Le réseau Bitcoin

2022-03-08

# Le réseau Bitcoin

## Architecture réseau pair à pair

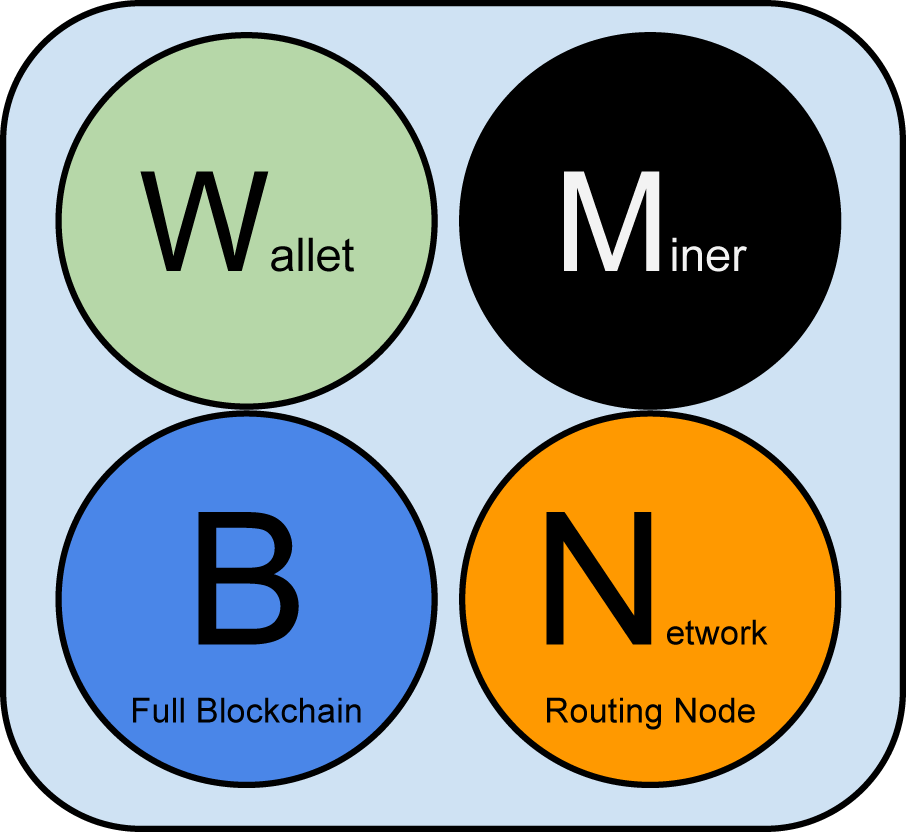
Bitcoin est structuré comme une architecture de réseau peer-to-peer au-dessus de l'Internet. Le terme peer-to-peer, ou P2P, signifie que les ordinateurs qui participent au réseau sont des pairs les uns des autres, qu'ils sont tous égaux, qu'il n'y a pas de nœuds "spéciaux" et que tous les nœuds partagent la charge de fournir services réseau. Les nœuds du réseau s'interconnectent dans un réseau maillé avec une topologie "plate". Il n'y a pas de serveur, pas de service centralisé et pas de hiérarchie au sein du réseau. Les nœuds d'un réseau P2P fournissent et consomment des services en même temps, la réciprocité agissant comme une incitation à la participation. Les réseaux P2P sont intrinsèquement résilients, décentralisés et ouverts. Un exemple prééminent d'une architecture de réseau P2P était le début d'Internet lui-même, où les nœuds du réseau IP étaient égaux. L'architecture Internet d'aujourd'hui est plus hiérarchique, mais le protocole Internet conserve toujours son essence de topologie plate. Au-delà du bitcoin, l'application la plus importante et la plus réussie des technologies P2P est le partage de fichiers, avec Napster comme pionnier et BitTorrent comme l'évolution la plus récente de l'architecture.

L'architecture du réseau P2P de Bitcoin est bien plus qu'un choix de topologie. Bitcoin est un système de paiement numérique P2P de par sa conception, et l'architecture du réseau est à la fois le reflet et le fondement de cette caractéristique essentielle. La décentralisation du contrôle est un principe de conception de base qui ne peut être atteint et maintenu que par un réseau consensuel P2P plat et décentralisé.

Le terme "réseau Bitcoin" fait référence à l'ensemble de nœuds exécutant le protocole bitcoin P2P. En plus du protocole bitcoin P2P, il existe d'autres protocoles tels que Stratum qui sont utilisés pour le minage et les portefeuilles légers ou mobiles. Ces protocoles supplémentaires sont fournis par des serveurs de routage de passerelle qui accèdent au réseau Bitcoin à l'aide du protocole bitcoin P2P, puis étendent ce réseau aux nœuds exécutant d'autres protocoles. Par exemple, les serveurs Stratum connectent les nœuds de minage Stratum via le protocole Stratum au réseau Bitcoin principal et relient le protocole Stratum au protocole bitcoin P2P. Nous utilisons le terme « réseau Bitcoin étendu » pour désigner le réseau global qui comprend le protocole Bitcoin P2P, les protocoles de pool-mining, le protocole Stratum et tout autre protocole connexe reliant les composants du système Bitcoin.

## Types de nœuds et rôles

Bien que les nœuds dans le bitcoin Les réseaux P2P sont égaux, ils peuvent jouer différents rôles en fonction de la fonctionnalité qu'ils prennent en charge. Un nœud Bitcoin est un ensemble de fonctions : le routage, la base de données blockchain, le minage et les services de portefeuille. Un nœud complet avec ces quatre fonctions est affiché dans <<full\_node\_reference> >.



Un nœud de réseau Bitcoin avec les quatre fonctions : portefeuille, mineur, base de données blockchain complète et routage réseau

Tous les nœuds incluent la fonction de routage pour participer au réseau et peuvent inclure d'autres fonctionnalités. Tous les nœuds valident et propagent les transactions et les blocs, et découvrent et maintiennent les connexions aux pairs. Dans l'exemple de nœud complet dans <<full\_node\_reference> >, la fonction de routage est indiquée par un cercle nommé "Network Routing Node" ou par la lettre "N".

Certains nœuds, appelés nœuds complets, conservent également une copie complète et à jour de la blockchain. Les nœuds complets peuvent vérifier de manière autonome et autoritaire toute transaction sans référence externe. Certains nœuds ne conservent qu'un sous-ensemble de la blockchain et vérifient les transactions à l'aide d'une méthode appelée *simplified-payment-verification*, ou SPV. Ces nœuds sont appelés nœuds SPV ou nœuds légers. Dans l'exemple de nœud complet de la figure, la fonction de base de données de blockchain de nœud complet est indiquée par un cercle appelé "Full Blockchain" ou la lettre "B". Dans <<bitcoin\_network> >, les nœuds SPV sont dessinés sans le cercle "B", montrant qu'ils n'ont pas une copie complète de la blockchain.

Les nœuds de minage se font concurrence pour créer de nouveaux blocs en exécutant du matériel spécialisé pour résoudre l'algorithme de preuve de travail. Certains nœuds de minage sont également des nœuds complets, conservant une copie complète de la blockchain, tandis que d'autres sont des nœuds légers participant au minage de pool et dépendant d'un serveur de pool pour maintenir un nœud complet. La fonction d'exploration de données est affichée dans le nœud complet sous la forme d'un cercle appelé "Miner" ou de la lettre "M".

Les portefeuilles d'utilisateurs peuvent faire partie d'un nœud complet, comme c'est généralement le cas avec les clients Bitcoin de bureau. De plus en plus, de nombreux portefeuilles d'utilisateurs, en particulier ceux fonctionnant sur des appareils à ressources limitées tels que les smartphones, sont des nœuds SPV. La fonction de portefeuille est affichée dans <<full\_node\_reference> > comme un cercle appelé "Wallet" ou la lettre "W."

En plus des principaux types de nœuds sur le protocole bitcoin P2P, il existe des serveurs et des nœuds exécutant d'autres protocoles, tels que des protocoles de pool de minage spécialisés et des protocoles d'accès client légers.

<<node\_type\_ledgend> > montre les types de nœuds les plus courants sur le réseau Bitcoin étendu.

## Le réseau Bitcoin étendu

Le principal réseau Bitcoin, en cours d'exécution le protocole bitcoin P2P, se compose de 5 000 à 8 000 nœuds d'écoute exécutant différentes versions du client de référence bitcoin (Bitcoin Core) et de quelques centaines de nœuds exécutant diverses autres implémentations du protocole bitcoin P2P, telles que Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin, btcd et bcoin. Un petit pourcentage des nœuds du réseau bitcoin P2P sont également des nœuds de minage, en concurrence dans le processus de minage, en validant les transactions et en créant de nouveaux blocs. Diverses grandes entreprises s'interfacent avec le réseau Bitcoin en exécutant des clients à nœud complet basés sur le client Bitcoin Core, avec des copies complètes de la blockchain et un nœud de réseau, mais sans fonctions de minage ou de portefeuille. Ces nœuds agissent comme des routeurs de périphérie de réseau, permettant à divers autres services (échanges, portefeuilles, explorateurs de blocs, traitement des paiements marchands) d'être construits par-dessus.

