Quick Notification of Block Generation Using Bloom Filter in a Blockchain

Tsuyoshi Hasegawa
Kyoto University
Kyoto, Japan

Akira Sakurai
Tokyo Institute of Technology
Tokyo, Japan

Kazuyuki Shudo
Kyoto University
Kyoto, Japan

***Abstract*—Forks in a blockchain sacrifice security. In this paper, we propose a protocol for quickly propagating block generation notifications in the blockchain network quickly to reduce the fork rate. Block generation notifications contain a Bloom filter that represents transactions in the generated block. Thus, when nodes receive a block generation notification, they can start mining the next block. In experiments in which a simulator is used, we compared the propagation time of a block generation notification with that of a block in the existing protocol. As a result, the propagation time of the 50 %ile is 41.1 % of an existing protocol, the 90 %ile is 39.2 % of the existing protocol, and the fork rate calculated from the average propagation time is 40.8 % of the existing protocol.**

***Index Terms*—blockchain, block propagation, fork rate**

I. Introduction

Blockchain is a decentralized system that is difficult to tamper with and utilized on a cryptocurrency basis. Currently, a disadvantage of blockchains is that they can only approve few transactions in a unit of time. Specifically, the performance of Bitcoin [1] transaction approval was initially only 7 TPS (transactions / sec). If the block generation interval is shortened to resolve the lack of TPS, the blockchain security is sacrificed [2]. The solution increases the possibility of a new block being generated by a node that has not received the generated block and divergence in the blockchain. The divergence in the blockchain is referred to as a fork and the rate of fork occurrence is referred to as a fork rate. By shortening the block propagation time, the fork rate can be reduced and the block generation interval can be shortened.

We propose a protocol that quickly propagates a block generation notification, which informs nodes that a block has been generated. We assume blockchains in which a fork can occur because all nodes probabilistically generate blocks.

In Section II, we show related research on block propagation between nodes. In Section III, we describe the proposed protocol. In Section IV, we show the results of simulation experiments. In Section V, we summarize our research.

II. Related Research

In the blockchain, the block consists of a header and a set of transactions. Nodes mined the block, and if they succeeded, transmitted the block to their neighbor nodes. For this node-to-node transfer protocol, first, we describe the normal node-to-node protocol (legacy protocol). Second, we describe Compact Block Relay (CBR) [3] and Graphene [4], which are efficient block transfer protocols between nodes.

A. Legacy Protocol

In the legacy protocol, a block itself is transmitted between nodes. First, the block sender node sends an inventory (`inv`) message that informs which block will be transmitted to a receiver node, and the receiver sends a `getdata` message if the block is needed. The sender transmits the block itself when receiving the `getdata` message. The size of the block is limited by 1 MiB [5].

B. Compact Block Relay

In the Compact Block Relay (CBR), a compact block that consists of a header and set of transaction ids is sent instead of a full block. Transactions have been sent to the blockchain network and nodes collect them in their transaction pool (mempool), thus, the node receiving the compact block can reconstruct the full block using the set of transaction ids. If the receiver does not have all transactions in the full block, the node request lacks transactions for the sender. The size of the compact block is smaller than that of the full block, approximately 18 KiB [6], thus it can be transmitted at a faster rate than the full block.

C. Graphene

Graphene is a protocol that uses two probabilistic data structures, the Bloom filter [7] and Invertible Bloom Lookup Table (IBLT) [8], to efficiently encode the transactions contained in a block. The sender transmits a Bloom filter S and IBLT I that encode the transactions in the block, and the receiver tries to decode them. If the receiver succeeds in decoding, the node can know all transactions in the block. In the case of a decoding failure, the receiver sends another Bloom filter R, and the sender sends another IBLT J and expectes to lack transaction set C. Then, the receiver tries to decode IBLT J. If the second attempt also fails, a normal block transfer is performed. In Graphene, the size of the data transferred between nodes is less than the CBR in total.

III. Proposed Protocol

This section describes the proposed protocol to reduce the fork rate in blockchains.

In this paper, we propose a protocol to quickly notify the nodes in the blockchain network that a block has been

Proc. 28th IEEE Symposium on Computers and Communications
(IEEE ISCC 2023), July 2023

TABLE I
STRUCTURE OF A BLOCK GENERATION NOTIFICATION (BLOCK HEADER).

