Consensus BFT sans dirigeant extensible et probabiliste par métastabilité

Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, Emin Gün Sirer
Cornell University[*]
Traduit par Grégory Lemercier et Jean Zundel

Résumé

Cet article présente une famille de protocoles à tolérance de faute byzantine sans dirigeant, bâtis autour d'un mécanisme métastable *via* sous-échantillonnage réseau. Ces protocoles fournissent une forte garantie de sûreté probabiliste en présence d'adversaires byzantins, tandis que leur nature simultanée et sans dirigeant leur permet d'être performants et extensibles. Contrairement aux blockchains fondées sur la preuve de travail (*proof-of-work*), ils sont économes et écologiques. Contrairement aux protocoles de consensus traditionnels où, typiquement, un ou plusieurs nœuds traitent linéairement l'information pour chaque décision, aucun nœud ne traite l'information d'une manière plus complexe que logarithmiquement. Il ne requiert pas de connaissance précise de tous les participants et propose de nouveaux compromis et améliorations en termes de sûreté de fonctionnement et de vitalité pour la construction de nouveaux protocoles de consensus.

Cet article décrit la famille de protocoles Snow, analyse ses garanties et décrit la manière dont il peut être employé pour bâtir le cœur d'un système de paiement électronique à l'échelle de l'internet appelé Avalanche, lequel est évalué lors d'un déploiement à grande échelle. Les expériences montrent que le système peut atteindre un haut débit (3400 tps), fournir une faible latence de confirmation (1,35 s) et correctement passer à l'échelle par rapport à d'autres systèmes offrant des fonctionnalités similaires. Le goulet d'étranglement du système dans l'implémentation et la configuration retenues est la vérification des transactions.

1 Introduction

L'obtention d'un accord au sein d'un ensemble d'hôtes distribués se retrouve au cœur d'innombrables applications allant de services à l'échelle de l'Internet servant des milliards de gens [13, 33] à des cryptomonnaies valant des milliards de dollars [1]. À ce jour, il existe deux familles principales de solutions à ce problème. Les protocoles de consensus traditionnels sont fondés sur une communication entre chaque pair pour s'assurer que tous les nœuds correct arrivent aux mêmes décisions avec une certitude absolue. Comme ils impliquent en général un nombre de communications croissant de façon quadratique et une connaissance exacte des membres du réseau, il s'est avéré difficile de l'étendre à de nombreux participants. D'un autre côté, les protocoles de consensus Nakamoto [9, 27, 29, 37, 45–48, 55–57] ont été popularisés par Bitcoin. Ces protocoles fournissent une garantie de sûreté probabiliste : les décisions par consensus Nakamoto peuvent s'inverser avec une certaine probabilité ε. Un paramètre du protocole permet d'abaisser à volonté cette probabilité, ce qui permet de bâtir des systèmes financiers de grande valeur sur ces fondations. Cette famille est conçue pour des configurations ouvertes et en libre accès où tout nœud peut rejoindre le système à tout moment. Cependant ces protocoles sont coûteux, gaspilleurs et limités en performance. Par construction, ils sont tenus de fonctionner même à vide : leur sécurité se base sur une participation constante des mineurs, même quand aucune décision ne doit être prise. Bitcoin consomme actuellement autour de 63.49 TWh/an [23], environ deux fois l'énergie consommée par le Danemark [17]. De plus, ces protocoles souffrent d'un goulet d'étranglement intrinsèque pour le passage à l'échelle qui s'avère difficile à dépasser par la simple reparamétrisation [20]. Cet article présente une nouvelle famille de protocoles de consensus appelés Snow. Inspirée par les algorithmes de *gossip* ou bavardage, cette famille possède des propriétés grâce à un mécanisme délibérément métastable. En particulier, le système opère en échantillonnant répétitivement le réseau au hasard, et en dirigeant les nœuds corrects vers un résultat commun. L'analyse montre que ce mécanisme métastable est puissant : il peut rapidement changer l'état d'un vaste réseau de manière irréversible, où l'irréversibilité implique qu'une part suffisamment importante du réseau a accepté une proposition et qu'une proposition différente ne sera acceptée qu'avec une probabilité au maximum négligeable.

De même que le consensus Nakamoto, la famille des protocoles Snow fournit une garantie de sûreté probabiliste, grâce à un paramètre de sécurité configurable qui peut amenuiser à volonté la possibilité d'un échec du consensus. Contrairement au consensus Nakamoto, ces protocoles sont écologiques, économes et efficients ; ils ne dépendent pas d'une preuve de travail [26] et ne consomment pas d'énergie quand il n'y a pas de décision à prendre. L'efficacité des protocoles provient en partie de l'élimination du goulet d'étranglement des dirigeants : chaque nœud ne demande qu'un nombre de communications correspondant à $O(1)$ par tour et une prévision de $O(\log n)$ tours, alors que, dans les protocoles de consensus classiques,

[*] *À la vitesse de la lumière !* — Team Rocket
Une version antérieure de cet article, publié le 16 mai 2018, IPFS, était intitulée *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*.

un ou plusieurs nœuds requièrent $O(n)$ communications par tour. De plus, la famille Snow tolère des incohérences dans la connaissance des participants, comme nous le verrons plus loin. Par contraste, les protocoles de consensus classiques exigent une connaisssance complète et exacte de n comme fondation de la sûreté de fonctionnement.

Le mécanisme de vote sous-échantillonné possède deux propriétés additionnelles supérieures par rapport aux approches antérieures du consensus. Alors que la sûreté de fonctionnement dans les approches basées sur un quorum s'effondre immédiatement quand le seuil prédéterminé f est dépassé, la garantie de sûreté probabiliste de Snow se dégrade progressivement quand les participants byzantins dépassent f. Cela facilite le choix du seuil critique f. Cela expose également de nouveaux compromis entre sûreté et vitalité : la famille Snow est plus efficace quand la fraction de nœuds byzantins est petite, et cela peut être paramétré pour tolérer plus d'un tiers de nœuds byzantins en sacrifiant la vitalité.

Pour démontrer le potentiel de cette famille de protocoles, nous l'illustrons par un système pratique de paiement pair à pair, Avalanche. Dans les faits, Avalanche exécute plusieurs instances de Snowball (l'un des protocoles de la famille Snow) à l'aide d'un graphe orienté acyclique (*Directed Acyclic Graph* ou DAG). Le DAG sert à empiler plusieurs instances en faisant passer le coût de $O(\log n)$ à $O(1)$ par nœud et en optimisant le chemin là où il n'existe pas de transactions conflictuelles.

En résumé, la principale contribution de cet article consiste à présenter une nouvelle famille de protocoles de consensus fondés sur un échantillonnage aléatoire et des décisions métastables. La section suivante fournit le modèle, les buts et les hypothèses nécessaires pour les nouveaux protocoles. La section 3 expose l'intuition ayant mené à ces protocoles, suivie de leurs spécifications complètes, La section 4 fournit la méthodologie employée par notre analyse formelle de sûreté de fonctionnement et de vitalité en annexe A, La section 5 décrit Avalanche, un système de paiement similaire à Bitcoin, La section 6 évalue Avalanche, La section 7 présente les travaux existants en rapport avec cette étude et, finalement, la section 8 résume nos contributions.

2 Modèle et buts

Garanties clefs

Sûreté de fonctionnement À l'inverse des protocoles de consensus classiques, et à l'instar des protocoles de consensus fondés sur la chaîne la plus longue [45], nous adoptons une garantie de sûreté de fonctionnement ε qui est probabiliste. En pratique, cette garantie probabiliste est aussi forte que les garanties de sûreté traditionnelles puisque le choix d'un petit ε peut rendre la défaillance du consensus négligeable, inférieure à la probabilité d'une panne matérielle due à des événements imprévisibles. La figure 1 montre comment la portion (f/n) de participants (ou de puissance informatique) problématique affecte la probabilité d'une défaillance de la sûreté du système (décision concernant deux propositions incompatibles), étant donné un choix de finalités.

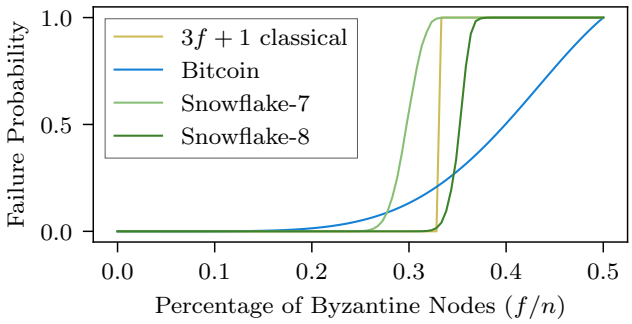

Figure 1: Les protocoles BFT classiques qui tolèrent f défaillances rencontrent une défaillance totale de leur sûreté de fonctionnement quand le seuil est dépassé même d'un seul nœud supplémentaire. La courbe de Bitcoin montre un choix de finalité typique pour Bitcoin où un bloc est considéré final quand il est « enterré » dans une branche sous 6 blocs supplémentaires par rapport aux autres branches concurrentes. Snowflake appartient à la famille Snow et est configuré avec $k = 10$, $\beta = 150$. Snowflake-7,8 utilise $\alpha = 7$ et $\alpha = 8$ respectivement (voir en section 3 pour la définition de k, α et β.

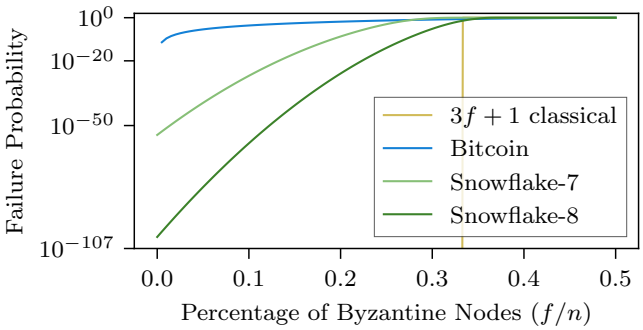

Figure 2: Figure 1 avec l'axe des y en log.

Vitalité Tous nos protocoles fournissent une garantie de probabilité de terminaison dans un laps de temps donné supérieure à zéro. Cette garantie de limite est similaire à celle de divers protocoles comme Ben-Or [8] et les protocoles de chaîne la plus longue. En particulier, pour le consensus Nakamoto, le nombre de blocs requis pour une transaction augmente exponentiellement avec le nombre de nœuds antagonistes, avec une asymptote à $f = n/2$ où le nombre est infini. En d'autres termes, le temps nécessaire à la finalité approche ∞ quand f approche $n/2$ (figure 3). De plus, le nombre requis de tours est calculable à l'avance, ce qui permet au concepteur du système de régler la vitalité aux dépens de la sûreté de fonctionnement. Enfin, contrairement aux protocoles de consensus traditionnels et similairement à Nakamoto, nos protocoles bénéficient d'une moindre présence antagoniste,

comme le détaille la propriété P3 ci-dessous.

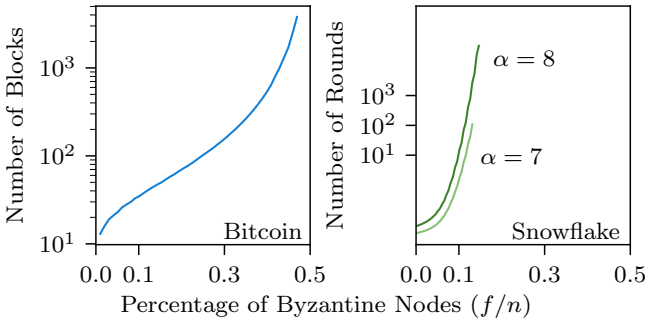

Figure 3: Relation entre f/n et la vitesse de convergence pour $\varepsilon = 10^{-20}$. La figure de gauche montre le nombre attendu de blocs pour garantir ε dans Bitcoin, qui, contrairement à la croyance populaire, n'est pas une constante 6, mais dépend de la taille de l'adversaire pour maintenir le même ε. La figure de droite montre le nombre maximum de tours requis par Snowflake où, étant différent de Bitcoin, l'asymptote se trouve sous 0.5 et varie selon les paramètres.

Garantie formelle : Soit un paramétrage du système pour une probabilité de défaillance de la sûreté de fonctionnement ε pour un nombre maximal attendu f de nœuds antagonistes. Soit $O(\log n) < t_{max} < \infty$ la limite supérieure de l'exécution du protocole. Les protocoles Snow fournissent alors les garanties suivantes :

P1. Sûreté de fonctionnement. Quand les décisions sont prises par deux nœuds corrects, ils décident des transactions conflictuelles avec une probabilité négligeable ($\leq \varepsilon$).

P2. Vitalité (limite supérieure). Les protocoles Snow se terminent avec une probabilité strictement positive en au plus t_{max} tours.

P3. Vitalité (forme forte). Si $f \leq O(\sqrt{n})$, alors les protocoles Snow se terminent avec une forte probabilité ($\geq 1-\varepsilon$) en $O(\log n)$ tours.

Réseau Dans la définition standard de l'asynchronie [8], le temps de transmission des messages est fini, mais la distribution n'est pas spécifiée (donc le temps de délivrance peut être non borné pour certains messages). Cela implique que l'ordonnancement de la transmission des messages lui-même peut être quelconque, voire potentiellement malveillant (pour une complète asynchronie). Nous employons une version modifiée de ce modèle, lequel est bien accepté [7, 25, 28, 35, 41] dans l'analyse des réseaux épidémiques et des systèmes stochastiques à base de *gossip*. En particulier, nous fixons la distribution des délais de messages à celle de la distribution exponentielle. Nous notons que, tout comme dans le modèle asynchrone standard, il existe une probabilité strictement non-nulle que tout nœud correct ne peut exécuter son prochain tour local qu'après qu'un laps de temps arbitrairement long a passé. De plus, notons également que l'ordonnancement ne s'applique qu'aux nœuds corrects, et que l'adversaire peut s'exécuter arbitrairement, comme nous le verrons plus loin.

Objectifs de vitalité Les consensus classiques qui fonctionnent en asynchronie ne restent pas bloqués lors d'une phase de vote car l'initiateur du vote interroge toujours tous les participants connus et attend $n-f$ réponses. Dans notre système, cependant, les nœuds opèrent par sous-échantillonnage, d'où la possibilité pour un seul échantillon qu'une majorité de nœuds antagonistes soit sélectionné, ce qui contraint le nœud à attendre en vain une réponse. Pour s'assurer de la vitalité, un nœud doit pouvoir limiter son attente. Enfin, il faut noter que le consensus Nakamoto est synchrone et que la difficulté requise pour la preuve de travail dépend du délai maximum du réseau [46].

Adversité Les nœuds antagonistes s'exécutent avec un ordonnanceur interne qui leur est propre, non limité en vitesse, ce qui signifie que tout nœud antagoniste peut s'exécuter à un moment précis infiniment petit, au contraire des nœuds corrects. L'adversaire peut voir l'état de chaque nœud correct à tout moment et peut instantanément modifier l'état de tout nœud antagoniste. Il ne peut, cependant, ordonnancer ou modifier la communication entre les nœuds corrects. Enfin, nous ne faisons aucune hypothèse sur le comportement de l'adversaire, ce qui signifie qu'il peut choisir la stratégie d'exécution qui lui convient. En résumé, l'adversaire est limité en puissance informatique (il ne peut pas forger des signatures numériques) mais, hormis cela, il n'est pas limité en matière informationnelle (il connaît tout de l'état) et il est adaptatif en matière de tours (il peut modifier sa stratégie à tout moment).

Attaques Sybil Les protocoles de consensus fournissent leurs garanties en se fondant sur l'hypothèse que seule une fraction des participants sont antagonistes. Ces limites peuvent être violées si le réseau est naïvement laissé ouvert à des participants quelconques. En particulier, une attaque Sybil [24], où un grand nombre d'identités sont générées par un adversaire, peut être employée pour dépasser ces limites.