Le réseau Bitcoin étendu comprend le réseau exécutant le protocole bitcoin P2P, décrit précédemment, ainsi que des nœuds exécutant des protocoles spécialisés. Un certain nombre de serveurs de pool et de passerelles de protocole qui connectent des nœuds exécutant d'autres protocoles sont attachés au réseau principal bitcoin P2P. Ces autres nœuds de protocole sont principalement des nœuds de pool mining (voir <<mining> >) et les clients de portefeuille légers, qui ne portent pas une copie complète de la blockchain.

<<bitcoin\_network> > montre le réseau Bitcoin étendu avec les différents types de nœuds, serveurs de passerelle, routeurs de périphérie et clients de portefeuille et les différents protocoles qu'ils utilisent pour se connecter les uns aux autres.

image

images/mbc2\_0802.png["BitcoinNodeTypes"]

image

images/mbc2\_0803.png["BitcoinNetwork"]

## Réseaux relais Bitcoin

Alors que le réseau bitcoin P2P répond aux besoins généraux d'une grande variété de types de nœuds, il présente une latence de réseau trop élevée pour les besoins spécialisés des nœuds de minage de bitcoins.

Les mineurs de Bitcoin sont engagés dans une compétition urgente pour résoudre le problème de la preuve de travail et étendre la blockchain (voir <<mining> >). Tout en participant à cette compétition, les mineurs de bitcoins doivent minimiser le temps entre la propagation d'un bloc gagnant et le début du prochain tour de compétition. Dans le secteur minier, la latence du réseau est directement liée aux marges bénéficiaires.

Un *Bitcoin Relay Network* est un réseau qui tente de minimiser la latence dans la transmission des blocs entre les mineurs. Le <https://www.bitcoinrelaynetwork.org> [Bitcoin Relay Network] original a été créé par le développeur principal Matt Corallo en 2015 pour permettre une synchronisation rapide des blocs entre les mineurs avec une latence très faible. Le réseau se composait de plusieurs nœuds spécialisés hébergés sur l'infrastructure Amazon Web Services à travers le monde et servait à connecter la majorité des mineurs et des pools de minage.

Le réseau de relais Bitcoin d'origine a été remplacé en 2016 par l'introduction du *Fast Internet Bitcoin Relay Engine* ou https:/ /bitcoinfibre.org[*FIBRE*], également créé par le développeur principal Matt Corallo. FIBRE est un réseau de relais basé sur UDP qui relaie les blocs au sein d'un réseau de nœuds. FIBRE met en œuvre l'optimisation *bloc compact* pour réduire davantage la quantité de données transmises et la latence du réseau.

Les réseaux de relais ne remplacent pas le réseau P2P de Bitcoin. Au lieu de cela, ce sont des réseaux superposés qui fournissent une connectivité supplémentaire entre les nœuds ayant des besoins spécialisés. Comme les autoroutes ne remplacent pas les routes rurales, mais plutôt des raccourcis entre deux points à fort trafic, vous avez toujours besoin de petites routes pour vous connecter aux autoroutes.

## Découverte du réseau

Lorsqu'un nouveau nœud démarre, il doit découvrir d'autres nœuds Bitcoin sur le réseau afin de participer. Pour démarrer ce processus, un nouveau nœud doit découvrir au moins un nœud existant sur le réseau et s'y connecter. L'emplacement géographique des autres nœuds n'est pas pertinent ; la topologie du réseau Bitcoin n'est pas définie géographiquement. Par conséquent, tous les nœuds Bitcoin existants peuvent être sélectionnés au hasard.

Pour se connecter à un pair connu, les nœuds établissent une connexion TCP, généralement au port 8333 (le port généralement connu comme celui utilisé par bitcoin), ou un autre port s'il en existe un. Lors de l'établissement d'une connexion, le nœud commencera une "prise de contact" (voir <<network\_handshake> >) en transmettant un message version, qui contient des informations d'identification de base, notamment :

nVersion

La version du protocole bitcoin P2P que le client "parle" (par exemple, 70002)

nLocalServices

Une liste des services locaux pris en charge par le nœud, actuellement uniquement NODE\_NETWORK

nTime

L'heure actuelle

addrYou

L'adresse IP du nœud distant vue depuis ce nœud

addrMe

L'adresse IP du nœud local, telle que découverte par le nœud local

subver

Une sous-version indiquant le type de logiciel exécuté sur ce nœud (par exemple, /Satoshi:0.9.2.1/ )

BestHeight

La hauteur de bloc de la blockchain de ce nœud

(Voir [GitHub](https://bit.ly/1qlsC7w) pour un exemple du message réseau version.)

Le message version est toujours le premier message envoyé par un pair à un autre pair. L'homologue local recevant un message version examinera la nVersion signalée par l'homologue distant et décidera si l'homologue distant est compatible. Si le pair distant est compatible, le pair local accusera réception du message version et établira une connexion en envoyant un message verack.

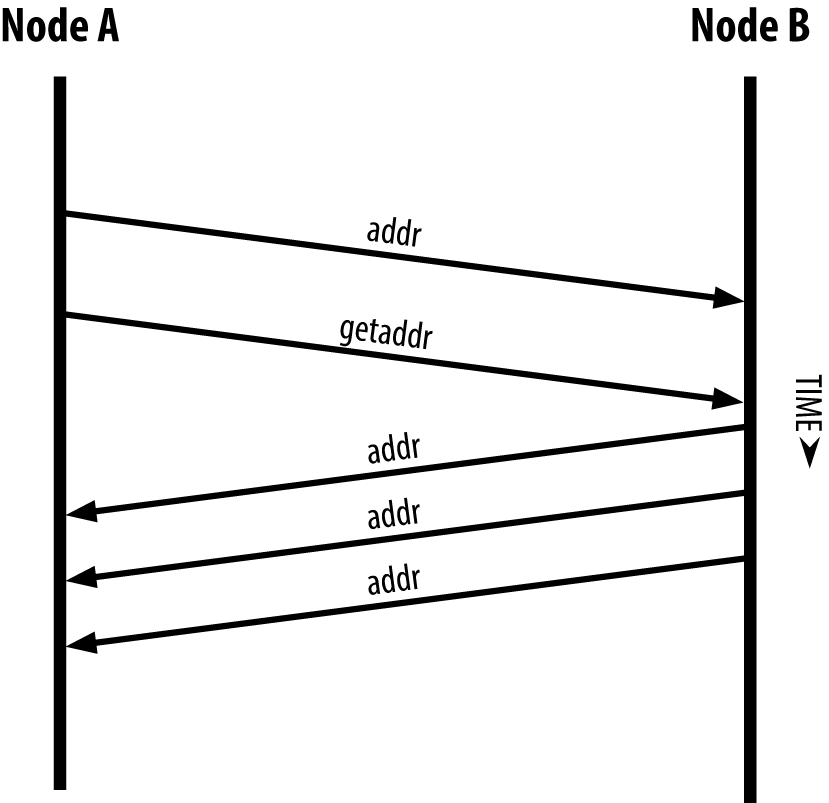
Comment un nouveau nœud trouve-t-il des pairs ? La première méthode consiste à interroger DNS à l'aide d'un certain nombre de "graines DNS", qui sont des serveurs DNS fournissant une liste d'adresses IP de nœuds Bitcoin. Certaines de ces graines DNS fournissent une liste statique d'adresses IP de nœuds d'écoute bitcoin stables. Certaines des graines DNS sont des implémentations personnalisées de BIND (Berkeley Internet Name Daemon) qui renvoient un sous-ensemble aléatoire à partir d'une liste d'adresses de nœuds Bitcoin collectées par un robot ou un nœud Bitcoin de longue durée. Le client Bitcoin Core contient les noms de neuf graines DNS différentes. La diversité de propriété et la diversité de mise en œuvre des différentes graines DNS offrent un haut niveau de fiabilité pour le processus d'amorçage initial. Dans le client Bitcoin Core, l'option d'utilisation des graines DNS est contrôlée par le commutateur d'option -dnsseed (réglé sur 1 par défaut, pour utiliser la graine DNS).

Alternativement, un nœud d'amorçage qui ne sait rien du réseau doit recevoir l'adresse IP d'au moins un nœud Bitcoin, après quoi il peut établir des connexions via d'autres introductions. L'argument de ligne de commande -seednode peut être utilisé pour se connecter à un nœud uniquement pour les introductions en l'utilisant comme graine. Une fois que le nœud de départ initial a été utilisé pour former des présentations, le client s'en déconnecte et utilise les pairs nouvellement découverts.

