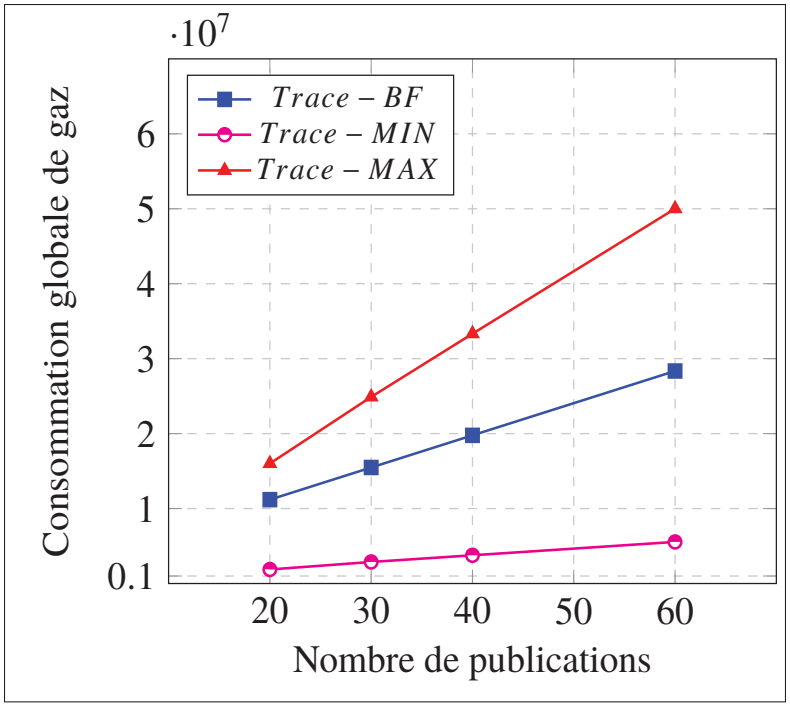

Figure 4.3 Effet de la fréquence de publication sur le coût global

figure 4.3 montre que la consommation globale de gaz par minute augmente linéairement pour toutes les solutions proposées avec un taux d'augmentation de 92000 pour Trace-MIN, 850000 pour Trace-MAX et 429000 pour Trace-BF. Cette augmentation est due à l'augmentation du nombre de transactions proportionnellement au nombre de publications générées.

En effet :

- Trace-MAX consomme plus de gaz en raison de son modèle d'enregistrement qui implique que tous les participants enregistrent chaque étape du partage de données. Le courtier sauvegarde toutes les publications qu'il reçoit et toutes les livraisons qu'il achemine avec le maximum d'informations utiles, de même pour l'éditeur et l'abonné ;
- Trace-BF consomme moins de gaz que Trace-MAX. Les interprétations directes de ce résultat sont liées à la quantité d'informations de journalisation écrites sur la chaîne de blocs. Effectivement, les sauvegardes effectuées sur la chaîne de blocs dans Trace-BF, contiennent beaucoup moins d'informations. Lors de la publication d'un message, un éditeur envoie une transaction qui contient uniquement les positions à modifier dans le filtre de Bloom des éditeurs. De plus, le courtier enregistre dans son filtre de Bloom les publications reçues par les éditeurs et donc il effectue des transactions en moins par rapport aux livraisons ;
- Trace-MIN consomme le moins de gaz par rapport aux deux autres solutions. En effet, ce modèle nécessite un enregistrement minimal sur la chaîne de blocs. Ainsi, les informations écrites sur la chaîne influencent la taille des transactions et affectent la consommation de gaz.

4.2.2 Scénario 2 : Impacte du nombre d'abonnés

Ce scénario consiste à évaluer l'impacte du nombre d'abonnés sur la consommation de gaz du système. Nous considérons un scénario de test impliquant un éditeur avec une fréquence de publication fixe de 20 publications par minute sur un sujet spécifique. Ensuite, nous augmentons le nombre d'abonnés inscrits sur ce sujet et nous calculons la consommation totale de gaz. Les résultats obtenus sont démontrés dans le graphe de la figure 4.4.

La figure 4.4 montre que la consommation de gaz de Trace-MAX augmente plus rapidement que les deux autres solutions. Ce résultat est dû à l'impact du nombre d'abonnés sur le nombre total d'enregistrements écrits sur la chaîne de blocs. En effet, chaque abonné doit confirmer les publications reçues sur la chaîne de blocs dans Trace-MAX. De même, le courtier enregistre sur la chaîne de blocs toutes les livraisons de publications effectuées pour chaque abonné. Par contre

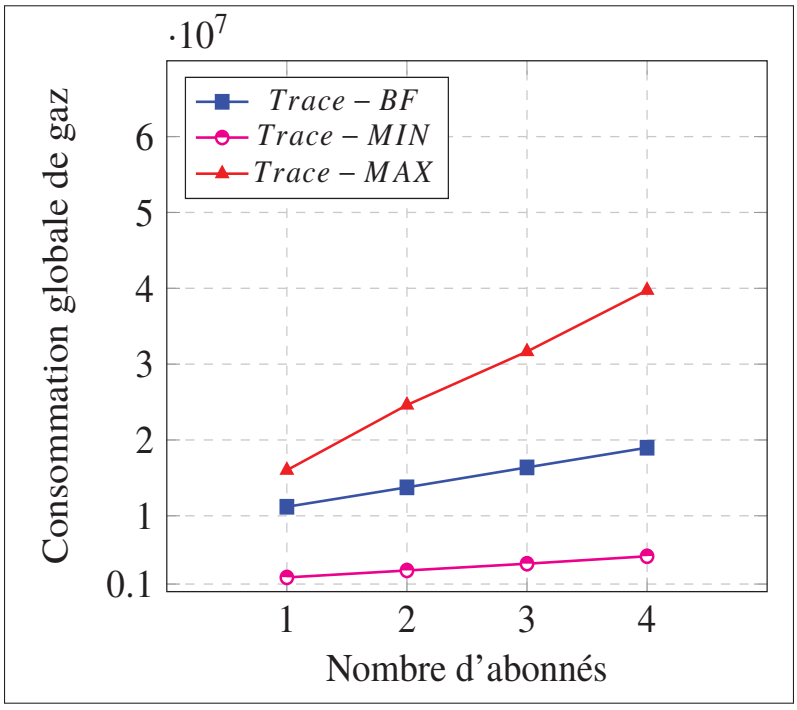

Figure 4.4 Impacte du nombre d'abonnés sur le coût de gaz global

dans Trace-BF, l'augmentation du nombre d'abonnés augmente le nombre d'insertions dans le filtre de Bloom des abonnées mais elle n'influence pas les écritures du courtiers sur la chaîne de blocs. Enfin, pour Trace-MIN, l'augmentation du nombre d'abonnés agit sur les opérations de calcul effectuées dans le contrat intelligent. En effet, le contrat est responsable d'incrémenter automatiquement le nombre de livraisons pour les abonnés lors de la réception d'une transaction initiée par le courtier pour enregistrer une nouvelle publication sur un sujet spécifique.

4.2.3 Scénario 3 : Influence du nombre d'éditeurs

Dans ce troisième test, nous étudions l'influence du nombre d'éditeurs sur la consommation du gaz. Nous fixons le nombre d'abonnés et nous augmentons le nombre d'éditeurs à chaque itération. Chaque éditeur envoie 20 publications par minute sur le même sujet. La figure 4.5 décrit les résultats de cette expérimentation.

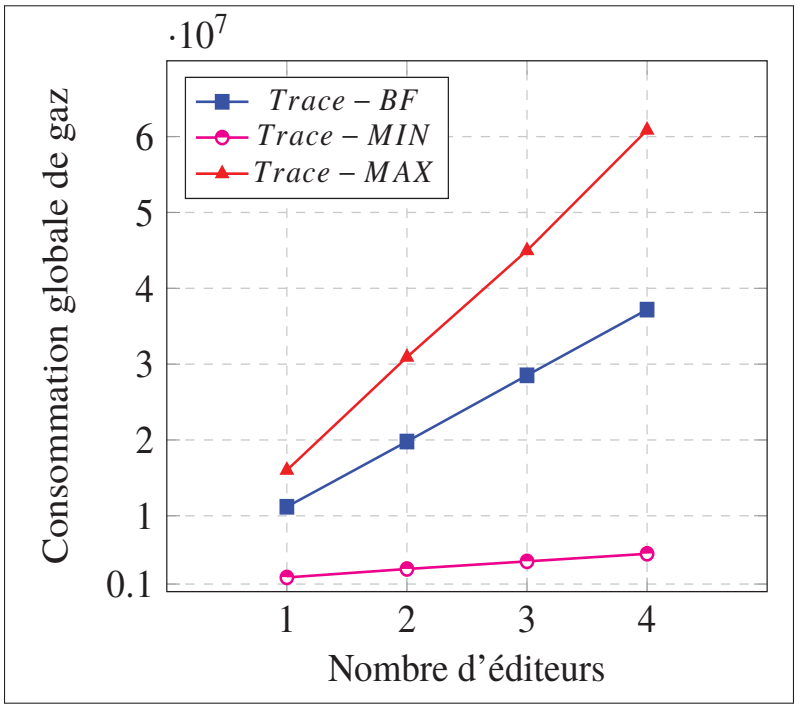

Figure 4.5 Influence du nombre d'éditeurs sur le coût global

Nous notons une augmentation plus importante pour Trace-MAX et Trace-BF par rapport à Trace-MIN. Cela se justifie par l'augmentation du nombre de publications ainsi que le nombre de livraisons suite à l'augmentation du nombre d'éditeurs agissant sur le système. De ce fait, le courtier, les éditeurs ainsi que les abonnées effectuent plus d'enregistrements sur la chaîne de blocs pour enregistrer les publications et les livraisons dans les modèles Trace-MAX et Trace-BF.

4.3 Évaluation des coûts de la vérification

Le but de cette expérience est de faire une enquête plus approfondie sur le processus de vérification de la cohérence des informations échangées dans le système. Nous suivons les mêmes étapes conduites pour calculer l'évaluation globale de la consommation de gaz et nous nous concentrons dans les scénarios de test sur la consommation de gaz dans notre processus de vérification.

4.3.1 Scénario 1 : Effet de la fréquence de publication

Comme expliqué dans la section précédente, nous fixons dans ce premier scénario le nombre d'abonnés et d'éditeurs et nous modifions le nombre de messages publiés par minute. La figure 4.6 représente le coût de vérification en termes de gaz par minute en fonction de la fréquence des publications pour nos trois modèles.

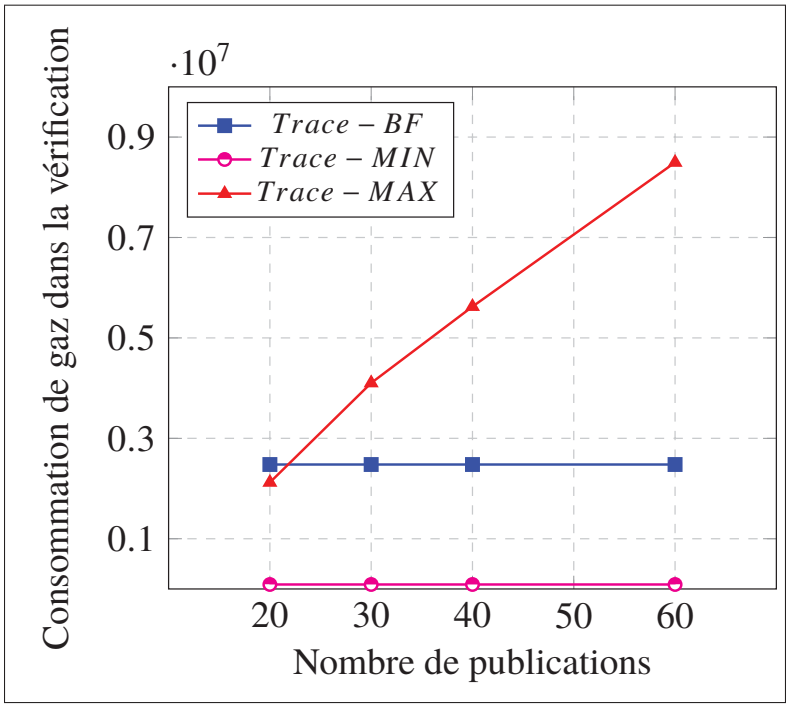

Figure 4.6 Effet de la fréquence des publications sur le coût de la vérification

Nous observons que la consommation du gaz dans le processus de vérification de Trace-MAX augmente avec une grande pente alors que la consommation de la vérification dans Trace-MIN et Trace-BF restent constantes. Cela est dû à la complexité du processus de vérification dans Trace-MAX. En effet, nous effectuons un audit complet sur chaque publication dans Trace-MAX en comparant les enregistrements de l'éditeur, du courtier et de l'abonné. Dans Trace-BF, la vérification effectuée consiste à comparer le degré de similitude entre les trois filtres de Bloom comme détaillé dans la section 3.4.3.2. Par ailleurs, le processus de vérification en Trace-MIN

consiste à comparer le nombre d'octets publiés et livrés. Les opérations de vérifications des deux solutions Trace-MIN et Trace-BF ne dépendent pas de la fréquence des publications.

4.3.2 Scénario 2 : Impacte du nombre d'abonnés

En augmentant le nombre d'abonnés sur un sujet spécifique, nous obtenons les résultats suivants traduisant l'impact du nombre d'abonnés sur le coût du gaz de vérification tel illustré dans la figure 4.7.

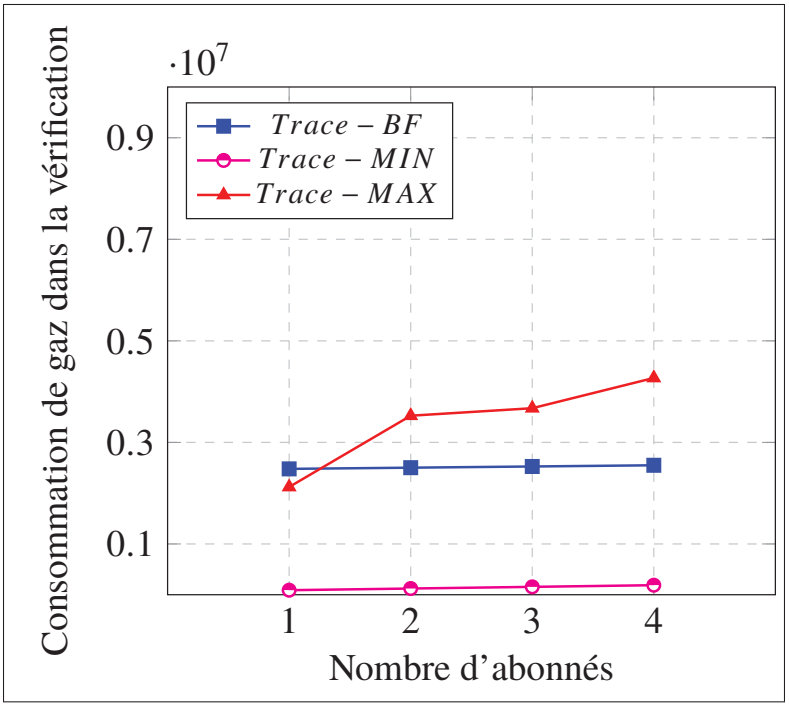

Figure 4.7 Impacte du nombre d'abonnés sur le coût de la vérification

Nous constatons une augmentation de la consommation du gaz au processus de vérification dans Trace-MAX ainsi qu'une légère augmentation dans Trace-MIN. Cependant, la consommation du gaz du processus de vérification pour le modèle Trace-BF reste constante.

