SUPRA, un protocole publish/subscribe distribué

Jean-Philippe Abegg, Quentin Bramas, Thomas Noel

SUPRA, un protocole publish/subscribe distribué

Jean-Philippe ABEGG[1], Quentin BRAMAS[1] et Thomas NOEL[1]

[1] *ICUBE, Université de Strasbourg, France*

Publish-subscribe est un paradigme de communication utilisé pour partager de l'information. Dans les protocoles utilisant ce paradigme, l'intégralité des messages passe par le *broker*. Ce dernier est au coeur d'une topologie en étoile, et donc une source de problèmes de confiance dans le système. Il existe des propositions de protocoles *publish-subscribe* utilisant une blockchain pour corriger ces problèmes de confiance. Ces solutions ont une utilisation intensive de la blockchain, ce qui les rend coûteuses sur le long terme, à cause des frais de transactions.

Dans cet article, nous présentons SUPRA, un protocole *publish-subscribe* distribué. Il possède les mêmes garanties de confiance que d'autres solutions se reposant sur la blockchain, mais en réduisant le nombre de messages envoyés dans le registre. De plus, nous présentons une amélioration du protocole pour réduire le nombre de signatures réalisées par le *publisher* pour partager des données.

Mots-clefs : blockchain, protocole publish/subscribe, tolérance aux pannes, synchronisation

1 Introduction

Le modèle *publish-subscribe* est un paradigme de communication. Ce modèle est plus économe en ressources et s'adapte mieux à des communications entre de nombreux noeuds que le modèle *request-reply* [EFGK03]. Il y a différents types de protocoles *publish-subscribe* et nous allons nous concentrer sur les protocoles *publish-subscribe* utilisant les *topics*. Un *topic* est un identifiant, representé par une chaîne de caractères, associé à la donnée crée par le *publisher*. Dans ces protocoles, le *subscriber* annonce son intérêt pour les données associées à un *topic*. Lorsqu'une nouvelle donnée de ce *topic* est générée par le *publisher*, alors la donnée est envoyée au *subscriber*. Les communications de ce modèle sont unidirectionnelles, du *publisher* vers les *subscribers*.

Le *broker* est l'entité centrale de ce modèle, car elle est chargée de transférer les messages du *publisher* vers les *subscribers*. Ce tier de confiance crée des problèmes de sécurité dans le modèle et il existe plusieurs protocoles remplaçant ce broker par une blockchain [RWZ+19, ABBN21].

Travaux connexes. Trinity [RWZ+19] est, à notre connaissance, le premier protocole distribué *publish-subscribe* utilisant la blockchain. Dans cette proposition, il y a plusieurs *brokers* chacun étant un nœud d'un réseau blockchain. Le *publisher* signe les données qu'il crée, et il les envoie à son *broker* (en lequel il a toute confiance). Les données seront par la suite ajoutées dans un bloc de la chaîne. Une fois le bloc crée, les autres *brokers* envoient les données à leurs *subscribers* locaux abonnés à ces données. La blockchain permet ici d'avoir une confiance complète dans la donnée et la communication. Les données sont signées par le *publisher* pour confirmer leur origine, le *broker* ne possède pas la clé privée du *publisher*. La blockchain fournie aussi un ordonnancement total et immuable pour les données.

Le problème de cette solution est l'usage intensif de la blockchain. Chaque donnée est envoyée sur la blockchain par le biais d'une transaction. Cette dernière possède un coût en jetons spécifique à la blockchain utilisée. Envoyer un grand nombre de données augmente ce coût en jetons qui doit être payé par le *publisher* ou le *broker* associé. De plus, les données sur la blockchain sont conservées durant toute la durée de vie de la blockchain. Au fur et à mesure des publications, la taille de la blockchain va croître, ce qui va augmenter les ressources de stockage nécessaires pour les noeuds.

2 SUPRA

SUPRA [ABBN21] est un protocole distribué publish/subscribe utilisant la blockchain. À l'inverse des autres propositions, ce protocole essaie de réduire au maximum le nombre de messages envoyés sur la

blockchain, qu'on appelle *on-chain*. Ceci afin de réduire le coût des frais de transaction et l'impact du temps de validation des transactions sur les performances du protocole. De plus, le protocole souhaite garantir, pour le publisher, la livraison des messages avant un délai T depuis le premier envoi et, pour le subscriber, l'ordre et l'origine des messages. L'intégralité des messages est horodatée et signée par la source. De plus, les clés publiques pour vérifier les signatures sont enregistrées dans la blockchain.