**La poignée de main initiale entre pairs**

image ::images/mbc2\_0804.png["Prise de contact réseau"]

Une fois qu'une ou plusieurs connexions sont établies, le nouveau nœud enverra un message addr contenant sa propre adresse IP à ses voisins. Les voisins transmettront à leur tour le message addr à leurs voisins, garantissant que le nœud nouvellement connecté devient bien connu et mieux connecté. De plus, le nœud nouvellement connecté peut envoyer getaddr aux voisins, leur demandant de renvoyer une liste d'adresses IP d'autres pairs. De cette façon, un nœud peut trouver des pairs auxquels se connecter et annoncer son existence sur le réseau pour que d'autres nœuds le trouvent. <<address\_propagation> > montre le protocole de découverte d'adresse.



Propagation et découverte d'adresses

Un nœud doit se connecter à quelques pairs différents afin d'établir divers chemins dans le réseau Bitcoin. Les chemins ne sont pas persistants - les nœuds vont et viennent - et le nœud doit donc continuer à découvrir de nouveaux nœuds à mesure qu'il perd d'anciennes connexions et à aider les autres nœuds lors de leur démarrage. Une seule connexion est nécessaire pour démarrer, car le premier nœud peut proposer des introductions à ses nœuds pairs et ces pairs peuvent proposer d'autres introductions. Il est également inutile et gaspilleur de ressources réseau de se connecter à plus d'une poignée de nœuds. Après le démarrage, un nœud se souviendra de ses dernières connexions homologues réussies, de sorte que s'il est redémarré, il peut rapidement rétablir les connexions avec son ancien réseau homologue. Si aucun des anciens pairs ne répond à sa demande de connexion, le nœud peut utiliser les nœuds de départ pour redémarrer.

Sur un nœud exécutant le client Bitcoin Core, vous pouvez lister les connexions homologues avec la commande getpeerinfo :

$ bitcoin-cli getpeerinfo

[  
 {  
 &quot;adresse&quot; : &quot;85.213.199.39:8333&quot;,  
 &quot;prestation&quot; : &quot;00000001&quot;,  
 &quot;dernier envoi&quot; : 1405634126,  
 &quot;lastrecv&quot; : 1405634127,  
 &quot;bytessent&quot; : 23487651,  
 &quot;bytesrecv&quot; : 138679099,  
 &quot;conntime&quot; : 1405021768,  
 &quot;pingtime&quot; : 0.00000000,  
 &quot;version&quot; : 70002,  
 &quot;subver&quot; : &quot;/Satoshi:0.9.2.1/&quot;,  
 &quot;entrant&quot; : faux,  
 &quot;hauteur de départ&quot; : 310131,  
 &quot;banscore&quot; : 0,  
 &quot;syncnode&quot; : vrai  
 },  
 {  
 &quot;adresse&quot; : &quot;58.23.244.20:8333&quot;,  
 &quot;prestation&quot; : &quot;00000001&quot;,  
 &quot;dernier envoi&quot; : 1405634127,  
 &quot;lastrecv&quot; : 1405634124,  
 &quot;bytessent&quot; : 4460918,  
 &quot;bytesrecv&quot; : 8903575,  
 &quot;conntime&quot; : 1405559628,  
 &quot;pingtime&quot; : 0.00000000,  
 &quot;version&quot; : 70001,  
 &quot;subver&quot; : &quot;/Satoshi:0.8.6/&quot;,  
 &quot;entrant&quot; : faux,  
 &quot;hauteur de départ&quot; : 311074,  
 &quot;banscore&quot; : 0,  
 &quot;syncnode&quot; : faux  
 }  
]

Pour outrepasser la gestion automatique des pairs et spécifier une liste d'adresses IP, les utilisateurs peuvent fournir l'option +-connect=<IPAddress> + et spécifiez une ou plusieurs adresses IP. Si cette option est utilisée, le nœud se connectera uniquement aux adresses IP sélectionnées, au lieu de découvrir et de maintenir automatiquement les connexions homologues.

S'il n'y a pas de trafic sur une connexion, les nœuds enverront périodiquement un message pour maintenir la connexion. Si un nœud n'a pas communiqué sur une connexion pendant plus de 90 minutes, il est supposé être déconnecté et un nouveau pair sera recherché. Ainsi, le réseau s'adapte dynamiquement aux nœuds transitoires et aux problèmes de réseau, et peut croître et rétrécir de manière organique selon les besoins sans aucun contrôle central.

## Nœuds complets

Les nœuds complets sont des nœuds qui maintenir une blockchain complète avec toutes les transactions. Plus précisément, ils devraient probablement être appelés "nœuds de chaîne de blocs complets". Dans les premières années de bitcoin, tous les nœuds étaient des nœuds complets et actuellement, le client Bitcoin Core est un nœud de blockchain complet. Au cours des deux dernières années, cependant, de nouvelles formes de clients Bitcoin ont été introduites qui ne maintiennent pas une blockchain complète mais fonctionnent comme des clients légers. Nous les examinerons plus en détail dans la section suivante.

Les nœuds de blockchain complets maintiennent un copie à jour de la blockchain Bitcoin avec toutes les transactions, qu'ils construisent et vérifient indépendamment, en commençant par le tout premier bloc (bloc de genèse) et en remontant jusqu'au dernier bloc connu du réseau. Un nœud de chaîne de blocs complet peut vérifier de manière indépendante et autoritaire toute transaction sans recours ni recours à aucun autre nœud ou source d'informations. Le nœud de la blockchain complète s'appuie sur le réseau pour recevoir des mises à jour sur les nouveaux blocs de transactions, qu'il vérifie ensuite et intègre dans sa copie locale de la blockchain.

L'exécution d'un nœud blockchain complet vous offre l'expérience bitcoin pure : une vérification indépendante de toutes les transactions sans avoir besoin de s'appuyer sur, ou de faire confiance, à d'autres systèmes. Il est facile de savoir si vous exécutez un nœud complet car il nécessite plus de cent gigaoctets de stockage persistant (espace disque) pour stocker la blockchain complète. Si vous avez besoin de beaucoup de disque et que la synchronisation avec le réseau prend deux à trois jours, vous exécutez un nœud complet. C'est le prix d'une indépendance totale et de la liberté vis-à-vis de l'autorité centrale.

Il existe quelques implémentations alternatives de clients Bitcoin à chaîne complète, construites à l'aide de différents langages de programmation et architectures logicielles. Cependant, l'implémentation la plus courante est le client de référence Bitcoin Core, également connu sous le nom de client Satoshi. Plus de 75 % des nœuds du réseau Bitcoin exécutent différentes versions de Bitcoin Core. Il est identifié comme « Satoshi » dans la chaîne de sous-version envoyée dans le message version et affiché par la commande getpeerinfo comme nous l'avons vu précédemment ; par exemple, /Satoshi:0.8.6/.

## Échange "Inventaire"

La première chose qu'un nœud complet fera une fois qu'il se connecte à des pairs est d'essayer de construire une blockchain complète. S'il s'agit d'un tout nouveau nœud et qu'il n'a aucune chaîne de blocs, il ne connaît qu'un seul bloc, le bloc de genèse, qui est intégré de manière statique dans le logiciel client. En commençant par le bloc #0 (le bloc de genèse), le nouveau nœud devra télécharger des centaines de milliers de blocs pour se synchroniser avec le réseau et rétablir la blockchain complète.

Le processus de synchronisation de la blockchain commence par le message version, car il contient BestHeight, la blockchain actuelle d'un nœud hauteur (nombre de blocs). Un nœud verra les messages + version + de ses pairs, saura combien de blocs ils ont chacun et pourra comparer avec le nombre de blocs qu'il a dans sa propre blockchain. Les nœuds appairés échangeront un message getblocks contenant le hachage (empreinte digitale) du bloc supérieur sur leur blockchain locale. L'un des pairs pourra identifier le hash reçu comme appartenant à un bloc qui n'est pas au sommet, mais appartient plutôt à un bloc plus ancien, en déduisant ainsi que sa propre blockchain locale est plus longue que celle de son pair.

Le pair qui a la blockchain la plus longue a plus de blocs que l'autre nœud et peut identifier les blocs dont l'autre nœud a besoin pour "rattraper son retard". Il identifiera les 500 premiers blocs à partager et transmettra leurs hachages à l'aide d'un message inv (inventaire). Le nœud manquant ces blocs les récupérera ensuite, en émettant une série de messages getdata demandant les données complètes du bloc et en identifiant les blocs demandés à l'aide des hachages du message inv.

Supposons, par exemple, qu'un nœud n'ait que le bloc genesis. Il recevra alors un message inv de ses pairs contenant les hachages des 500 prochains blocs de la chaîne. Il commencera à demander des blocs à tous ses pairs connectés, répartissant la charge et s'assurant qu'il ne submerge aucun pair de demandes. Le nœud garde une trace du nombre de blocs "en transit" par connexion homologue, c'est-à-dire des blocs qu'il a demandés mais non reçus, en vérifiant qu'il ne dépasse pas une limite (MAX\_BLOCKS\_IN\_TRANSIT\_PER\_PEER). De cette façon, s'il a besoin de beaucoup de blocs, il n'en demandera de nouveaux que lorsque les demandes précédentes seront satisfaites, permettant aux pairs de contrôler le rythme des mises à jour et de ne pas submerger le réseau. Au fur et à mesure que chaque bloc est reçu, il est ajouté à la blockchain, comme nous le verrons dans <<blockchain> >. Au fur et à mesure que la blockchain locale se construit, davantage de blocs sont demandés et reçus, et le processus se poursuit jusqu'à ce que le nœud rattrape le reste du réseau.