Une longue suite de travaux, comme PBFT [15], sépare le problème de Sybil de celui du consensus, à juste titre puisque les mécanismes de contrôle de Sybil sont distincts du protocole d'accord plus complexe sous-jacent[1]. En fait, à notre connaissance, le consensus Nakamoto est le seul à avoir intégré la protection anti-Sybil dans son consensus, chose rendue possible par la preuve de travail chaînée [5] qui demande aux mineurs de continuellement mettre en jeu un investissement matériel. D'autres protocoles, dont il est question en section 7, se basent sur la preuve d'enjeu (*proof of stake* ou PoS, par un argument économique) ou la preuve d'autorité (*proof of authority* ou PoA, par un argument administratif) qui rend le système « permissionné ». Les protocoles de consensus présentés dans cet article peuvent adopter n'importe quel mécanisme de contrôle de Sybil, bien que la preuve d'enjeu soit la plus en ligne avec le caractère économe de leur opération.

[1]Cela n'implique pas que tout protocole de consensus puisse être couplé ou découplé de tout mécanisme de contrôle de Sybil.

On peut utiliser un mécanisme en preuve d'enjeu déjà établi. Le déploiement complet d'un système de paiement autonome en P2P incorporant des mécanismes d'enjeu dépasse le cadre de cet article, dont l'objet est un nouveau paradigme de conception de l'algorithme de consensus.

Attaques par inondation Les attaques par inondation et par spam représentent un problème pour tout système distribué. Sans mécanisme de protection, un attaquant peut générer un grand nombre de transactions et inonder les structures de données du protocole, consommant ainsi des ressources de stockage. Il existe une multitude de techniques pour repousser ces attaques comme la protection de la couche réseau, la preuve d'autorité, la preuve de travail local, ainsi que des mécanismes économiques. Dans Avalanche, nous utilisons les frais de transaction, rendant ainsi ces attaques coûteuses même si l'attaquant renvoie l'argent aux adresses qu'il contrôle.

Hypothèses supplémentaires Nous ne supposons pas que tous les membres du réseau sont connus de tous les participants, car la vue du réseau qu'ont ces derniers peut souffrir d'incohérences temporaires. Nous en quantifions les bornes en annexe A.7. Nous supposons un mécanisme de démarrage initial sûr, similaire à celui de Bitcoin, qui permet à un nœud de se connecter à suffisamment de nœuds corrects pour acquérir une vue statistiquement non biaisée du réseau. Nous ne prévoyons pas une PKI. Enfin, nous nous reposons sur les hypothèses standard en matière de cryptographie dans le domaine des signatures numériques et des fonctions de hachage.

3 Conception du Protocole

Nous partons d'un protocole non résistant au problème des généraux byzantins appelé Slush, en amenant progressivement Snowflake et Snowball, tous basés sur le même mécanisme métastable de vote basé sur une majorité. Ces derniers sont des protocoles de consensus à simple décret présentant plusieurs niveaux de robustesse. Nous détaillons les caractéristiques complètes de ces protocoles dans cette section, suivie de leur analyse en section suivante, les preuves formelles étant disponibles en annexe.

3.1 Slush : présentation de la métastabililté

Le principe de notre approche est un protocole de consensus à simple décret inspiré des protocoles dits épidémiques ou *gossip*. Le protocole le plus simple, Slush, est la fondation de cette famille, comme le montre la figure 4. Slush *n'est pas* tolérant aux fautes byzantines, seulement aux défauts de fonctionnement (CFT), mais sert à mettre en avant les autres protocoles tolérants aux fautes byzantines qui vont suivre. Dans un souci de simplicité de présentation, nous décrirons le fonctionnement de Slush en le comparant à une décision à prendre entre deux couleurs incompatibles, le rouge et le bleu.

Avec Slush, un nœud démarre dans un état de couleur neutre. Après réception d'une transaction envoyée par un client, un

```
1:  procedure onQuery(v, col')
2:      if col = ⊥ then col := col'
3:      répondre(v, col)
4:  procedure slushLoop(u, col_0 ∈ {R, B, ⊥})
5:      col := col_0 // initialise avec une couleur
6:      for r ∈ {1...m} do
7:          // Si ⊥, saute jusqu'à ce que onQuery applique la couleur
8:          if col = ⊥ then continue
9:          // échantillonne aléatoirement parmi les nœuds connus
10:         K := échantillon(N\u, k)
11:         P := [requête(v, col)   pour v ∈ K]
12:         for col' ∈ {R, B} do
13:             if P.nombre(col') ≥ α then
14:                 col := col'
15:     accepte(col)
```

Figure 4: Protocole Slush. Temps d'expiration omis dans un souci de lisibilité.

nœud de couleur neutre adopte la couleur définie dans la transaction et initie une requête. Pour lancer une requête, le nœud choisit un échantillon réduit et de taille constante (k) de nœuds appartenant au réseau suivant une distribution uniformément aléatoire puis initie sa propre requête, tandis qu'un nœud déja coloré répond simplement avec sa couleur actuelle. Une fois que le nœud qui fait la requête récupère k réponses, il vérifie si une proportion $\geq \alpha$ de l'échantillon retourne la même couleur, $\alpha > \lfloor k/2 \rfloor$ étant un paramètre du protocole. Si le seuil α est dépassé et que la couleur dominante revenant de l'échantillon sollicité est différente de la couleur propre du nœud à l'origine de la requête, ce nœud adopte cette nouvelle couleur. Il revient ensuite à l'étape de génération de la requête et lance un nouveau tour de requête jusqu'à un total de m tours. En définitive, le nœud décide de sa couleur finale à l'instant m.

Slush possède quelques propriétés intéressantes. Premièrement, il n'a quasiment *pas besoin de mémoire* : un nœud ne stocke aucun état intermédiaire entre chaque tour autre que sa couleur actuelle, et plus particulièrement ne stocke aucun historique de ses interactions avec les autres nœuds. Deuxièmement, contrairement aux protocoles de consensus traditionnels qui sollicitent tous les participants, chaque tour n'implique qu'un échantillon du réseau réduit et de taille constante choisi aléatoirement. Troisièmement, Slush progresse quelque soit la configuration du réseau (même depuis un état entièrement bivalent, c'est-à-dire un partage 50/50 entre les deux couleurs) puisque des perturbations aléatoires sur l'échantillonnage auront pour conséquence de fournir un léger avantage à l'une des deux couleurs, lequel sera ensuite amplifié au fur et à mesure des échantillonnages. Finalement, si m est choisi pour être suffisamment élevé, Slush s'assure que tous les nœuds convergent vers la même couleur avec une forte probabilité. Chaque nœud nécessite un nombre de communications par tour constant et prévisible, et m croît de manière logarithmique suivant n.

Le protocole Slush ne fournit pas de fortes garanties

```
1:  procedure SNOWFLAKELOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, cnt := 0
3:      while undecided do
4:          if col = ⊥ then continue
5:          K := SAMPLE(N\u, k)
6:          P := [QUERY(v, col)   for v ∈ K]
7:          maj := false
8:          for col' ∈ {R, B} do
9:              if P.COUNT(col') ≥ α then
10:                 maj := true
11:                 if col' ≠ col then
12:                     col := col', cnt := 1
13:                 else cnt++
14:                 if cnt ≥ β then ACCEPT(col')
15:         if maj = false then cnt := 0
```

Figure 5: Snowflake.

quant à la sûreté de fonctionnement en présence de nœuds byzantins. En particulier, si les nœuds légitimes développent une préférence pour une certaine couleur, un adversaire byzantin pourra inverser la couleur des nœuds afin de maintenir le système à l'équilibre, empêchant ainsi de prendre une décision. Nous étudions ce problème dans notre premier protocole BFT qui ajoute un stockage de l'état sur les nœuds.

3.2 Snowflake : BFT

Snowflake améliore Slush en ajoutant un compteur unique qui reflète la force de conviction d'un nœud concernant sa couleur actuelle. Ce compteur associé à chaque nœud stocke le nombre d'échantillons du réseau consécutifs que le nœud a sollicité et qui ont retourné la même couleur. Un nœud accepte la couleur actuelle lorsque son compteur atteint β, un autre paramètre de sécurité. La figure 5 montre le protocole amélioré qui inclut les modifications suivantes :

1. Chaque nœud maintient un compteur *cnt* ;
2. À chaque changement de couleur, le compteur *cnt* se réinitialise à 0 ;
3. Après chaque requête réussie qui renvoie $\geq \alpha$ réponses en faveur de la même couleur que le nœud, ce dernier incrémente *cnt*.

Quand le protocole est correctement paramétré pour un seuil donné de nœuds byzantins et un niveau de garantie ε, il peut garantir à la fois la sécurité de fonctionnement (P1) et la vitalité (P2, P3). Comme nous le démontrons plus loin, il existe un état irréversible à partir duquel une décision est inévitable. Les nœuds légitimes commencent à se positionner après cet état irréversible et adoptent la même couleur. Une intuition supplémentaire, que nous ne détaillerons pas dans ce papier, indique l'existence d'un point de changement de phase à partir duquel les nœuds byzantins perdent toute capacité à maintenir le réseau dans un état bivalent.

3.3 Snowball : ajout de la confiance

La notion d'état de Snowflake est éphémère : le compteur est réinitialisé à chaque changement de couleur. Snowball

```
1:  procedure SNOWBALLLOOP(u, col_0 ∈ {R, B, ⊥})
2:      col := col_0, lastcol := col_0, cnt := 0
3:      d[R] := 0, d[B] := 0
4:      while undecided do
5:          if col = ⊥ then continue
6:          K := SAMPLE(N\u, k)
7:          P := [QUERY(v, col)   for v ∈ K]
8:          maj := false
9:          for col' ∈ {R, B} do
10:             if P.COUNT(col') ≥ α then
11:                 maj := true
12:                 d[col']++
13:                 if d[col'] > d[col] then
14:                     col := col'
15:                 if col' ≠ lastcol then
16:                     lastcol := col', cnt := 1
17:                 else cnt++
18:                 if cnt ≥ β then ACCEPT(col')
19:         if maj = false then cnt := 0
```

Figure 6: Snowball.

améliore Snowflake avec des *compteurs de confiance* qui reflètent le nombre de requêtes ayant renvoyé un résultat en faveur de la couleur du nœud au dessus d'un certain seuil (figure 6). Un nœud prend une décision s'il obtient β réponses consécutives en faveur d'une couleur. Cependant, il change sa préférence uniquement sur la base de la confiance totale accumulée. Les différences entre Snowflake et Snowball sont les suivantes :

1. Après chaque requête réussie, le nœud incrémente son compteur de confiance associé à la couleur retournée ;
2. Un nœud change de couleur quand la confiance associée à sa couleur actuelle devient inférieure à la valeur de la confiance associée à la nouvelle couleur.

4 Analyse

Pour des raisons de manque de place, nous déplaçons certains détails importants vers l'annexe A, dans laquelle nous démontrons que sous certaines hypothèses distinctes et indépendantes, la famille de protocoles de consensus Snow possède des propriétés de sûreté de fonctionnement (P1) et de vitalité (P2, P3). Dans cette section, nous résumons nos résultats principaux en fournissant des ébauches de preuves.

Notation En considérant que le réseau consiste en un ensemble de n nœuds (représentés par l'ensemble $\mathcal{N}$), où c sont des nœuds légitimes (représentés par l'ensemble $\mathcal{C}$) et f sont les nœuds byzantins (représentés par l'ensemble $\mathcal{B}$). Soient $u, v \in \mathcal{C}$ des références à deux nœuds légitimes pris au hasard dans le réseau. Soient $k, \alpha, \beta \in \mathbb{Z}^+$ des entiers positifs où $\alpha > \lfloor k/2 \rfloor$. A partir de maintenant, k fera toujours référence à la taille des échantillons du réseau, où $k \leq n$, et α sera le seuil de majorité requis pour considérer l'expérience de vote comme « réussie ». En général, nous définissons $\mathcal{S}$ comme étant l'état (ou la configuration) du réseau à tout moment.

Framework de modélisation Afin de modéliser nos protocoles de manière formelle, nous utilisons des processus de Markov à temps continu (CTMC). L'espace contenant les états est dénombrable (et fini), et les transitions entre états s'opèrent sur un espace de temps continu. Les CTMC proposent une modélisation naturelle de nos protocoles sachant que les transitions entre états ne se font pas par étapes et en marche ordonnée pour chaque nœud (à la fin de chaque unité de temps) mais arrivent plutôt à n'importe quel moment et indépendamment des autres nœuds.

Nous concentrons l'étude sur un consensus binaire, même si les résultats en termes de sûreté de fonctionnement se généralisent pour plus de deux valeurs. Nous pouvons imaginer le réseau comme un ensemble de nœuds colorés, soit en rouge, soit en bleu, et nous ferons référence à cette configuration à l'instant t comme étant $\mathcal{S}_t$. Nous modélisons nos protocoles à travers un processus à temps continu contenant deux états, dans lequel tous les nœuds sont rouges ou tous les nœuds sont bleus. L'espace $\mathcal{S}$ des états du processus stochastique est une version condensée de l'espace de configuration complet, où chaque état $\{0,\dots,n\}$ représente le nombre total de nœuds bleus dans le système.

La simplification qui nous permet d'analyser ce système consiste à s'affranchir du besoin de garder un historique de tous les chemins d'exécution, ainsi que de toutes les stratégies d'attaque possibles, et de se concentrer entièrement sur un unique état quelle que soit la manière d'arriver à celui-ci. Plus spécifiquement, l'idée principale à retenir de notre analyse est l'identification d'un état *d'irréversibilité* du système, cet état à partir duquel un tel nombre de nœuds légitimes a adopté soit la couleur rouge soit la couleur bleue que la probabilité de revenir à la couleur minoritaire est très faible.

4.1 Sûreté de fonctionnement

Slush Nous partons du principe que tous les nœuds partagent le même $\mathcal{N}$, et nous expliquons en annexe A.7 comment assouplir l'exigence de la connaissance des participants. Nous modélisons la dynamique du système grâce à un processus à temps continu contenant deux états, tous les nœuds étant rouges ou tous les nœuds étant bleus. Soit $\{X_{t\geq 0}\}$ la variable aléatoire qui décrit l'état du système à l'instant t, où $X_0 \in \{0,\dots,c\}$. Nous commençons par détailler le résultat le plus important de la dynamique de la sûreté de fonctionnement de notre processus : les probabilités de *réversibilité* du processus **Slush**. Tous les autres résultats formels de ce papier sont, en simplifiant, des dérivations et améliorations intuitives de ce résultat.

Théorème 1. *Prenons la configuration du système à l'instant t comme étant $\mathcal{S}_t = n/2+\delta$, ce qui signifie que le réseau a dérivé de 2δ nœuds bleus de plus que les nœuds rouges ($\delta = 0$ signifie que rouges et bleus sont en nombre égal). ξ_δ étant la probabilité d'un basculement vers l'état où tous les nœuds sont rouges, nous avons donc pour tout $0 \leq \delta \leq n/2$*

$$\begin{aligned}\xi_\delta &\leq \left(\frac{1/2-\delta/n}{\alpha/k}\right)^{\alpha}\left(\frac{1/2+\delta/n}{1-\alpha/k}\right)^{k-\alpha} \\ &\leq e^{-2((\alpha/k)-(1/2)+(\delta/n))^2 k}\end{aligned} \tag{1}$$

Preuve. Cette borne suit les bornes de queue dérivées de Hoeffding de la distribution hypergéométrique définie par Chvatal [18]. □

Nous notons que les bornes de Chvatal sont introduites ici dans un but de simplification et sont extrêmement faibles. Nous laissons l'expression dans sa forme complète dans le théorème 2 disponible en annexe, qui est aussi significativement plus fort que la borne de Chvatal. Néanmoins, l'utilisation de la borne de Chvatal nous permet de faire l'observation qui consiste à dire qu'au fur et à mesure que la déviation δ augmente, pour un α et un k constants, la probabilité de basculer vers la valeur minoritaire décroît *de manière exponentielle* (en fait encore plus rapidement puisqu'il y a un terme quadratique dans l'exposant inverse). De plus, cette même observation est valable si l'on fait croître α en gardant k constant.