– dans Trace-MAX, le contrat intelligent doit boucler sur les enregistrements de chaque abonné afin de vérifier la cohérence des informations que chaque abonné a écrit sur la chaîne de

blocs. Par conséquent, la consommation de gaz augmente dans ce modèle en fonction du nombre d'abonnés ;

– dans Trace-MIN, il y a une augmentation mineure de 32839 gaz avec des abonnés supplémentaires suite aux opérations de comparaison du nombre d'octets publiés sur un sujet et d'octets livrés pour chaque abonné ;
– dans Trace-BF, le coût de la vérification ne dépend pas du nombre d'abonnés, étant donné que tout en stockant les enregistrements, tous les abonnés opèrent sur le même filtre de Bloom. Et pour vérifier, le processus agit sur le filtre de Bloom qui conserve la même taille quelque soit le nombre d'abonnés ou d'éléments à insérer. Ainsi, la quantité d'informations enregistrées n'augmente pas la complexité de la vérification.

4.3.3 Scénario 3 : Influence du nombre d'éditeurs

Dans ce troisième scénario, nous évaluons l'influence du nombre d'éditeurs sur la consommation du gaz dans le processus de vérification. Nous constatons un comportement similaire du processus de vérification au deuxième scénario où nous augmentons le nombre d'abonnés. Comme illustré dans la figure 4.8 ci-dessous, le nombre d'éditeurs n'affecte pas le comportement de la consommation du gaz au processus de vérification dans le modèle Trace-BF. Cependant, nous observons une augmentation dans Trace-MAX et Trace-MIN à cause de l'augmentation des opérations de vérification. En effet, l'augmentation du nombre d'éditeurs se traduit par une augmentation du nombre de publications et des informations écrites dans la chaîne de blocs qui doivent être vérifiées pour des fins de monétisation.

4.4 Discussion des résultats obtenus

Grâce à ces scénarios, nous observons que Trace-BF a une consommation de gaz modérée par rapport à la solution maximale et minimale. Bien que Trace-MIN consomme moins de gaz, il reste peu fiable comme certains comportements malveillants ne peuvent pas être détectés. Tandis que Trace-MAX offre une traçabilité rigoureuse avec une détection des altérations et impose de lourdes amendes ce qui permet d'éviter toutes manipulations malveillantes. Cependant, le

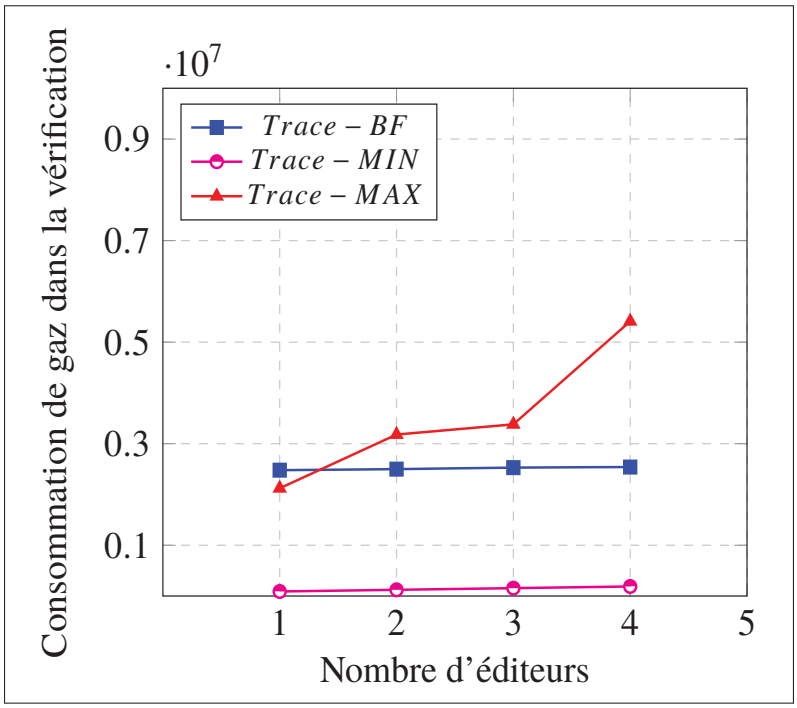

Figure 4.8 Influence du nombre d'éditeurs sur le coût de la vérification

système ne s'adapte pas en termes de consommation de gaz avec l'augmentation de la fréquence des publications et du nombre d'éditeurs et d'abonnés.

En revanche, le modèle de traçabilité Trace-BF établit un équilibre entre le coût et le niveau de traçabilité en fournissant une détection et une prévention des fraudes avec un coût modéré. De plus, le modèle évolue avec l'augmentation de la fréquence des publications, du nombre d'abonnés et du nombre d'éditeurs surtout dans le processus de vérification de l'échange de données. Néanmoins, le processus de vérification implique une suspension de service d'environ 2 min36 s par jour qui correspond à un taux de disponibilité de 99,82% . Cela peut ne pas être suffisant dans certains cas d'utilisation.

CHAPITRE 5

EXPÉRIENCE PERSONNELLE

Dans ce chapitre, je détaille mon parcours durant la réalisation de la maîtrise avec mémoire à l'École de technologie supérieure. Je présente tout d'abord les étapes suivies afin de fixer et atteindre mes objectifs. Ensuite, j'énumère les défis que j'ai pu surmonter pour mener à bien le projet et je finis par les épreuves qui m'ont permis de forger une expérience enrichissante sur le plan tant personnel que professionnel.

Au début de ma maîtrise, la première étape consistait à identifier la proposition de recherche, sélectionner les cours que je vais suivre et enfin définir le plan de maîtrise. Pour ce faire, la phase de revue de littératures était cruciale pour encadrer l'énoncé du problème et les objectifs de recherche . En effet, j'ai commencé par parcourir toutes les ressources scientifiques disponibles afin de former une base sur les recherches qui ont été réalisées dans le domaine de l'Internet des Objets et converger vers des résultats efficaces dans mon travail. En fait, j'ai repéré plusieurs défis à résoudre dans cette technologie à savoir les problèmes de ressources limitées, de connectivité, de sécurité. Par ailleurs, j'ai également observé d'autres problèmes qui ne sont pas liés à la conception des IdO mais à l'exploitation de ces objets et à la génération des quantités énormes de données en temps réel qui ne sont pas utilisées. À cet effet, j'ai réussi à définir la proposition de recherche qui consiste à mettre en place un système de monétisation de données en temps réel pour les systèmes de communications Pub/Sub de manière fiable et traçable en utilisant la technologie de registre distribué.

De là, j'ai commencé à apprendre les notions de base et les technologies que je vais utiliser et maîtriser les outils de travails nécessaires pour la mise en œuvre du système. En effet, j'ai réalisé des recherches approfondies sur les protocoles de messagerie basés sur les publications/abonnements et j'ai choisi le protocole MQTT. Ensuite, j'ai suivi un cours d'*applications et systèmes décentralisés* en plus des lectures des communications afin de choisir la plateforme à considérer et comprendre les propriétés de chaque plateforme. D'où le choix des contrats intelligents et de la plateforme Ethereum. Enfin, j'ai entamé l'apprentissage du langage de programmation des

contrats intelligents *Solidity*, et les environnements de développement *Truffle* et *Ganache*. La prochaine étape consistait à concevoir les modèles de traçabilité et le protocole de monétisation. J'ai tout d'abord réussi à définir les deux premières solutions Trace-MAX et Trace-MIN et fixer les exigences et les propriétés de chaque modèle par rapport à la traçabilité, la vérification et la monétisation. Ensuite, je me suis focalisé à proposer une troisième solution adaptative qui doit fournir un haut niveau de traçabilité avec un coût modéré. De cette façon, j'ai étudié plusieurs systèmes cryptographiques tels la preuve à divulgation nulle de connaissance, les accumulateurs et les filtres de Bloom et j'ai choisi les filtres de Bloom en raison de son efficacité temporelle et spatiale. Sous cet angle, j'ai développé les contrats intelligents et les clients MQTT ainsi que le courtier MQTT. En dernier lieu, j'ai optimisé mes contrats intelligents afin de minimiser le coût de gaz et j'ai évalué la consommation de chaque modèle dans différents scénarios. Grâce aux résultats obtenus, j'ai rédigé une communication scientifique mettant en œuvre notre contribution et publiée dans *la conférence internationale de l'IEEE sur les chaînes de blocs et les cryptomonnaies*. Suite à l'acceptation de notre communication, j'ai pu présenter notre travail et discuter les résultats avec un grand public participant. La dernière étape fut la rédaction de ce mémoire afin de décrire mon projet de recherche et détailler le travail réalisé durant ma maîtrise.

Afin de réaliser ce projet, j'ai fait face à plusieurs défis. En premier lieu, j'ai commencé à réaliser le projet avec le langage de programmation $python$ et donc j'ai utilisé la bibliothèque $web3.py$ pour communiquer avec le contrat intelligent. Cependant, cette librairie était instable et j'ai du ré-implémenter le travail avec $javascript$ et la bibliothèque $web3.js$ qui était bien plus documenté. En deuxième lieu, en développant le contrat intelligent de la solution de traçabilité maximale, j'ai constaté que les lignes de codes que j'ai développées afin de vérifier rigoureusement les informations inscrites sur la chaîne de blocs, ont causé le dépassement de la taille du contrat limite autorisé pour déployer le contrat sur la chaîne de blocs. En effet, le coût de gaz pour déployer le contrat dépassait les 8 millions qui est la limite de gaz sur chaque bloc. Par conséquent, j'ai été mis au défi d'optimiser au maximum les fonctions du contrat intelligent tout en assurant les mêmes fonctionnalités. En dernier lieu, j'ai investigué les implémentations de filtres de Bloom afin de concevoir ma propre implémentation puisque celles existantes ne

convenaient pas à mon système d'échange de données. En effet, les implémentations existantes permettent l'insertion de 27 données alors qu'il existe beaucoup plus de données à insérer dans le contexte d'Internet des Objets. Dans cette perspective, j'ai mis en place un modèle d'enregistrement de données basé sur le filtre de Bloom avec le langage de programmation Solidity qui peut prendre en charge un nombre illimité de données.

Pour finir, ma maîtrise est une expérience enrichissante tant sur le plan personnel que du point de vue professionnel. J'ai pu gagner de nouvelles compétences techniques, et découvrir le domaine de registres distribués, des plateformes de chaînes de blocs et de cryptomonnaies. Même si je suis arrivé à la fin de ma maîtrise, je considère à continuer à travailler sur ce domaine et à développer des applications décentralisées à base de contrats intelligents. Par ailleurs, ma maîtrise avec mémoire m'a permis de découvrir le monde de la recherche pour comprendre des domaines en cours d'évolution et contribuer à ces domaines. La recherche m'a permis de fonder un esprit innovant et un regard critique dans les domaines scientifiques. Pour terminer, plusieurs compétences de communications, de rédaction et de travail d'équipe ont été développées durant mes échanges avec mes superviseurs, mes collègues et d'autres chercheurs de différents secteurs.

CONCLUSION ET RECOMMANDATIONS

En raison de l'omniprésence des appareils connectés, une masse de données sensibles et utiles est générée de manière continue et exponentielle. Ces données sont désormais considérées comme un atout important pour les producteurs et les consommateurs de données. Ainsi, le défi consiste à concevoir un système de partage de données IdO transparent et fiable, et une approche rentable pour monétiser les échanges.

Dans notre mémoire, nous mettons en œuvre un marché de données de l'Internet des Objets, partagées en temps réel en se basant sur les systèmes de publications/abonnements et la technologie de registres distribués. Nous utilisons MQTT comme protocole pub/sub léger pour assurer le transfert de données IdO. Par ailleurs, notre marché de données offre trois modèles de traçabilité et de monétisation différents. Nous utilisons la chaîne de blocs comme étant un registre distribué, publique et immuable pour tracer le partage de données de manière transparente. En effet, nous développons des contrats intelligents d'Ethereum pour enregistrer les informations des échanges effectués et assurer la vérification de ces informations selon le modèle utilisé. Dans notre logique de monétisation, nous valorisons ces échanges avec des transactions basées sur la cryptomonnaie Ether.

Enfin, nous évaluons les performances de nos trois solutions proposées. Les résultats obtenus montrent une consommation de gaz différente pour les trois modèles avec un coût élevé pour la solution maximale et une vérification peu fiable pour la solution minimale. En outre, notre modèle de filtre de Bloom proposé atteint les meilleures performances avec une traçabilité précise et un coût de vérification constant. En résumé, le choix de la solution est un compromis entre le niveau de traçabilité souhaité et les frais généraux, notamment le coût d'exécution du contrat intelligent.

Dans notre marché de données, nous accordons aux utilisateurs le choix de la solution de monétisation en fonctions de leurs besoins. Comme travaux futurs, nous comptons concevoir

un mécanisme de recommandation basé sur l'apprentissage intelligent qui permet de proposer au client le modèle de traçabilité adéquat qui répond à ses exigences. En outre, notre système proposé adresse les communications de publications/abonnements de plusieurs à plusieurs avec un seul courtier. Nous nous intéressons à adapter notre système pour supporter des cas d'utilisation de plusieurs courtiers qui acheminement les données entre différents producteurs et consommateurs. Pour terminer, nous considérons augmenter la disponibilité dans la solution de filtres de Bloom étant donné que nous mettons le système en pause d'environ 2 min36s. De ce fait, nous envisageons une réplication du contrat intelligent et la conception d'un mécanisme de commutation de service qui basculent entre les contrats intelligents lors du processus de vérification afin de ne pas interrompre le service.

ANNEXE I

ARTICLE PUBLIÉ DANS UNE CONFÉRENCE

Ce mémoire fait partie d'une publication, intitulé "Monetization using Blockchains for IoT Data Marketplace", qui a été publié à IEEE International Conference on Blockchain and Cryptocurrency 2020. (ICBC 2020, Toronto)

Monetization using Blockchains for IoT Data Marketplace

Wiem Badreddine
École de Technologie Supérieure
Montreal, Quebec, Canada
wiem.badreddine.1@ens.etsmtl.ca

Kaiwen Zhang
École de Technologie Supérieure
Montreal, Quebec, Canada
kaiwen.zhang@etsmtl.ca

Chamseddine Talhi
École de Technologie Supérieure
Montreal, Quebec, Canada
chamseddine.talhi@etsmtl.ca

***Abstract*—The number of Internet of Things devices is growing dramatically, generating a huge amount of data which is becoming a valuable asset for data analysts. This trend culminates towards the creation of an IoT data marketplace, where streams of data from heterogeneous sources are sent in real time to various data consumers and are metered for monetization purposes. Publish/subscribe systems, such as Message Queuing Telemetry Transport (MQTT), are a promising solution to act as a transport layer for real-time data streams in a decoupled and large scale manner. However, pub/sub systems lack two key properties for an IoT data marketplace: (1) it does not provide any monetization logic; (2) it assumes that the pub/sub brokers are trusted entities, which is not the case in a decentralized or federated marketplace setting. In this paper, we address these issues using a reliable and transparent monetization system based on Distributed Ledger Technology (DLT) and smart contracts. We propose three monetization solutions and demonstrate the trade-off between the overhead of tracking IoT data on a blockchain vs. the accuracy of the monetization for data producers and consumers. In particular, we provide a Bloom filter-based solution for efficient verification of data exchange. We implement our system using Ethereum and Solidity and evaluate with respect to contract gas cost.**

I. Introduction

With the exponential growth of the Internet of Things and the ubiquitous connectivity and interaction of IoT devices, a voluminous amount of sensitive information is continuously produced and transformed to valuable assets. Real-time streams of IoT data create promising opportunities to build applications in different domain such as medical research [1], targeted marketing [2] and smart cities [3]. This trend culminates in the creation of IoT data marketplaces.