Ce processus de comparaison de la blockchain locale avec les pairs et de récupération des blocs manquants se produit chaque fois qu'un nœud se déconnecte pendant une période donnée. Qu'un nœud soit hors ligne depuis quelques minutes et qu'il manque quelques blocs, ou un mois et qu'il manque quelques milliers de blocs, il commence par envoyer getblocks, reçoit une réponse inv et commence à télécharger les blocs manquants. <<inventory\_synchronization> > affiche l'inventaire et le protocole de propagation des blocs.

**Node synchronisant la blockchain en récupérant les blocs d'un pair**

image ::images/mbc2\_0806.png["Synchronisation de l'inventaire"]

## Nœuds de vérification simplifiée des paiements (SPV)

Tous les nœuds n'ont pas la capacité de stocker la blockchain complète. De nombreux clients Bitcoin sont conçus pour fonctionner sur des appareils à espace et puissance limités, tels que les smartphones, les tablettes ou les systèmes embarqués. Pour ces appareils, une méthode de *vérification de paiement simplifiée* (SPV) est utilisée pour leur permettre de fonctionner sans stocker la blockchain complète. Ces types de clients sont appelés clients SPV ou clients légers. À mesure que l'adoption du bitcoin augmente, le nœud SPV devient la forme la plus courante de nœud Bitcoin, en particulier pour les portefeuilles bitcoin.

Les nœuds SPV téléchargent uniquement les en-têtes de bloc et ne téléchargent pas les transactions incluses dans chaque bloc. La chaîne de blocs qui en résulte, sans transactions, est 1 000 fois plus petite que la blockchain complète. Les nœuds SPV ne peuvent pas construire une image complète de tous les UTXO disponibles pour les dépenses car ils ne connaissent pas toutes les transactions sur le réseau. Les nœuds SPV vérifient les transactions en utilisant une méthode légèrement différente qui s'appuie sur des pairs pour fournir des vues partielles des parties pertinentes de la blockchain à la demande.

Par analogie, un nœud complet est comme un touriste dans une ville étrange, équipé d'une carte détaillée de chaque rue et de chaque adresse. En comparaison, un nœud SPV est comme un touriste dans une ville étrange demandant à des inconnus au hasard des indications détaillées tout en ne connaissant qu'une seule avenue principale. Bien que les deux touristes puissent vérifier l'existence d'une rue en la visitant, le touriste sans carte ne sait pas ce qui se trouve dans les rues latérales et ne sait pas quelles autres rues existent. Positionné en face du 23 Church Street, le touriste sans carte ne peut pas savoir s'il existe une dizaine d'autres adresses « 23 Church Street » dans la ville et si celle-ci est la bonne. La meilleure chance pour le touriste sans carte est de demander à suffisamment de gens et d'espérer que certains d'entre eux n'essaient pas de l'agresser.

SPV vérifie les transactions en se référant à leur *profondeur* dans la blockchain au lieu de leur *hauteur*. Alors qu'un nœud de blockchain complet construira une chaîne entièrement vérifiée de milliers de blocs et de transactions descendant dans la blockchain (remontant dans le temps) jusqu'au bloc de genèse, un nœud SPV vérifiera la chaîne de tous les blocs (mais pas toutes les transactions) et lier cette chaîne à la transaction d'intérêt.

Par exemple, lors de l'examen d'une transaction dans le bloc 300 000, un nœud complet relie les 300 000 blocs au bloc de genèse et crée une base de données complète d'UTXO, établissant la validité de la transaction en confirmant que l'UTXO reste non dépensé. Un nœud SPV ne peut pas valider si l'UTXO n'est pas dépensé. Au lieu de cela, le nœud SPV établira un lien entre la transaction et le bloc qui la contient, en utilisant un *merkle path* (voir <<merkle\_trees> >). Ensuite, le nœud SPV attend de voir les six blocs 300 001 à 300 006 empilés au-dessus du bloc contenant la transaction et le vérifie en établissant sa profondeur sous les blocs 300 006 à 300 001. Le fait que d'autres nœuds du réseau aient accepté le bloc 300 000 et aient ensuite effectué le travail nécessaire pour produire six autres blocs en plus est la preuve, par procuration, que la transaction n'était pas une double dépense.

Un nœud SPV ne peut pas être persuadé qu'une transaction existe dans un bloc alors que la transaction n'existe pas en fait. Le nœud SPV établit l'existence d'une transaction dans un bloc en demandant une preuve de chemin Merkle et en validant la preuve de travail dans la chaîne de blocs. Cependant, l'existence d'une transaction peut être "masquée" à un nœud SPV. Un nœud SPV peut certainement prouver qu'une transaction existe mais ne peut pas vérifier qu'une transaction, telle qu'une double dépense du même UTXO, n'existe pas car il n'a pas d'enregistrement de toutes les transactions. Cette vulnérabilité peut être utilisée dans une attaque par déni de service ou pour une attaque à double dépense contre les nœuds SPV. Pour se défendre contre cela, un nœud SPV doit se connecter de manière aléatoire à plusieurs nœuds, pour augmenter la probabilité qu'il soit en contact avec au moins un nœud honnête. Ce besoin de se connecter de manière aléatoire signifie que les nœuds SPV sont également vulnérables aux attaques de partitionnement de réseau ou aux attaques Sybil, où ils sont connectés à de faux nœuds ou à de faux réseaux et n'ont pas accès à des nœuds honnêtes ou au vrai réseau Bitcoin.

Dans la plupart des cas pratiques, les nœuds SPV bien connectés sont suffisamment sécurisés, trouvant un équilibre entre les besoins en ressources, l'aspect pratique et la sécurité. Pour une sécurité infaillible, cependant, rien ne vaut l'exécution d'un nœud de blockchain complet.

Un nœud de blockchain complet vérifie une transaction en vérifiant toute la chaîne de milliers de blocs en dessous afin de garantir que l'UTXO n'est pas dépensé, tandis qu'un nœud SPV vérifie à quelle profondeur le bloc est enterré par une poignée de blocs au-dessus.

Pour obtenir les en-têtes de bloc, les nœuds SPV utilisent un message getheaders au lieu de getblocks. L'homologue qui répond enverra jusqu'à 2 000 en-têtes de bloc à l'aide d'un seul message headers. Le processus est par ailleurs le même que celui utilisé par un nœud complet pour récupérer des blocs complets. Les nœuds SPV définissent également un filtre sur la connexion aux pairs, pour filtrer le flux des futurs blocs et transactions envoyés par les pairs. Toutes les transactions intéressantes sont récupérées à l'aide d'une requête getdata. Le pair génère un message tx contenant les transactions, en réponse. <<spv\_synchronization> > montre la synchronisation des en-têtes de bloc.

Étant donné que les nœuds SPV doivent récupérer des transactions spécifiques afin de les vérifier de manière sélective, ils créent également un risque pour la confidentialité. Contrairement aux nœuds de blockchain complets, qui collectent toutes les transactions au sein de chaque bloc, les demandes de données spécifiques du nœud SPV peuvent révéler par inadvertance les adresses de leur portefeuille. Par exemple, un tiers surveillant un réseau pourrait suivre toutes les transactions demandées par un portefeuille sur un nœud SPV et les utiliser pour associer des adresses Bitcoin à l'utilisateur de ce portefeuille, détruisant ainsi la vie privée de l'utilisateur.

Noeud .SPV synchronisant les en-têtes de bloc image::images/mbc2\_0807.png["Synchronisation SPV"]

Peu de temps après l'introduction des nœuds SPV/légers, les développeurs de bitcoins ont ajouté une fonctionnalité appelée *filtres de floraison* pour faire face aux risques de confidentialité des nœuds SPV. Les filtres Bloom permettent aux nœuds SPV de recevoir un sous-ensemble des transactions sans révéler précisément les adresses qui les intéressent, grâce à un mécanisme de filtrage qui utilise des probabilités plutôt que des modèles fixes.("", startref="simple08")

## Filtres Bloom

Un filtre bloom est un filtre de recherche probabiliste qui offre un moyen efficace d'exprimer un modèle de recherche tout en protégeant la confidentialité. Ils sont utilisés par les nœuds SPV pour demander à leurs pairs des transactions correspondant à un modèle spécifique, sans révéler exactement quelles adresses, clés ou transactions ils recherchent.