Les conclusions de ce théorème démontrent une propriété clef du système: une fois que le réseau perd sa bivalence (par exemple : $\delta > 0$), il tend à se renverser et à converger rapidement vers la couleur majoritaire, et il devient alors suffisamment improbable qu'il rebascule vers la couleur minoritaire. C'est cette propriété fondamentale que nous exploitons dans nos protocoles, et c'est ce qui les rend sûrs malgré le fait que l'échantillonnage se fait sur une petite partie du réseau de taille fixe. Le résultat principal qui s'ensuit pour les garanties de sûreté de fonctionnement de Snowflake consiste à trouver des régions (à partir de paramètres spécifiquement choisis) où la réversibilité n'est possible qu'à partir d'une probabilité maximale ε même en conditions d'adversité.

Snowflake Dans le cas de Snowflake, nous prenons comme hypothèse qu'une partie des nœuds est antagoniste. Avec Slush, une fois que le réseau obtient une majorité significative pour l'une des propositions (par exemple : la couleur bleue), il devient improbable qu'une proposition minoritaire (par exemple : la couleur rouge) puisse être adoptée dans le futur (irréversibilité). De plus, les nœuds Slush ont simplement besoin d'exécuter le protocole pour un nombre déterministe m de tours qui est connu en amont de l'exécution du protocole. Lorsqu'on introduit des nœuds antagonistes qui suivent des stratégies arbitraires, les nœuds ne peuvent tout simplement plus exécuter le protocole pour un nombre déterministe de tours puisque les adversaires peuvent affecter la valeur de m de manière non déterministe. En revanche, les nœuds légitimes doivent implémenter un mécanisme pour explicitement détecter que l'irréversibilité a été atteinte. Dans ce but, chaque nœud légitime de Snowflake implémente une fonction de décision, $\mathcal{D}(u,\mathcal{S}_t,blue) \rightarrow \{0,1\}$, qui est une variable aléatoire qui retourne 1 si le nœud u détecte que l'état d'irréversibilité

a été atteint à l'instant t pour bleu. Ce mécanisme de décision est probabiliste, ce qui veut dire qu'il peut échouer, même s'il est conçu pour que la probabilité de cet échec soit négligeable. Nous pouvons maintenant ébaucher la preuve de Snowflake.

Ébauche de preuve. Nous définissons la stratégie d'échec comme étant l'événement lors duquel deux nœuds légitimes u et v décident entre le bleu et le rouge, par exemple : $\mathcal{D}(u, \mathcal{S}_t, bleu) \to 1$ et $\mathcal{D}(v, \mathcal{S}_{t'}, rouge) \to 1$, pour deux instants t et t'. Nous modélisons une fois de plus le système comme un processus à temps aléatoire. L'espace des états est défini de la même manière que pour Slush. Néanmoins, nous observons des subtilités critiques. Tout d'abord, même si tous les nœuds légitimes acceptent une couleur, les nœuds byzantins peuvent toujours basculer vers l'autre couleur. Puis, nous devons aussi prendre en considération le mécanisme de décision $\mathcal{D}(*)$. Pour cette analyse, nous nous affranchissons du besoin de garder l'historique de toutes les configurations du réseau soumises à toutes les stratégies antagonistes et prenons pour hypothèse qu'un nœud u décide d'abord d'adopter la couleur bleue. Ensuite, en fonction de l'état du réseau suite à la décision de u, nous calculons la probabilité qu'un autre nœud v adopte le rouge, qui est fonction à la fois de la probabilité que le réseau bascule vers l'état minoritaire bleu et que v bascule vers cet état. Nous démontrons qu'en choisissant correctement k, α, et β, nous pouvons construire des instances extrêmement sécurisées de Snowflake (par exemple : échec de sécurité avec une probabilité de $\leq \varepsilon$) quand le réseau atteint un biais de δ, comme le montre la figure 7. Un exemple concret est présenté en figure 1.

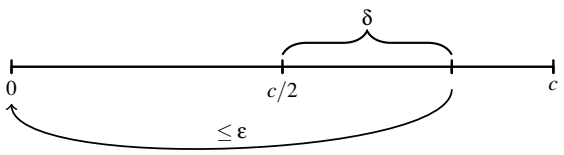

Figure 7: Représentation de l'état d'irréversibilité qui existe quand – même en la présence de f nœuds byzantins – le nombre de nœuds légitimes bleus dépasse le nombre de nœuds légitimes rouges de plus de 2δ.

Snowball Snowball est une amélioration en terme de sécurité par rapport à Snowflake, où les perturbations aléatoires dans les échantillons du réseau sont réduites par l'introduction d'une forme limitée d'historique que l'on appelle confiance.

Ebauche de preuve. Nous appliquons des inégalités de concentration pour prouver qu'une fois le système ayant atteint l'état d'irréversibilité, alors l'évolution de la confiance envers la couleur majoritaire adoptée va augmenter constamment et dériver de plus en plus loin des nœuds de couleur minoritaire, jusqu'à rendre la réversibilité de plus en plus improbable avec le temps. Si cette dérive finit par se retourner, alors l'analyse de la réversibilité devient identique à celle de Snowflake.

4.2 Vitalité

Nous prenons pour hypothèse la présence antagoniste observée $0 \leq f' \leq n(k-\alpha-\psi)/k \leq f$, où ψ représente la zone tampon. Plus ψ est grand, plus la capacité du mécanisme de décision à tendre vers une valeur est rapide. Bien sûr, si ψ tend vers zéro ou devient négatif, alors nous dépassons la limite supérieure de tolérance antagoniste pour le système paramétré, et en conséquence l'attaquant peut, avec une haute probabilité, bloquer la convergence en choisissant simplement de ne pas répondre, ceci même si les garanties de sûreté de fonctionnement s'appliquent toujours.

En partant de ψ strictement positif, la convergence est strictement finie pour toutes les configurations où une proposition a un support d'au moins α. De plus, non seulement la convergence est finie avec une probabilité de un, mais nous avons aussi une probabilité strictement positive de convergence jusqu'à un instant borné t_{max}, comme décrit dans la proposition 4, qui découle du théorème 3. Cela formalise la propriété de vitalité P2.

Ébauche de preuve. En utilisant la construction du système pour prouver l'irréversibilité, nous caractérisons la distribution de la moyenne du temps passé (les périodes de séjour) dans chaque état avant que le système ne termine son exécution en adoptant l'un des états. Ce temps de convergence est l'union de ces périodes.

Pour les transactions non conflictuelles, l'attaquant étant incapable de générer un conflit, le temps de décision est simplement le temps de mélange $O(\log n)$. La vitalité guarantit qu'une configuration de réseau entièrement bivalent atteint un temps de convergence optimal de $O(\log n)$ tours si le nombre d'attaquants est au plus $O(\sqrt{n})$, pour $\alpha = \lfloor k/2 \rfloor + 1$. Nous donnons plus de détails en lemme 5. Quand le nombre de nœuds antagonistes dépasse $O(\sqrt{n})$, le nombre le moins favorable de tours grandit de manière polynomiale, et quand f tend vers $n/2$, ce nombre de tours tend vers des taux de convergence exponentiels.

Ebauche de preuve. Nous modifions le théorème 3 en y incluant le nombre de nœuds antagonistes, ce qui annule tout déséquilibre dans le réseau en le gardant entièrement bivalent.

Consensus multi-valeurs Notre protocole de consensus binaire pourrait supporter des consensus multi-valeurs en exécutant des instances binaires logarithmiques, une pour chaque bit de la valeur proposée. Cependant, une telle réduction théorique n'est pas forcément efficace en pratique. À la place, nous pourrions directement incorporer des valeurs multiples comme des couleurs multiples dans le protocole en permettant toujours la généralisation de l'analyse de sûreté de fonctionnement.

Pour ce qui est de la vitalité, nous ébauchons en annexe A.6 un mécanisme d'initialisation sans dirigean, qui nécessite $O(\log n)$ tours attendus dans l'hypothèse que le réseau est synchronisé. Bien que la conception de ce mécanisme soit intéressante, il est à noter qu'un tel mécanisme n'est pas néces-

saire pour un système de paiement décentralisé, comme nous le démontrons en section 5. Finalement, nous développons le roulement, dans le sens de renouvellement des nœuds, et les divergences de vues en annexe A.7.

5 Système de paiement pair à pair

Grâce au consensus Snowball, nous avons implémenté un système de paiement élémentaire, Avalanche, qui supporte les transactions Bitcoin. Dans cette section, nous en décrivons la conception et schématisons la manière dont l'implémentation peut supporter la primitive de transfert de valeur au cœur des cryptomonnaies. Le déploiement complet d'une cryptomonnaie implique l'initialisation, l'émission, le dépôt, le retrait et le contrôle de l'inflation. Nous disposons de solutions à ces problèmes mais leur description détaillée sort du cadre de cet article, dont l'objet est la nouvelle famille de consensus Snow.

Dans une configuration de cryptomonnaie, les signatures cryptographiques imposent que seul le possesseur de la clef soit capable de créer une transaction dépensant un dépôt donné. Comme les clients corrects suivent le protocole tel qu'il est prescrit et ne peuvent pratiquer la double dépense, ils ont la garantie tant de la sûreté de fonctionnement que de la vitalité pour leurs transactions *vertueuses*. Par contraste, la vitalité n'est pas garantie pour les transactions *malveillantes* soumises par des clients byzantins, lesquelles entrent mutuellement en conflit. Ces décisions peuvent arrêter le réseau mais n'ont pas d'impact sur les transactions vertueuses. Nous montrons que ce compromis est raisnnable et que le système qui en résulte suffit à bâtir des systèmes de paiement complexes.

5.1 Avalanche : ajout d'un DAG

Avalanche consiste en plusieurs instance Snowball à décret unique instanciées en tant que protocole multi-décrets qui maintient un graphe orienté acyclique (*Directed Acyclic Graph*, DAG) dynamique en ajout seul de toutes les transactions connues. Le DAG possède un nœud terminal (*sink*) unique qui est le *genesis vertex*, le sommet genèse. L'emploi d'un DAG procure deux avantages : D'abord, il est plus efficient car un vote sur un sommet vote implicitement pour toutes les transactions se trouvant sur le chemin du sommet genèse. Ensuite, il améliore également la sécurité parce que le DAG lie fortement le sort des transactions, similaire en cela à la blockchain de Bitcoin. Cela rend les décisions passées difficiles à défaire sans l'approbation des nœuds corrects.

Quand un client crée une transaction, il nomme un ou plusieurs *parents*, et ceux-ci sont inclus de façon inséparable dans la transaction et forment les bords du DAG. Les relations parent-enfant codées dans le DAG peuvent, sans obligation, correspondre à des dépendances spécifiques aux applications ; par exemple, la transaction d'un enfant ne dépense pas nécessairement, ni n'a forcément de relation avec les fonds reçus lors de la transaction parent. Nous utilisons le terme *ensemble ancêtre* pour nous référer à toutes les transactions accessibles par les arêtes parent historisées, et *progéniture* pour nous référer à toutes les transactions enfant et leur descendance.

```
1:  procedure INIT
2:      𝒯 := ∅   // l'ensemble des transactions connues
3:      Q := ∅   // l'ensemble des transactions interrogées
4:  procedure ONGENERATETX(data)
5:      edges := {T' ← T : T' ∈ PARENTSELECTION(𝒯)}
6:      T := TX(data, edges)
7:      ONRECEIVETX(T)
8:  procedure ONRECEIVETX(T)
9:      if T ∉ 𝒯 then
10:         if 𝒫_T = ∅ then
11:             𝒫_T := {T}, 𝒫_T.pref := T
12:             𝒫_T.last := T, 𝒫_T.cnt := 0
13:         else 𝒫_T := 𝒫_T ∪ {T}
14:         𝒯 := 𝒯 ∪ {T}, c_T := 0.
```

Figure 8: Avalanche : génération des transactions.

Le principal défi de la maintenance du DAG est de choisir parmi des *transactions conflictuelles*. La notion de conflit est définie par l'application et est transitive, formant une relation d'équivalence. Dans notre application de cryptomonnaie, les transactions qui dépensent les mêmes fonds (*double-dépense*) entrent en conflit et forment un *ensemble de conflits* (régions ombrées en figure 11), duquel une seule peut être acceptée. On note que l'ensemble de conflits d'une transaction vertueuse est toujours un singleton.

Avalanche instancie une instance de Snowball pour chaque ensemble de conflit. Alors que Snowball utilise des requêtes répétées et plusieurs compteurs pour refléter le degré de confiance associé aux transactions conflictuelles (couleurs), Avalanche profite de la structure du DAG et utilise la progéniture d'une transaction. En particulier, quand on interroge une transaction T, toutes les transactions accessible depuis T en suivant les arêtes du DAG font implicitement partie de la requête. Un nœud ne répond positivement à la requête que si T et toute son ascendance représente actuellement l'option préférentielle dans leurs ensembles de conflits respectifs. Si un nombre de nœuds répondant positivement est supérieur au seuil, on dit que la transaction récolte un *chit* (un bon). Les nœuds calculent ensuite leur *confiance* en comptant le nombre de chits dans la progéniture de cette transaction. Ils n'envoient qu'une seule requête et se basent sur les nouveaux sommets les chits possibles, ajoutés à la progéniture, pour fonder leur confiance. Les liens sont brisés par une préférence initiale pour les transactions vues en premier. On note que les chits sont découplés de la structure du DAG, rendant le protocole immune aux attaques où un attaquant génère de grands sous-graphes remplis de fausses valeurs.

5.2 Avalanche : Spécification

Chaque nœud correct u conserve la trace de toutes les transactions dont il a été informé dans l'ensemble $\mathcal{T}_u$, partitionné en ensembles de conflits mutuellement exclusifs $\mathcal{P}_T$, $T \in \mathcal{T}_u$. Comme les conflits sont transitifs, si T_i et T_j sont en conflit,

```
1: procedure AVALANCHELOOP
2:     while true do
3:         trouver T satisfaisant T ∈ 𝒯 ∧ T ∉ 𝒬
4:         𝒦 ≔ SAMPLE(𝒩\u, k)
5:         P ≔ Σ_{v∈𝒦} QUERY(v, T)
6:         if P ≥ α then
7:             c_T ≔ 1
8:             // mettre à jour les préférences pour les ancêtres
9:             for T′ ∈ 𝒯 : T′ ←* T do
10:                if d(T′) > d(𝒫_{T′}.pref) then
11:                    𝒫_{T′}.pref ≔ T′
12:                if T′ ≠ 𝒫_{T′}.last then
13:                    𝒫_{T′}.last ≔ T′, 𝒫_{T′}.cnt ≔ 1
14:                else
15:                    ++𝒫_{T′}.cnt
16:        else
17:            for T′ ∈ 𝒯 : T′ ←* T do
18:                𝒫_{T′}.cnt ≔ 0
19:        // sinon, c_T reste éternellement à 0
20:        𝒬 ≔ 𝒬 ∪ {T}   // marquer T comme interrogé
```

Figure 9: Avalanche : boucle principale.

```
1: function ISPREFERRED(T)
2:     return T = 𝒫_T.pref
3: function ISSTRONGLYPREFERRED(T)
4:     return ∀T′ ∈ 𝒯, T′ ←* T : ISPREFERRED(T′)
5: function ISACCEPTED(T)
6:     return
   ((∀T′ ∈ 𝒯, T′ ← T : ISACCEPTED(T′))
    ∧ |𝒫_T| = 1 ∧ 𝒫_T.cnt ≥ β1)   // engagement préalable sûr
   ∨(𝒫_T.cnt ≥ β2)   // compteur consécutif

7: procedure ONQUERY(j, T)
8:     ONRECEIVETX(T)
9:     RESPOND(j, ISSTRONGLYPREFERRED(T))
```

Figure 10: Avalanche : primitives de vote et de décision.

ils appartiennent au même ensemble de conflits, c'est-à-dire $\mathcal{P}_{T_i} = \mathcal{P}_{T_j}$. Cette relation peut sembler contre-intuitive : les transactions en conflit ont la même relation d'*équivalence* car ce sont des équivoques dépensant les *mêmes* fonds.