Element	Description
Transaction filter	Bloom filter with UTXOs utilized in transactions in the generated block as an element
Prerequisite block indices	Indices of prerequisite blocks at generated block
Version	Software / protocol version number
Previous Block Hash	Hash of the parent block
Merkle Root	Root hash of the Merkle Tree for all transactions in the block
Timestamp	Generation time of the block
Difficulty Target	Difficulty of proof of work during block generation
Nonce	Counters used in proof of work

generated. We refer to this notification as a block generation notification. The block generation notification includes information about the generated block, and the nodes that receive the notification start the next mining based on the notification. The rapid propagation of block generation notifications and the notion that the nodes that receive them start the next mining with the block corresponding to the notification as its parent reduces the fork rate. Hence, a space-efficient data structure, the Bloom filter, is selected to represent the information of the generated block and is included in the block generation notification.

Table I shows the structure of a block generation notification. In the proposed protocol, the block header is extended to be a block generation notification. The structure is similar to the Bitcoin header, with the exception that a transaction filter and prerequisite block indices are added. The miner mines using the extended header and propagates the header as a block generation notification when the block is successfully generated. In this way, the node receiving the block generation notification can check the metadata of the generated block and confirm that the creator of the block generation notification has indeed successfully mined the proof of work. The transaction filter is discussed in more detail in Subsection III-A, and the prerequisite blocks are discussed in more detail in Subsection III-B. Figure 1 shows a summary of the proposed protocol.

A possible method for implementing block generation notifications in Bitcoin is to include them in transactions made by miners and propagate them to the blockchain network.

A. Transaction Filter

The transaction filter F_h is a Bloom filter whose elements are the UTXOs consumed by the transactions in the generated block B_h. A node that receives a block generation notification N_h determines the UTXOs consumed by each transaction in its own transaction pool using transaction filter F_h. If all consuming UTXOs for each transaction are negative, the node determines that it is not included in the generated block B_h and includes it in the next mining.

Bloom filters have the potential for false positives. Ways to address false positives are discussed in Subsection III-B.

Algorithm 1 Validation of block generation notification N_h

1: Validate one previous block generation notification N_{h-1}.
2: Validate blocks activated by N_h.
3: Validate each header element of N_h (no validation of the Merkle root, otherwise the same as Bitcoin header validation).

B. Activation of Blocks

Bloom filters have the problem of false positives, which means that transactions that are false positives by a certain transaction filter can no longer be included in the mining. A transaction that is a false positive by a transaction filter F_h of a certain height h can be determined to be a false positive by checking transactions at the corresponding same height block B_h. The transaction filter F_h can be inactivated by block B_h. Ensuring the existence of block B_h and inactivating the corresponding transaction filter F_h is referred to as enabling block B_h. Activating a block held by each node at mining inactivates the corresponding transaction filter and solves the problem of false positives.

However, the block propagation status is not the same for each node. When verifying block B_h, each node needs to know which blocks were activated when the block B_h was generated. Hence, the proposed method is to include in the block generation notification the information of the blocks that each node has received and for which the corresponding transaction filter has been inactivated at mining. This information is referred to as the prerequisite block indices. By using the prerequisite block indices in block generation notification N_h of height h and activating these blocks, the status of the active blocks at height h can be identified for all nodes. Conversely, the block is unconfirmed until it is activated by a block generation notification.

A possible attack by malicious nodes is to propagate only block generation notification N_h and not block B_h upon successful mining of height h. By this attack, the corresponding transaction filter F_h cannot be inactivated because block B_h cannot be propagated to ensure that transactions that are positive due to F_h can not be included in the mining. To counter this attack, a time limit is set on block activation. Specifically, if block B_h is not activated for b consecutive blocks, i.e., if block B_h is not included in the prerequisite block information of a block generation notification of height $h+1$ to $h+b$, the transaction filter F_h is disabled and B_h cannot be activated thereafter. Since block B_h can no longer be activated, all transactions that were positive by transaction filter F_h are included in the mining. Subsection III-F describes how the system-wide parameter b is determined.

C. Validation of Block Generation Notification

Algorithm 1 shows the algorithm for validation of the block generation notification, which is performed by the node receiving block generation notification N_h of height h.