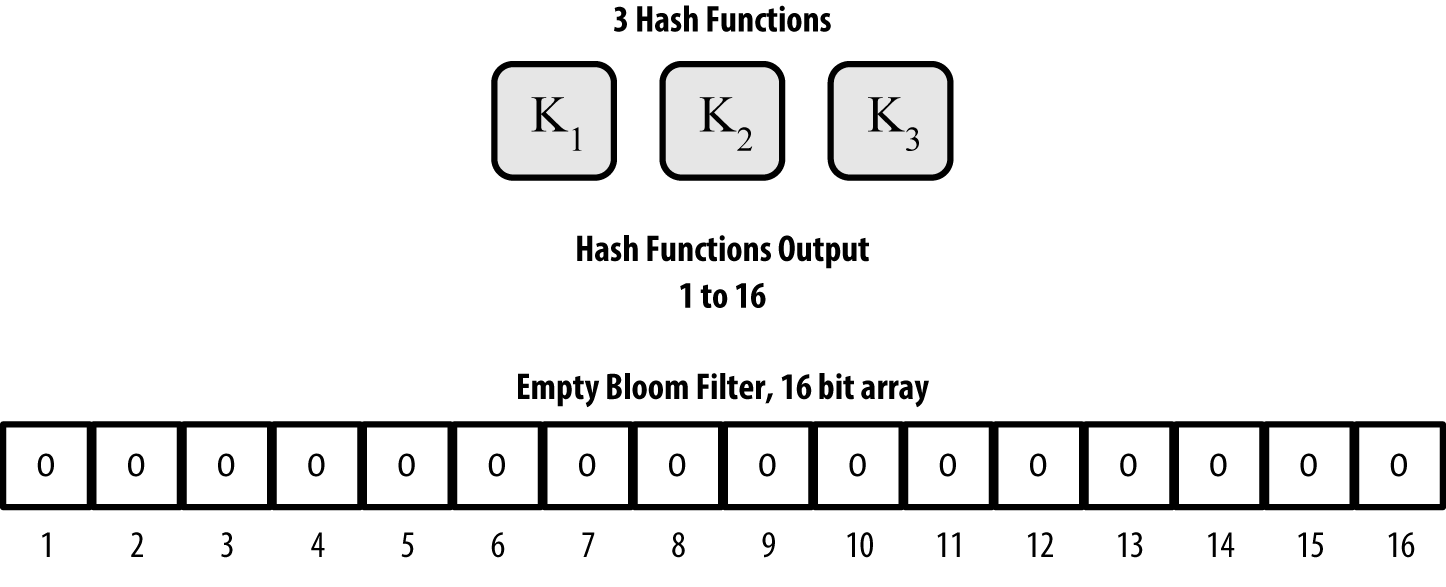
Dans notre analogie précédente, un touriste sans carte demande son chemin vers une adresse spécifique, "23 Church St." Si elle demande à des inconnus son chemin vers cette rue, elle révèle par inadvertance sa destination. Un filtre Bloom revient à demander : "Y a-t-il des rues dans ce quartier dont le nom se termine par RCH ?" Une question comme celle-là en dit un peu moins sur la destination souhaitée que de demander "23 Church St." En utilisant cette technique, un touriste pourrait spécifier l'adresse souhaitée plus en détail comme "se terminant par URCH" ou moins en détail comme "se terminant par H". En faisant varier la précision de la recherche, le touriste révèle plus ou moins d'informations, au détriment d'obtenir des résultats plus ou moins précis. Si elle demande un modèle moins spécifique, elle obtient beaucoup plus d'adresses possibles et une meilleure confidentialité, mais de nombreux résultats ne sont pas pertinents. Si elle demande un modèle très spécifique, elle obtient moins de résultats mais perd sa vie privée.

Les filtres Bloom remplissent cette fonction en permettant à un nœud SPV de spécifier un modèle de recherche pour les transactions qui peuvent être réglés vers la précision ou la confidentialité. Un filtre bloom plus spécifique produira des résultats précis, mais au détriment de la révélation des modèles qui intéressent le nœud SPV, révélant ainsi les adresses détenues par le portefeuille de l'utilisateur. Un filtre bloom moins spécifique produira plus de données sur plus de transactions, dont beaucoup ne sont pas pertinentes pour le nœud, mais permettra au nœud de maintenir une meilleure confidentialité.

### Comment fonctionnent les filtres Bloom

Les filtres Bloom sont implémentés sous la forme d'un tableau de taille variable de N chiffres binaires (un champ de bits) et d'un nombre variable de M fonctions de hachage. Les fonctions de hachage sont conçues pour produire toujours une sortie comprise entre 1 et N, correspondant au tableau de chiffres binaires. Les fonctions de hachage sont générées de manière déterministe, de sorte que tout nœud implémentant un filtre bloom utilisera toujours les mêmes fonctions de hachage et obtiendra les mêmes résultats pour une entrée spécifique. En choisissant des filtres de bloom de différentes longueurs (N) et un nombre différent (M) de fonctions de hachage, le filtre de bloom peut être réglé, en faisant varier le niveau de précision et donc de confidentialité.

Dans <<bloom1> >, nous utilisons un très petit tableau de 16 bits et un ensemble de trois fonctions de hachage pour démontrer le fonctionnement des filtres bloom.

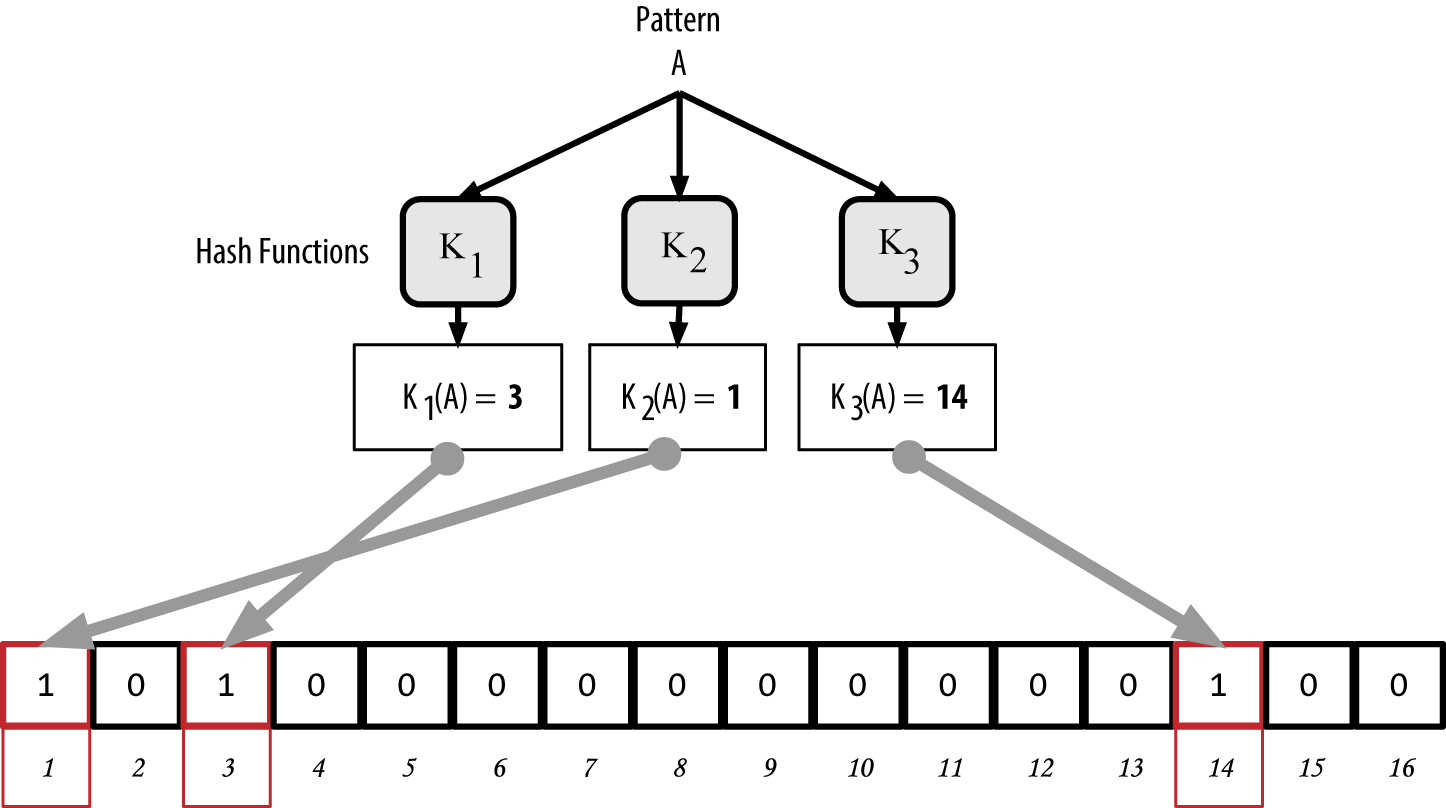


Un exemple de filtre bloom simpliste, avec un champ 16 bits et trois fonctions de hachage

Le filtre de bloom est initialisé de sorte que le tableau de bits ne soit que des zéros. Pour ajouter un motif au filtre bloom, le motif est haché tour à tour par chaque fonction de hachage. L'application de la première fonction de hachage à l'entrée donne un nombre compris entre 1 et N. Le bit correspondant dans le tableau (indexé de 1 à N) est trouvé et défini sur 1, enregistrant ainsi la sortie de la fonction de hachage. Ensuite, la fonction de hachage suivante est utilisée pour définir un autre bit et ainsi de suite. Une fois que toutes les fonctions de hachage M ont été appliquées, le modèle de recherche sera "enregistré" dans le filtre bloom sous forme de M bits qui ont été modifiés de 0 à 1.