The transport of real-time streams of IoT data can be accomplished using publish/subscribe systems, such as MQTT [4]. Pub/sub systems allow data consumers (subscribers) to express their interest through subscriptions, which are matched against incoming traffic of publications from data producers (publishers). The matching and routing process is accomplished through brokers, which can realize these tasks efficiently at a large scale [5]. Prominent publish/subscribe systems used in practice include Facebook Wormhole [6], Google Cloud Pub/Sub [7], Redis [8], Apache Kafka [9], and Yahoo! Pulsar [10].

978-1-7281-6680-3/20/$31.00

Publish/subscribe systems are originally intended to be used in a logically centralized environment, where the information is exchanged between data producers and consumers belonging to the same company. As a consequence, pub/sub systems lack two key properties when used in an IoT data marketplace setting. First, they do not offer any monetization logic. Although the brokers do keep track of usage information, such as the number of publications sent or received by various clients, or the amount of bandwidth used, the system does not innately use this information to calculate payments to and from publishers and subscribers, respectively. Second, pub/sub systems normally assume that the brokers correctly execute matching and routing tasks. In an IoT data marketplace, the data producers and consumers should ideally be able to transfer data in a decentralized manner [11]–[13]. The involvement of any third party in the process should be transparent and require a minimal amount of trust.

To jointly address both issues, we propose the use of Distributed Ledger Technology (DLT) to complement pub/sub in order to provide monetization of IoT data marketplace. Blockchains can provide immutable records of data sharing activities from the publishers, subscribers, and the brokers. Furthermore, smart contracts can be executed to provide a transparent and reliable mechanism to verify the consistency of the blockchain's records. Finally, we can leverage blockchains to enable cryptocurrency-based transactions which correspond to the data monetization logic.

The primary contributions of our work are as follows:

- We propose an IoT data monetization system where pub/sub activities are recorded, which triggers periodic smart contract execution to verify and calculate monetization payments (Section IV),
- We introduce three traceability solutions: Trace-MIN, Trace-MAX, and Trace-BF, which provide varying degrees of traceability and overhead (Sections V). In particular, Trace-BF leverages Bloom filter to provide efficient data storage and faster verification.
- We implement our three solutions with MQTT as the pub/sub broker and smart contracts written in Solidity deployed over Ethereum (Section VI). Our evaluation demonstrates the trade-off between monetization accuracy and contract gas cost.

The paper continues with Section II on related works. This is followed by Section III which is a review of the technologies used in our solution.

II. Related Works

In this section, we review major works related to our research. First, we describe related works for IoT data marketplace. Second, we focus on works related to data monetization. Finally, we present papers leveraging smart contracts for traceability purposes.

A. IoT Data Marketplace

IoT Data marketing is gaining popularity in the last few years as the market is expected to reach USD 3 trillion by 2030 [14]. Several IoT data specialized marketplaces exist that enable selling and buying IoT datasets, for instance, BDEX [15] and DAWEX [16]. Authors in [17] presented a marketing solution for IoT data that overcomes the lack of interoperability in the IoT ecosystem by offering a platform-independent marketplace model with a common API. Furthermore, numerous alternative decentralized marketplace systems that leverage blockchain and smart contracts have been studied. Datapace [11] is a decentralized solution based on Hyperledger Fabric which ensures the integrity of the traded data. It leverages smart contracts to define the conditions under which the data is transferred. IDMoB [12] is another decentralized IoT data marketplace that combines Ethereum blockchain as a smart contract platform and Swarm as a decentralized storage system to provide a platform for incentive collaboration between IoT data vendors and Artificial Intelligence/Machine Learning solution providers. Unlike the above works, our solution is integrated with a publish/subscribe system to deliver real-time streams of data relevant to interested consumers, while continuously monitoring the flow of information for monetization purposes.

B. Data Monetization Models

Many data monetization models have been studied and conceived depending on different requirements and use cases. In [13], the authors describe a reputation-based monetization system for IoT data using Ethereum blockchain and MQTT. According to their model, a data owner creates a smart contract containing information about the topics and the broker address. A customer who is interested subscribes by depositing to the smart contract and starts to get the data from the broker. At the end of the session, the customer can rate the data and leave a review. A similar approach is proposed in [18] which requires the customer to make a deposit and decide how the balance is used across subscribed topics. These papers focus on using a reputation mechanism to evaluate the quality of the data received through the marketplace. In our work, we do not consider using a reputation mechanism. Instead, we focus on tracking specific pub/sub activities on the blockchain and use a smart contract to verify the consistency of the traces collected and calculate the monetization outcomes. Nevertheless, a reputation mechanism could complement our work to assess the quality of the data as well. Authors in [19] propose a different approach, enabling trading parties to negotiate a data offer and store the agreement in the smart contract. The agreement contains streaming time interval and the total price which is based on the message unit price, message streaming rate and the streaming time interval. A receipt is sent to the smart contract by the customer after receiving a specified amount of data as an acknowledgment of receiving data to minimize the customer fraud effect. In our paper, we focus on analyzing the information available from MQTT and deciding which activities should be tracked on the blockchain. We provide three traceability solutions and demonstrate how a smart contract can leverage the information recorded on-chain to calculate monetization outcomes with varying degrees of precision.

C. Traceability using Smart Contracts

With its immutable nature, blockchains are widely used to keep track of operations for traceability and auditing purposes. Several works discuss its use, especially in supply chain (SC) traceability [20]–[22]. Thomas et al. propose an approach of varying the granularity of traceability based on the product characteristics, SC processes, and stakeholders' engagement [23]. In our work, we also investigate different levels of granularity for traceability to enable monetization for IoT data marketplace.

III. Background

In the following section, we give an overview of publish/subscribe systems and we detail Bloom filters technology that we use in one of our solutions (Trace-BF).

A. Publish/Subscribe systems

Publish/Subscribe is an asynchronous messaging service allowing data consumers or (*subscribers*) to subscribe to channels (*Topics*) to which data producers (*Publishers*) send data. Matching publications to topics and forwarding them to subscribers is performed by a pub/sub agent called *Broker*. Message Queuing Telemetry Transport (MQTT) is a widely-used Pub/Sub protocol [24]. It is specialized for machine-to-machine communication with lightweight messaging, many-to-many routing and compatibility with resources-constrained devices [25]–[27]. MQTT provides a tradeoff between transport overhead and delivery guarantee by offering 3 Quality of Service (QoS) levels:

- QoS 0: The message will be delivered only once
- QoS 1: The message will be delivered at least
- QoS 2: The message will be delivered at least once with a more reliable confirmation process.

MQTT is integrated into our work as a message transportation layer for the IoT data marketplace.

B. Bloom Filter

A Bloom filter is a probabilistic data structure that is used to determine whether an element is a member of a set or not. An element is either *probably in the data set* or *definitely not*

in the data set.
The main features of the Bloom filter are:

- Space efficiency: Regardless of the number of items inserted in the collection, the Bloom filter uses a fixed number of bits.
- Time efficiency: The time used to add an element or search an element from the Bloom filter is fixed and is independent of the number of elements contained inside.

False positives rate in a Bloom Filter is influenced by the size of the bit array and the number of hash functions used to insert an element. The number of hash functions can be calculated as follow: $k = \frac{m}{n} \times \ln 2$, with m the size of a Bloom filter in bits and n the number of elements to insert.

Given two Bloom filters, it is possible to determine whether two sets approximately match [28] by calculating the Dice coefficient:

$$DiceCoefficient = \frac{2 \times h}{a + b} \quad (1)$$

Where h is the number of positions set to 1 in both Bloom filters, a and b are the number of positions set to 1 in each Bloom filter. The Dice coefficient value must be between 0 and 1, where a higher number indicates higher similarity between the two filters.

In our work, we use Bloom filters in one of our proposed solutions (Trace-BF) to leverage its storage and cost-efficiency. We employ the Dice coefficient to calculate the similarity degree between various Bloom filters maintained by different entities (publishers, subscribers, and brokers) in order to detect inconsistencies in the traces.

IV. System Architecture

In this section, we propose a real-time IoT data sharing and monetization framework using MQTT protocol combined with Ethereum smart contracts. First, we highlight our system's main features and components. Then, we describe our three proposed solutions with different traceability levels to address different usage requirements. Finally, we provide a comparison and discussion of the three solutions.

A. System Components and Architecture

Our system uses smart contracts to support the IoT data sharing and to perform the verification and accounting process. We leverage the inherited immutability of smart contracts to provide a transparent and trustless IoT data trading environment and to provide fraud detection. In addition to the smart contract, our system contains the following components:

Broker: An MQTT component that handles the connections between the publishers and the subscribes and routes the received messages to the corresponding destinations. In our context, the broker is an untrusted party which must regularly submit traceability information to the blockchain.

Publisher: An IoT device (e.g., sensor) that continuously generates and shares data on a specific topic. The publisher will be paid according to the monetization scheme described below.

Subscriber: A data consumer that subscribes to topics and receives streams of publications. The subscriber will pay according to the amount of data received and the subscription active time.

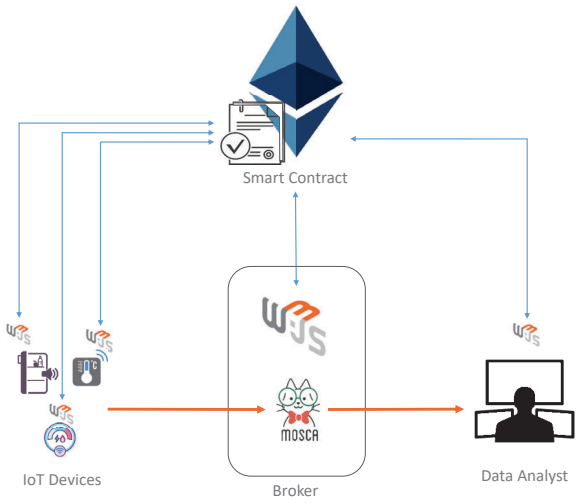

Fig. 1: System Architecture

In order to interact with the system, the broker, the publisher and the subscriber possess an Ethereum address. In addition to these components, we have the *system manager*, who owns the smart contract and confirms the registration of the smart devices; and the *devices owner*, who is responsible for attributing Ethereum addresses to his devices and registering them to the system. Each device must be registered in advance to the smart contract with its Ethereum address and the corresponding role that defines the topics on which it acts as a publisher or as a subscriber. In order to preserve clients' identity, the Ethereum address is hashed and we only record its hash on the blockchain as an identifier of the client. A registered entity can initiate a connection request to the smart contract that verifies its identity and notify the broker with its scope. Thenceforth, a publisher or a subscriber is authorized to start communicating with MQTT and exchange data.

In Figure 1, we illustrate the architecture of our system using a simple smart home use case: A smart home owner wants to use our system to sell its devices' data to a data analyst who processes the latter for smart home products amelioration purpose. As shown in the system architecture, the data analyst acting as a subscriber, the IoT devices acting as publishers and the broker interact with the smart contract, which is deployed on the Ethereum blockchain, to record the data sharing activity depending on the traceability model. To do so, these entities use a gateway (using $web.js$) to interact with Ethereum blockchain. On the other hand, the data generated by the IoT devices are sent to the broker which uses the Mosca module to collect the data and to route it to the data analyst.

B. Monetization Scheme

We define in our smart contract a standard pricing for publishers and subscribers based on the number of mes-

sages sent/received and the volume of data sent/received. The proposed accounting system is based on these equations to monetize the exchanged data for publishers and subscribers and have a decent profit. We calculate the bill of a subscriber and the total income of a publisher as follows:

$$B_{Subscriber} = f_1 \times t + f_2 \times NbrP + f_3 \times BytesR + f_4$$
$$I_{Publisher} = f'_1 \times t + f'_2 \times NbrP + f'_3 \times BytesS - f'_4$$

With $NbrP$ as the number of publications received for the subscriber and sent for the publisher, $BytesR$ as the total bytes received, $BytesS$ as the total bytes sent, f_i represents the fees per unit and f'_i represent the pay per unit ($i \in 1, 2, 3$). f_4 and f'_4 are fixed fees charged for the registration and the verification fees. These fees are fixed by the marketplace operator in order to achieve a target profit goal:

$$Profit = \sum B_{Subscriber} - \sum I_{Publishers} - \sum Gas_{Used}$$

The smart contract automatically calculates bills that can be paid using the native cryptocurrency (i.e., Ether). Subscribers are informed of their bills by a triggered event from the smart contract and they must deposit the required amount to the contract. Another event is triggered containing the publishers id and the corresponding revenue. The broker initiates the transfer of the specified value. When a transfer or a deposit is made, the bill of the client is reset.

The cost of recording publications, deliveries and the verification varies depending on the traceability solution chosen. Thus, each solution has its own prices and the client can choose the model to use based on these charges and the trade-off between traceability and cost.

V. Proposed Traceability Solutions

In this section, we provide details about the key contribution of the paper, which is the traceability mechanism used to record pub/sub activities on the blockchain. We present three solutions: Trace-MAX, Trace-MIN, and Trace-BF. For each solution, we present which data is shared on the blockchain, how verification is performed, and the impact of the choice of traceability on monetization, as described in the previous section.

A. Trace-MAX: Maximum Traceability

Trace-MAX provides maximal traceability of the data transferring process. All participants in the data marketplace will have to write detailed information on the Blockchain. Once written, the information is immutable and participants can not deny their actions.

Data sharing: Publishers keep track of all their sent messages, subscribers record all the received messages and the broker stores all the received and delivered messages. Whenever a publisher sends a publish message in a specific topic, it writes in the blockchain through calling the smart contract and saves all information related to its publication. We maintain in this model, for each publication, a structure with the message id, the topic name, the data hash, the data size and the block timestamp. At the same time, the broker also writes the publication on the blockchain with the same logging information written by the publisher. When the publication is routed to its topic subscribers, each subscriber confirms the delivery by recording on the blockchain what it has received. Since we set the QoS level in MQTT to 1, the subscriber sends a confirmation to the broker. The latter stores the delivery information in the smart contract. With this logic, every step is tracked on the blockchain for each client, for each publication and each delivery which ensures the correctness of the accounting.

Verification process: Our approach to verify the exchange process and mitigate repudiation and fraud risks is to have the maximum information logged in the blockchain. Our system is able to exert full verification periodically. it enables a monthly verification of the list of clients registered on the blockchain. Thus, our smart contract performs an exhaustive audit for each publisher and verify the coherence of what it wrote with what the broker wrote. Two situations can occur:

- The information recorded on the blockchain is coherent, the smart contract count these correct publications for billing.
- The system detects incoherent information for a certain publication in a specific topic, the smart contract performs a second verification through the deliveries assets of that topic to find whether one of the records coincides with the same data hash, data size and within the appropriate time interval.

On the other hand, the smart contract performs the same full verification for each subscriber. It compares all confirmations sent from the subscriber with the one recorded by the broker. In case of incoherent information for certain deliveries, a second verification is necessary to identify who is responsible for this incorrect records. Thus, the contract search for the odd delivery in order to find the corresponding publication. A fixed fine is charged to the entity at fault.

Monetization process: Our smart contract is able to detect the entities involved in information incoherence that costs the system extra verification operations. To prevent writing invalid data to the blockchain, our monetization system in this Trace-MAX model charges all these entities with the verification fees for every fake data.

B. Trace-MIN: Minimal Traceability

This model is based on the minimal traceability level possible for clients who don't need rigorous traceability in their exchange. In this case, the only entity writing on the blockchain is the broker and the information recorded is also minimal but it allows a basic trace and verification in order to monetize the trade with a minimum cost.

Data sharing: When the broker receives a new publication, he sends the id of the publisher and the size of the data published to the blockchain. Our smart contract adds this value to the total volume of data sent by the publisher, increments the total bytes published on that topic and updates the publisher billing information. On the other side, when the broker receives a confirmation from a subscriber of a delivered data, he sends the id of the subscriber and the size of data received to the

blockchain. Our smart contract adds the received data size to the tracked amount of bytes received by the subscriber on the specific topic and updates the billing information of the subscriber.