Nous écrivons $T' \leftarrow T$ si T a une arête commune avev la transaction T'. La relation "$\overset{*}{\leftarrow}$" est sa fermeture réflexive transitive, indiquant un chemin de T vers T'. Les DAG construits par des nœuds différents offrent la garantie d'être compatibles, bien qu'à un moment donné les nœuds puissent ne pas avoir une vue complète de tous les sommets dans le système. En particulier, si $T' \leftarrow T$, alors tout nœud dans le système qui possède T aura également T' et la même relation $T' \leftarrow T$; inversement, si f $T' \not\leftarrow T$, alors aucun nœud n'arrivera à $T' \leftarrow T$.

Chaque nœud u peut calculer une valeur de confiance $d_u(T)$

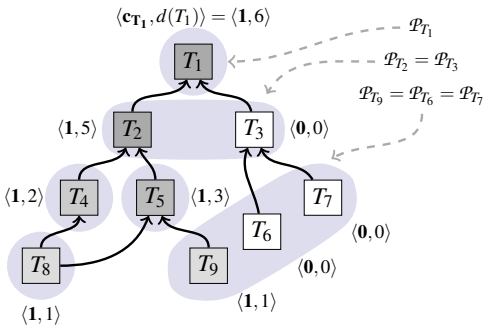

Figure 11: Exemple de valeurs pour $\langle \text{chit}, \text{confiance} \rangle$. Les boîtes sombres indiquent des transactions dont l'indice de confiance est supérieur. Au plus une transaction dans chaque région ombrée sera acceptée.

de sa progéniture comme suit :

$$d_u(T) = \sum_{T' \in \mathcal{T}_u, T \overset{*}{\leftarrow} T'} c_{uT'}$$

où $c_{uT'}$ représente la valeur de chit de T' pour le nœud u. Chaque transaction a un valeur de chit à 0 avant que le nœud obtienne les résultats des requêtes. Si le nœud récolte un seuil de α votes « oui » après la requête, la valeur $c_{uT'}$ est mise à 1, sinon elle reste éternellement à 0. Donc, une valeur de chit reflète le résultat de la requête à un moment donné de ses transactions associées et devient immuable après cela, alors que $d(T)$ peut augmenter au fur et à mesure de la croissance du DAG en collectant d'autres chits dans sa progéniture. Comme $c_T \in \{0,1\}$, les valeurs de confiance sont monotones.

De plus, le nœud u maintient sa propre liste locale de nœuds connus $\mathcal{N}_u \subseteq \mathcal{N}$ qui constituent le système. Dans un souci de simplicité, nous supposons pour l'instant que $\mathcal{N}_u = \mathcal{N}$, et nous éludons le u souscrit dans les contextes sans ambiguïté.

Chaque nœud implémente une machine à état orientée événements, centrée autour d'une requête qui sert à la fois à solliciter des votes pour chaque transaction et à notifier d'autres nœuds de l'existence de transactions nouvellement découvertes. En particulier, quand le nœud u découvre une transaction T au cours d'une requête, il commence un processus de requête unique en échantillonnant k pairs au hasard et en leur envoyant un message après que T a été délivré par ONRECEIVETX.

Le nœud u répond à une requête en vérifiant si chaque T' telle que $T' \overset{*}{\leftarrow} T$ est actuellement préférée par rapport aux transactions concurrentes $\forall T'' \in \mathcal{P}_{T'}$. Si chacun des ancêtres T' répond à ce critère, la transaction est dite *fortement préférée*, et reçoit un vote oui (1). Un manquement à ce critère pour toute T' engendre un vote non (0). Quand u accumule k réponses, il vérifie s'il existe Le processus ci-dessus étiquette le DAG avec une valeur de chit et une confiance associée pour chaque transaction T.

La figure 11 illustre un exemple de DAG construit par Avalanche. De même que pour Snowball, l'échantillonnage

dans Avalanche crée une rétroaction positive pour la préférence d'une seule transaction dans son ensemble de conflits. Par exemple, comme T_2 a un indice de confiance supérieur à celui de T_3, ses descendants seront plus à même de récolter des chits à l'avenir que T_3.

Comme Bitcoin, Avalanche laisse à l'application le soin de déterminer le point d'acceptation d'une transaction. Une application fournit un prédicat isAccepted qui peut prendre en compte la valeur en jeu dans la transaction et les chances de réversion d'une décision pour déterminer le moment de cette décision.

L'engagement concernant une transaction peut être effectué par un *engagement préalable sûr*. Pour les transactions vertueuses, T est acceptée quand elle est la seule transaction dans son ensemble de conflits et que sa confiance n'est pas inférieure à un seuil β_1. Comme dans Snowball, T peut aussi être acceptée après un nombre de requêtes réussies consécutives β_2. Si une transaction vertueuse échoue à être acceptée en raison d'un problème avec ses parents, elle peut être acceptée si elle est réémise avec des parents différents. La figure 8 montre comment Avalanche lie les transactions entre elles. Puisque les transactions qui consomment et génèrent la même UTXO n'entrent pas en conflit entre elles, toute transaction peut être réémise avec des parents différents.

La figure 9 illustre la boucle principale du protocole exécutée par chaque nœud. À chaque itération, le nœud tente de sélectionner une transaction T qui n'a pas encore été interrogée. S'il n'existe pas de telle transaction, la boucle se met en pause jusqu'à ce qu'une nouvelle transaction soit ajoutée à $\mathcal{T}$. Il sélectionne ensuite k pairs et les interroge. Si plus que α de ces pairs reviennent à une réponse positive, la valeur de chit est mise à 1. Après cela, il met à jour la transaction qu'il préfère de chaque ensemble de conflits des transactions dans son ascendance. Ensuite, T est ajoutée à l'ensemble Q afin de ne plus jamais être interrogée par le nœud. Le code qui sélectionne des pairs supplémentaires sur certains des k pairs ne répondent pas est omis dans un but de simplicité.

La figure 10 montre ce qui se produit quand un nœud reçoit une requête pour la transaction T du pair j. Il ajoute d'abord T à $\mathcal{T}$, à moins qu'il ne l'ait déjà. Il détermine ensuite si T est actuellement fortement préféré. Dans ce cas, le nœud renvoie une réponse positive au pair j. Sinon, il renvoie une réponse négative. On remarque que, dans le pseudocode, nous assumons que quand un nœud connaît T, il connaît aussi récursivement toute son ascendance. Cela peut être réalisé en retardant la délivrance de T jusqu'à ce que l'intégralité de son ascendance soit récursivement récupérée. En pratique, un processus *gossip* supplémentaire qui dissémine des transactions est utilisé en parallèle, mais n'est pas montré dans le pseudocode dans un but de simplicité.

5.3 Transactions à UTXO multi-entrées

En addition à la structure de DAG dans Avalanche, un graphe de *unspent transaction output* (UTXO) [45], ou sortie de transaction non dépensée, qui reflète les dépendances des dépenses est utilisé pour réaliser le registre du système de paiement. Pour lever toute ambiguïté, nous appelons les transactions qui codent les données de transfert d'argent *transactions* alors que nous appelons les transactions ($T \in \mathcal{T}$) dans le DAG d'Avalanche des *sommets*.

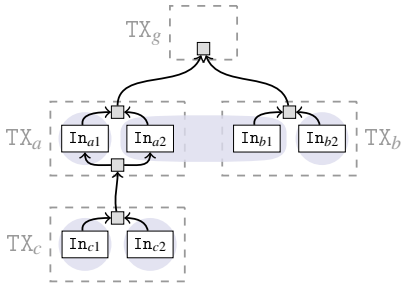

Figure 12: La structure de DAG logique sous-jacente utilisée par Avalanche. Les petits carrés avec des ombres sont des sommets fictifs qui ne servent qu'à former la topologie du DAG dans un but de clarté, et peuvent être remplacés par des arêtes directes. Les régions grises arrondies sont les ensembles de conflits.

Nous héritons des mécanismes de transaction et d'adresse de Bitcoin. En simplifiant à l'extrême, les transactions consistent en multiples entrées et sorties, avec les scripts d'échange correspondants. Les adresses sont identifiées par l'empreinte de leurs clefs publiques et les signatures sont générées par les clefs privées. Le langage de script est utilisé pour s'assurer qu'un script d'échange est authentifié pour dépenser une UTXO. Les UTXO sont totalement consommées par une transaction valide et peuvent engendrer de nouvelles UTXO que des destinataires nommés peuvent dépenser. Les transactions multi-entrées consomment plusieurs UTXO et, dans Avalanche, peuvent apparaître dans plusieurs ensembles de conflits. Pour les prendre correctement en compte, nous représentons les paires d'entrées de transactions (*transaction-input*), par exemple $\texttt{In}_{a1}$, comme des sommets Avalanche. La relation des conflits de paires de sorties de transation est transitive car une seule paire ne dépense qu'une sortie non dépensée. Nous utilisons la conjonction de isAccepted pour toutes les entrées d'une transaction pour nous assurer qu'aucune transaction ne sera acceptée à moins que toutes ses entrées ne soient acceptées (figure 12). En d'autres termes, une transaction n'est acceptée que si toutes ses paires d'entrées de transaction sont acceptées dans leurs ensembles de conflits Snowball respectifs. Selon cette idée, nous pouvons finalement implémenter le DAG des paires d'entrées de transactions tel que plusieurs transactions peuvent être regroupées par requête.

Optimisations Nous implémentons quelques optimisations pour que le système puisse passer à l'échelle. D'abord, nous utilisons des *lazy updates*, des mises à jour retardées, du DAG, car la définition récursive de la confiance pourrait engendrer une traversée du DAG coûteuse.

Nous maintenons la valeurs $d(T)$ courante pour chaque sommet actif sur le DAG, et nous ne le mettons à jour que quand un sommet de la progéniture obtient un *chit*. Comme le chemin de recherche peut être purgé sur des sommets acceptés, le coût d'une mise à jour est constant si les sommets rejetés ont un nombre limité de descendant et si la région non décidée du DAG reste de taille constante. Ensuite, l'ensemble de conflits peut être vaste en pratique, car un client malveillant peut générer un volume important de transactions conflictuelles. Au lieu de conserver une structure de données comme un conteneur pour chaque ensemble de conflits, nous créons une correspondance à partir de chaque UTXO vers la transaction préférée qui peut représenter tout l'ensemble de conflits. Cela permet à un nœud de déterminer rapidement les conflits à venir ainsi que la réponse appropriée aux requêtes. Enfin, nous accélérons le processus de requête en terminant au plus tôt, dès que le seuil α est atteint, sans attendre les k réponses.

DAG Comparé à Snowball, Avalanche introduit une structure de DAG qui lie fortement le sort d'ensembles de conflits sans relation entre eux, chacun étant une instance à décret unique. Cette intrication concrétise une tension : l'attachement d'une transaction à des parents non décidés contribue à la décision concernant les transactions, tandis que les transactions risquent des défaillances de vitalité quand les parents s'avèrent malveillants. Nous pouvons résoudre cette tension et fournir une garantie de vitalité à l'aide de deux mécanismes.

D'abord nous adoptons une stratégie adaptative de sélection des parents, où les transactions sont attachées sur le bord actif du DAG et sont réessayées avec de nouveaux parents plus proche du sommet genèse. Cette procédure garantit un achèvement avec des parents incontestés, assurant ainsi qu'une transaction ne peut souffrir de défaillance de vitalité en raison de transactions contestées et malveillantes. Un mécanisme secondaire s'assure que les transactions vertueuses avec une ascendance décidée reçoivent suffisamment de chits. Les nœuds corrects examinent le DAG pour y trouver des transactions vertueuses manquant d'une progéniture suffisante et émettent des transactions *no-op* pour accroître leur confiance. Ces deux mécanismes mis en place, il est facile de voir qu'au pire Avalanche dégénérera en instances distinctes de Snowball, fournissant ainsi la même garantie de vitalité pour les transactions vertueuses.

Contrairement à d'autres cryptomonnaies [50] qui utilisent directement les sommets du graphe comme votes, Avalanche n'utilise un DAG que pour regrouper les requêtes dans les instances Snowball sous-jacentes.

Comme la confiance se fonde sur les chits récoltés, et non simplement par la présence d'un sommet, la simple inondation du réseau par des sommets attachés à une partie rejetée du sous-graphe ne subvertit pas le protocole.

5.4 Complexité des communications

Soit un facteur de ramification attendu de p dans Avalanche, correspondant à la largeur du DAG, et déterminé par l'algorithme de sélection du parent. Étant donné le seuil de décision β_1 et β_2, une transaction qui vient d'atteindre le point de décision aura une progéniture associée $\mathcal{Y}$. Soit m la profondeur attendue de $\mathcal{Y}$. Si nous devions permettre au réseau Avalanche de progresser puis de geler le DAG à une profondeur y, alors il aurait à peu près py sommets/transactions, desquels on pourrait attendre que $p(y-m)$ soient décidés. Seules pm transactions récentes manqueraient de la progéniture nécessaires à une prise de décision. Pour chaque nœud, chaque requête demande k échantillons, et donc on attend un coût total des messages par transaction de $(pky)/(p(y-m)) = ky/(y-m)$. Comme m est une constante déterminée par la région non décidée du DAG pendant que le système progresse constamment, la complexité en messages par nœuds est $O(k)$ alors que la complexité totale est $O(kn)$.

6 Evaluation

6.1 Configuration de test

Nous conduisons nos expériences sur Amazon EC2 en faisant tourner une plage allant d'une centaine (125) à des milliers (2000) d'instances de machines virtuelles. Nous utilisons des instances `c5.large`, chacune simulant un noeud individuel. AWS fournit une bande passante allant jusqu'à 2 gigabits par seconde, sachant que le protocole Avalanche utilise au maximum 100 mégabits par seconde environ.

Notre implémentation supporte deux types de transactions : la première est le format d'UTXO customisé, tandis que l'autre repose directement sur le code de Bitcoin 0.16. Ces deux formats supportés utilisent la librairie de cryptographie secp256k1 issue du projet Bitcoin et exposent le même format d'adresse pour les portefeuilles. Toutes nos expériences utilisent le format customisé à l'exception de la géo-réplication, pour laquelle les résultats sont donnés pour les deux types de transactions.

Nous simulons un flux continu de nouvelles transactions venant des utilisateurs en créant des processus client séparés, chacun de ces processus gérant un portefeuille différent, qui génèrent des transactions à destination de nouvelles adresses de réception puis envoient les requêtes vers les noeuds Avalanche.