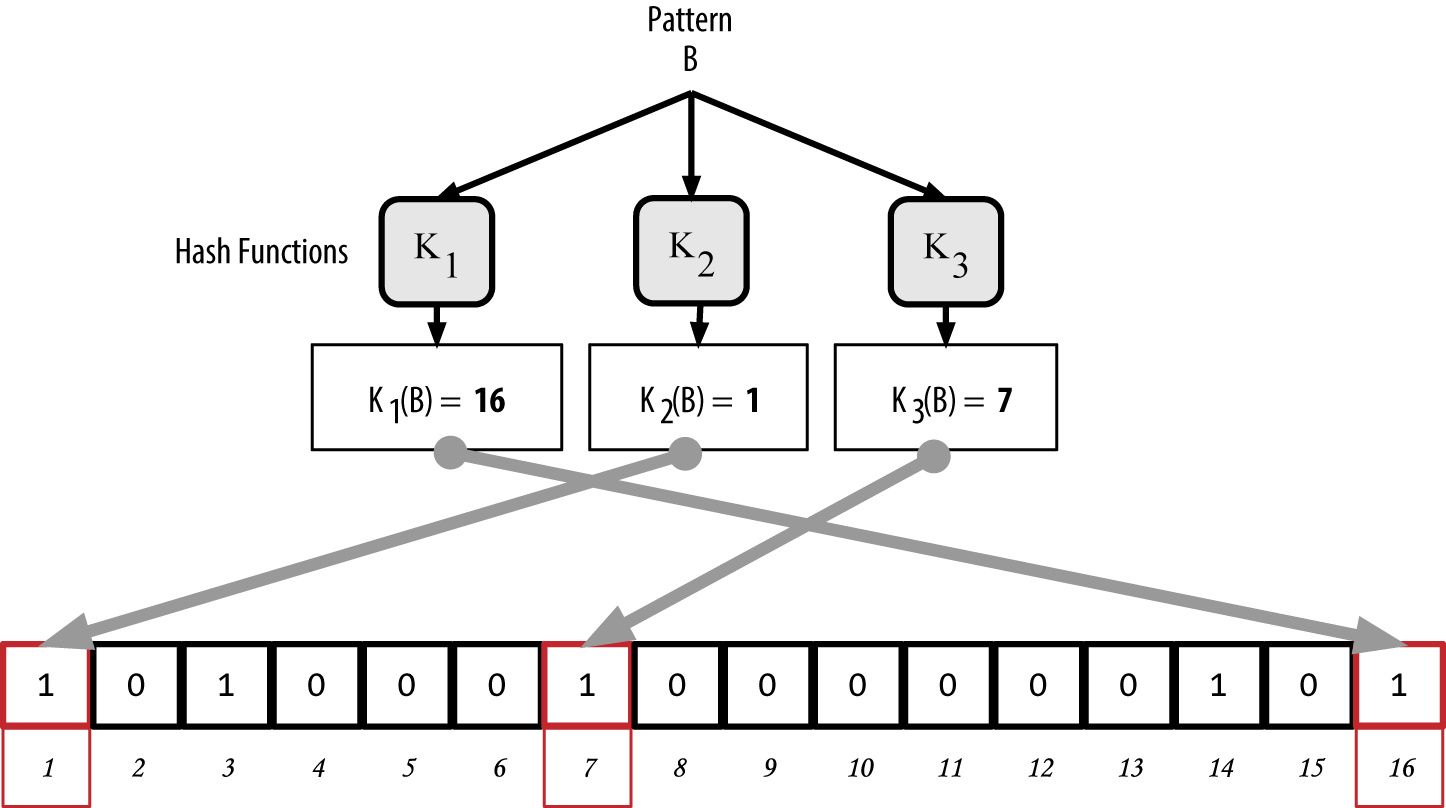
<<bloom2> > est un exemple d'ajout d'un motif "A" au filtre Bloom simple illustré dans <<bloom1> >.

L'ajout d'un deuxième motif est aussi simple que de répéter ce processus. Le motif est haché par chaque fonction de hachage tour à tour et le résultat est enregistré en réglant les bits sur 1. Notez qu'à mesure qu'un filtre bloom est rempli de plusieurs modèles, le résultat d'une fonction de hachage peut coïncider avec un bit déjà défini sur 1, auquel cas le bit n'est pas modifié. Essentiellement, à mesure que davantage de motifs sont enregistrés sur des bits qui se chevauchent, le filtre de bloom commence à être saturé avec davantage de bits réglés sur 1 et la précision du filtre diminue. C'est pourquoi le filtre est une structure de données probabiliste - il devient moins précis à mesure que d'autres modèles sont ajoutés. La précision dépend du nombre de motifs ajoutés par rapport à la taille du tableau de bits (N) et du nombre de fonctions de hachage (M). Un tableau de bits plus grand et plus de fonctions de hachage peuvent enregistrer plus de modèles avec une plus grande précision. Un tableau de bits plus petit ou moins de fonctions de hachage enregistrera moins de modèles et produira moins de précision.



Ajout d'un motif "A" à notre filtre bloom simple

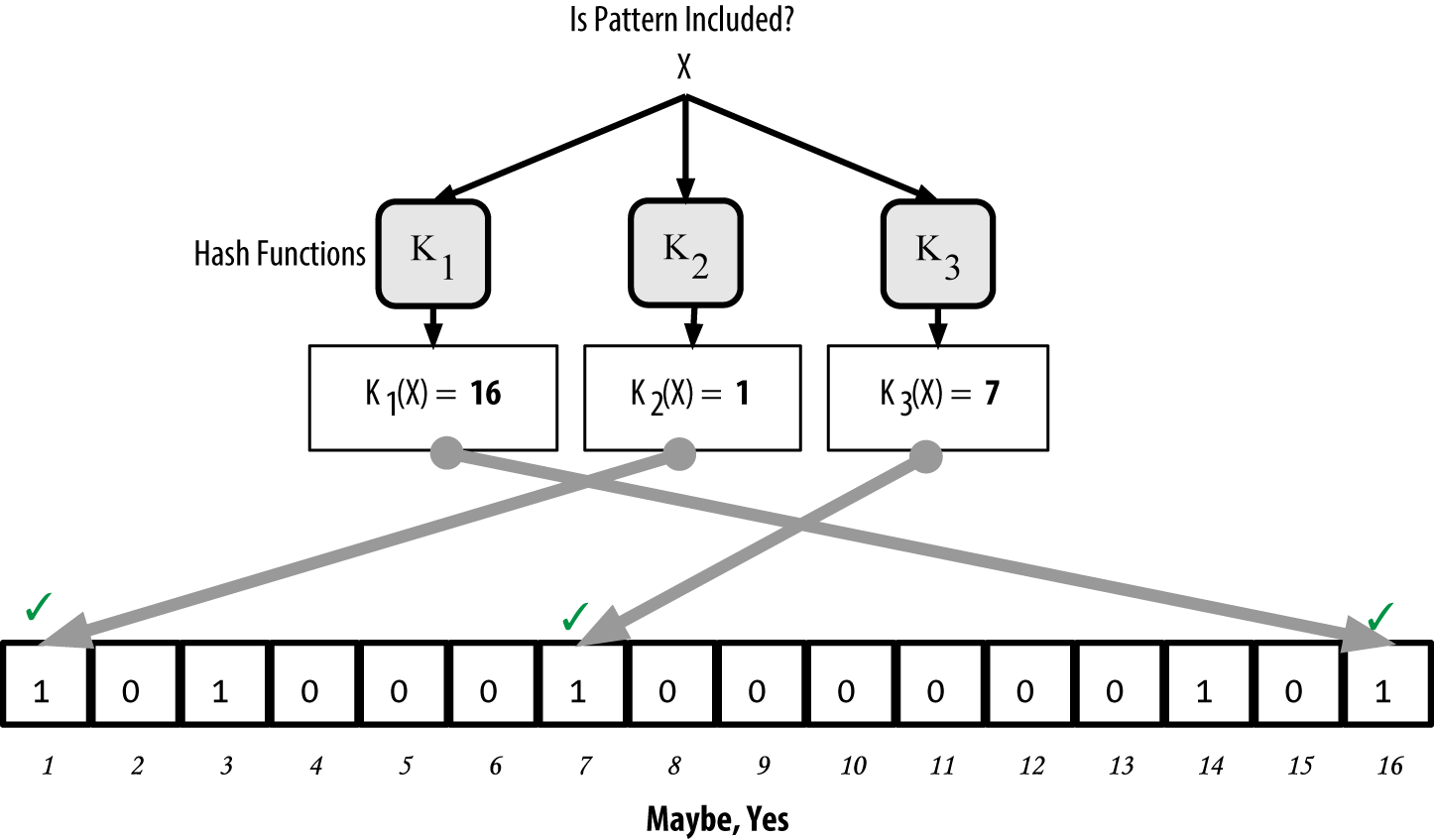
<<bloom3> > est un exemple d'ajout d'un deuxième motif "B" au filtre bloom simple.