Verification process: In this model and similar to the Trace-MAX model, we consider a periodic verification. The smart contract calculates the sum of the total bytes received by a specific subscriber on all his topics and compares it to the sum of total bytes published on all those topics.

Monetizing process: Since we are doing the verification in a captured interval of time, the total bytes received by a specific subscriber in that time interval should be equal to what was published on that topic at the same time interval.

- If the total bytes received calculated in the verification process is less then what was published, this means that the subscriber has sent less delivery confirmations compared to what he has received. The broker has no interest in writing less information in this case since this means that the subscriber will pay less.
- In the case the total bytes received are superior than what was published, the broker might be responsible for writing more information than what a subscriber confirmed since a subscriber has no interest in logging publications that he did not receive.
- In the case the total bytes received are equal to what was published, this means that there is a chance that everything is recorded correctly but this also might be the case where the broker is dishonestly writing more publications and deliveries. The system can not detect this behavior and this is Trace-MIN limitation.

In our Trace-MIN mode, the smart contract can only detects that an incoherence occurred and informs all parties of the issue. Then, it charges the subscriber of what was published and inform the latter of his billing information including the volume of data published on his topics, the volume of data received and the charged amount.

C. Trace-BF: Bloom Filter-Based Traceability

In order to overcome the shortcomings of the other two proposed models by providing a reliable traceability with a reduced cost, we leverage Bloom filters to maintain data hashes and perform a robust verification with fewer operations on the blockchain.

Existing implementations of the Bloom filters in Solidity have a maximum size of 256 bits which can hold up to 27 elements with a false positives probability of 10^{-2} [29]. These implementations are not adequate for an IoT environment that generates a huge amount of data (e.g., ten thousand events per day). In order to support 10000 elements with a false positive probability of 10^{-2}, a Bloom filter implementation must contain 96000 bits and a hash rate of 7.

For this purpose, we propose a new design of the bloom filter in the form of an array of 3000 Bytes32, which corresponds to 96000 bits. As shown in Figure 2, an element is represented by the concatenation of a message and its topic, which is hashed using sha3 function. A position is determined by calculating the modulus 96000 of the hash. This operation is executed 7 times to finally extract 7 positions to insert one element.

Data sharing: In our smart contract, we initiate the first three Bloom filters with the same size, one Bloom filter for publishers to write all their publications, one for the broker to keep track of the sent publications and one for subscribers to insert confirmation of publications reception. Each publication or delivery is added to the Bloom filters and recorded on the blockchain by following these steps:

- The first step consists of hashing the topic name concatenated with the data.
- Then, the resulted hash is inserted to each of the k hash functions. The modulus of these hashes are computed with respect to the Bloom filter bit size and the result corresponds to k positions in that length of set of bits.
- Our Bloom filter is an array of bytes32. Hence, we have to find from the calculated positions the corresponding position in the array and within the bytes32.
  To do so, the position in the bytes32 array corresponds to the position in the bit divided by 32 and the position in the bytes32 corresponds to the division remainder.
- The k array and bytes32 positions are sent to the blockchain in order to set these positions to 1 in the Bloom filter.

All the participants (e.g., the broker, the publisher and the subscriber) must perform these steps in order to modify their Bloom filters. When the smart contract updates the broker's Bloom filter with a new publication on a specific topic, it updates the number of sent publications for the publisher and the number of received publications for all subscribed clients on the topic.

Verification process: What makes this model robust is that our mechanism makes every intended fraudulent action unprofitable for every entity. We define a limited insertion number to prevent Bloom filters saturation and decrease false positives rate. This limit is set based on the maximal number of elements in a Bloom filter for a specific false positives rate. Malicious actions will only cause a non-similarity of one Bloom filter compared to the other two Bloom filters but no profit will be made. The smart contract is responsible for maintaining the number of publications and deliveries for all publishers and subscribers based on the broker insertions. In the case where the broker is inserting more publications than what he actually receives, his Bloom filter will defer from the other two Bloom filters. Thus, our monetization rules persuade the broker to hold a correct Bloom filter that matches at least one of the other Bloom filters. Therefore, when all the three Bloom filters reaches the maximum number of elements to be inserted, the smart contract starts the verification process. First, it calculates the similarity degree between the three Bloom filters according to Equation (1) in III-B, the resulting score of the coefficient must be between 0 and 1. When the value is closer to 1 it corresponds to greater similarity of the two Bloom filters. Finally, the contract sends the scores of similarities between

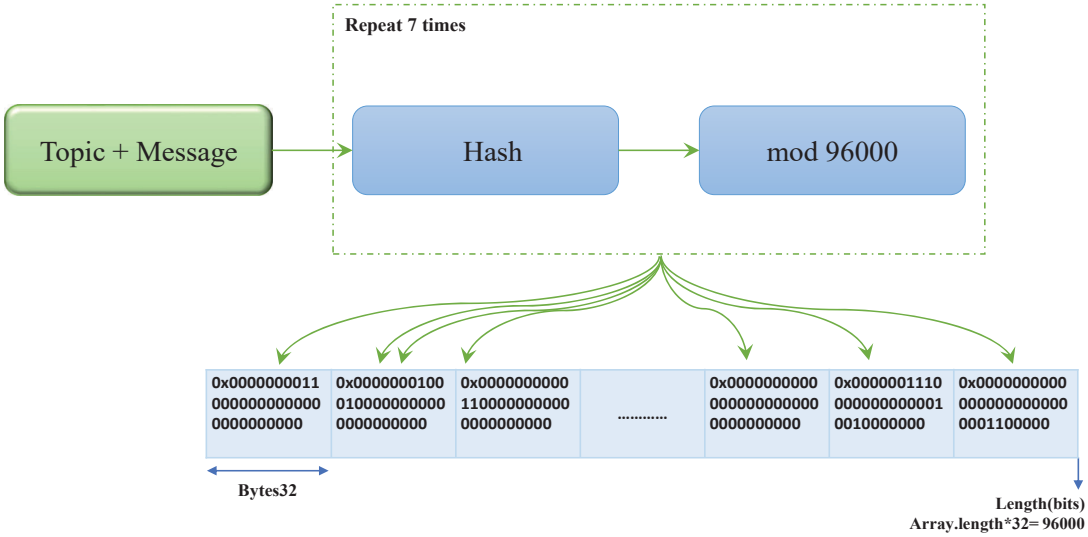

Fig. 2: Bloom Filter data insertion

publishers and broker Bloom filters, subscribers and broker Bloom filters and between publishers and subscribers Bloom filters. During this phase, the smart contract sends an event for all participants to inform them that all communications must be suspended in order to complete the verification and reset the Bloom filters.

Monetizing process: The broker initiates monetization based on the similarity scores that he receives from the smart contract. Our approach relies on having at least two similar Bloom filters (i.e., similarity score ≈ 1). In the case where the broker Bloom filter is equal to one of the two other Bloom filters, the contract proceeds to calculate the bill for the subscribers and the pay for the publishers based on the maintained volume of data sent/received during the insertion of the publications. However, in the case where the publishers and subscribers Bloom filters have a higher similarity degree and the broker Bloom filter score is closer to 0, this indicates that the broker's Bloom filter contains incorrect information. In this situation, the broker is responsible for paying the publishers.

D. Comparison and Discussion

Our proposed three models for monetizing IoT real-time data respect an essential characteristic in MQTT which is decoupling the clients sending the data from the clients that receive and buy the data. We design our solutions ensuring clients anonymity. Publishers and subscribers won't be able to identify each other thanks to the encryption technologies applied to all clients Ids and the fact that no interaction occurs between the two parties.

In our models, we propose different traceability levels and costs. Our Trace-MAX model provides accurate traceability of the data trading by storing each interaction information. The model ensures also a strong verification and offers inconsistency detection in all situations. However, keeping a complete trace requires recording all publications and deliveries on the blockchain with all the encrypted data and performing complex operations for an exhaustive auditing. On the other hand, the Trace-MIN model guarantees a low cost considering that the clients do not write on the blockchain and the verification process simply calls blockchain for checking without changing in the blockchain state. With this model, data is monetized while tracing the volume of data exchanged but no further auditing is provided. Finally, our Trace-BF model overcomes the limitations mentioned in the two other models. It gives an optimized data exchanged logging with a reduced cost compared to Trace-MAX solution. Besides, Trace-BF model performs full verification of the Bloom filters without changing in the blockchain state since billing is calculated while saving the publications. However, executing the verification process demands the suspension of all the communications which might affect the availability of the system while doing this process. Our three traceability models provide a transparent record of the data marketplace however it does not keep track of off-chain trades. As a matter of fact, system components can sell the data off-chain. Yet, they can not prove the provenance and the legitimacy of the data they are selling. We summarize these models properties in Table I.

TABLE I: Traceability Models Comparison

Models / Properties	**Trace-MAX**	**Trace-MIN**	**Trace-BF**
Traceability	Rigorous data sharing logging	Minimal logging	Representative logging
Recorded information	Data hash and size	Bytes exchanged	Data hash positions in the Bloom filter
Entities participating in recording	Broker, Publishers, Subscribers	Broker	Broker, Publishers, Subscribers
Fraudulence	Detection and prevention	Unreliable fraud detection	Detection and prevention
Cost	High	Low	Moderate

VI. Evaluation

In this section, we present the development environment used to implement our system. Then we give a detailed interpretation and comparison of our three models performance in terms of gas consumption.

A. Overview of the development environment

To implement our blockchain-based data marketplace, we use Truffle as our development environment and testing framework for Ethereum. We use Solidity as a development language. Then, our smart contract is deployed on Ganache personal Ethereum network. We define 10 gwei as our gas price which is the average according to *Eth gas station* [30] with an average confirmation time of 31 seconds. To implement our pub/sub environment, we use MQTT client library and Mosca $node.js$ implementation to define one MQTT broker. The number of publishers and subscribers vary depending on the testing scenario. Our system is executed on a laptop equipped with Intel(R) Core(TM) i7 2.00GHz processor running on Ubuntu 18.04.

B. Overall Execution Cost

The purpose of this experiment is to evaluate the total gas consumption of our three models. Thus, we run different testing scenarios for one hour and calculate the average gas consumption per minute.

1) Publication frequency effect: To evaluate the publication frequency impact on our system gas consumption, we consider a publisher that is generating and sending messages on a topic named $Temperature$ and a subscriber subscribed to receive publications on this topic. For each iteration, we increase the frequency of messages sent by the publisher per minute. We measure the overall gas consumption which encompasses all the transactions sent to the smart contract from all the system components.

Figure 3 illustrates the results of the experiment:

- The overall gas consumption increases linearly for all the proposed solutions with a slope of ~92000 for Trace-MIN, ~850000 for Trace-MAX and ~429000 Trace-BF. The increase of the overall gas consumption is due to the rise of the transaction's number proportionally to the number of generated publications.
- Trace-MAX consumes higher gas due to its recording pattern that implicates all the participants to log every step of the data sharing (e.g., the broker, the publisher and the subscriber).
- Trace-BF consumes less gas than Trace-MAX. The direct interpretations of this result are related to the quantity of logging information stored on the blockchain. In fact, in Trace-BF, the broker stores the publications received from the publisher but does not store the publications delivered to the subscriber as it is the case in Trace-MAX. Moreover, the records written on blockchain in Trace-BF contain much less information which influences the size of the transactions and effects the gas consumption. For instance, when publishing a message, a publisher sends a transaction that contains only the positions to be modified in publishers Bloom filter.
- Trace-MIN consumes less gas than the other two solutions since it requires minimal recording on the blockchain.

2) Subscribers number effect: To evaluate the number of subscribers effect on the system gas consumption, we created a testing scenario involving one publisher with a fixed publishing frequency of 20 publications per minute on a specific topic. Then, we increase the number of subscribers and calculate the total gas consumption. The results obtained in the second graph of Figure 3 shows that the gas consumption of Trace-MAX increases faster than the other two solutions. This result is due to the impact of the number of subscribers on the total number of records written. Indeed, each subscriber needs to confirm the received publications, likewise, the broker logs on the blockchain all publications deliveries for each subscriber.

3) Publishers number effect: In this third experiment, we fixed the number of subscribers and increased the number of publishers in each iteration. Each publisher sends 20 publications per minute on a specific topic. The third graph in Figure 3 describes the results of the experience. We note a bigger increase for the Trace-MAX and Trace-BF compared to Trace-MIN. This is justified by the rise of the publications' number following the increase of the publishers using our system which implies more records on the blockchain by the broker, the publishers and the subscriber for both Trace-MAX and Trace-BF solutions.

C. Verification Cost Evaluation

The purpose of this experiment is to make a deeper investigation over the core process of our systems which is the verification process. We follow the same steps conducted to calculate the overall gas consumption evaluation and we focus in the following tests on the gas consumption of our verification process.

1) Publication frequency effect: As explained in the first scenario, we fix the number of subscribers and publishers to one and change the number of messages published per minute. The first graph in Figure 4 represents the verification cost in terms of gas as a function of the publications' frequency for our three models. The gas consumption of Trace-MAX verification increases with a big slope whereas Trace-MIN and Trace-BF verification remain constant. This is due to the complexity of Trace-MAX verification process as it operates an audit of every publication by comparing the logs of the publisher, the broker and the subscriber. While in Trace-BF, the conducted verification consists of comparing the similarity level of the three Bloom filters as detailed in Section V-C. On the other hand, Trace-MIN verification process consists of comparing the number of published and delivered bytes.

2) Number of subscribers: The second graph of Figure 4 illustrates the impact of the number of subscribers on the verification gas cost. While Trace-MAX verification rises and Trace-MIN verification slightly increases, Trace-BF remains constant.

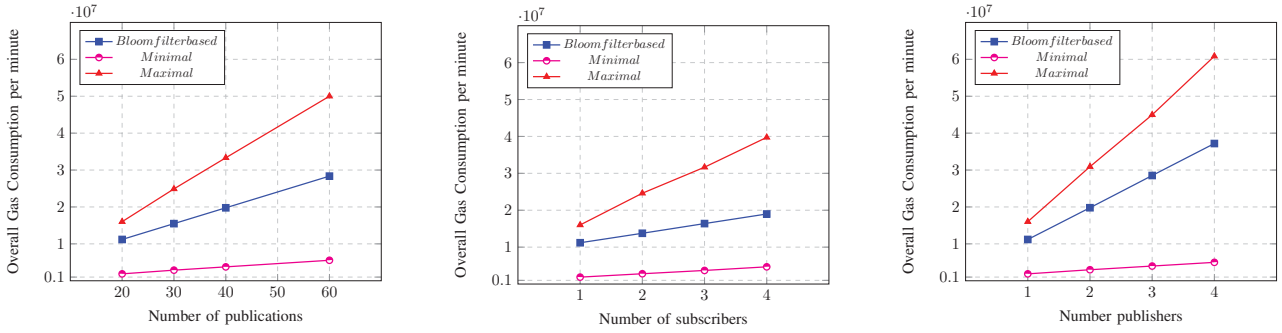

Fig. 3: Publication frequency, subscribers number and publishers number effect on the total gas consumption

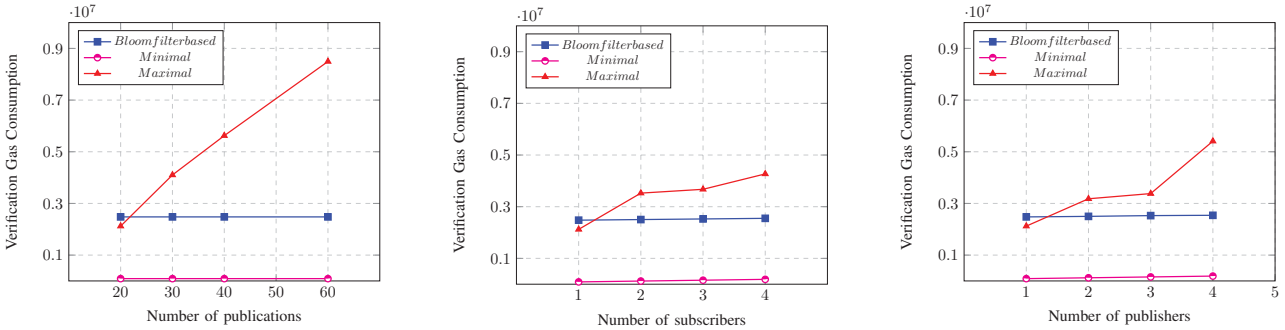

Fig. 4: Publication frequency, subscribers number and publishers number effect on the verification gas consumption

- In Trace-MAX, the smart contract needs to loop on the records of every subscriber data structure. Therefore, the verification gas consumption rises depending on the subscribers number.
- In Trace-MIN, there is a minor increase of 32839 gas with additional subscribers as a result of the comparison operations of the number of published bytes and delivered bytes for every subscriber.
- In Trace-BF, the verification cost does not depend on the subscribers number considering the fact that while storing the records, all the subscribers operates on the same Bloom filter. Thus, the amount of recorded information does not increase the complexity of the verification.