Pour les communications unidirectionnelles entre le publisher et le subscriber, le protocole utilise deux canaux de communication : un lien *off-chain*, qui est une connexion non-fiable entre le publisher et le subscriber (ne passant pas par la blockchain), et un lien *on-chain* qui est une connexion fiable à travers une blockchain (c'est-à-dire que le publisher, resp. le subscriber, est soit connecté de manière fiable à un nœud blockchain, soit est lui-même un nœud blockchain). Pour envoyer un message, le publisher utilise en premier lieu le lien off-chain et attend de recevoir un acquittement de la part du subscriber. L'acquittement du message est une preuve pour le publisher que le message a été reçu dans les temps. En cas de conflit, cette preuve peut être envoyée à un tiers qui sera alors capable de vérifier la bonne réception du message en utilisant uniquement des données publiques. En l'occurrence, les clés publiques utilisées pour signer les messages sont dans la blockchain, et l'acquittement contient la signature du message acquitté.

Si, après l'envoie d'une donnée au subscriber, l'acquittement ou le message d'origine sont perdus, le publisher utilise alors le canal on-chain, c'est-à-dire, envoie le message dans la blockchain pour qu'il soit inclu dans un bloc avant l'écoulement du délai T. La présence du message dans un bloc est une preuve de sa réception par le subscriber, car le protocole suppose que les utilisateurs sont connectés au réseau blockchain. Avec ce système, on réduit le nombre de messages envoyé dans la blockchain par rapport aux autres propositions de protocole.

Pour détecter les messages perdus, ou désordonnés, chaque message envoyé par le publisher aux subscribers est chaîné au précédent en répétant sa signature. Si le message répète une signature inconnue, le subscriber détecte alors une perte. À cause du réseau, le message manquant peut être retardé, mais si le subscriber attend un délai T et n'est toujours pas capable de trouver le message manquant, alors il sait que le publisher n'a pas respecté le protocole. En effet, le message manquant devrait au minimum être présent dans la blockchain.

En montrant à un tiers (ou même un smart-contract) le message utilisé pour détecter l'erreur et le dernier message correct envoyé par le publisher, le subscriber peut prouver que le publisher n'a pas respecté le protocole. Le tiers peut vérifier que les signatures ne correspondent pas, et que les messages manquants ne sont pas dans la blockchain. Pour éviter les fausses accusations, le tiers doit accorder un délai au publisher pour présenter une preuve de réception des messages manquants : un acquittement venant du subscriber. Pour que ce processus de détection fonctionne, il est important pour le subscriber de ne pas envoyer d'acquittement avant la réception de l'intégralité des messages précédents.

Cette version du protocole possède un défaut pour partager les données à un grand nombre de subscribers. Le chaînage des messages force le publisher à réaliser N signatures si il souhaite partager la même donnée à N *subscribers*. Ceci va à l'encontre du modèle publish/subscribe où l'on souhaite avoir une faible quantité de travail pour partager l'information. Nous allons présenter une méthode pour réduire ce nombre d'opérations à 1 pour N *subscriber*.

3 Réduction du nombre de signatures

3.1 Idée generale

Le problème du nombre de signatures réalisées par le *publisher* vient du format de messages et des fonctionnalités du protocole. Sur la Figure 1 est représentée une version graphique de notre solution à ce problème. Nous voulons que le *publisher* soit capable à partir de plusieurs connexions avec plusieurs subscribers pour un même topic TN de créer une seule chaîne de signatures commune entre toutes les connexions. Pour ce faire, nous devons vérifier les deux règles suivantes :

— R1 : après avoir unifié les chaînes de signatures, le message M_i est identique pour tous les *subscribers*.

— R2 : Si le message M_i est identique pour tous les *subscribers*, alors le message M_{i+1} l'est aussi.