Ajout d'un deuxième motif "B" à notre filtre bloom simple

Pour tester si un motif fait partie d'un filtre de bloom, le motif est haché par chaque fonction de hachage et le motif de bits résultant est testé par rapport au tableau de bits. Si tous les bits indexés par les fonctions de hachage sont mis à 1, alors le motif est *probablement* enregistré dans le filtre bloom. Étant donné que les bits peuvent être définis en raison du chevauchement de plusieurs modèles, la réponse n'est pas certaine, mais plutôt probabiliste. En termes simples, une correspondance positive du filtre bloom est un "Peut-être, Oui".

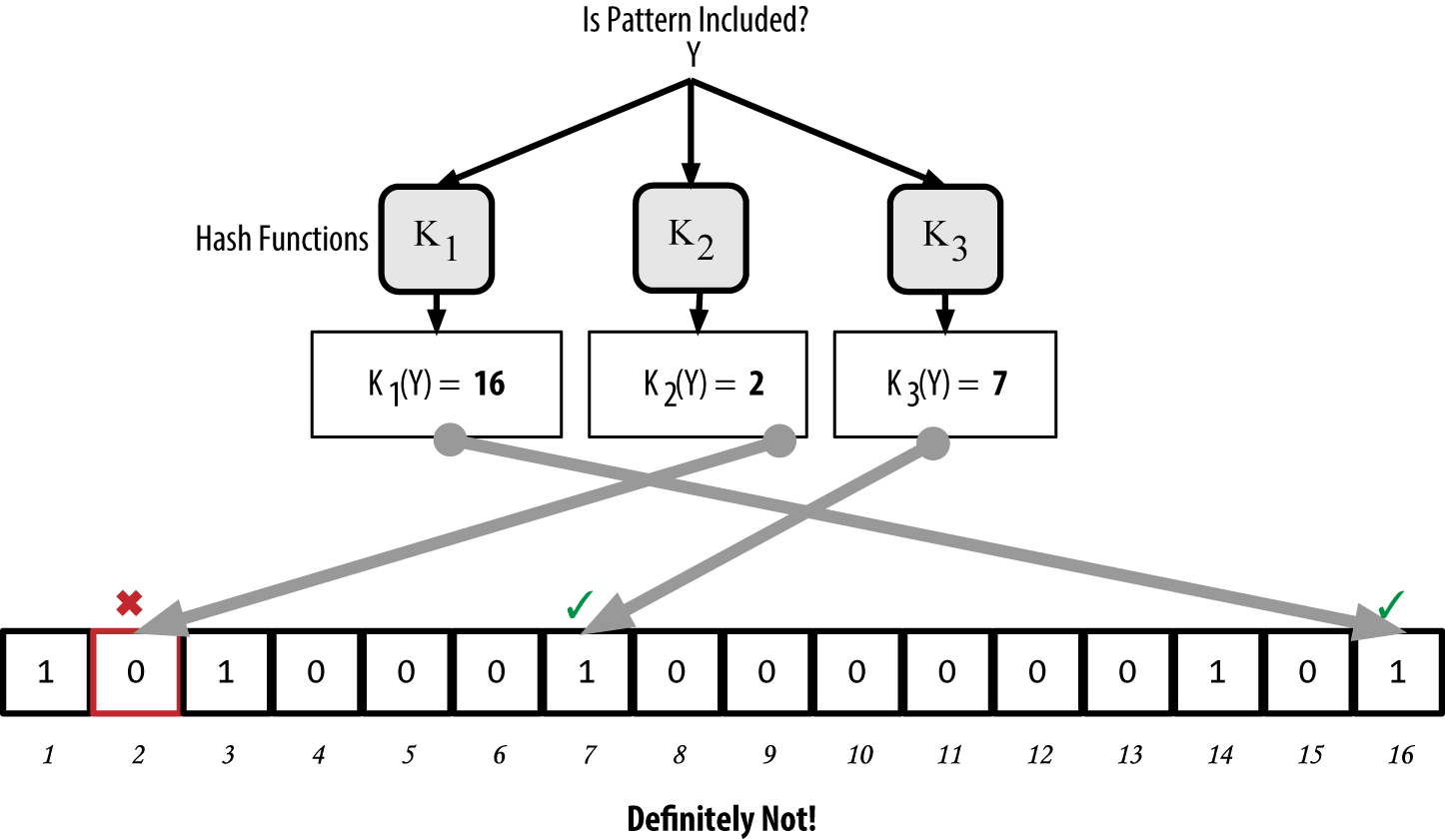
<<bloom4> > est un exemple de test de l'existence du motif "X" dans le filtre bloom simple. Les bits correspondants sont définis sur 1, donc le motif est probablement une correspondance.



Tester l'existence du motif "X" dans le filtre bloom. Le résultat est une correspondance positive probabiliste, ce qui signifie "Peut-être".

Au contraire, si un motif est testé par rapport au filtre bloom et que l'un des bits est mis à 0, cela prouve que le motif n'a pas été enregistré dans le filtre bloom. Un résultat négatif n'est pas une probabilité, c'est une certitude. En termes simples, une correspondance négative sur un filtre bloom est un "Certainement pas!"

<<bloom5> > est un exemple de test de l'existence du motif "Y" dans le filtre bloom simple. L'un des bits correspondants est défini sur 0, donc le motif n'est certainement pas une correspondance.



Tester l'existence du motif "Y" dans le filtre bloom. Le résultat est une correspondance négative définitive, ce qui signifie "Certainement pas !"

## Comment les nœuds SPV utilisent les filtres Bloom

Les filtres Bloom sont utilisés pour filtrer les transactions (et les blocs les contenant) qu'un nœud SPV reçoit de ses pairs, en sélectionnant uniquement les transactions qui intéressent le nœud SPV sans révéler les adresses ou les clés qui l'intéressent.

Un nœud SPV initialisera un filtre bloom comme "vide" ; dans cet état, le filtre bloom ne correspondra à aucun motif. Le nœud SPV établira ensuite une liste de toutes les adresses, clés et hachages qui l'intéressent. Il le fera en extrayant le hachage de clé publique et le hachage de script et les ID de transaction de tout UTXO contrôlé par son portefeuille. Le nœud SPV ajoute ensuite chacun d'entre eux au filtre de floraison, de sorte que le filtre de floraison "correspondra" si ces modèles sont présents dans une transaction, sans révéler les modèles eux-mêmes.

Le nœud SPV enverra alors un message filterload au pair, contenant le filtre bloom à utiliser sur la connexion. Sur le pair, les filtres bloom sont vérifiés par rapport à chaque transaction entrante. Le nœud complet vérifie plusieurs parties de la transaction par rapport au filtre Bloom, à la recherche d'une correspondance, notamment :

* L'identifiant de la transaction
* Les composants de données des scripts de verrouillage de chacune des sorties de transaction (chaque clé et hachage du script)
* Chacune des entrées de transaction
* Chacun des composants de données de signature d'entrée (ou scripts témoins)

En comparant tous ces composants, les filtres Bloom peuvent être utilisés pour faire correspondre les hachages de clés publiques, les scripts, les valeurs OP\_RETURN, les clés publiques dans les signatures ou tout futur composant d'un contrat intelligent ou d'un script complexe.

Une fois qu'un filtre est établi, le pair teste ensuite la sortie de chaque transaction par rapport au filtre Bloom. Seules les transactions qui correspondent au filtre sont envoyées au nœud.

En réponse à un message getdata du nœud, les pairs enverront un message merkleblock qui ne contient que des en-têtes de bloc pour les blocs correspondant au filtre et un chemin merkle (voir <<merkle\_trees> >) pour chaque transaction correspondante. Le pair enverra alors également des messages tx contenant les transactions mises en correspondance par le filtre.

Lorsque le nœud complet envoie des transactions au nœud SPV, le nœud SPV rejette tous les faux positifs et utilise les transactions correctement appariées pour mettre à jour son ensemble UTXO et le solde de son portefeuille. Au fur et à mesure qu'il met à jour sa propre vue de l'ensemble UTXO, il modifie également le filtre de floraison pour correspondre à toutes les transactions futures faisant référence à l'UTXO qu'il vient de trouver. Le nœud complet utilise ensuite le nouveau filtre bloom pour faire correspondre les nouvelles transactions et l'ensemble du processus se répète.