3) Number of publishers: In this third scenario, increasing the number of publishers results in the rise of the number of publications. Thus, the verification process behaves similarly to the second scenario where we increase the subscribers' number as shown in the third graph of Figure 4. Hence, the number of publishers does not impact Trace-BF behavior.

D. Discussion

Through the experiment, we observe that Trace-BF has a moderate gas consumption compared to the minimal and the maximal solution. Although Trace-MIN consumes less gas, it remains unreliable since some malicious behaviors cannot be detected (e.g., a broker recording fake deliveries on the blockchain). Otherwise, Trace-MAX offers a rigorous traceability with tampering detection and heavy fines to prevent malicious manipulations. However, the system does not scale in terms of gas consumption with the rise of the publications frequency and the number of publishers and subscribers.

Trace-BF traceability model balances between the cost and the traceability level by providing tamper detection and prevention at a moderate cost. Moreover, it scales well with the publications' frequency, the number of subscribers and the number of publishers; Nonetheless, this solution may not provide enough availability for certain use cases. The verification process involves a service pause of about 2min36sec per day, which corresponds to an availability rate of 99.82%. As an alternative solution, we propose a smart contract replication and service switch at the verification process. This latter is suggested as a future work.

VII. Conclusion

In this paper, we propose and implement an IoT data marketplace with three different traceability and monetization models based on publish/subscribe (MQTT), DLTs (Ethereum) and smart contracts (Solidity).

The results show different gas consumption for the three models with a high cost for the maximal solution and unreliable verification for the minimal solution. Our proposed Bloom filter model achieves the best performance with accurate traceability and constant verification cost. The choice of solution is a trade-off between the desired traceability level and the overhead cost, notably the smart contract execution cost.

REFERENCES

[1] O. Albahri, A. Zaidan, B. Zaidan, M. Hashim, A. Albahri, and M. Alsalem, "Real-time remote health-monitoring systems in a medical centre: A review of the provision of healthcare services-based body sensor information, open challenges and methodological aspects," *Journal of medical systems*, vol. 42, no. 9, p. 164, 2018.

[2] S. Lessmann, J. Haupt, K. Coussement, and K. W. De Bock, "Targeting customers for profit: An ensemble learning framework to support marketing decision-making," *Information Sciences*, 2019.

[3] B. Ahlgren, M. Hidell, and E. C.-H. Ngai, "Internet of things for smart cities: Interoperability and open data," *IEEE Internet Computing*, vol. 20, no. 6, pp. 52–56, 2016.

[4] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi, "Internet of things: Survey and open issues of mqtt protocol," in *2017 International Conference on Engineering & MIS (ICEMIS)*. IEEE, 2017, pp. 1–6.

[5] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[6] Y. Sharma, P. Ajoux, P. Ang, D. Callies, A. Choudhary, L. Demailly, T. Fersch, L. A. Guz, A. Kotulski, S. Kulkarni *et al.*, "Wormhole: Reliable pub-sub to support geo-replicated internet services," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 351–366.

[7] S. Krishnan and J. L. U. Gonzalez, "Google cloud pub/sub," in *Building Your Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 277–292.

[8] "Redis pub/sub," https://redis.io/topics/pubsub.

[9] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[10] J. Francis and M. Merli, "Open-sourcing pulsar, pub-sub messaging at scale," https://yahooeng.tumblr.com/post/150078336821/open-sourcing-pulsar-pub-sub-messaging-at-scale, 2016.

[11] D. Draskovic and G. Saleh, "Decentralized data marketplace based on blockchain," White Paper, 2017. [Online]. Available: https://www.datapace.io/datapace_whitepaper.pdf

[12] K. R. Özyilmaz, M. Doğan, and A. Yurdakul, "Idmob: Iot data marketplace on blockchain," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 2018, pp. 11–19.

[13] A. Javaid, M. Zahid, I. Ali, R. J. U. H. Khan, Z. Noshad, and N. Javaid, "Reputation system for iot data monetization using blockchain," in *International Conference on Broadband and Wireless Computing, Communication and Applications*. Springer, 2019, pp. 173–184.

[14] D. Tang, J. Fujii-Hwang, E. Colwill, S. Arora, A. Shah, A. Mendizabal, and R. Callejas, "Value of data: the dawn of the data marketplace.[online] accenture," 2018.

[15] "Bdex," https://www.bdex.com/.

[16] "Dawex," https://www.dawex.com/.

[17] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente, "Enabling iot ecosystems through platform interoperability," *IEEE software*, vol. 34, no. 1, pp. 54–61, 2017.

[18] A. Suliman, Z. Husain, M. Abououf, M. Alblooshi, and K. Salah, "Monetization of iot data using smart contracts," *IET Networks*, vol. 8, no. 1, pp. 32–37, 2018.

[19] S. Bajoudah, C. Dong, and P. Missier, "Toward a decentralized, trust-less marketplace for brokered iot data trading using blockchain."

[20] K. Behnke and M. Janssen, "Boundary conditions for traceability in food supply chains using blockchain technology," *International Journal of Information Management*, 2019.

[21] J. Lin, Z. Shen, A. Zhang, and Y. Chai, "Blockchain and iot based food traceability for smart agriculture," in *Proceedings of the 3rd International Conference on Crowd Science and Engineering*. ACM, 2018, p. 3.

[22] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart contract-based product traceability system in the supply chain scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019.

[23] T. K. Dasaklis, F. Casino, and C. Patsakis, "Defining granularity levels for supply chain traceability based on iot and blockchain," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. ACM, 2019, pp. 184–190.

[24] "Message queuing telemetry transport," http://mqtt.org/.

[25] S. Pooja, D. Uday, U. Nagesh, and S. G. Talekar, "Application of mqtt protocol for real time weather monitoring and precision farming," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. IEEE, 2017, pp. 1–6.

[26] R. Atmoko, R. Riantini, and M. Hasin, "Iot real time data acquisition using mqtt protocol," in *Journal of Physics: Conference Series*, vol. 853, no. 1. IOP Publishing, 2017, p. 012003.

[27] K. Chooruang and P. Mangkalakeeree, "Wireless heart rate monitoring system using mqtt," *Procedia Computer Science*, vol. 86, pp. 160–163, 2016.

[28] S. M. Randall, A. M. Ferrante, J. H. Boyd, J. K. Bauer, and J. B. Semmens, "Privacy-preserving record linkage on large real world datasets," *Journal of biomedical informatics*, vol. 50, pp. 205–212, 2014.

[29] "Bloom filter calculator," https://hur.st/Bloomfilter/.

[30] "Eth gas station," https://ethgasstation.info/.