In the validation of block generation notification N_h, first, one previous block generation notification N_{h-1} is checked

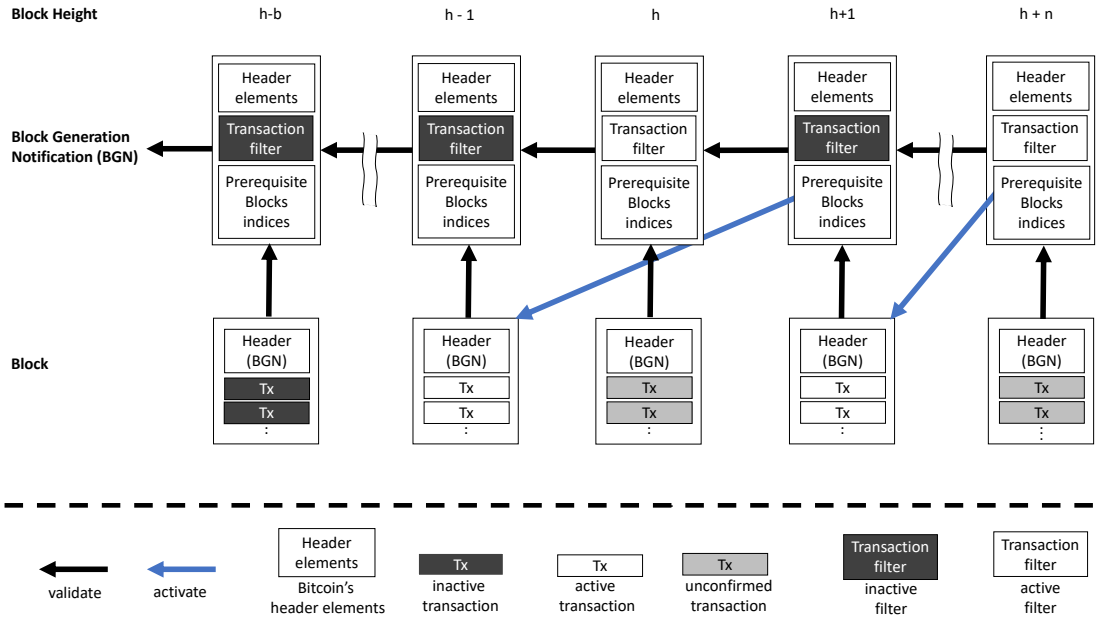

Fig. 1. Summary of the proposed protocol.

Algorithm 2 Validation of block B_h

1: Block generation notification N_h id validated.
2: All transactions in block B_h are checked to ensure that they pass through the integration filter I_h.
3: Each transaction in block B_h id validated.

Algorithm 3 Creation of integration filter I_h

1: An integration filter I_h = an empty Bloom filter that is a bit array, all set to 0.
2: $r = h$
3: **while** $r \geq 0$ **do**
4: **if** F_r is not inactivated **then**
5: $T_h = Unite(T_h, F_r)$ ▷ $Unite$ is a function that takes the logical OR of each bit.
6: **end if**
7: $r = r - 1$
8: **end while**
9: **return** T_h

to ensure that it has been validated. Thus, to validate N_h, it is recursively required that all previous block generation notifications have been validated. Second, the prerequisite block indices in N_h are checked, and the activated blocks are validated. If the node receiving N_h has not received these blocks, the node requests them for the N_h sender. Last, each header element of N_h are validated. In this validation, the Merkle root is not validated because block B_h is required for this validation; otherwise, the same is true for the validation of the Bitcoin header. Block B_h is not validated in the N_h validation.

D. Validation of Block

Algorithm 2 shows the algorithm for validation of block B_h of height h. The node receiving a block generation notification higher than height h activating block B_h validates block B_h.

In the validation of block B_h, first, the corresponding block generation notification N_h is validated. Second, the next step is to check that all transactions in block B_h pass through the integration filter I_h. The integration filter is the integration of the active transaction filters at height h, and the algorithm for creating the integrated filter is given by Algorithm 3. Last, each transaction in block B_h is validated. The validation of one previous block B_{h-1} is not performed because in the proposed protocol, blocks are activated by a block generation notification, and the prerequisite block when block B_h was generated is indicated in block generation notification N_h and is validated in the first N_h verification (Algorithm 1).

E. Mining

Algorithm 4 shows the algorithm by which the node receives block generation notification N_h and starts the next mining.