Le nœud définissant le filtre bloom peut ajouter de manière interactive des motifs au filtre en envoyant un message filteradd. Pour effacer le filtre bloom, le nœud peut envoyer un message filterclear. Puisqu'il n'est pas possible de supprimer un modèle d'un filtre bloom, un nœud doit effacer et renvoyer un nouveau filtre bloom si un modèle n'est plus souhaité.

Le protocole réseau et le mécanisme de filtre de bloom pour les nœuds SPV sont définis dans [BIP-37 (Peer Services)](https://bit.ly/1x6qCiO).

## Nœuds SPV et confidentialité

Les nœuds qui implémentent SPV ont une confidentialité plus faible qu'un nœud complet. Un nœud complet reçoit toutes les transactions et ne révèle donc aucune information indiquant s'il utilise une adresse dans son portefeuille. Un nœud SPV reçoit une liste filtrée des transactions liées aux adresses qui se trouvent dans son portefeuille. En conséquence, cela réduit la vie privée du propriétaire.

Les filtres Bloom sont un moyen de réduire la perte de confidentialité. Sans eux, un nœud SPV devrait répertorier explicitement les adresses qui l'intéressent, créant ainsi une grave violation de la vie privée. Cependant, même avec des filtres Bloom, un adversaire surveillant le trafic d'un client SPV ou s'y connectant directement en tant que nœud du réseau P2P peut collecter suffisamment d'informations au fil du temps pour apprendre les adresses dans le portefeuille du client SPV.

## Connexions cryptées et authentifiées

La plupart des nouveaux utilisateurs de bitcoin supposent que les communications réseau d'un nœud Bitcoin sont cryptées. En fait, la mise en œuvre originale du bitcoin communique entièrement en clair. Bien que ce ne soit pas un problème majeur de confidentialité pour les nœuds complets, c'est un gros problème pour les nœuds SPV.

Afin d'augmenter la confidentialité et la sécurité du réseau bitcoin P2P, il existe deux solutions qui fournissent le cryptage des communications : *Tor Transport* et *P2P Authentication and Encryption* avec BIP-150/151.

### Transport Tor

Tor, qui signifie *The Onion Routing network*, est un projet logiciel et un réseau qui offre le cryptage et l'encapsulation des données via des chemins de réseau qui offrent l'anonymat, l'intraçabilité et la confidentialité.

Bitcoin Core offre plusieurs options de configuration qui vous permettent d'exécuter un nœud Bitcoin avec son trafic transporté sur le réseau Tor. De plus, Bitcoin Core peut également offrir un service caché Tor permettant à d'autres nœuds Tor de se connecter à votre nœud directement via Tor.

À partir de la version 0.12 de Bitcoin Core, un nœud proposera automatiquement un service Tor caché s'il est capable de se connecter à un service Tor local. Si vous avez installé Tor et que le processus Bitcoin Core s'exécute en tant qu'utilisateur disposant des autorisations adéquates pour accéder au cookie d'authentification Tor, il devrait fonctionner automatiquement. Utilisez le drapeau debug pour activer le débogage de Bitcoin Core pour le service Tor comme ceci :

$ bitcoind --daemon --debug=tor

Vous devriez voir "tor : ADD\_ONION réussi" dans les journaux, indiquant que Bitcoin Core a ajouté un service caché au réseau Tor.

Vous pouvez trouver plus d'instructions sur l'exécution de Bitcoin Core en tant que service caché de Tor dans la documentation de Bitcoin Core (*docs/tor.md*) et divers didacticiels en ligne.

### Authentification et cryptage peer-to-peer

Deux propositions d'amélioration Bitcoin, BIP-150 et BIP-151, ajoutent la prise en charge de l'authentification et du cryptage P2P dans le réseau bitcoin P2P. Ces deux BIP définissent des services optionnels pouvant être proposés par des nœuds Bitcoin compatibles. BIP-151 active le chiffrement négocié pour toutes les communications entre deux nœuds qui prennent en charge BIP-151. BIP-150 offre une authentification par les pairs en option qui permet aux nœuds de s'authentifier mutuellement à l'aide d'ECDSA et de clés privées. BIP-150 exige qu'avant l'authentification, les deux nœuds aient établi des communications cryptées conformément à BIP-151.

Depuis février 2021, BIP-150 et BIP-151 ne sont pas implémentés dans Bitcoin Core. Cependant, les deux propositions ont été mises en œuvre par au moins un client Bitcoin alternatif nommé bcoin.

BIP-150 et BIP-151 permettent aux utilisateurs d'exécuter des clients SPV qui se connectent à un nœud complet de confiance, en utilisant le chiffrement et l'authentification pour protéger la confidentialité du client SPV.

De plus, l'authentification peut être utilisée pour créer des réseaux de nœuds Bitcoin de confiance et empêcher les attaques Man-in-the-Middle. Enfin, le cryptage P2P, s'il est déployé à grande échelle, renforcerait la résistance du bitcoin à l'analyse du trafic et à la surveillance portant atteinte à la vie privée, en particulier dans les pays totalitaires où l'utilisation d'Internet est fortement contrôlée et surveillée.

La norme est définie dans https:/ /github.com/bitcoin/bips/blob/master/bip-0150.mediawiki[BIP-150 (authentification par les pairs)] et  [BIP-151 (chiffrement des communications poste à poste)](https://github.com/bitcoin/bips/blob/master/bip-0151.mediawiki).

## Groupes de transactions

Presque tous les nœuds du réseau Bitcoin maintiennent une liste temporaire de transactions non confirmées appelées *memory pool*, *mempool* ou *transaction pool*. Les nœuds utilisent ce pool pour suivre les transactions connues du réseau mais qui ne sont pas encore incluses dans la blockchain. Par exemple, un nœud de portefeuille utilisera le pool de transactions pour suivre les paiements entrants dans le portefeuille de l'utilisateur qui ont été reçus sur le réseau mais qui n'ont pas encore été confirmés.

Au fur et à mesure que les transactions sont reçues et vérifiées, elles sont ajoutées au pool de transactions et relayées aux nœuds voisins pour se propager sur le réseau.

Certaines implémentations de nœuds maintiennent également un pool séparé de transactions orphelines. Si les entrées d'une transaction font référence à une transaction qui n'est pas encore connue, comme un parent manquant, la transaction orpheline sera stockée temporairement dans le pool orphelin jusqu'à ce que la transaction parent arrive.

Lorsqu'une transaction est ajoutée au pool de transactions, le pool orphelin est vérifié pour tous les orphelins qui référencent les sorties de cette transaction (ses enfants). Tous les orphelins correspondants sont ensuite validés. S'ils sont valides, ils sont supprimés du pool orphelin et ajoutés au pool de transactions, complétant ainsi la chaîne qui a commencé avec la transaction parent. À la lumière de la transaction nouvellement ajoutée, qui n'est plus orpheline, le processus est répété de manière récursive à la recherche de tout autre descendant, jusqu'à ce qu'aucun autre descendant ne soit trouvé. Grâce à ce processus, l'arrivée d'une transaction parente déclenche une reconstruction en cascade de toute une chaîne de transactions interdépendantes en réunissant les orphelins avec leurs parents tout au long de la chaîne.

Le pool de transactions et le pool orphelin (le cas échéant) sont stockés dans la mémoire locale et ne sont pas enregistrés sur le stockage persistant ; ils sont plutôt renseignés dynamiquement à partir des messages réseau entrants. Lorsqu'un nœud démarre, les deux pools sont vides et se remplissent progressivement avec les nouvelles transactions reçues sur le réseau.

Certaines implémentations du client Bitcoin maintiennent également une base de données ou un pool UTXO, qui est l'ensemble de toutes les sorties non dépensées sur la blockchain. Les utilisateurs de Bitcoin Core le trouveront dans le dossier chainstate/ du répertoire de données de leur client. Bien que le nom "pool UTXO" ressemble au pool de transactions, il représente un ensemble de données différent. Contrairement aux pools de transactions et orphelins, le pool UTXO n'est pas initialisé vide mais contient à la place des millions d'entrées de sorties de transactions non dépensées, tout ce qui n'est pas dépensé depuis le bloc de genèse. Le pool UTXO peut être hébergé dans la mémoire locale ou sous la forme d'une table de base de données indexée sur un stockage persistant.

Alors que les pools de transactions et orphelins représentent la perspective locale d'un seul nœud et peuvent varier considérablement d'un nœud à l'autre en fonction du moment où le nœud a été démarré ou redémarré, le pool UTXO représente le consensus émergent du réseau et variera donc peu entre les nœuds. De plus, les pools de transactions et orphelins ne contiennent que des transactions non confirmées, tandis que le pool UTXO ne contient que des sorties confirmées.