ANNEXE II

CONTRATS INTELLIGENTS

1. Trace-MAX

```
pragma solidity >=0.4.22 <0.6.0;
pragma experimental ABIEncoderV2;

contract TraceMaxContract{

    address sysOwner;

    uint256 BP_DiceCoef = 0;
    uint256 BS_DiceCoef = 0;
    uint256 PS_DiceCoef = 0;

    uint256 feesperduration = 0.01*10000;
    uint256 feespernumber = 0.0009*10000;
    uint256 registerFees = 0.002*10000;
    uint256 feesperbytes = 0.0001*10000;
    uint256 verificationFees = 0.1*10000;
    uint256 payperduration = 0.001*100000;
    uint256 paypernumber = 0.0006*100000;
    uint256 registerFeesP = 0.001*100000;
    uint256 payperbytes = 0.00007*100000;

    struct broker
    {
        address brokerAddress;
        uint256 maxClientsConnected;
        uint256 maxPublicationsReceived;
        mapping(string => client) idToClients;
    }
    struct client
    {
        uint256 cnxtime;
        uint256 discnxtime;
        mapping(string => message[]) topicToMsgReceived;
        mapping(string => message[]) topicToMsgPublished;
        mapping(string => bill) topicToBill;
    }
    struct message
    {
```

```
        string messageID;
        bytes32 dataHash;
        uint256 dataSize;
        uint256 t;
    }
    struct delivery
    {
        message deliveredMsg;
        uint256 nbDeliveries;
    }
    struct topic
    {
        message[] publications;
        delivery[] deliveries;
        uint256 totalBytes;
    }
    struct bill
    {
        uint256 totalBytes;
        uint256 totalNumber;
        uint256 numberDeliveries;
        uint256 verifFees;
        uint256 finalAmount;
        bool paid;
    }
    struct RegisteredClients
    {
        string id;
        string[] subscriptions;
        string[] publications;
        uint index;
    }
    broker b;
    string[] private registerIndex;
    mapping(string => client) private idToClient;
    mapping (string => topic) private topicToPublications;
    mapping(string => RegisteredClients) private idToRegisteredClients;
    mapping (string => address) private idTorecoveredAddresss;
    mapping(address => uint256) balance;


    modifier onlyOwner{
        require(msg.sender == sysOwner);
        _;
```

```
}

modifier onlyBroker{
    require(msg.sender == b.brokerAddress);
    _;
}
/****Events****/
event Authenticated(string id, string[] subscriptions, string[] publications);
event PayBill(string topic,uint256 bytesR, uint256 numberR,uint256 amount);
event Payed(uint256 amount, uint256 balances);

constructor() public
{
    sysOwner = msg.sender;
    b.brokerAddress = 0x5b465e496270D1Ae3c90e18E3e85b6AcfBF6b27E;
}

function registerDevice(string memory deviceId, string memory dtype, string memory dtopic)
     public onlyOwner returns(uint index)
{
    if(isRegistred(deviceId))
    {
        for(uint i = 0; i<idToRegisteredClients[deviceId].subscriptions.length; i++)
        {
            require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                subscriptions[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(
                abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber"))));
        }
        for(uint i = 0; i<idToRegisteredClients[deviceId].publications.length; i++)
        {
            require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                publications[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(abi
                .encodePacked(dtype)) == keccak256(abi.encodePacked("publisher"))));
        }
    }
    if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("publisher")))
        {idToRegisteredClients[deviceId].publications.push(dtopic);}

    if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber")))
        {idToRegisteredClients[deviceId].subscriptions.push(dtopic);}

    idToRegisteredClients[deviceId].index = registerIndex.push(deviceId)-1;

    return registerIndex.length-1;
```

```
}

function isRegistred(string memory deviceId) public view returns(bool isIndeed)
{
    if(registerIndex.length == 0) return false;

    return (keccak256(abi.encodePacked(registerIndex[idToRegisteredClients[deviceId].index
        ])) == keccak256(abi.encodePacked(deviceId)));
}

function getDevice(string memory deviceId) public internal returns(string[] memory
    subscriptions, string[] memory publications)
{
    if(!isRegistred(deviceId)) revert("Not registred");

    return(idToRegisteredClients[deviceId].subscriptions, idToRegisteredClients[deviceId].
        publications);
}

function IdentifyDevice(string memory _id, uint8 v, bytes32 r, bytes32 s) public returns (
    address)
{
    if (isRegistred(_id))
    {
        bytes32 recoveredMessage = keccak256(abi.encodePacked("\x19Ethereum Signed Message
            :\n32", keccak256(abi.encodePacked(msg.sender))));
        address recoveredAddress = ecrecover(recoveredMessage, v, r, s);
        if(recoveredAddress == msg.sender)
        {
            (string[] memory subscriptions, string[] memory publications) = getDevice(_id)
                ;
            idTorecoveredAddresss[_id] = recoveredAddress;
            emit Authenticated(_id, subscriptions, publications);
            return(recoveredAddress);
        }
        else
        {
            revert('Could not authenticate device');
        }
    }
    else
        {revert('Not registred');}
}
function saveConnexionTime(string memory _id) public onlyBroker
```

```
{
    b.idToClients[_id].cnxtime = now;
}
function saveConnexionTimeClient(string memory _id) public
{
    idToClient[_id].cnxtime = now;
}
//Echange de donnees
function savePublication(string memory _id,string memory _topic,string memory _msgID,
    bytes32 _dataHash,uint256 _dataSize) public onlyBroker
{
    b.idToClients[_id].topicToMsgPublished[_topic].push(message(_msgID,_dataHash,_dataSize
        ,now));
}
function saveDeliveredPub(string memory _id,string memory _topic,string memory _msgID,
    bytes32 _dataHash,uint256 _dataSize) public onlyBroker
{
    b.idToClients[_id].topicToMsgReceived[_topic].push(message(_msgID,_dataHash,_dataSize,
        now));
}
function publishingMessage(string memory _id,string memory _topic,string memory _msgID,
    bytes32 _dataHash,uint256 _dataSize) public
{
    idToClient[_id].topicToMsgPublished[_topic].push(message(_msgID,_dataHash,_dataSize,
        now));
    topicToPublications[_topic].publications.push(message(_msgID,_dataHash,_dataSize,now))
        ;
    topicToPublications[_topic].totalBytes += _dataSize;
}
function receivedPublication(string memory _id,string memory _topic,string memory _msgID,
    bytes32 _dataHash,uint256 _dataSize) public
{
    idToClient[_id].topicToMsgReceived[_topic].push(message(_msgID,_dataHash,_dataSize,now
        ));
    //Confirmer livraison
    uint i = 0;
    while(i<topicToPublications[_topic].deliveries.length && topicToPublications[_topic].
        deliveries[i].deliveredMsg.dataHash != _dataHash)
        {i++;}
    if(i < topicToPublications[_topic].deliveries.length && topicToPublications[_topic].
        deliveries[i].deliveredMsg.dataHash == _dataHash)
    {
        topicToPublications[_topic].deliveries[i].nbDeliveries += 1;
    }
```

```
        else if(i == topicToPublications[_topic].deliveries.length)
            {topicToPublications[_topic].deliveries.push(delivery(message(_msgID,_dataHash,
                _dataSize,now),1));}
    }

    function testBrokerSubPub(string memory _id, string memory _topic, uint i) internal
    {
                uint k = 0;
                bool verif = false;
                while(k < topicToPublications[_topic].publications.length && verif == false &&
                     idToClient[_id].topicToMsgReceived[_topic][i].t < topicToPublications[
                    _topic].publications[k].t + 1000 && topicToPublications[_topic].
                    publications[k].t <= idToClient[_id].topicToMsgReceived[_topic][i].t)
                {
                    if((idToClient[_id].topicToMsgReceived[_topic][i].dataHash ==
                        topicToPublications[_topic].publications[k].dataHash) && (idToClient[
                        _id].topicToMsgReceived[_topic][i].dataSize == topicToPublications[
                        _topic].publications[k].dataSize))
                    {
                        updateBill(_id,_topic,idToClient[_id].topicToMsgReceived[_topic][i].
                            dataSize,0);
                        verif = true;
                    }
                    else
                    {k++;}
                }
                if (k == topicToPublications[_topic].publications.length && verif == false)
                {
                   testBrokerPublisher(_id,_topic,i);
                }
    }
    function testBrokerPublisher(string memory _id, string memory _topic, uint i) internal
    {
        uint j = 0;
        bool verify = false;
        while(j < topicToPublications[_topic].publications.length && verify == false && b.
            idToClients[_id].topicToMsgReceived[_topic][i].t < topicToPublications[_topic].
            publications[j].t + 1000 && topicToPublications[_topic].publications[j].t <= b.
            idToClients[_id].topicToMsgReceived[_topic][i].t)
        {
            if((b.idToClients[_id].topicToMsgReceived[_topic][i].dataHash ==
                topicToPublications[_topic].publications[j].dataHash) && (b.idToClients[_id].
                topicToMsgReceived[_topic][i].dataSize == topicToPublications[_topic].
                publications[j].dataSize))
```

```
            {
                updateBill(_id,_topic,b.idToClients[_id].topicToMsgReceived[_topic][i].
                    dataSize,1);
                verify = true;
            }
            else {j++;}
        }
    }
    //Verifier les confirmations du subscriber
    function checkSubscriber(string memory _id, string memory _topic, uint256 _startTime,
        uint256 _endTime) public
    {
        resetBill(_id,_topic);
        if(b.idToClients[_id].topicToMsgReceived[_topic].length <= idToClient[_id].
            topicToMsgReceived[_topic].length)
        {
            for(uint i = 0; i<idToClient[_id].topicToMsgReceived[_topic].length; i++)
            {
                while(i < b.idToClients[_id].topicToMsgReceived[_topic].length && idToClient[
                    _id].topicToMsgReceived[_topic][i].t >= _startTime && idToClient[_id].
                    topicToMsgReceived[_topic][i].t < _endTime)
                {
                    if(b.idToClients[_id].topicToMsgReceived[_topic][i].dataHash != idToClient
                        [_id].topicToMsgReceived[_topic][i].dataHash && b.idToClients[_id].
                        topicToMsgReceived[_topic][i].dataSize != idToClient[_id].
                        topicToMsgReceived[_topic][i].dataSize)
                    {
                            testBrokerSubPub(_id,_topic,i);
                    }
                }
                if(i >= b.idToClients[_id].topicToMsgReceived[_topic].length && idToClient[_id
                    ].topicToMsgReceived[_topic][i].t >= _startTime && idToClient[_id].
                    topicToMsgReceived[_topic][i].t < _endTime)
                {
                    uint k = 0;
                    bool verif = false;
                    while(k < topicToPublications[_topic].publications.length && verif ==
                        false && idToClient[_id].topicToMsgReceived[_topic][i].t <
                        topicToPublications[_topic].publications[k].t + 1000 &&
                        topicToPublications[_topic].publications[k].t <= idToClient[_id].
                        topicToMsgReceived[_topic][i].t)
                    {
                        if((idToClient[_id].topicToMsgReceived[_topic][i].dataHash ==
                            topicToPublications[_topic].publications[k].dataHash) && (
```

```
                        idToClient[_id].topicToMsgReceived[_topic][i].dataSize ==
                        topicToPublications[_topic].publications[k].dataSize))
                    {
                        updateBill(_id,_topic,idToClient[_id].topicToMsgReceived[_topic][i
                            ].dataSize,0);
                        verif = true;
                    }
                    else
                    {k++;}
                }
            }
        }
    }
    if(b.idToClients[_id].topicToMsgReceived[_topic].length > idToClient[_id].
        topicToMsgReceived[_topic].length)
    {
        for(uint i = 0; i<idToClient[_id].topicToMsgReceived[_topic].length; i++)
        {
            while(idToClient[_id].topicToMsgReceived[_topic][i].t >= _startTime &&
                idToClient[_id].topicToMsgReceived[_topic][i].t < _endTime)
            {
                if(b.idToClients[_id].topicToMsgReceived[_topic][i].dataHash != idToClient
                    [_id].topicToMsgReceived[_topic][i].dataHash && b.idToClients[_id].
                    topicToMsgReceived[_topic][i].dataSize != idToClient[_id].
                    topicToMsgReceived[_topic][i].dataSize)
                {
                    testBrokerSubPub(_id,_topic,i);
                }
                else
                {updateBill(_id,_topic,idToClient[_id].topicToMsgReceived[_topic][i].
                    dataSize,0);}
            }
        }
        for(uint p = idToClient[_id].topicToMsgReceived[_topic].length+1; p<b.idToClients[
            _id].topicToMsgReceived[_topic].length; p++)
        {
            while(b.idToClients[_id].topicToMsgReceived[_topic][p].t >= _startTime && b.
                idToClients[_id].topicToMsgReceived[_topic][p].t < _endTime)
            {
                testBrokerPublisher(_id,_topic,p);
            }
        }
    }
}
```

```
function testPublisherSubscriber(string memory _id, string memory _topic, uint i) internal
{
    uint k = 0;
    bool v = false;
    while(k < topicToPublications[_topic].deliveries.length && v == false && idToClient[
        _id].topicToMsgPublished[_topic][i].t < topicToPublications[_topic].deliveries[k
        ].deliveredMsg.t && topicToPublications[_topic].deliveries[k].deliveredMsg.t -
        1000 <= idToClient[_id].topicToMsgPublished[_topic][i].t)
    {
        if((idToClient[_id].topicToMsgPublished[_topic][i].dataHash == topicToPublications
            [_topic].deliveries[k].deliveredMsg.dataHash) && (idToClient[_id].
            topicToMsgPublished[_topic][i].dataSize == topicToPublications[_topic].
            deliveries[k].deliveredMsg.dataSize) && topicToPublications[_topic].
            deliveries[k].nbDeliveries > 1)
        {
            updateDeliveriesBill(_id,_topic,idToClient[_id].topicToMsgPublished[_topic][i
                ].dataSize,topicToPublications[_topic].deliveries[k].nbDeliveries,0);
            v = true;
        }
        else
        {k++;}
    }
}
function testPublisherBroker(string memory _id, string memory _topic, uint i) internal
{
    bool verify = false;
    uint j = 0;
        while(j < topicToPublications[_topic].deliveries.length && verify == false && b.
            idToClients[_id].topicToMsgPublished[_topic][i].t < topicToPublications[
            _topic].deliveries[j].deliveredMsg.t && topicToPublications[_topic].
            publications[j].t - 1000 <= b.idToClients[_id].topicToMsgPublished[_topic][i
            ].t)
        {
            if((b.idToClients[_id].topicToMsgPublished[_topic][i].dataHash ==
                topicToPublications[_topic].deliveries[j].deliveredMsg.dataHash) && (b.
                idToClients[_id].topicToMsgPublished[_topic][i].dataSize ==
                topicToPublications[_topic].deliveries[j].deliveredMsg.dataSize) &&
                topicToPublications[_topic].deliveries[j].nbDeliveries > 1)
            {
                updateDeliveriesBill(_id,_topic,b.idToClients[_id].topicToMsgPublished[
                    _topic][i].dataSize,topicToPublications[_topic].deliveries[j].
                    nbDeliveries,1);
                verify = true;
            }
```

```
            else {j++;}
        }
}
function testPubSubBroker(string memory _id, string memory _topic, uint i) internal
{
    uint k = 0;
    bool verif = false;
    while(k < topicToPublications[_topic].deliveries.length && verif == false &&
        idToClient[_id].topicToMsgPublished[_topic][i].t < topicToPublications[_topic].
        deliveries[k].deliveredMsg.t && topicToPublications[_topic].deliveries[k].
        deliveredMsg.t - 1000 <= idToClient[_id].topicToMsgPublished[_topic][i].t)
    {
        if((idToClient[_id].topicToMsgPublished[_topic][i].dataHash == topicToPublications
            [_topic].deliveries[k].deliveredMsg.dataHash) && (idToClient[_id].
            topicToMsgPublished[_topic][i].dataSize == topicToPublications[_topic].
            deliveries[k].deliveredMsg.dataSize) && topicToPublications[_topic].
            deliveries[k].nbDeliveries > 1)
        {
            verif = true;
            updateDeliveriesBill(_id,_topic,idToClient[_id].topicToMsgPublished[_topic][i
                ].dataSize,topicToPublications[_topic].deliveries[k].nbDeliveries,0);
        }
        else
        {k++;}
    }
    if (k == topicToPublications[_topic].deliveries.length && verif == false)
    {
        testPublisherBroker(_id,_topic,i);
    }
}
//Verifier les publications du publisher
function checkPublisher(string memory _id, string memory _topic, uint256 _startTime,
    uint256 _endTime) public
{
    resetBill(_id,_topic);
    if(b.idToClients[_id].topicToMsgPublished[_topic].length <= idToClient[_id].
        topicToMsgPublished[_topic].length)
    {
        for(uint i = 0; i<idToClient[_id].topicToMsgPublished[_topic].length; i++)
        {
            while(i < b.idToClients[_id].topicToMsgPublished[_topic].length && idToClient[
                _id].topicToMsgPublished[_topic][i].t >= _startTime && idToClient[_id].
                topicToMsgPublished[_topic][i].t < _endTime)
            {
```

```
                if(b.idToClients[_id].topicToMsgPublished[_topic][i].dataHash ==
                    idToClient[_id].topicToMsgPublished[_topic][i].dataHash && b.
                    idToClients[_id].topicToMsgPublished[_topic][i].dataSize ==
                    idToClient[_id].topicToMsgPublished[_topic][i].dataSize)
                {
                    testPublisherSubscriber(_id, _topic,i);
                }
                else
                {
                    testPubSubBroker(_id,_topic,i);
                }
            }
            if(i >= b.idToClients[_id].topicToMsgPublished[_topic].length && idToClient[
                _id].topicToMsgPublished[_topic][i].t >= _startTime && idToClient[_id].
                topicToMsgPublished[_topic][i].t < _endTime)
            {
                testPublisherSubscriber(_id,_topic,i);
            }
        }
    }
    if(b.idToClients[_id].topicToMsgPublished[_topic].length > idToClient[_id].
        topicToMsgPublished[_topic].length)
    {
        for(uint i = 0; i<b.idToClients[_id].topicToMsgPublished[_topic].length; i++)
        {
            while(i < idToClient[_id].topicToMsgPublished[_topic].length && idToClient[_id
                ].topicToMsgPublished[_topic][i].t >= _startTime && idToClient[_id].
                topicToMsgPublished[_topic][i].t < _endTime)
            {
                if(b.idToClients[_id].topicToMsgPublished[_topic][i].dataHash ==
                    idToClient[_id].topicToMsgPublished[_topic][i].dataHash && b.
                    idToClients[_id].topicToMsgPublished[_topic][i].dataSize ==
                    idToClient[_id].topicToMsgPublished[_topic][i].dataSize)
                {
                    testPublisherSubscriber(_id, _topic,i);
                }
                else
                {
                    testPubSubBroker(_id,_topic,i);
                }

            }
```

```
            if(i >= idToClient[_id].topicToMsgPublished[_topic].length && b.idToClients[
                _id].topicToMsgPublished[_topic][i].t >= _startTime && b.idToClients[_id
                ].topicToMsgPublished[_topic][i].t < _endTime)
            {
                testPublisherBroker(_id,_topic,i);
            }
        }
    }
}
function resetBill(string memory _id, string memory _topic) public
{
    idToClient[_id].topicToBill[_topic].totalBytes = 0;
    idToClient[_id].topicToBill[_topic].totalNumber = 0;

}
function updateBill(string memory _id,string memory _topic, uint256 _dataSize, uint _verif
    ) public
{
    idToClient[_id].topicToBill[_topic].totalBytes += _dataSize;
    idToClient[_id].topicToBill[_topic].totalNumber += 1;
    idToClient[_id].topicToBill[_topic].verifFees = _verif;
}
function updateDeliveriesBill(string memory _id,string memory _topic, uint256 _dataSize,
    uint256 nbDeliveries,uint _verif) public
{
    updateBill(_id,_topic,_dataSize,_verif);
    idToClient[_id].topicToBill[_topic].numberDeliveries += nbDeliveries;
}
//Lancer la verification
function verificationProcess(string memory _id, uint256 _startTime, uint256 _endTime)
    public
{
    uint256 duration;
    if(_startTime > b.idToClients[_id].cnxtime)
    {
        duration = _endTime - _startTime;
    }
    else
    {
        duration = _endTime - b.idToClients[_id].cnxtime;
    }
    for(uint i = 0; i<idToRegisteredClients[_id].subscriptions.length; i++)
    {
```

```
            checkSubscriber(_id,idToRegisteredClients[_id].subscriptions[i],_startTime,
                _endTime);
            calculateFees(_id,idToRegisteredClients[_id].subscriptions[i], duration);
        }
        for(uint i = 0; i<idToRegisteredClients[_id].publications.length; i++)
        {
            checkPublisher(_id,idToRegisteredClients[_id].publications[i],_startTime,_endTime)
                ;
            calculateIncome(_id,idToRegisteredClients[_id].publications[i],duration);
        }
    }
    function calculateFees(string memory _id, string memory _topic, uint256 duration) public
    {
        uint256 pay = feesperduration*duration + idToClient[_id].topicToBill[_topic].
            totalBytes*feesperbytes + idToClient[_id].topicToBill[_topic].totalNumber*
            feespernumber + registerFees;

        if(idToClient[_id].topicToBill[_topic].verifFees == 0)
        {
            idToClient[_id].topicToBill[_topic].finalAmount = pay;
            idToClient[_id].topicToBill[_topic].paid = false;
            emit PayBill(_topic,idToClient[_id].topicToBill[_topic].totalBytes,idToClient[_id
                ].topicToBill[_topic].totalNumber,idToClient[_id].topicToBill[_topic].
                finalAmount);
        }
        if(idToClient[_id].topicToBill[_topic].verifFees == 1)
        {
            idToClient[_id].topicToBill[_topic].finalAmount = pay + verificationFees;
            idToClient[_id].topicToBill[_topic].paid = false;
            emit PayBill(_topic,idToClient[_id].topicToBill[_topic].totalBytes,idToClient[_id
                ].topicToBill[_topic].totalNumber,idToClient[_id].topicToBill[_topic].
                finalAmount);
        }
    }
    function calculateIncome(string memory _id, string memory _topic, uint256 duration) public
    {
        uint256 pay = payperduration*duration + idToClient[_id].topicToBill[_topic].totalBytes
            *payperbytes + idToClient[_id].topicToBill[_topic].numberDeliveries*paypernumber
            - registerFeesP;
        transferMoneyToPublisher(_id, pay);

    }
    function deposit(string memory _id) public payable
    {
```

```
        balance[idTorecoveredAddresss[_id]] += msg.value;
    }
    function transferMoneyToPublisher(string memory _id, uint256 _pay) public
    {
        uint256 amount = (_pay/100000) * 1000000000000000000;
        address payable wallet = address(uint160(idTorecoveredAddresss[_id]));
        address(wallet).transfer(amount);
        emit Payed(amount, balance[idTorecoveredAddresss[_id]]);
    }
    }
```