De ce double objectif, découle un nouveau format de message qui ne contient plus d'informations désignant un subscriber en particulier. Dans la version précédente du protocole, le message contenait par

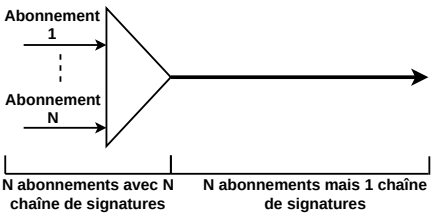

FIGURE 1 – Le *publisher* possède plusieurs abonnement actifs, possédant chacun une chaîne de signature, et il fusionne ces chaînes en une seule.

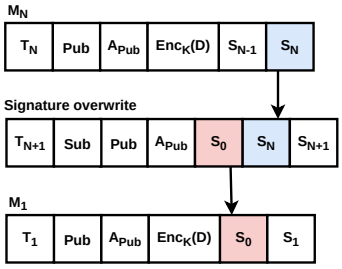

FIGURE 2 – Le *publisher* utilise le message *SigOver* pour choisir la valeur de Pre_i

exemple l'identifiant du subscriber. Cette information est unique par subscriber et force le publisher à réaliser une signature par subscriber. Nous proposons ce nouveau format :

$$M_i = T_i||Pub||A_{Pub}||Enc_K(D)||Pre_i||S_i.$$

T_i est l'horodatage du message M_i. *Pub* est l'identifiant du publisher. A_{Pub} est l'alias choisi par le publisher (l'équivalent d'un numéro de port). $Enc_K(D)$, la donnée chiffrée par une clé symetrique. S_i est la signature du message et Pre_i est la signature du message précédent, $Pre_i = S_{i-1}$.

Ce nouveau format ne contient pas d'information identifiant un *subscriber* en particulier, ce qui signifie qu'en l'utilisant, si nous sommes capables de faire un message M_i identique pour tous les *subscribers*, alors le message M_{i+1} sera lui aussi identique (R2).

3.2 Synchronisation des signatures

Pour résoudre le problème de signatures, nous devons maintenant faire en sorte de créer une version unique du message M_i pour tous les *subscribers* du même topic. Dans SUPRA, chaque message M_i répète la signature du message précédent dans le champ $Pre_i = S_{i-1}$. La première valeur de Pre_i est la signature du dernier message de la poignée de mains utilisée pour mettre en place l'abonnement. Cette signature est réalisée par le *subscriber*, il y a donc N versions de Pre_i quand les connexions entre le *publisher* et les *subscribers* sont mises en place.

SUPRA ne fait aucune supposition sur l'algorithme de signature. Il peut engendrer ou non des collisions. Si les collisions sont impossibles, alors il devient impossible d'unifier la valeur de Pre_i, mais, même si l'algorithme réalise des collisions, unifier la valeur de Pre_i reste une opération aléatoire et difficile.

Pour parvenir à unifier la valeur de Pre_i, quelle que soit l'hypothèse sur l'algorithme de signature utilisé, nous ajoutons un nouveau message de contrôle, et nous avons représenté son fonctionnement sur la Figure 2. Ce message se nomme *Signature Overwrite* et il possède le format suivant :

$$SigOver = T_{SigOver}||Sub||Pub||A_{Pub}||S_{Over}||Pre_{SigOver-1}||S_{SigOver}$$

En utilisant ce message, le *publisher* peut partager une valeur S_{Over} à tous les subscribers. Comme représentée sur la Figure 3, cette valeur va être utilisée dans le message suivant M_i où $Pre_i = S_{Over}$. Si le publisher partage la même valeur *SigOver* à tous les *subscribers* d'un même topic, il est alors certains d'avoir transformé les chaînes de signatures de chaque subscriber en une seule chaîne unique pour tous les subscribers.

Grâce à nos modifications dans le format de message et à ce nouveau message de contrôle, le *publisher* est capable de réduire le nombre d'opérations de signature et de chiffrement de N à 1 pour partager la même donnée à N *subscribers*.