Nous utilisons plusieurs de ces processus client pour atteindre la limite de capacité de notre système. Le nombre de récepteurs pour chaque transaction est paramétré pour obtenir une taille de transaction moyenne d'environ 250 octets (1–2 entrées/sorties par transaction en moyenne et une taille d'UTXO fixe), la taille de transaction moyenne actuelle sur Bitcoin. Afin d'utiliser le réseau efficacement, nous regroupons jusqu'à 40 transactions en une requête, tout en gardant des valeurs de confiance à un niveau de granularité équivalent à une seule transaction.

Tous les indicateurs rapportés ici reflètent les mesures de bout en bout prises depuis le point de vue de tous les clients. Plus précisément, les clients examinent le nombre total de

transactions confirmées par seconde pour mesurer le débit global, et, pour chaque transaction, ils soustraient l'horodatage d'initiation de la transaction à l'horodatage de confirmation de celle-ci pour en mesurer la latence. Chaque expérience de mesure du débit est répétée 5 fois et l'écart type est mentionné dans chaque graphique.

Pour ce qui est des paramètres de sécurité, nous prenons $k = 10$, $\alpha = 0.8$, $\beta_1 = 11$, $\beta_2 = 150$, ce qui donne un MTTF de ~10^{24} années.

6.2 Débit de transactions

Nous mesurons d'abord le débit du système en le saturant avec des transactions tout en examinant la fréquence à laquelle les transactions sont confirmées à l'état nominal. Pour cette expérience, nous lançons d'abord Avalanche sur 125 noeuds exécutant chacun 10 processus client qui génèrent un nombre de 400 transactions sortantes à tout moment.

Comme le montre le premier groupe de barres dans la figure 13, le système atteint 6851 transactions par seconde (tps) pour une taille de batch de 20 et plus de 7002 tps pour une taille de batch de 40. Notre système sature plus vite avec des petites tailles de batch en comparaison avec d'autres blockchains dont les performances sont connues : Bitcoin regroupe des batchs de plusieurs milliers de transactions par bloc, Algorand [30] gère des blocks de 2-10 mégaoctets, c'est-à-dire 8.4–41.9K transactions par batch et Conflux [40] gère des blocs de 4 mégaoctets, c'est-à-dire 16.8K tx/batch. Ces systèmes sont relativement lents pour prendre une simple décision, et demandent de surcroît une taille de batch (et donc de bloc) importante pour obtenir de meilleures performances. Atteindre un haut niveau de débit transactionnel avec une petite taille de batch implique une latence réduite, comme nous le démontrons plus loin.

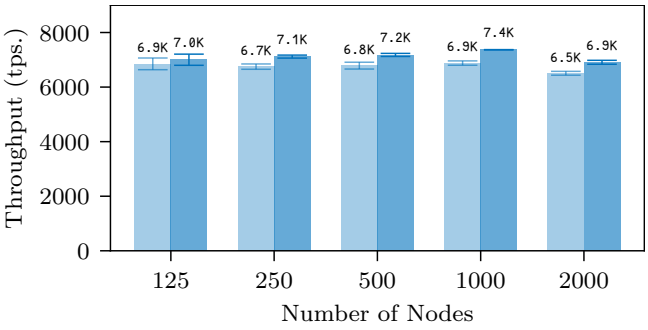

Figure 13: Débit transactionnel en fonction de la taille du réseau. Chaque paire de barres est produite avec une taille de batch de 20 et 40, de gauche à droite.

6.3 Mise à l'échelle

Afin d'observer la manière dont le système passe à l'échelle en terme de nombre de noeuds participant au consensus Avalanche, nous effectuons des expériences en gardant des paramètres identiques tout en faisant varier le nombre de noeuds de 125 jusqu'à 2000.

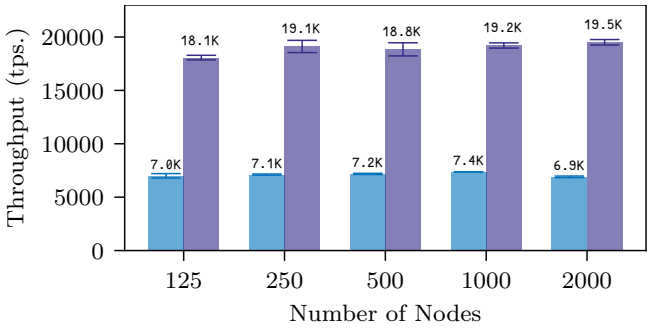

Figure 14: Débit transactionnel pour une taille de batch de 40, avec (gauche) et sans (droite) vérification de la signature.

La figure 13 montre que le débit global se dégrade d'environ 1.34% à 6909 tps lorsque le réseau augmente d'un facteur 16 pour atteindre $n = 2000$. Cette dégradation est mineure si on la compare au taux d'augmentation du réseau. Notez que l'axe X est logarithmique.

Avalanche tire sa bonne mise à l'échelle de plusieurs sources : premièrement en maintenant un ordre partiel qui capture uniquement les relations entre les dépenses ce qui permet une meilleure parallélisation qu'un journal BFT répliqué classique qui traite toutes les transactions de manière linéaire ; deuxièmement l'absence de dirigeant permet d'éviter les goulets d'étranglement ; et finalement le nombre de messages que chaque noeud doit gérer par décision est $O(k)$ et n'augmente pas au fur et à mesure que le réseau s'agrandit.

6.4 Engorgement dû à la cryptographie

Nous examinons ensuite où se situent les goulets d'étranglement dans notre implémentation. La barre violette à la droite de chaque groupe dans la figure 14 montre le débit d'Avalanche lorsque la vérification des signatures est désactivée. Les débits s'améliorent d'environ 2.6x, par par rapport à la barre bleue sur la gauche. Cela révèle que le surplus de calcul dû à la vérification cryptographique est le principal goulet d'étranglement de l'implémentation de notre système. On peut pallier ce problème en déléguant la vérification des transactions à un GPU. Même en l'absence d'une telle optimisation, 7K tps dépasse de loin ce qui existe en matière de blockchain.

6.5 Latence

La latence d'une transaction est le temps écoulé à partir du moment où elle est soumise au réseau jusqu'à ce qu'elle soit confirmée et donc acceptée par le réseau.

La figure 15 représente un histogramme montrant la distribution des latences en utilisant la même configuration que pour les mesures de débits à partir de 2000 noeuds. L'axe X représente le temps en secondes tandis que l'axe Y est la portion des transactions qui sont finalisées dans la période de temps correspondante. Cette figure met également en avant la

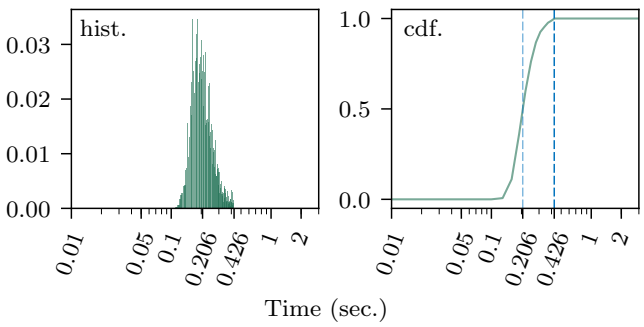

Figure 15: Distribution de la latence de transaction pour $n = 2000$. L'axe X représente la latence de transaction en secondes sur une échelle logarithmique, tandis que l'axe Y est la portion des transactions qui tombe dans la période de confirmation (normalisée à 1). L'histogramme de toutes les latences de transaction pour un client est visible sur la gauche avec 100 bins, et son CDF visible sur la droite.

Fonction de Distribution Cumulée (CDF) en accumulant le nombre de transactions finalisées sur la période.

Cette expérience démontre que la plupart des transactions sont confirmées en moins de 0.3 secondes environ. La majorité des latences se trouve aux alentours des 206 ms avec une variance relativement faible, ce qui indique que les noeuds convergent sur la décision finale en tant que groupe environ au même moment. La deuxième ligne verticale montre la latence maximale observée, qui se situe aux alentours des 0.4 secondes.

La figure 16 montre la latence de transaction pour un nombre variable de noeuds. Les arêtes horizontales des boîtes représentent respectivement du bas vers le haut le minimum, le premier quartile, la médiane, le troisième quartile, et le maximum de latence. En définitive, les données expérimentales montrent que la latence médiane est plus ou moins indépendante de la taille du réseau.

6.6 Clients malveillants

Nous évaluons ensuite comment des transactions conflictuelles générées par des clients malveillants effectuant des double dépenses sur des sorties non dépensées peuvent affecter la latence des transactions légitimes créées par des clients corrects. Nous adoptons une stratégie qui simule des clients malveillants où une fraction (de 0% à 25%) des transactions en attente sont en conflit avec d'autres transactions existantes.

Le processus client effectue ceci en désignant des flux de transactions contenant des double dépenses parmi toutes les transactions simulées en attente, et envoie les transactions conflictuelles vers différents noeuds. Nous utilisons pour cela les mêmes paramètres avec $n = 1000$ comme dans les expériences précédentes, et nous mesurons uniquement le débit et la latence des transactions confirmées.

La latence d'Avalanche est seulement légèrement influencée par les clients malveillants, comme le montre la figure 17. Étonnamment, la latence maximale baisse légèrement lorsque

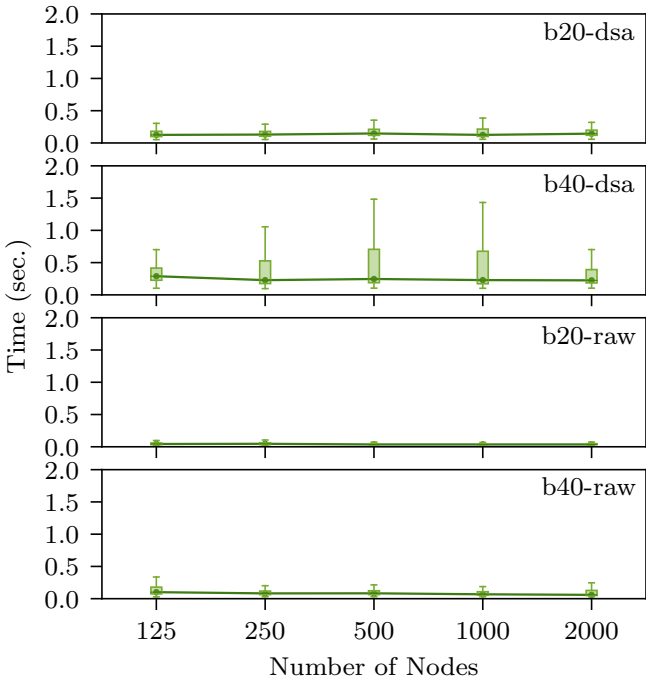

Figure 16: Latence de transaction en fonction de la taille du réseau. "b" indique la taille de batch et "raw" est l'occurence sans la vérification des signatures.

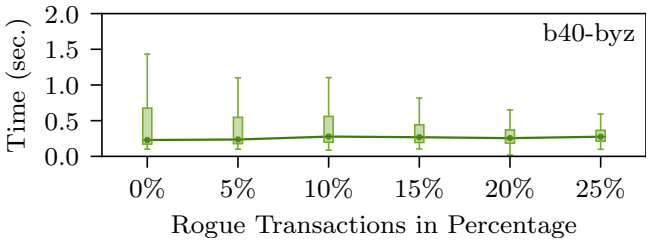

Figure 17: Latence en fonction du taux de transactions malveillantes.

le pourcentage de transactions conflictuelles augmente. Ce comportement s'explique par le fait que lorsqu'on introduit de telles transactions, le débit *effectif* est réduit et soulage donc la charge du système.

Cette observation est confirmée par la figure 18 qui montre que le débit (de transactions légitimes) decroît avec le taux de transactions conflictuelles. De plus, la baisse du débit apparaît proportionnelle au nombre de clients malveillants, ce qui démontre que les attaques potentielles ne bénéficient pas d'un effet de levier.

6.7 Géo-réplication

L'expérience suivante place le système dans l'émulation d'un scénario géo-répliqué, modélisé sur le même scénario détaillé dans un travail préliminaire [30]. Nous avons sélectionné 20 grandes villes qui semblent être localisées à proximité d'un nombre conséquent de noeuds Bitcoin joignables, d'après [10]. Ces villes couvrent l'Amérique du Nord, l'Europe, l'Asie

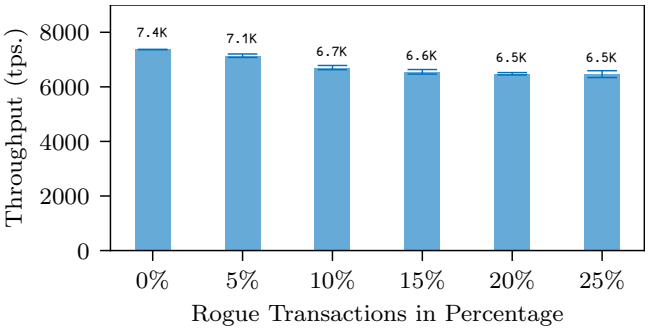

Figure 18: Débit de transactions en fonction du taux de transactions conflictuelles.

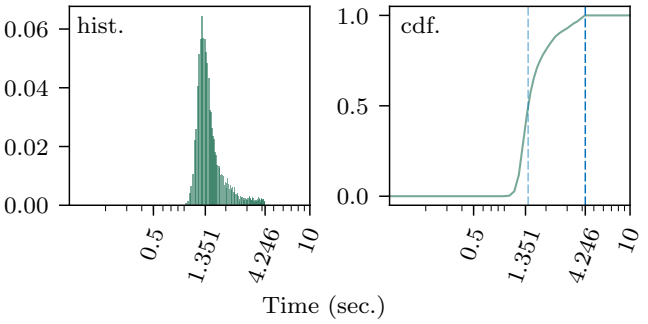

Figure 19: Histogramme de latence/CDF pour $n = 2000$ réparties dans 20 villes.

occidentale, l'Asie orientale, l'Océanie, et incluent également les 10 villes où l'on trouve le plus grand nombre de noeuds joignables. Nous utilisons les données de latence obtenues depuis [60] et nous émulons la latence des paquets réseau dans le noyau Linux en utilisant les commandes `tc` et `netem`. 2000 noeuds sont distribués équitablement parmi toutes les villes, sans émuler de latence supplémentaire entre les noeuds d'une même ville. Comme pour l'évaluation d'Algorand, nous limitons aussi la bande passante par processus à 20 mégabits par seconde dans le but de simuler des paramètres à l'échelle d'internet d'un réseau comportant de nombreux liens. Nous assignons ensuite un processus client à chaque ville, qui génère 400 transactions sortantes par ville à tout moment.

Dans ce scénario, Avalanche atteint un débit de transactions moyen de 3401 tps, avec un écart type de 39 tps. Comme le montre la figure 19, la latence de transaction médiane est de 1.35 secondes, avec une latence maximale de 4.25 secondes. Nous supportons également le code initial de Bitcoin pour les transactions, qui dans ce cas donne un débit de 3530 tps avec $\sigma = 92$ tps.

6.8 Comparaison avec d'autres systèmes

Même s'il existe une abondance de protocoles blockchain et de cryptomonnaies, la plupart d'entre eux présentent une ébauche de leur protocole sans proposer d'implémentation pratique ou de résultats d'évaluation. En outre, parmi ceux qui proposent effectivement des résultats, la plupart ne sont pas évalués dans des conditions réalistes ou à grande échelle (des centaines de milliers de noeuds complets participant au consensus).

En conséquent, nous choisissons Algorand et Conflux pour notre étude comparative. Algorand, Conflux et Avalanche sont tous fondamentalement différents dans leur conception. L'algorithme de consensus par comité d'Algorand repose sur un accord byzantin basé sur un quorum, et Conflux supplémente le consensus de Nakamoto par une structure de type graphe orienté acyclique (DAG) pour permettre un meilleur débit, tandis qu'Avalanche appartient à une nouvelle famille de protocoles basés sur la méta-stabilité. Par ailleurs, nous utilisons Bitcoin [45] comme base technique.