2. Trace-MIN

```
pragma solidity >=0.4.22 <0.6.0;
pragma experimental ABIEncoderV2;

contract TraceMinContrat{

    address sysOwner;
    uint256 feesperduration = 0.01*1000000;
    uint256 feespernumber = 0.0009*1000000;
    uint256 registerFees = 0.02*1000000;
    uint256 registerFeesP = 1*1000000;
    uint256 feesperbytes = 0.0001*1000000;
    uint256 verificationFees = 0.1*1000000;
    uint256 payperbytes = 0.000085*1000000;
    uint256 payperduration = 0.005*1000000;

    struct broker
    {
        address brokerAddress;
        mapping (string => topic) topicNameToClients;
        mapping (string => timeUsingSystem) clientToDuration;
        uint256 maxPublicationsReceived;
    }
    struct timeUsingSystem
    {
        uint256 connection;
        uint256 disconnection;
    }
    struct topic
    {
        mapping (string => uint256) idToBytesReceived ;
        mapping (string => uint256) idToBytesSent;
```

```
    uint256 totalBytesPublished;
}
struct RegisteredClients
{
    string id;
    string[] subscriptions;
    string[] publications;
    uint index;
}
struct billSubcriber
{
    uint256 totalBytesReceived;
    uint256 finalAmount;
    bool paid;
}
struct billPublisher
{
    uint256 totalBytesSent;
    uint256 finalAmount;
    bool sent;
}
broker b;

mapping (string => address) idTorecoveredAddresss;

modifier onlyOwner{
    require(msg.sender == sysOwner);
    _;
}
modifier onlyBroker{
    require(msg.sender == b.brokerAddress);
    _;
}
event Authenticated(string id, string[] subscriptions, string[] publications);
event PayBillSubcriber(string id, uint256 bytesR,uint256 amount, bool paid);
event PayBillPublisher(string id, uint256 bytesS,uint256 amount, bool sent);
event Payed(string id, uint256 amount, uint256 balances);

constructor() public
{
    sysOwner = msg.sender;
    b.brokerAddress = 0xFFcf8FDEE72ac11b5c542428B35EEF5769C409f0;
}
```

```
function registerDevice(string memory deviceId, string memory dtype, string memory dtopic)
     public onlyOwner returns(uint index)
{
    if(isRegistred(deviceId))
    {
        for(uint i = 0; i<idToRegisteredClients[deviceId].subscriptions.length; i++)
        {
            require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                subscriptions[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(
                abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber"))));
        }
        for(uint i = 0; i<idToRegisteredClients[deviceId].publications.length; i++)
        {
            require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                publications[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(abi
                .encodePacked(dtype)) == keccak256(abi.encodePacked("publisher"))));
        }
    }
    if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("publisher")))
        {idToRegisteredClients[deviceId].publications.push(dtopic);}

    if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber")))
        {idToRegisteredClients[deviceId].subscriptions.push(dtopic);}

    idToRegisteredClients[deviceId].index = registerIndex.push(deviceId)-1;

    return registerIndex.length-1;
}
function isRegistred(string memory deviceId) public view returns(bool isIndeed)
{
    if(registerIndex.length == 0) return false;
    return (keccak256(abi.encodePacked(registerIndex[idToRegisteredClients[deviceId].index
        ])) == keccak256(abi.encodePacked(deviceId)));
}
function getDevice(string memory deviceId) public view returns(string[] memory
    subscriptions, string[] memory publications)
{
    if(!isRegistred(deviceId)) revert("Not registred");
    return(idToRegisteredClients[deviceId].subscriptions, idToRegisteredClients[deviceId].
        publications);
}
function authenticateDevice(string memory _id, uint8 v, bytes32 r, bytes32 s) public
    returns (address)
{
```

```
        if (isRegistred(_id))
        {
            bytes32 messag = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32",
                keccak256(abi.encodePacked(msg.sender))));
            address recoveredAddress = ecrecover(messag, v, r, s);
            if(recoveredAddress == msg.sender)
            {
                (string[] memory subscriptions, string[] memory publications) = getDevice(_id)
                    ;
                idTorecoveredAddresss[_id] = recoveredAddress;
                emit Authenticated(_id, subscriptions, publications);
                return(recoveredAddress);
            }
            else
            {
                revert('Could not authenticate');
            }
        }
        else
            {revert('Not registred');}
    }
    function saveConnexionTime(string memory _id) public onlyBroker
    {
        b.clientToDuration[_id].connection = now;
    }
    function savePublication(string memory _id,string memory _topic,uint256 _dataSize) public
        onlyBroker
    {
        b.topicNameToClients[_topic].idToBytesSent[_id] += _dataSize;
        idPublisherToBill[_id].totalBytesSent += _dataSize;
        b.topicNameToClients[_topic].totalBytesPublished += _dataSize;
    }
    function saveDeliveredPub(string memory _id,string memory _topic,uint256 _dataSize) public
         onlyBroker
    {
        b.topicNameToClients[_topic].idToBytesReceived[_id] += _dataSize;
        idSubscriberToBill[_id].totalBytesReceived += _dataSize;
    }
     function verificationProcess(uint256 _startTime, uint256 _endTime) public onlyBroker
    {
        for(uint256 j = 0; j<registerIndex.length; j++)
        {
            uint256 duration;
      uint256 totalBytesChecked;
```

```
        if(_startTime > b.clientToDuration[registerIndex[j]].connection)
        {
            duration = _endTime - _startTime;
        }
        else {duration = _endTime - b.clientToDuration[registerIndex[j]].connection;}

    if(idSubscriberToBill[registerIndex[j]].totalBytesReceived > 0)
        {
      for(uint i = 0; i<idToRegisteredClients[registerIndex[j]].subscriptions.length; i++)
      {
        string topic = idToRegisteredClients[registerIndex[j]].subscriptions[i];
        if( b.topicNameToClients[_topic].idToBytesReceived[registerIndex[j]] == b.
            topicNameToClients[topic].totalBytesPublished)
        {
          totalBytesChecked += b.topicNameToClients[_topic].idToBytesReceived[
              registerIndex[j]];
        }
      }
      uint256 bill = feesperduration*duration + idSubscriberToBill[registerIndex[j]].
          totalBytesReceived*feesperbytes + registerFees;
      idSubscriberToBill[registerIndex[j]].finalAmount += bill;
      idSubscriberToBill[registerIndex[j]].paid = false;
      emit PayBillSubcriber(registerIndex[j],idSubscriberToBill[registerIndex[j]].
          totalBytesReceived,idSubscriberToBill[registerIndex[j]].finalAmount,
          totalBytesChecked);
    }
    if(idPublisherToBill[registerIndex[j]].totalBytesSent > 0)
    {
        uint256 pay = payperduration*duration + idPublisherToBill[registerIndex[j]].
            totalBytesSent*payperbytes + registerFeesP;
        idPublisherToBill[registerIndex[j]].finalAmount += pay;
        idPublisherToBill[registerIndex[j]].sent = false;
        emit PayBillPublisher(registerIndex[j], idPublisherToBill[registerIndex[j]].
            totalBytesSent, idPublisherToBill[registerIndex[j]].finalAmount,
            idPublisherToBill[registerIndex[j]].sent);
    }
      }
      resetBill();
}
function resetBill() internal
{
    for(uint256 j = 0; j < registerIndex.length; j++)
    {
        delete idSubscriberToBill[registerIndex[j]].totalBytesReceived;
```

```
            delete idPublisherToBill[registerIndex[j]].totalBytesSent;
        }
    }
    function deposit(string memory _id) public payable
    {
        idSubscriberToBill[_id].paid = true;
        balance[idTorecoveredAddresss[_id]] += msg.value;
        delete idSubscriberToBill[_id].finalAmount;
    }
    function transferMoneyToPublisher(string memory _id, uint256 _pay) public
    {
        address payable wallet = address(uint160(idTorecoveredAddresss[_id]));
        address(wallet).transfer(_pay);
        idPublisherToBill[_id].sent = true;
        delete idPublisherToBill[_id].finalAmount;
        emit Payed(_id,_pay, balance[idTorecoveredAddresss[_id]]);
    }
    }
```

3. Trace-BF

```
pragma solidity >=0.4.22 <0.6.0;
pragma experimental ABIEncoderV2;

contract TraceBFContract
{
    address sysOwner;
    uint256 feespernumber = 0.09*100000;
    uint256 registerFees = 0.2*100000;
    uint256 paypernumber = 0.06*100000;
    uint256 registerFeesP = 0.1*100000;

    struct BloomFilter
    {
        uint256 nbHash;
        uint256 maxElements;
        uint256 BFnbrElements;
        uint256 sizeBit;
        uint256 sizeBytes;
    }
    struct broker
    {
        address brokerAddress;
        uint256 maxClientsConnected;
```

```
        uint256 maxPublicationsReceived;
        mapping(string => uint256) idsubscriberToNbReceptions;
        mapping(string => uint256) idpublisherToNbDeliveries;
    }
    struct topic
    {
        mapping (bytes32 => string[]) topicToSubscribers;
        mapping (string => uint) subscriberToIndex;
    }
    struct RegisteredClients
    {
        string id;
        string[] subscriptions;
        string[] publications;
        uint index;
    }
    struct bill
    {
        uint256 totalNumberReceived;
        uint256 totalNumberSent;
        uint256 finalAmount;
        bool paid;
        bool received;
    }

    broker b;
    BloomFilter bloomFilter;
    bytes32[] brokerBF;
    bytes32[] publishersBF;
    bytes32[] subscribersBF;
    topic subscribersList;
    string[] private registerIndex;

    mapping (string => address) private idTorecoveredAddresss;
    mapping(string => RegisteredClients) private idToRegisteredClients;
    mapping(string => bill) idToBill;
    mapping(address => uint256) balance;

    modifier onlyBroker{
        require(msg.sender == b.brokerAddress);
        _;
    }
    modifier onlyOwner{
        require(msg.sender == sysOwner);
```

```
    _;
}
event Authenticated(string id, string[] subscriptions, string[] publications);
event BFlimit(uint256 BFnbrElements, bool stop);
event PayBillPublisher(string id, uint256 amount);
event Payed(string _id, uint256 pay,uint256 amount, uint256 balances);
//event VerifyNumberElements(uint256 publishers, uint256 subscribers, uint256 Broker);

constructor() public
{
    sysOwner = msg.sender;
    b.brokerAddress = 0xeB065bE5f37A4C1E5eB20451aEF6f16cEAf07Fbd;
    setBloomFilters();
}
function setBloomFilters() internal
{
    bloomFilter.nbHash = 7;
    bloomFilter.maxElements = 10000;
    bloomFilter.BFnbrElements = 0;
    bloomFilter.sizeBit = 96000 ;
    bloomFilter.sizeBytes = 375;
    bytes32[] memory array = new bytes32[](bloomFilter.sizeBytes);
    subscribersBF = array;
    bytes32[] memory array1 = new bytes32[](bloomFilter.sizeBytes);
    publishersBF = array1;
    bytes32[] memory array2 = new bytes32[](bloomFilter.sizeBytes);
    brokerBF = array2;
}
function isRegistred(string memory deviceId) public view returns(bool isIndeed)
{
    if(registerIndex.length == 0) return false;
    return (keccak256(abi.encodePacked(registerIndex[idToRegisteredClients[deviceId].index
        ])) == keccak256(abi.encodePacked(deviceId)));
}
function registerDevice(string memory deviceId, string memory dtype, string memory dtopic)
     public returns(uint index)
{
    if(isRegistred(deviceId))
    {
        for(uint i = 0; i<idToRegisteredClients[deviceId].subscriptions.length; i++)
        {
            require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                subscriptions[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(
                abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber"))));
```

```
            }
            for(uint i = 0; i<idToRegisteredClients[deviceId].publications.length; i++)
            {
                require(!(keccak256(abi.encodePacked(idToRegisteredClients[deviceId].
                    publications[i])) == keccak256(abi.encodePacked(dtopic)) && keccak256(abi
                    .encodePacked(dtype)) == keccak256(abi.encodePacked("publisher"))));
            }
        }
        if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("publisher")))
            {idToRegisteredClients[deviceId].publications.push(dtopic);}

        if(keccak256(abi.encodePacked(dtype)) == keccak256(abi.encodePacked("subscriber")))
            {idToRegisteredClients[deviceId].subscriptions.push(dtopic);}

        idToRegisteredClients[deviceId].index = registerIndex.push(deviceId)-1;

        return registerIndex.length-1;
    }
    function getDevice(string memory deviceId) public view returns(string[] memory
        subscriptions, string[] memory publications)
    {
        if(!isRegistred(deviceId)) revert("Not registred");
        return(idToRegisteredClients[deviceId].subscriptions, idToRegisteredClients[deviceId].
            publications);
    }
    function authenticateDevice(string memory _id, uint8 v, bytes32 r, bytes32 s) public
        returns (address)
    {
        if (isRegistred(_id))
        {
            bytes32 recoveredMessage = keccak256(abi.encodePacked("\x19Ethereum Signed Message
                :\n32", keccak256(abi.encodePacked(msg.sender))));
            address recoveredAddress = ecrecover(recoveredMessage, v, r, s);
            if(recoveredAddress == msg.sender)
            {
                (string[] memory subscriptions, string[] memory publications) = getDevice(_id)
                    ;
                idTorecoveredAddresss[_id] = recoveredAddress;
                emit Authenticated(_id, subscriptions, publications);
                return(recoveredAddress);
            }
            else
            {
                revert('Could not authenticate');
```

```
        }
    }
    else
        {revert('Not registred');}
}
function subscribe(string memory _id, string memory _topic) public
{
    subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))].push(_id);
    subscribersList.subscriberToIndex[_id] = subscribersList.topicToSubscribers[keccak256(
        abi.encodePacked(_topic))].length - 1;
}
function unsubscribe(string memory _id, string memory _topic) public
{
    for(uint i = subscribersList.subscriberToIndex[_id]; i<subscribersList.
        topicToSubscribers[keccak256(abi.encodePacked(_topic))].length-1; i++)
    {
        subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))][i] =
            subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))][i+1];
    }
    delete subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))][
        subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))].length
        -1];
    subscribersList.topicToSubscribers[keccak256(abi.encodePacked(_topic))].length--;
}
function incrementNbdelivered(string memory _id,string memory _topic) internal
{
    for(uint256 i = 0; i < subscribersList.topicToSubscribers[keccak256(abi.encodePacked(
        _topic))].length; i++)
    {
        b.idsubscriberToNbReceptions[subscribersList.topicToSubscribers[keccak256(abi.
            encodePacked(_topic))][i]] += 1;
        b.idpublisherToNbDeliveries[_id] += 1;
    }
}
function addElementB(string memory _id,string memory _topic, uint256[] memory posTab,
    uint256[] memory posByte) public
{
    if(bloomFilter.BFnbrElements < bloomFilter.maxElements)
    {
        bloomFilter.BFnbrElements += 1;

        for(uint i = 0; i < bloomFilter.nbHash; i++)
        {
            bytes32 leByte = brokerBF[posTab[i]];
```

```
            uint256 shiftPos = 1 << posByte[i];
            bytes32 L = leByte | bytes32(shiftPos);
            brokerBF[posTab[i]] = L;
        }
        if(bloomFilter.BFnbrElements == bloomFilter.maxElements)
        {
            emit BFlimit(bloomFilter.BFnbrElements, true);
        }

    }
    incrementNbdelivered(_id,_topic);
}
function addElementS(uint256[] memory posTab, uint256[] memory posByte) public
{
    if(bloomFilter.BFnbrElements <= bloomFilter.maxElements)
    {
        for(uint i = 0; i < bloomFilter.nbHash; i++)
        {
            bytes32 leByte = subscribersBF[posTab[i]];
            uint256 shiftPos = 1 << posByte[i];
            bytes32 L = leByte | bytes32(shiftPos);
            subscribersBF[posTab[i]] = L;
        }
    }
}
function addElementP(uint256[] memory posTab, uint256[] memory posByte) public
{
    if(bloomFilter.BFnbrElements < bloomFilter.maxElements)
    {
        for(uint i = 0; i < bloomFilter.nbHash; i++)
        {
            bytes32 leByte = publishersBF[posTab[i]];
            uint256 shiftPos = 1 << posByte[i];
            bytes32 L = leByte | bytes32(shiftPos);
            publishersBF[posTab[i]] = L;
        }
    }
}
function checkingBloomFilters() public returns (uint256, uint256,uint256)
{
    uint256 nbrofOnesBr = 0;
    uint256 nbrofOnesPb = 0;
    uint256 nbrofOnesSb = 0;
    uint256 PubBr = 0;
```

```
uint256 PubSub = 0;
uint256 SubBr = 0;

for(uint256 i = 0; i < bloomFilter.sizeBytes; i++)
{
    if((publishersBF[i] ^ brokerBF[i]) == 0)
    {
        if(publishersBF[i] ^ subscribersBF[i] == 0)
        {
            for(uint256 j = 0; j < 32; j++)
            {
                bool bitPSet = (uint256(publishersBF[i]) & (1 << j) == 1 << j);
                if(bitPSet)
                {
                    nbrofOnesBr += 1;
                    nbrofOnesPb += 1;
                    nbrofOnesSb += 1;
                    PubBr += 1;
                    PubSub += 1;
                    SubBr += 1;
                }
            }
        }
        else
        {
            for(uint256 j = 0; j < 32; j++)
            {
                bool bitPSet = (uint256(publishersBF[i]) & (1 << j) == 1 << j);
                bool bitSSet = (uint256(subscribersBF[i]) & (1 << j) == 1 << j);
                if(bitPSet)
                {
                    nbrofOnesBr += 1;
                    nbrofOnesPb += 1;
                    PubBr += 1;
                    if(bitSSet)
                    {
                        nbrofOnesSb += 1;
                        PubSub += 1;
                        SubBr += 1;
                    }
                }
                if(bitPSet == false && bitSSet == true)
                {
                    nbrofOnesSb += 1;
```

```
                }
            }

        }
    }
    else
    {
        if(publishersBF[i] ^ subscribersBF[i] == 0)
        {
            for(uint256 j = 0; j < 32; j++)
            {
                bool bitPSet = (uint256(publishersBF[i]) & (1 << j) == 1 << j);
                bool bitBSet = (uint256(brokerBF[i]) & (1 << j) == 1 << j);
                if(bitPSet)
                {
                    nbrofOnesSb += 1;
                    nbrofOnesPb += 1;
                    PubSub += 1;
                    if(bitBSet)
                    {
                        nbrofOnesBr += 1;
                        PubBr += 1;
                        SubBr += 1;
                    }
                }
                if(bitPSet == false && bitBSet == true)
                    {
                        nbrofOnesBr += 1;
                    }
                }
        }
        else if(brokerBF[i] ^ subscribersBF[i] == 0)
        {
            for(uint256 j = 0; j < 32; j++)
            {
                bool bitPSet = (uint256(publishersBF[i]) & (1 << j) == 1 << j);
                bool bitBSet = (uint256(brokerBF[i]) & (1 << j) == 1 << j);
                if(bitBSet)
                {
                    nbrofOnesSb += 1;
                    nbrofOnesBr += 1;
                    SubBr += 1;
                    if(bitPSet)
                    {
```

```
                                nbrofOnesPb += 1;
                                PubSub += 1;
                                PubBr += 1;
                            }
                        }
                        if(bitBSet == false && bitPSet == true)
                        {
                                nbrofOnesPb += 1;
                        }
                    }
                }
            }
        }
        BP_DiceCoef = (2*PubBr/(nbrofOnesPb+nbrofOnesBr));
        BS_DiceCoef = (2*SubBr/(nbrofOnesSb+nbrofOnesBr));
        PS_DiceCoef = (2*PubSub/(nbrofOnesSb+nbrofOnesPb));
        return(BP_DiceCoef,BS_DiceCoef, PS_DiceCoef);
    }
    function monetizing(int256 BP_DiceCoef, uint256 BS_DiceCoef, uint256 PS_DiceCoef) public
    {
        if(BP_DiceCoef == 1 || BS_DiceCoef == 1 || (BP_DiceCoef == 0 && BS_DiceCoef == 0 &&
            PS_DiceCoef == 0))
        {
            for(uint256 i = 0; i < registerIndex.length; i++)
            {
                if(b.idsubscriberToNbReceptions[registerIndex[i]] > 0)
                {
                    idToBill[registerIndex[i]].totalNumberReceived += b.
                        idsubscriberToNbReceptions[registerIndex[i]];
                    calculateFees(registerIndex[i]);
                }
                if(b.idpublisherToNbDeliveries[registerIndex[i]] > 0)
                {
                    idToBill[registerIndex[i]].totalNumberSent += b.idpublisherToNbDeliveries[
                        registerIndex[i]];
                    uint256 pay = idToBill[registerIndex[i]].totalNumberSent*paypernumber +
                        registerFeesP;
                    idToBill[registerIndex[i]].received = false;
                    emit PayBillPublisher(registerIndex[i], pay);
                }
            }
        }
        else if(PS_DiceCoef == 1 && BP_DiceCoef == 0 && BS_DiceCoef == 0)
        {
```

```
        for(uint256 i = 0; i < registerIndex.length; i++)
        {
            if(b.idpublisherToNbDeliveries[registerIndex[i]] > 0)
            {
                idToBill[registerIndex[i]].totalNumberSent += b.idpublisherToNbDeliveries[
                    registerIndex[i]];
                uint256 pay = idToBill[registerIndex[i]].totalNumberSent*paypernumber +
                    registerFeesP;
                idToBill[registerIndex[i]].received = false;
                emit PayBillPublisher(registerIndex[i], pay);
                //calculateIncomes(registerIndex[i]);
            }
        }
    }
    resetBF();
    emit BFlimit(bloomFilter.BFnbrElements,false);
}
function resetBF() internal
{
    bloomFilter.BFnbrElements = 0;
    for(uint256 i = 0; i < registerIndex.length; i++)
    {
        b.idsubscriberToNbReceptions[registerIndex[i]] = 0;
        b.idpublisherToNbDeliveries[registerIndex[i]] = 0;
    }
    for(uint256 i = 0; i < bloomFilter.sizeBytes; i++)
    {
        delete brokerBF[i];
        delete publishersBF[i];
        delete subscribersBF[i];
    }
}
function calculateFees(string memory _id) public
{
    uint256 pay = idToBill[_id].totalNumberReceived*feespernumber + registerFees;
    idToBill[_id].finalAmount += pay;
    idToBill[_id].paid = false;
    emit PayBill(_id,idToBill[_id].totalNumberReceived,idToBill[_id].finalAmount);

}
function calculateIncomes(string memory _id) internal
{
    uint256 pay = idToBill[_id].totalNumberSent*paypernumber + registerFeesP;
```

```
        idToBill[_id].received = false;
        transferMoneyToPublisher(_id, pay);
    }
    function deposit(string memory _id) public payable
    {
        idToBill[_id].paid = true;
        balance[idTorecoveredAddresss[_id]] += msg.value;
    }
    function transferMoneyToPublisher(string memory _id, uint256 _pay) public
    {
        address payable wallet = address(uint160(idTorecoveredAddresss[_id]));
        wallet.transfer(_pay);
        idToBill[_id].received = true;
        emit Payed(_id, _pay);
    }
    }
```