When ntheode that received a block generation notification N_h starts the next mining, it validates the received N_h (Algorithm 1). Then, if any block B_i up to the previous b blocks already received has not yet been activated, the corresponding transaction filter F_i is inactivated, and the prerequisite block indices are updated (there can be multiple blocks to be activated). Subsequently, an integrated filter I_h is generated (Algorithm 3). The generated integrated filter I_h is then used to determine transactions in the transaction pool of the receiver. The UTXOs consumed by each transaction are determined by the integrated filter I_h and at least one positive transaction is excluded from mining. From the remaining transactions,

Algorithm 4 Mining

1: Block generation notification N_h is validated.
2: Transaction filters corresponding to blocks that have not been activated up to b blocks already received are inactivated.
3: References of blocks that have not been activated up to b blocks already received are added to the prerequisite block indices.
4: An integrated filter I_h is created.
5: Transactions in the transaction pool are determined by I_h.
6: Transactions included in the next block B_{h+1} are determined.
7: A transaction filter F_{h+1} is created.
8: The transaction filter F_{h+1} and prerequisite block indices in the block header are included, and then mining is started.
9: Mining success.
10: The block header of the generated block B_{h+1} is propagated as a block generation notification.
11: Block B_{h+1} is propagated.

a new transaction filter F_{h+1} is created to determine the transactions to be included in the next block B_{h+1}. Mining is then started by including the transaction filter and prerequisite block indices in the extended block header. If the mining is successful, the block header of the generated block B_{h+1} is propagated as block generation notification N_{h+1} and then block B_{h+1} is propagated.

F. Size of a transaction filter

We consider the size of a transaction filter included in the block generation notification. The transaction filter should be as small as possible to minimize the propagation time of the block generation notification.

The transaction filter F_h is inactivated if the corresponding block B_h is not activated for b blocks (Subsection III-B). Hence, the number of active transaction filters is highest when b consecutive blocks have not been activated by block generation notifications, and the number is b. In this paper, we set the false positive rate of the Bloom filter so that the ratio of the highest number of false positive transactions to the total number of transactions that have not been included in the generated block and should be available for mining is less than or equal to the parameter r. The size of the smallest Bloom filter is $-\frac{n \ln f}{\ln^2 2}$ bits when the number of filter elements is n and the false positive rate is f.

The average number of UTXOs consumed by a transaction is three [9]. We use this figure to determine the size of the Bloom filter. When at least one of the consuming UTXOs is positive, the transaction is not included in the next mining. Therefore, the probability f' of an originally usable transaction becoming a false positive transaction is

$$f' = 1 - (1 - f)^3, \tag{1}$$

where f is the false positive rate of the Bloom filter.

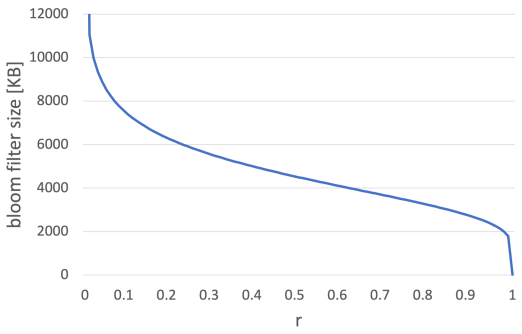

Fig. 2. Size of the Bloom filter when r is varied.

For the ratio of false positive transactions to the total number of transactions that should be available for mining to be less than or equal to r, the inequality satisfied by f' is

$$\sum_{k=1}^{b} f'(1 - f')^{k-1} \leq r. \tag{2}$$

Substituting Equation (1) into inequality (2) and rearranging, we obtain

$$f \leq 1 - (1 - r)^{\frac{1}{3b}}. \tag{3}$$

The false positive rate and size in a Bloom filter is a trade-off. Thus, the false positive rate of the Bloom filter is maximized to be as small as possible. Thus, the false positive rate of the Bloom filter is

$$f = 1 - (1 - r)^{\frac{1}{3b}}. \tag{4}$$

We describe parameter b. From Equation (4), f decreases as b increases when r is fixed. Since the size of a Bloom filter increases as the false positive rate decreases, the size increases as b increases. Therefore, to make the size as small as possible, we obtain the smallest acceptable b. Malicious nodes can inactivate the $b + 1$ previous block by creating blocks without activating this block b consecutive times. If b is small, the possibility of the success of such an attack increases. We set b such that the possibility that malicious nodes succeed in generating block b consecutive times is lower than the possibility of a successful double-spending attack. The possibility that malicious nodes succeed in generating blocks b consecutive times is h^b when h is the hash rate of malicious nodes. With reference to the possibility of a successful double-spending attack depending on the hash rate by Chaudhary et al. [10], $b = 4$ is sufficient.

Figure 2 shows the size of the transaction filter when r is varied, $b = 4$, and the number of transactions included in a block is 2000. The transaction filter size is 7.181 KiB when $r = 0.1$, and 4.391 KiB when $r = 0.5$.

The size of the block generation notification is sufficiently small; thus, the proposed protocol does not use `inv` messages when sending block generation notifications.