Les évaluations d'Algorand et d'Avalanche reposent toutes deux sur un réseau décisionnel de 2000 noeuds sur EC2. Notre évaluation se fait sur des instances `c5.large` partagées, tandis qu'Algorand utilise des instances `m4.2xlarge`. Ces deux plateformes sont très similaires à l'exception d'un léger avantage en terme de fréquence CPU pour le `c5.large`, qui au final n'est que partiellement utilisé puisque nos processus ne consomment que 30% de temps processeur lors de ces expériences. Les paramètres de sécurité choisis pour nos expériences garantissent une probabilité de violation de sécurité en dessous de 10^{-9} en présence de 20% de noeuds byzantins, en comparaison de l'évaluation Algorand qui garantit une probabilité de violation en dessous de 5×10^{-9} avec 20% de noeuds byzantins.

Ni Algorand ni Conflux ne prennent en compte le surplus de calcul lié à la vérification cryptographique dans leurs études. Leurs évaluations utilisent des blocs qui contiennent des mégaoctets de données factices et présentent les mesures de débit en mégaoctets ou gigaoctets par heure. Nous utilisons donc la taille moyenne d'une transaction Bitcoin, 250 octets, pour en déduire leurs débits de transactions. Par contraste, nos expériences transportent des données de transactions réelles et prennent entièrement en compte le temps de calcul des vérifications cryptographiques.

Le débit de transactions est de 3-7 tps pour Bitcoin, 874 tps pour Algorand (avec des blocs de 10 mégaoctets), 3355 tps pour Conflux (le papier fait mention d'un débit 3.84 fois supérieur au débit d'Algorand dans les mêmes conditions).

En comparaison, Avalanche obtient constamment plus de 3400 tps sur un réseau allant jusqu'à 2000 noeuds sans aucun comité de décision ni preuve de travail. Pour ce qui est de la latence, une transaction est confirmée après 10–60 minutes pour Bitcoin, environ 50 secondes pour Algorand, 7.6–13.8 minutes avec Conflux, et 1.35 secondes pour Avalanche.

Avalanche se comporte bien mieux qu'Algorand à la fois en terme de débit et de latence étant donné qu'Algorand utilise une fonction aléatoire vérifiable pour élire ses comités, et maintient un journal ordonné pendant qu'Avalanche établit un ordre partiel uniquement. Bien que l'entité à la tête d'Algorand soit anonyme et change continuellement, elle reste néanmoins basée sur un dirigeant unique qui pourrait être un facteur limitant pour le passage à l'échelle, tandis qu'Avalanche n'est

pas contrôlé par une entité unique.

Avalanche a un débit similaire à Conflux, mais sa latence est 337–613 fois meilleure. Conflux utilise également une structure DAG pour amortir le coût du consensus et augmenter le débit de transactions, cependant il est toujours ancré sur un consensus de Nakamoto (PoW) le rendant incapable de confirmer des transactions instantanément comme le fait Avalanche.

Dans un système à base de blockchain, on ne peut généralement améliorer le débit des transactions qu'aux dépens de la latence à travers le regroupement par batchs. Le vrai facteur limitant au niveau de la performance est le nombre de décisions qu'un système est en mesure de prendre en une seconde, et ce paramètre est fondamentalement limité soit par l'accord byzantin (BA*) chez Algorand soit par le consensus de Nakamoto pour Conflux.

7 Travaux associés

Bitcoin [45] est une cryptomonnaie qui utilise une blockchain basée sur la preuve de travail (PoW pour *Proof Of Work*) et qui tient un livre de comptes des « sorties de transaction non dépensées », ou UTXO (*Unspent Transaction Output*). Bien que des techniques basées sur la preuve de travail [5, 26], et même d'autres cryptomonnaies dont l'émission est basée sur la preuve de travail [51, 59], aient été explorées avant l'apparition de Bitcoin, ce dernier fut le premier à incorporer la preuve de travail dans son processus de consensus. Contrairement aux protocoles résistants aux fautes byzantines traditionnels, Bitcoin apporte une garantie de sûreté probabiliste et considère que la majorité de la puissance de calcul agit de manière honnête plutôt que de contrôler les participants au réseau, ce qui a permis de faire émerger un protocole sans permission à l'échelle d'internet. Bien qu'il soit sans permission et résilient face aux attaques, Bitcoin souffre d'un débit de transaction faible (~3 tps) et d'une latence élevée (~5.6 heures pour un réseau comportant 20% de nœuds byzantins et une garantie de sûreté de 2^{-32}). De plus la preuve de travail nécessite une puissance de calcul considérable qui est consommée dans le seul et unique but de maintenir la sûreté de fonctionnement.

D'innombrables cryptomonnaies utilisent la preuve de travail [5, 26] dans le but de tenir un livre de comptes distribué. Tout comme Bitcoin, elles souffrent des mêmes limitations lorsqu'il s'agit de monter en charge. Il existe plusieurs propositions de protocoles essayant de faire meilleur usage des progrès apportés par la preuve de travail. Bitcoin-NG [27] et la version sans permission de Thunderella [48] utilise un consensus similaire à celui de Nakamoto pour élire un dirigeant qui va dicter les écritures dans le journal répliqué pour un temps donné assez long dans le but de fournir un meilleur débit de transactions. En outre, Thunderella apporte une borne optimiste qui, en considérant 3/4 de puissance de calcul et un dirigeant élu honnête, permet de confirmer les transactions rapidement. ByzCoin [37] sélectionne un échantillon réduit de participants de manière périodique et exécute ensuite un protocole similaire à PBFT entre les nœuds sélectionnés.

Les protocoles basés sur un accord byzantin [39, 49] se basent généralement sur un quorum et nécéssitent une connaissance précise des membres du réseau. PBFT [15, 16], l'un de ces protocoles les plus connus, nécessite un nombre quadratique d'échanges de messages avant de parvenir à un accord. Le protocole Q/U [2] et la réplication HQ [19] utilisent une approche basée sur un quorum afin d'optimiser le protocole pour obtenir des cas de fonctionnement sans contention et parvenir à un consensus en un seul tour de communications. Malgré tout, même si ces protocoles améliorent les performances, ils se dégradent fortement lorsqu'ils sont soumis à une forte contention. Zyzzyva [38] ajoute au protocole BFT une exécution spéculative pour tendre vers un cas de fonctionnement sans erreurs. Les travaux antérieurs portant sur les systèmes résistants aux fautes byzantines avec contrôle d'accès nécessitent typiquement au moins $f+1$ répliques. CheapBFT [34] tire profit de composants matériels certifiés pour construire un protocole nécessitant exactement $f+1$ répliques.

D'autres travaux tentent d'introduire de nouveaux protocoles qui redéfinissent et relâchent les contraintes du modèle de résistance aux fautes byzantines. Large-scale BFT [52] modifie PBFT pour autoriser un nombre quelconque de répliques et un seuil d'échec en fournissant une garantie de vitalité probabilitse pour un taux d'échec donné tout en protégeant la sûreté de fonctionnement avec une probabilité élevée. Dans une autre forme de relâchement des contraintes, Zeno [54] introduit un protocole de réplication de machine à état résistante aux fautes byzantines qui échange la cohérence pour une meilleure disponibilité. Plus spécifiquement, Zeno garantit une cohérence assurée au final plutôt qu'une linéarisation de celle-ci, ce qui veut dire que les participants n'ont pas besoin d'être cohérents mais doivent se mettrent finalement d'accord une fois le réseau stabilisé. En fournissant une garantie de cohérence encore plus faible, appelée « cohérence causale par *fork-join* », Depot [42] définit un protocole garantissant la sûreté de fonctionnement pour $2f+1$ répliques.

NOW [31] uttilise des sous-quorums pour diriger des instances de consensus réduites. Cet article enseigne que de petits quorums de taille logarithmique peuvent être extraits à partir d'un échantillon potentiellement large de nœuds du réseau, ce qui permet à des instances de protocole de consensus plus petites de tourner en parallèle.

Snow White [21] et Ouroboros [36] font partie des premiers protocoles à la sûreté prouvée basés sur la preuve d'enjeu (PoS ou *Proof of Stake*). Ouroboros utilise un protocole de bascule de jeton multi-parties sécurisé dans le but d'incorporer un caractère aléatoire dans l'élection du dirigeant. Son évolution Ouroboros Praos [22] apporte la sûreté en présence d'attaquants entièrement adaptatifs. HoneyBadger [44] fournit une bonne vitalité dans un réseau comportant des latences hétérogènes.

Tendermint [11, 12] change de dirigeant à chaque bloc et

a démontré ses qualités dans une configuration à 64 nœuds. Ripple [53] a une faible latence grâce à l'utilisation de sous-réseaux certifiés collectivement au sein d'un réseau plus large. La société Ripple génère une liste de nœuds vérifiés par défaut renouvelée peu souvent, qui en fait un réseau essentiellement centralisé.

HotStuff [61, 62] améliore les coûts de communication d'une distribution quadratique vers une distribution linéaire et simplifie drastiquement les caractéristiques du protocole, même si les limitations liées à l'élection d'un dirigeant persistent. Facebook utilise Hotstuff comme le consensus principal de son projet Libra.

Dans une configuration synchrone, inspirée par Hoststuff, Sync HotStuff [3] atteint un consensus en un temps 2Δ avec un coût quadratique et contrairement à d'autres protocoles synchrones à étapes verrouillées, il opère aussi rapidement que le réseau se propage. Stellar [43] se base sur un accord byzantin fédéré dans lequel des *tranches de quorum* permettent une confiance hétérogène entre des nœuds différents. La sûreté de fonctionnement est garantie lorsque les transactions peuvent être reliées entre elles de manière transitive par des tranches de quorum. Algorand [30] utilise une fonction aléatoire vérifiable pour sélectionner un comité de nœuds qui participent à un protocole de consensus byzantin innovant.

Certains protocoles utilisent une structure de graphe orienté acyclique (DAG) en lieu et place d'une chaîne linéaire pour atteindre un consensus [6, 9, 55–57]. Au lieu de choisir la chaîne la plus longue comme dans Bitcoin, GHOST [56] définit des règles de sélection de la chaîne qui permettent de prendre en considération des transactions non présentes sur la chaîne principale, ce qui augmente son efficacité. SPECTRE [55] utilise des transactions sur le DAG pour voter de manière récursive via la preuve de travail pour atteindre un consensus, suivi par SPECTRE [55] qui obtient un ordre linéaire à partir de tous les blocs. Tout comme PHANTOM, Conflux obtient également un ordre linéaire entre les transactions via la preuve de travail utilisée sur une structure DAG, avec une meilleure résistance aux attaques à la vitalité [40]. Avalanche est différent en ce que le résultat du vote est un chit à usage unique déterminé par une requête sans preuve de travail, alors que les votes dans PHANTOM ou Conflux sont purement déterminés par le PoW dans la structure des transactions. A la manière de Thunderella, Meshcash [9] combine un protocole lent basé sur la preuve de travail et un protocole de consensus rapide qui apporte une fréquence de génération de blocs élevée quelle que soit la latence du réseau, offrant un temps de confirmation rapide. Hashgraph [6] est un protocole sans dirigeant qui construit un DAG sur la base d'un *gossip* aléatoire. Il nécessite une connaissance complète des membres du réseau à tout moment, et comme Ben-Or [8], il demande un nombre de messages exponentiel [4, 14] pour fonctionner.

8 Conclusion

Cet article a présenté une nouvelle famille de protocoles de consensus, accompagnée des outils mathématiques appropriés pour leur analyse. Ces protocoles sont extrêmement efficaces et robustes, et rassemblent les meilleures fonctionnalités des consensus classiques et Nakamoto. Ils passent bien à l'échelle, permettent un haut débit et une finalité rapide, ne nécessitent pas une connaissance précise des participants et leur fonctionnement se dégrade progressivement lors d'attaques catastrophiques.

Il reste beaucoup à faire dans ce domaine de recherche. Une amélioration possible consisterait à introduire un coordinateur d'attaques sur le réseau. Une autre serait de caractériser les garanties du système face à un adversaire dont les pouvoirs seraient limités de manière réaliste, à la suite de quoi les performances se verraient encode améliorées. D'autres mécanismes d'initialisation sophistiqués seraient souhaitables pour améliorer la vitalité d'un consensus multi-valeurs. En résumé, nous espérons que les protocoles et les techniques d'analyse présentés ici renforceront l'arsenal des développeurs de systèmes distribués et permettront de bâtir de nouveaux mécanismes légers et évolutifs.

Références

[1] Crypto-currency market capitalizations. https://coinmarketcap.com. Accessed: 2017-02.

[2] Abd-El-Malek, M., Ganger, G. R., Goodson, G. R., Reiter, M. K., and Wylie, J. J. Fault-scalable byzantine fault-tolerant services. In *ACM SIGOPS Operating Systems Review* (2005), vol. 39, ACM, pp. 59–74.

[3] Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Yin, M. Sync hotstuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Report 2019/270, 2019. https://eprint.iacr.org/2019/270.

[4] Aspnes, J. Randomized protocols for asynchronous consensus. *Distributed Computing 16*, 2-3 (2003), 165–175.

[5] Aspnes, J., Jackson, C., and Krishnamurthy, A. Exposing computationally-challenged byzantine impostors. Tech. rep., Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, 2005.

[6] Baird, L. Hashgraph consensus: fair, fast, byzantine fault tolerance. Tech. rep., Swirlds Tech Report, 2016.

[7] Banerjee, S., Chatterjee, A., and Shakkottai, S. Epidemic thresholds with external agents. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications* (2014), IEEE, pp. 2202–2210.

[8] Ben-Or, M. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing* (1983), ACM, pp. 27–30.

[9] Bentov, I., Hubácek, P., Moran, T., and Nadler, A. Tortoise and Hares Consensus: the Meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive 2017* (2017), 300.

[10] Bitnodes. Global Bitcoin nodes distribution. https://bitnodes.earn.com/. Accessed: 2018-04.

[11] Buchman, E. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[12] Buchman, E., Kwon, J., and Milosevic, Z. The latest gossip on bft consensus, 2018.

[13] Burrows, M. The chubby lock service for loosely-coupled distributed systems. In *7th Symposium on Operating Systems Design and Implementation (OSDI'06), November 6-8, Seattle, WA, USA* (2006), pp. 335–350.

[14] Cachin, C., and Vukolic, M. Blockchain consensus protocols in the wild. *CoRR abs/1707.01873* (2017).

[15] Castro, M., and Liskov, B. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999* (1999), pp. 173–186.

[16] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst. 20*, 4 (2002), 398–461.

[17] Central Intelligence Agency. The world factbook. https://www.cia.gov/library/publications/the-world-factbook/geos/da.html. Accessed: 2018-04.

[18] Chvátal, V. The tail of the hypergeometric distribution. *Discrete Mathematics 25*, 3 (1979), 285–287.

[19] Cowling, J., Myers, D., Liskov, B., Rodrigues, R., and Shrira, L. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th symposium on Operating systems design and implementation* (2006), USENIX Association, pp. 177–190.

[20] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A. E., Miller, A., Saxena, P., Shi, E., Sirer, E. G., Song, D., and Wattenhofer, R. On scaling decentralized blockchains - (a position paper). In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers* (2016), pp. 106–125.

[21] Daian, P., Pass, R., and Shi, E. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. https://eprint.iacr.org/2016/919.

[22] David, B., Gazi, P., Kiayias, A., and Russell, A. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II* (2018), pp. 66–98.

[23] Digiconomist. Bitcoin energy consumption index. https://digiconomist.net/bitcoin-energy-consumption. Accessed: 2018-04.