RÉFÉRENCES

Ahlgren, B., Hidell, M. & Ngai, E. C.-H. (2016). Internet of things for smart cities : Interoperability and open data. *IEEE Internet Computing*, 20(6), 52–56.

Albahri, O. et al. (2018). Real-time remote health-monitoring Systems in a Medical Centre : A review of the provision of healthcare services-based body sensor information, open challenges and methodological aspects. *Journal of medical systems*, 42(9), 164.

Atmoko, R., Riantini, R. & Hasin, M. (2017). IoT real time data acquisition using MQTT protocol. *Journal of Physics : Conference Series*, 853(1), 012003.

Bajoudah, S., Dong, C. & Missier, P. Toward a Decentralized, Trust-less Marketplace for Brokered IoT Data Trading using Blockchain.

BDEX. Data Exchange Platform, Buy Consumer Data to Empower Human Connectivity. Repéré à https://www.bdex.com/.

Behnke, K. & Janssen, M. (2019). Boundary conditions for traceability in food supply chains using blockchain technology. *International Journal of Information Management*.

Bröring, A., Schmid, S., Schindhelm, C.-K., Khelil, A., Käbisch, S., Kramer, D., Le Phuoc, D., Mitic, J., Anicic, D. & Teniente, E. (2017). Enabling IoT ecosystems through platform interoperability. *IEEE software*, 34(1), 54–61.

Chan, H. & Williams, N. (2019). Three ways that blockchain can improve the quality of market research. Repéré à https://www.researchworld.com/three-ways-that-blockchain-can-improve-the-quality-of-market-research/.

Chang, T. & Meling, H. (2012). Byzantine fault-tolerant publish/subscribe : A cloud computing infrastructure. *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 454–456.

Choi, S., Ghinita, G. & Bertino, E. (2010). A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations. *International Conference on Database and Expert Systems Applications*, pp. 368–384.

Chooruang, K. & Mangkalakeeree, P. (2016). Wireless heart rate monitoring system using MQTT. *Procedia Computer Science*, 86, 160–163.

Dasaklis, T. K., Casino, F. & Patsakis, C. (2019). Defining granularity levels for supply chain traceability based on IoT and blockchain. *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, pp. 184–190.

Dawex. Sell, buy and share data. Repéré à https://www.dawex.com/.

Dobreva, V. & Albutiu, M.-C. (2010). Put all eggs in one basket : an OLTP and OLAP database approach for traceability data. *Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, pp. 31–36.

Draskovic, D. & Saleh, G. (2017). *Decentralized data marketplace based on blockchain*. Repéré à https://www.datapace.io/datapace_whitepaper.pdf.

Esposito, C. & Ciampi, M. (2014). On security in publish/subscribe services : A survey. *IEEE Communications Surveys & Tutorials*, 17(2), 966–997.

ETHGasStation. ETH Gas Station. Repéré à https://ethgasstation.info/.

Eugster, P. T., Felber, P. A., Guerraoui, R. & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2), 114–131.

Foundation, I. (2020). Trinity Attack Incident Part 1 : Summary and next steps. IOTA. Repéré à https://blog.iota.org/trinity-attack-incident-part-1-summary-and-next-steps-8c7ccc4d81e8.

Francis, J. & Merli, M. (2016). Open-sourcing pulsar, pub-sub messaging at scale.

Geravand, S. & Ahmadi, M. (2013). Bloom filter applications in network security : A state-of-the-art survey. *Computer Networks*, 57(18), 4047–4064.

Greve, A. (2019). IOTA Announces Coordicide Solution. IOTA. Repéré à https://blog.iota.org/coordicide-e039fd43a871.

Heilman, E. et al. (2019). Cryptanalysis of Curl-P and Other Attacks on the IOTA Cryptocurrency. *IACR Cryptology ePrint Archive*, 2019, 344.

Hurst, T. Bloom Filter Calculator. Repéré à https://hur.st/bloomfilter/?n=&p=1.0E-2&m=96000&k=7.

Insights, D. (2019). Deloitte's 2019 Global Blockchain Survey. *Blockchain Gets Down to Business. Deloitte*.

Javaid, A. et al. (2019). Reputation System for IoT Data Monetization Using Blockchain. *International Conference on Broadband and Wireless Computing, Communication and Applications*, pp. 173–184.

Katsikeas, S. et al. (2017). Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol. *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1193–1200.

Kreps, J. et al. (2011). Kafka : A distributed messaging system for log processing. *Proceedings of the NetDB*, pp. 1–7.

Krishnamachari, B., Power, J., Kim, S. H. & Shahabi, C. (2018). I3 : An iot marketplace for smart communities. *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 498–499.

Krishnan, S. & Gonzalez, J. L. U. (2015). Google cloud pub/sub. Dans *Building Your Next Big Thing with Google Cloud Platform* (pp. 277–292). Springer.

Küsters, R., Rausch, D. & Simon, M. Accountability in a Permissioned Blockchain : Formal Analysis of Hyperledger Fabric (Full Version).

Lessmann, S., Haupt, J., Coussement, K. & De Bock, K. W. (2019). Targeting customers for profit : An ensemble learning framework to support marketing decision-making. *Information Sciences*.

Lin, J., Shen, Z., Zhang, A. & Chai, Y. (2018). Blockchain and iot based food traceability for smart agriculture. *Proceedings of the 3rd International Conference on Crowd Science and Engineering*, pp. 3.

Mišura, K. & Žagar, M. (2016). Data marketplace for Internet of Things. *2016 International Conference on Smart Systems and Technologies (SST)*, pp. 255–260.

Moscajs. moscajs/mosca. Repéré à https://github.com/moscajs/mosca/wiki/Authentication-&-Authorization.

MQTT. Message Queuing Telemetry Transport. Repéré à http://mqtt.org/.

Nakamoto, S. et al. (2008). Bitcoin : A peer-to-peer electronic cash system.

Onica, E., Felber, P., Mercier, H. & Rivière, E. (2016). Confidentiality-preserving publish/subscribe : A survey. *ACM computing surveys (CSUR)*, 49(2), 1–43.

Özyilmaz, K. R., Doğan, M. & Yurdakul, A. (2018). IDMoB : IoT Data Marketplace on Blockchain. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 11–19.

Palatinus, M., Rusnak, P., Voisine, A. & Bowe, S. (2019). BIP39 : Mnemonic code for generating deterministic keys, 2013. *URL https ://github. com/bitcoin/bips/blob/master/bip-0039. mediawiki. Last visited December*, 10.

Pooja, S., Uday, D., Nagesh, U. & Talekar, S. G. (2017). Application of MQTT protocol for real time weather monitoring and precision farming. *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pp. 1–6.

Quan, W., Xu, C., Guan, J., Zhang, H. & Grieco, L. A. (2013). Scalable name lookup with adaptive prefix bloom filter for named data networking. *IEEE Communications Letters*, 18(1), 102–105.

Ramachandran, G. S., Wright, K.-L., Zheng, L., Navaney, P., Naveed, M., Krishnamachari, B. & Dhaliwal, J. (2019). Trinity : A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence. *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 227–235.

Randall, S. M., Ferrante, A. M., Boyd, J. H., Bauer, J. K. & Semmens, J. B. (2014). Privacy-preserving record linkage on large real world datasets. *Journal of biomedical informatics*, 50, 205–212.

Redis. Redis Pub/Sub. Repéré à https://redis.io/topics/pubsub.

Saint-Andre, P. (2011). *Extensible messaging and presence protocol (XMPP) : Address Format*.

Sennoun, Y. (2018). Internet des Objets : Quels protocoles applicatifs utiliser ? (1/2). Publicis Sapient Engineering - Engineering Done Right. Repéré à https://blog.engineering.publicissapient.fr/2018/04/16/internet-des-objets-quels-protocoles-applicatifs-utiliser-1-2/.

Sharma, Y. et al. (2015). Wormhole : Reliable pub-sub to support geo-replicated internet services. *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 351–366.

Suliman, A., Husain, Z., Abououf, M., Alblooshi, M. & Salah, K. (2018). Monetization of IoT data using smart contracts. *IET Networks*, 8(1), 32–37.

Tang, D., Fujii-Hwang, J., Colwill, E., Arora, S., Shah, A., Mendizabal, A. & Callejas, R. (2018). Value of data : the dawn of the data marketplace.[online] Accenture.

Vinoski, S. (2006). Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), 87–89.

Wang, S., Li, D., Zhang, Y. & Chen, J. (2019). Smart contract-based product traceability system in the supply chain scenario. *IEEE Access*, 7, 115122–115133.

Yassein, M. B., Shatnawi, M. Q., Aljwarneh, S. & Al-Hatmi, R. (2017). Internet of Things : Survey and open issues of MQTT protocol. *2017 International Conference on Engineering & MIS (ICEMIS)*, pp. 1–6.

Yusuf, Z. et al. (2020). Unleashing the Power of Data with IoT and Augmented Reality : BCG. Repéré à https://www.bcg.com/publications/2020/unleashing-the-power-of-data-with-iot-and-augmented-reality.aspx.

Zupan, N., Zhang, K. & Jacobsen, H.-A. (2017). Hyperpubsub : a decentralized, permissioned, publish/subscribe service using blockchains. *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference : Posters and Demos*, pp. 15–16.