4 Sécurité

SUPRA est un protocole qui assure une traçabilité des données. Il est toujours possible pour un utilisateur de prouver la provenance d'un message et sa destination. Cette traçabilité permet d'utiliser un smart-contract pour résoudre les possibles conflits entre les utilisateurs. C'est-à-dire, si un message du *publisher*

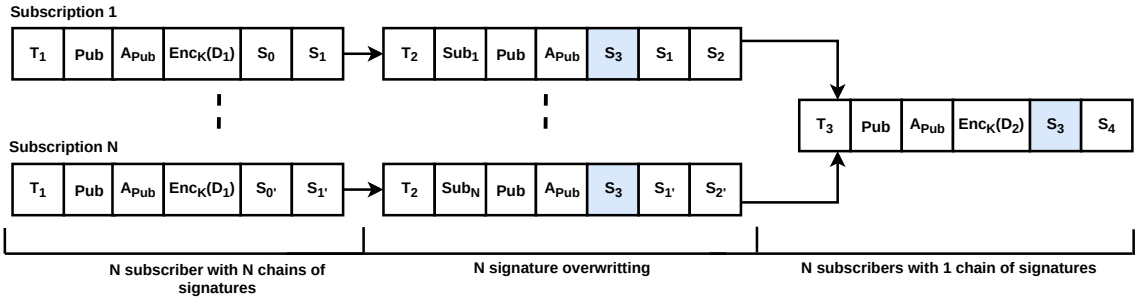

FIGURE 3 – Le publisher unifie les signatures de publishers abonnements.

n'est pas délivré après un délai T depuis son premier envoi. Avec notre nouveau format de message, notamment avec l'absence de l'identifiant du *subscriber*, on peut se demander si cette traçabilité est toujours garantie.

Pour assurer la résolution des conflits, SUPRA demande à ses utilisateurs de conserver la poignée de main en trois temps utilisée pour mettre en place la connexion entre le *subscriber* et le *publisher*. Dans cette poignée de main, sont échangés le nom du topic et l'alias. Dans ces messages, les identifiants du subscriber et du publisher apparaissent. En présentant cette poignée de main, il est toujours possible de prouver l'existence d'une connexion entre le *subscriber* et le *publisher*.

Pour prouver la réception d'un message, le *publisher* doit posséder un acquittement pour ce message ou le message doit être présent dans la blockchain. L'acquittement du message M_i par le *subscriber* n'est que la signature, par le *subscriber*, de S_i. Si on considère S'_i la signature acquittée par un ancien acquittement, un *publisher* malveillant peut essayer de modifier Pre_i pour que $S_i = S'_i$. En réalité, on peut montrer que cette opération est impossible, grâce à l'horodatage des messages. Le protocole présume que si M_i et M_j sont deux messages, avec $i \neq j$, alors $T_i \neq T_j$. Ce qui signifie si $S_i = S'_i$ alors $T_i = T'_i$ et donc $i = i'$. Si ce n'est pas le cas, les horodatages de l'acquittement et du faux message ne correspondent pas, et un tiers (ou un smart-contrat) peut facilement détecter cette tentative de fraude.

Une analyse plus précise de la sécurité de la proposition est disponible dans la version complète de l'article [ABBN22].

5 Conclusion

SUPRA est un protocole *publish/subscribe* utilisant la blockchain pour ajouter des garanties sur l'origine et la livraison des données. La blockchain est ici un médium neutre permettant le partage public d'information non-critique pour mettre en place des garanties de sécurité aux utilisateurs.

La première version du protocole souffre d'un problème de passage à l'échelle, et force le *publisher* à réaliser un nombre d'opérations ne respectant pas le modèle *publish/subscribe*. Dans cet article, nous avons présenté le protocole SUPRA ainsi qu'une amélioration permettant de corriger ce problème, tout en gardant la sécurité de la version d'origine.

Références

[ABBN21] Jean-Philippe ABEGG, Quentin BRAMAS, Timothée BRUGIÈRE, and Thomas NOEL. Supra, a distributed publish/subscribe protocol with blockchain as a conflict resolver. In *2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 41–42, 2021.

[ABBN22] Jean-Philippe ABEGG, Quentin BRAMAS, Timothée BRUGIÈRE, and Thomas NOEL. Distributed publish/subscribe protocol with minimum number of encryption. In *23rd International Conference on Distributed Computing and Networking*, ICDCN 2022, page 117–123, New York, NY, USA, 2022. Association for Computing Machinery.

[EFGK03] Patrick Th Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2) :114–131, 2003.

[RWZ+19] Gowri Sankar Ramachandran, Kwame Lante Wright, Licheng Zheng, Pavas Navaney, Muhammad Naveed, Bhaskar Krishnamachari, and Jagjit Dhaliwal. Trinity : A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence. *ICBC 2019 - IEEE International Conference on Blockchain and Cryptocurrency*, pages 227–235, 2019.