[24] Douceur, J. R. The sybil attack. In *International Workshop on Peer-to-Peer Systems* (2002), Springer, pp. 251–260.

[25] Draief, M., Ganesh, A., and Massoulié, L. Thresholds for virus spread on networks. In *Proceedings of the 1st international conference on Performance evaluation methodologies and tools* (2006), ACM, p. 51.

[26] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings* (1992), pp. 139–147.

[27] Eyal, I., Gencer, A. E., Sirer, E. G., and van Renesse, R. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016* (2016), pp. 45–59.

[28] Ganesh, A., Massoulié, L., and Towsley, D. The effect of network topology on the spread of epidemics. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* (2005), vol. 2, IEEE, pp. 1455–1466.

[29] Garay, J. A., Kiayias, A., and Leonardos, N. The Bitcoin Backbone Protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II* (2015), pp. 281–310.

[30] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements

for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017* (2017), pp. 51–68.

[31] Guerraoui, R., Huc, F., and Kermarrec, A.-M. Highly dynamic distributed computing with byzantine failures. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing* (2013), ACM, pp. 176–183.

[32] Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association 58*, 301 (1963), 13–30.

[33] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free coordination for internet-scale systems. In *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010* (2010).

[34] Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S. V., Schröder-Preikschat, W., and Stengel, K. Cheapbft: resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems* (2012), ACM, pp. 295–308.

[35] Keeling, M. J., and Rohani, P. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2011.

[36] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I* (2017), pp. 357–388.

[37] Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.* (2016), pp. 279–296.

[38] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. L. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst. 27*, 4 (2009), 7:1–7:39.

[39] Lamport, L., Shostak, R. E., and Pease, M. C. The byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (1982), 382–401.

[40] Li, C., Li, P., Xu, W., Long, F., and Yao, A. C. Scaling nakamoto consensus to thousands of transactions per second. *CoRR abs/1805.03870* (2018).

[41] Liggett, T. M., et al. Stochastic models of interacting systems. *The Annals of Probability 25*, 1 (1997), 1–29.

[42] Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., and Walfish, M. Depot: Cloud storage with minimal trust. *ACM Transactions on Computer Systems (TOCS) 29*, 4 (2011), 12.

[43] Mazieres, D. The Stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* (2015).

[44] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. The Honey Badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016* (2016), pp. 31–42.

[45] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, 2008.

[46] Pass, R., Seeman, L., and Shelat, A. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II* (2017), pp. 643–673.

[47] Pass, R., and Shi, E. Fruitchains: A fair blockchain. *IACR Cryptology ePrint Archive 2016* (2016), 916.

[48] Pass, R., and Shi, E. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II* (2018), pp. 3–33.

[49] Pease, M. C., Shostak, R. E., and Lamport, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (1980), 228–234.

[50] Popov, S. The tangle. https://www.iota.org/research/academic-papers. Accessed: 2018-04.

[51] Rivest, R., and Shamir, A. Payword and micromint: Two simple micropayment schemes. In *Security protocols* (1997), Springer, pp. 69–87.

[52] Rodrigues, R., Kouznetsov, P., and Bhattacharjee, B. Large-scale byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd Workshop on Hot Topics in System Dependability* (2007).

[53] Schwartz, D., Youngs, N., Britto, A., et al. The Ripple protocol consensus algorithm. *Ripple Labs Inc White Paper 5* (2014).

[54] Singh, A., Fonseca, P., Kuznetsov, P., Rodrigues, R., and Maniatis, P. Zeno: Eventually consistent byzantine-fault tolerance. In *Proceedings of the 6th USENIX*

Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA (2009), pp. 169–184.

[55] Sompolinsky, Y., Lewenberg, Y., and Zohar, A. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive 2016* (2016), 1159.

[56] Sompolinsky, Y., and Zohar, A. Secure high-rate transaction processing in Bitcoin. In *Financial Cryptography and Data Security, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers* (2015), pp. 507–527.

[57] Sompolinsky, Y., and Zohar, A. PHANTOM: A scalable blockdag protocol. *IACR Cryptology ePrint Archive 2018* (2018), 104.

[58] Tan, W. On the absorption probabilities and absorption times of finite homogeneous birth-death processes. *Biometrics* (1976), 745–752.

[59] Vishnumurthy, V., Chandrakumar, S., and Sirer, E. G. Karma: A secure economic framework for peer-to-peer resource sharing. In *Workshop on Economics of Peer-to-peer Systems* (2003), vol. 35.

[60] WonderNetwork. Global ping statistics: Ping times between wondernetwork servers. https://wondernetwork.com/pings. Accessed: 2018-04.

[61] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., and Abraham, I. Hotstuff: Bft consensus in the lens of blockchain, 2018.

[62] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., and Abraham, I. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2019), PODC '19, ACM, pp. 347–356.

A Analyse

Cette annexe contient une analyse de Slush, Snowflake et Snowball.

A.1 Préliminaires

Nous supposons le modèle réseau discuté en section 2. Soient R ("rouge") et B ("bleu") représentant deux choix conflictuels génériques. Sans perdre en généralité, nous portons notre attention sur les décomptes de B, c'est-à-dire le nombre de nœuds qui préfèrent le bleu.

Distribution hypergéométrique Chaque requête réseau de k pairs correspond à un échantillon sans remplacement depuis un réseau de n nœuds, également appelé un échantillon hypergéométrique. La variable aléatoire $\mathcal{H}(\mathcal{N},x,k) \to \{0,\dots,k\}$ note les décomptes résultant de B dans l'échantillon (sauf mention spécifique), où x est le nombre total de B dans la population. La probabilité que la requête atteigne le seuil requis d'au moins α votes est donnée par :

$$P(\mathcal{H}(\mathcal{N},x,k) \geq \alpha) = \sum_{j=\alpha}^{k} \binom{x}{j}\binom{n-x}{k-j} \Big/ \binom{n}{k} \tag{2}$$

Pour faciliter la notation, nous surchargeons $\mathcal{H}(*)$ en nous référant implicitement à $P(\mathcal{H}(\mathcal{N},x,k) \geq \alpha)$ par $\mathcal{H}(\mathcal{N},x,k,\alpha)$.

Bornes de queue sur la distribution hypergéométrique Nous pouvons un peu réduire la complexité de l'équation 2 en introduisant une borne sur la distribution hypergéométrique induite par $\mathcal{H}^k_{\mathcal{N},x}$. Soit $p = x/n$ la proportion du support pour B dans la population. Nous attendons exactement kp pour $\mathcal{H}(\mathcal{N},x,k)$. Donc, la probabilité que $\mathcal{H}(\mathcal{N},x,k)$ dévie de la moyenne de plus d'une petite constante ψ est donnée par la borne de queue de Hoeffding [32], comme suit,

$$\begin{aligned} P(\mathcal{H}(\mathcal{C},x,k) \leq (p-\psi)k) &\leq e^{-k\mathcal{D}(p-\psi,p)} \\ &\leq e^{-2(p-\psi)^2 k} \end{aligned} \tag{3}$$

où $\mathcal{D}(p-\psi,p)$ est la divergence de Kullback-Leibler, mesurée par

$$\mathcal{D}(a,b) = a\log\frac{a}{b} + (1-a)\log\frac{1-a}{1-b} \tag{4}$$

Concentration de sous-martingales Soit $\{X_{\{t\geq 0\}}\}$ une sous-martingale et $|X_t - X_{t-1}| < c_t$ presque sûrement. Alors, pour tout réel positif ψ et tout entier positif t,

$$P(X_t \geq X_0 + \psi) \leq e^{-\psi^2/2\Sigma_{i=1}^{t} c_i^2} \tag{5}$$

A.2 Slush

Slush opère dans une configuration non-byzantine, c'est-à-dire $f = 0$, $c = n$. Dans cette section, nous caractériserons les propriétés d'irréversibilité de Slush (qui apparaissent dans Snowflake et Snowball), ainsi que la distribution de taux de convergence précise. La distribution de la sûreté de fonctionnement et de vitalité de Slush se traduisent bien dans une configuration byzantine.

La version procédurale de Slush en figure 4 faisait usage d'un paramètre m, le nombre de tours d'un nœud exécutant des requêtes Slush. Nous voulons en définitive extraire le nombre total de tours ϕ que l'ordonnanceur doit exécuter afin de garantir que l'entièreté du réseau se retrouve de la même couleur, ceci avec une forte probabilité.

Nous analysons le système en utilisant principalement un processus à temps continu. Soit $\{X_{\{t\geq 0\}}\}$ un processus de Markov à temps continu. L'espace d'état $\mathcal{S}$ du processus stochastique est une version condensée de tout l'espace de configuration, où chaque état $\{0,\dots,n\}$ représente le nombre total de nœuds bleus dans le système.

Soit $\mathcal{F}_{X_s}$ la filtration, ou l'historique afférent au processus, jusqu'au temps s. Ce processus est Markovien et homogène dans le temps, se conformant à

$$P\{X_t = j|\mathcal{F}_{X_s}\} = P\{X_t = j|X_s\} = P\{X_t = j|X_0\}$$

Dans cet article, nous utilisons la notation $Q \equiv (q_{ij}, i, j \in \mathcal{S})$ pour nous référer au générateur infinitésimal du processus, où les taux de mort ($i \to i-1$) et de naissance ($i \to i+1$) de transitions de configuration sont notées via μ_i, et λ_i (λ_i est disctint du paramètre d'horloge λ et sera sorti du contexte). Ces taux sont

$$\begin{cases} \mu_i = i\ \mathcal{H}(\mathcal{N}, c-i, k, \alpha), & \text{for } i \to i-1 \\ \lambda_i = (c-i)\ \mathcal{H}(\mathcal{N}, i, k, \alpha), & \text{for } j \to i+1 \end{cases}$$

pour $1 \leq i \leq c-1$, et où $i = 0$ et $i = c$ sont absorbants. Soit $p_{ij}(t)$ une référence à la probabilité de transition de l'état i à j à l'instant t. Nous supposons toujours que

$$p_{ij}(t) = \begin{cases} \lambda_i t + o(t), & \text{pour } j = i+1 \\ \mu_i t + o(t), & \text{pour } j = i-1 \\ 1 - (\lambda_i + \mu_i)t + o(t), & \text{pour } j = i \\ o(t), & \text{sinon} \end{cases}$$

où tous les $o(t)$ sont uniformes en i.

Irréversibilité En section 4, nous avons traité la borne de Chtaval qui fournit une compréhension intuitive de la dynamique de forte irréversibilité au cœur de notre mécanisme de sous-échantillonnage. En particulier, une fois que le réseau tend vers une valeur majoritaire donnée, il ne peut s'inverser qu'avec une probabilité exponentiellement petite. Nous calculons l'expression de forme fermée pour la réversibilité, et nous montrons qu'elle est exponentiellement petite.

Théorème 2. *Soit ξ_δ la probabilité d'absorption dans l'état totalement rouge (s_0), commençant par un drift de δ (c'est-à-dire que δ tendent à s'éloigner de $n/2$). Alors, en supposant $\delta > 1$,*

$$\xi_\delta = 1 - \frac{\sum_{l=1}^{\delta} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \lambda_j}{2 \sum_{l=1}^{n/2} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \mu_j} \tag{6}$$

et

$$\begin{aligned} \frac{\xi_\delta - \xi_{\delta+1}}{\xi_{\delta+1} - \xi_{\delta+2}} &= \sqcap_{\delta+1} = \frac{\lambda_{\delta+1}}{\mu_{\delta+1}} \\ &\approx \frac{n-\delta-1 \sum_{j=\alpha}^{k} \frac{(n-\delta-1)^k (\delta+1)^{k-j}}{n^{2k-j}}}{\delta+1 \sum_{j=\alpha}^{k} \frac{(\delta+1)^k (n-\delta-1)^{k-j}}{n^{2k-j}}} \end{aligned} \tag{7}$$

où à partir de maintenant nous nous référons à $\sqcap_{\delta+1}$ comme étant le drift du processus.

Preuve. Nos résultats sont dérivés en se fondant sur les constructions de Tan [58]. Nous construisons une sous-matrice de Q, notée B, comme on le voit en figure 20. Soit $W_1' = (\mu_1, 0, \ldots, 0)$, $W_{n-1}' = (0, \ldots, 0, \lambda_{n-1})$. Alors nous pouvons exprimer Q par

$$Q = \begin{bmatrix} 0 & \ldots & 0 \\ W_1 & B & W_{n-1} \\ 0 & \ldots & 0 \end{bmatrix}$$

Rappelons que la distribution stationnaire peut être trouvée *via* $\lim_{t\to\infty} P(t) = e^{Qt}$, où nous avons

$$e^{Qt} = \sum_{i=0}^{\infty} \frac{t^i}{i!} Q^i = \sum_{i=0}^{\infty} \frac{t^i}{i!} \begin{bmatrix} 0 & \ldots & 0 \\ B^{i-1} W_1 & B^i & B^{i-1} W_{n-1} \\ 0 & \ldots & 0 \end{bmatrix}$$

Comme le montre Tan (eq. 2.3), nous avons

$$\xi(t) = B^{-1} \left[\sum_{i=0}^{\infty} B^i - \mathbb{I}_{n-1} \right] W_1$$

Comme nous voulons les probabilités ultimes, nous avons

$$\xi = \lim_{t\to\infty} \xi(t) = -B^{-1} W_1$$

Nous pouvons explicitement calculer ξ_δ en termes de nos taux μ_i et λ_i, obtenant ainsi

$$\xi_\delta = \frac{\sum_{l=1}^{n-\delta} \prod_{i=1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j}{\sum_{l=1}^{n} \prod_{i=1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j}$$

Cependant, notons que $u_i = \lambda_{n-i}$. La manipulation algébrique à partir de cette observation mène aux deux équations du théorème. Cette expression est strictement inférieure aux bornes de Chtaval utilisées en section 4. □

En utilisant la construction pour les probabilités d'absorption (et d'(ir)réversibilité) précédemment traitées, il est naturel de poursuivre par un calcul concernant le *temps de convergence moyen*. Soit $T_z(t) = \inf\{t \geq 0 : X_t = \{0, n\} | X_0 = z\}$, et soit $\tau_z = \mathbb{E}[T_z(t)]$. τ_z est le temps moyen pour atteindre l'un ou l'autre des états absorbants, à partir de l'état z, qui correspond au temps moyen de convergence. Le théorème suivant caractérise cette distribution.

Théorème 3. *Soit τ_z le temps attendu de convergence, à partir de l'état $z > n/2$, vers l'un des deux états convergents dans le réseau (totalement rouge ou totalement bleu). Alors,*

$$\tau_z = \frac{\sum_{d=1}^{n-1} x(d) y(d)}{2 \sum_{l=1}^{n/2} \prod_{i=1}^{l-1} \mu_i^2 \prod_{j=l}^{n-l} \mu_j} \tag{8}$$

$$B = \begin{bmatrix} -(\lambda_1+\mu_1) & \lambda_1 & 0 & \cdots & \cdots & 0 \\ \mu_2 & -(\lambda_2+\mu_2) & \lambda_2 & 0 & \cdots & 0 \\ 0 & \mu_3 & -(\lambda_3+\mu_3) & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \mu_{n-3} & -(\lambda_{n-2}+\mu_{n-2}) & \lambda_{n-3} & 0 \\ \vdots & \cdots & 0 & \mu_{n-1} & -(\lambda_{n-2}+\mu_{n-2}) & \lambda_{n-2} \\ 0 & \cdots & 0 & 0 & \mu_{n-1} & -(\lambda_{n-1}+\mu_{n-1}) \end{bmatrix}$$

Figure 20: Matrice B.

où $x(d)$ et $y(d)$ sont

$$\begin{aligned} x(d) &= \sum_{l=1}^{\min(z,d)} \prod_{i=1}^{l-1} \mu_i \prod_{j=l}^{d-1} \lambda_j \\ y(d) &= \sum_{l=1}^{n-d-\max(z-d,0)} \prod_{i=d+1}^{n-l} \mu_i \prod_{j=n-l+1}^{n-1} \lambda_j \end{aligned} \tag{9}$$

Preuve. D'après les calculs qui précèdent, $-B^{-1}$ à la rangée z fournit le nombre de traversées vers chaque autre état en partant de z. En calculant leur somme, nous avons notre résultat. L'équation ci-dessus est l'expression complète de la somme des rangées des matrices. □

Le théorème 3 mène au lemme suivant qui formalise la propriété P2, en suppposant qu'au début du protocole, une proposition possède un support d'au moins α dans le réseau.

Lemme 4. *Slush atteint presque sûrement un état absorbant dans un temps fini.*

Preuve. En partant d'un quelconque état non-absorbant et transitoire, il y a une probabilité non nulle d'absorption. De plus, comme la terminaison est finie et différentiable partout, le théorème 3 implique aussi que la probabilité de terminaison de toute configuration réseau où une proposition a un support $\geq \alpha$ dans un temps borné t_{max} est strictement positive. □

A.3 Snowflake

Dans Snowflake, l'ensemble des nœuds échantillonnés comprend des nœuds byzantins. Nous introduisons finalement une fonction appelée $\mathcal{A}(\mathcal{S}_t)$ qui est construite en faisant conserver à chaque nœud la trace du nombre total de fois consécutives qu'il a échantillonné une majorité de la couleur d'échantillon (β). Enfin, nous introduisons une fonction appelée $\mathcal{A}(\mathcal{S}_t)$, la stratégie contradictoire, qui prend comme paramètres toute la configuration du réseau à l'instant t, ainsi que l'ensemble suivant de nœuds choisi par l'ordonnanceur pour exécution, avec pour effet secondaire la modification de l'ensemble de nœuds $\mathcal{B}$ vers une configuration arbitraire de couleurs.

Afin d'appliquer notre framework antérieur à Snowflake, nous devons porter notre attention sur une subtilité essentielle. Contrairement à Slush, où il est clair qu'une fois que le réseau a atteint l'un des états convergents, il ne reviendra pas en arrière, cela ne s'applique pas à Snowflake, puisque tout adversaire $f \geq \alpha$ a une probabilité strictement positive d'inverser la tendance du système, fût-elle infinitésimale. Le processus de Markov à temps continu est assez souple pour s'accommoder d'un système où il n'existe qu'un seul état absorbant, mais le comportement du système sur le long terme n'est plus significatif puisque, après un laps de temps infini, le système doit s'inverser, violant ainsi la sécurité de fonctionnement. Nous pouvons borner trivialement l'intervalle de temps et montrer la sûreté de fonctionnement grâce à ce bornage en nous contentant de caractériser la distribution de e^{tQ}, où Q est le générateur. Nous pouvons cependant faire l'observation suivante : si la probabilité d'aller de l'état c (totalement bleu) à $c-1$ est exponentiellement petite, alors il faudra à l'attaquant un temps exponentiel (attendu ; on note que c'est une borne inférieure et, en réalité, il faudra beaucoup plus longtemps) pour réussir à inverser l'état du système. Nous pouvons donc suppposer qu'une fois que tous les nœuds légitimes ont adopté la même couleur, l'attaque de l'adversaire se termine puisqu'il serait irréaliste de la poursuivre.

En fait, dans un cadre temporel raisonnablement borné, la distance variationnelle entre l'approche exacte et l'approximation est très petite. Nous laissons les détails à l'article qui accompagne celui-ci, mais nous discutons brièvement du déroulement de l'analyse pour Snowflake.

Comme il en a été fait mention en section 4, la manière d'analyser l'adversaire en employant la même construction que dans Slush consiste à conditionner la réversibilité sur le premier nœud u se décidant pour le bleu, ce qui peut se produire à n'importe quel état (comme le spécifie $\mathcal{D}(*)$). À ce point, la stratégie contradictoire se réduit à une seule fonction, qui est de continuellement voter pour rouge. Les probabilités de réversibilité, pour tous les états $\{1,\ldots,c-1\}$ doivent coder la probabilité que des nœuds bleus s'engagent, anisi que la fonction unique de l'adversaire. Les taux de naissance et de mort sont transformés comme suit :

$$\begin{cases} \mu_i = & i(1-\mathbb{I}[\mathcal{D}(*,i,\mathbb{B})])\ \mathcal{H}(\mathcal{N},c-i+f,k,\alpha) \\ \lambda_i = & (c-i)(1-\mathbb{I}[\mathcal{D}(*,c-i,\mathbb{R})])\ \mathcal{H}(\mathcal{N},i,k,\alpha) \end{cases}$$

À partir de maintenant, l'analyse est la même que pour Slush. Pour divers k et β, nous pouvons trouver le α minimal qui fournit au système des propriétés de forte irréversibilité.

Le lemme suivant formalise P3 et la preuve procède du théorème central limite.

Lemme 5. *Si $f < O(\sqrt{n})$, et $\alpha = \lfloor k/2 \rfloor + 1$, alors Snowflake se termine en $O(\log n)$ tours avec une forte probabilité.*

Preuve. Le résultat procède du théorème central limite, où pour $\alpha = \lfloor k/2 \rfloor + 1$, la polarisation attendue dans le réseau après échantillonnage sera $O(\sqrt{n})$. Un adversaire plus petit que ce résultat sera incapable de conserver le réseau dans un état totalement bivalent pendant plus qu'un nombre constant de tours. Le facteur logarithmique provient de la limite inférieure du temps de mélange. □

A.4 Snowball

Nous faisons l'observation suivante : si la confiance entre rouge et bleu est équivalente, alors l'adversaire a la même influence sur l'irréversibilité du système que dans Snowflake, quelle que soit la configuration du réseau. En fait, Snowflake peut être vu comme Snowball où les déviations des valeurs de confiance n'excèdent jamais un. La même analyse s'applique à Snowball comme à Snowflake, avec la contrainte supplémentaire de bornage des comportements des confiances à long terme sur le réseau. L'analyse suit à cette fin en employant des inégalités de concentration de martingales, notamment celle qui a été introduite en équation 5. Snowball peut être vu comme un système à deux urnes où chaque urne est une sous-martingale. Les garanties qui peuvent en être extraites ci-après sont que les confiances en la valeur engagée par la majorité (toujours bleu dans notre cadre de référence) croissent toujours plus que celles de la valeur minoritaire, avec une forte probabilité, tendant à s'écarter alors que $t \to t_{max}$.

A.5 Engagement préalable sûr

Comme nous l'avons précédemment déduit, chaque ensemble de conflits dans Avalanche peut être vu comme une instance de Snowball où chaque instance de la progéniture vote itérativement pour tout le chemin de l'ascendance. Cette caractéristique offre plusieurs avantages ; cependant, elle peut aussi mener à ce que des transactions vertueuses dépendantes d'une transaction malveillante subissent le même sort que celle-ci. En particulier, les transactions malveillantes peuvent s'insérer entre des transactions vertueuses et réduire la possibilité que celles-ci atteignent le prédicat isAccepted nécessaire. En guise d'expérience mentale, supposons qu'une transaction T_i nomme un ensemble de transactions parentes qui sont toutes décidées, selon leur point de vue local. Si T_i est échantillonnée sur un ensemble suffisamment grand de requêtes réussies sans découvrir aucun conflit, alors, puisque par supposition toute l'ascendance de T_i est décidée, il s'ensuit (de manière probabiliste) que nous sommes arrivés à l'irréversibilité.

Pour ensuite mesurer statistiquement l'assurance que T_i a été acceptée par un grand pourcentage de nœuds corrects sans aucun conflit, nous nous servons d'un processus de naissance à sens unique, où une naissance se produit quand une nouveau nœud correct découvre le conflit de T_i. Les morts ne peuvent pas exister dans ce modèle, car une transaction conflictuelle ne peut pas être escamotée une fois qu'un nœud correct la découvre. Nos naissances se font comme suit :

$$\lambda_i = \frac{c-i}{c}\left(1 - \frac{\binom{n-i}{k}}{\binom{n}{k}}\right) \tag{10}$$

La résolution du temps attendu pour atteindre l'état final de naissance fournit une borne inférieure au paramètre β_1 dans la branche de décision rapide isAccepted. La table ci-dessous montre un exemple d'analyse pour $n = 2000$, $\alpha = 0.8k$ et divers k, où $\varepsilon \ll 10^{-9}$, et où β est la valeur minimale requise avant la décision.

k	10	20	30	40
β	10.87625	10.50125	10.37625	10.25125

Globalement, un très petit nombre d'itérations est suffisant pour le prédicat d'engagement préalable sûr. Cela confirme le choix de β dans notre évaluation.

A.6 Heuristique d'initialisation

Pour améliorer la vitalité, nous pouvons utiliser des hypothèses de forte synchronie. L'heuristique fonctionne comme suit. Chaque nœud opère en deux phases : dans la première, elle diffuse et collecte des propositions pendant $O(\log n)$ tours, où chaque tour dure le temps maximum d'un message, ce qui garantit que la proposition d'un nœud correct se propage vers presque tous les autres nœuds corrects ; dans la seconde phase, chaque nœud arrête de collecter les propositions dont il a localement la connaissance, en recherchant ceux qui ont une majorité α, ordonnée de manière déterministe, comme par la valeur de leur empreinte cryptographique. Il sélectionne ensuite la première valeur dans l'ordre pour en faire son état initial quand il commence vraiment le protocole de consensus. Dans le cas d'une cryptomonnaie, la fonction de tri déterministe incorporerait les frais payés pour toute nouvelle proposition, ce qui signifie que l'adversaire serait financièrement limité pour lancer une attaque d'impartialité contre l'initialisation.

A.7 Roulement et incohérence de vues

Les systèmes réalistes doivent s'accommoder du départ et de l'arrivée des nœuds. Jusqu'à présent, nous avons simplifié notre analyse en supposant une connaissance précise des membres du réseau, c'est-à-dire $\mathcal{L}(u) = \mathcal{N}$. Nous allons maintenant démontrer que les nœuds corrects peuvent admettre un roulement caractérisé d'entrants et de sortants en montrant comment choisir des paramètres tels que les nœuds Avalanche puissent différer dans leurs vues respectives du réseau tout

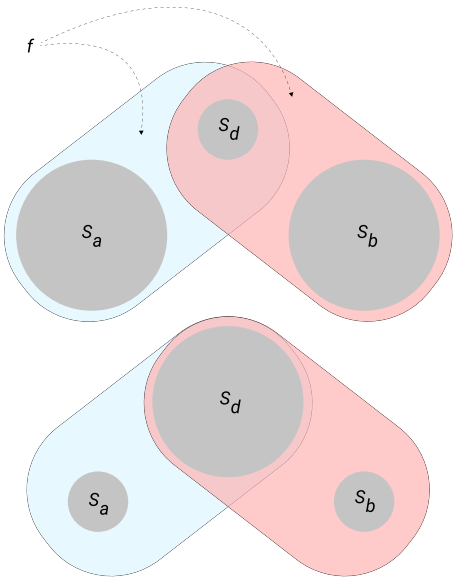

Figure 21: Changements de la vue du réseau en se basant sur la taille de S_d. Toutes les preuves antérieures jusqu'à présent représentent la variante où $S_a = S_b = \varnothing$. Avec la nouvelle construction, la probabilité de violation de la sécurité de fonctionnement n'est qu'une application directe des ensembles antérieurs de preuves pour les nouveaux sous-ensembles.

en maintenant leur capacité à prendre des décisions en toute sécurité.

Pour caractériser le roulement, nous utilisons une construction généralisée d'intersection d'ensembles qui nous permet d'établir des arguments concernant les éclatements de vues du réseau. Avant de le formaliser, nous partons de cette intuition : supposons que nous éclations le réseau en deux sous-ensembles entièrement indépendants mais parfaitement connectés. Il est clair que l'adversaire byzantin gagne avec une probabilité de un puisqu'il peut envoyer deux transactions conflictuelles à chaque réseau indépendant, et chacun finaliserait immédiatement les transactions. Cela représente le cas d'éclatement le plus défavorable. Nous pouvons généraliser ceci à d'autres éclatements du réseau, et même récursivement dans chaque sous-ensemble. Les preuves de sûreté de fonctionnement sont alors une question de caractérisation de la probabilité que les rouges et les bleus s'engagent dans les deux sous-ensembles (éventuellement indépendants).

Supposons que nous divisions l'ensemble de nœuds corrects en trois sous-ensembles, S_a, S_d, S_b. Nous surchargeons $\mathcal{L}(S_*)$ pour représenter les vues de tout nœud dans l'ensemble d'entrées. La vue de tous les nœuds dans S_a est $\mathcal{L}(S_a) = S_a \cup S_d \cup \mathcal{B}$, celle de tous les nœuds dans S_b est $\mathcal{L}(S_b) = S_b \cup S_d \cup \mathcal{B}$, et la vue de S_d est $\mathcal{N}$. Nous supposons le cas le plus défavorable, ce qui signifie que les nœuds de l'adversaire sont communs à tous les sous-ensembles. Quand $S_d = \varnothing$, cela représente une division du réseau en deux sous-ensembles égaux où $|S_a| = |S_b| = n/2$ [2]. Si S_d est uniquement composé de nœuds corrects, alors $|S_a| = |S_b| = n$. Cette construction est démontrée visuellement en figure 21.

Lemme 6. *Soit $\tau \in \mathbb{Z}^+$. Soit $|S_d| = n - \tau$, et donc $|S_{\{a,b\}}| = \tau/2$. Il existe un taille maximale de τ telle que la probabilité pour que deux nœuds quelconques $u, v \in S_a,\ S_b,\ S_d$ finalisant des transactions équivoques soit inférieure à ε.*

Preuve. Nous supposons que l'adversaire a le contrôle total des éclatements de la vue du réseau, ce qui signifie qu'il peut choisir sans contrainte de créer $S_a,\ S_b,\ S_d$. Pour prouver la sécurité de fonctionnement, il nous suffit de réutiliser exactement la même construction qu'en section A.3, mais nous remplaçons l'ensemble originel $\mathcal{N}$ avec le nouvel ensemble qui nous intéresse, nommément $S_a \cup S_d \cup \mathcal{B}$ (c'est-à-dire que nous excluons S_b) [3]. Donc, pour trouver le τ maximal, nous nous contentons de remplacer u_i et λ_i par

$$\begin{cases} \bar{\mu}_i = i\ \mathcal{H}(S_a \cup S_d \cup \mathcal{B}, c - (\tau/2) - i, k, \alpha), & \text{for } i \to i-1 \\ \lambda_i = (c-i)\ \mathcal{H}(S_a \cup S_d \cup \mathcal{B}, i, k, \alpha), & \text{for } j \to i+1 \end{cases} \tag{11}$$

où

$$P(\mathcal{H}(S_a \cup S_d \cup \mathcal{B}, x, k) \geq \alpha) = \sum_{j=\alpha}^{k} \binom{x}{j} \binom{c-(\tau/2)-x}{k-j} \Big/ \binom{c-(\tau/2)}{k} \tag{12}$$

et nous appliquons la même construction qu'en section A.3. Alors que τ augmente, il est clair que S_a (et, réciproquement, S_b) deviennent plus indépendants et moins dépendents des valeurs proposées par les membres de S_d, augmentant ainsi les capacités de l'adversaire. □

[2] $|S_a|$ et $|S_b|$ ne sont pas nécessairement égaux, nous posons ceci pour la démonstration.

[3] Les ensembles sont symétriques dans cet exemple.