G. Impact of attacks and countermeasures

Possible attacks on the proposed method, their impact and countermeasures are described.

1) Attacks that do not propagate blocks: As described in Sections III-B and III-F, when a block is generated, there is a possible attack where only the block generation notification is propagated and the block is not propagated. To address this attack, blocks that have not been activated for b blocks inactivate the corresponding transaction filter (Section III-B). However, since the corresponding transaction filter is not inactivated until b blocks have been accumulated, there is a problem that for this period the transactions made positive by that transaction filter cannot be included in the mining. We propose a method for setting the size of the transaction filter so that the number of worst-case false positive transactions is below a certain percentage.

2) Attacks that do not activate blocks: As described in Section III-F, a possible attack occres when malicious nodes inactivate the $b+1$ previous block by creating blocks without activating this block b consecutive times. Since the success probability of this attack decreases as the parameter b increases, in this paper the value of the parameter b is set such that the success probability of this attack is lower than that of the double-spending attack.

3) Attacks that keep certain transactions positive: It is possible to prevent certain transactions from being used by creating a transaction filter such that certain transactions are positive when malicious nodes succeed in mining. This step could lead to an attack whereby successful mining occurs again before the b blocks stack up, keeping certain transactions positive and preventing them from being included in the block. This attack has a trade-off with the attack in which the blocks above are not propagated, as the probability of success increases as b increases.

4) Attacks that propagate a malformed transaction filter: By propagating a malformed transaction filter such that all UTXOs are positive, all transactions become unusable. To address this attack, a possible approach is to limit the percentage of bits with a value of 1, assuming that the initial value of the Bloom filter is 0 and that the positive bits have a value of 1. It is possible that the number of bits with a value of 1 may exceed the limit even for nonmalicious nodes by chance. In this case, we allow the reconfiguration of the Bloom filter and include information in the notification indicating the number of reconfigurations.

IV. Experiment

We show that block generation notifications in the proposed protocol quickly propagate. A simulator is used to measure the propagation time of the proposed protocol and compare related studies.

A. Experimental Method

We use SimBlock [11], [12] a blockchain network simulator for experiments. The Legacy protocol and CBR have been implemented in SimBlock, so Graphene and the proposed protocol are additionally implemented, and the propagation time is measured. For the proposed protocol, the propagation times of the block generation notification and the block body are measured. The propagation time of the block generation notification is employed for comparison with other methods because miners stop the current mining and start the next mining at the time of receiving the block generation notification. Since the experiments are intended to reveal the fork rate for each protocol, it is sufficient to consider only the propagation time of the block generation notification for the proposed protocol. Table II shows the parameters of the blockchain network in the simulation. Network parameters other than those in Table II include node distribution, bandwidth, and network delay, the values of which are obtained from Nagayama et al. [6].

TABLE II
NETWORK PARAMETERS.

Number of nodes	10000
Block generation interval	10 min
Size of block	1.0 MiB
Distribution of hash rate	Normal distribution
Average of hash rate	400000 / sec
Variance of hash rate	100000 / sec

TABLE III
CBR PARAMETERS.

Compact block size	18 KiB
Ratio of churn nodes	0.97
Block reconstruction failure rate in churn nodes	0.27
Block reconstruction failure rate in control nodes	0.1

1) Implemention of CBR: Table III shows the CBR parameters in SimBlock. Control nodes are nodes that always participate in the blockchain network, while churn nodes are nodes that repeatedly join and leave the blockchain network. Churn nodes and control nodes have different rates of block reconstruction failure in the CBR, and the distribution of the number of missing transactions in the event of block reconstruction failure is also different for control nodes and churn nodes. Nagayama et al. [6] calculate the cost of sending the missing transactions based on each distribution. These parameters are utilized in this experiment.

2) Implemention of Graphene: Table IV shows the parameters of Graphene in SimBlock implemented in this experiment. The number of transactions per block and the number of transactions in the transaction pool of a node are referred to in [5]. The ratio of churn nodes is the same as the CBR (Table III). From these parameters, we calculated the size of the Bloom filters S, R, and IBLT I and J, as described in subsection II-C. For simplicity, the size of the missing transactions in the CBR is applied. The decoding of IBLT I is assumed to succeed 100 % of the time if the percentage of the number of missing transactions is less than 2 % (scenario 1) since the decoding will almost succeed in scenario 1. The decoding succeed probability is assumed to be 0 % when the percentage of missing transactions exceeds 2 % (scenario 2)

TABLE IV
GRAPHENE PARAMETERS.

Number of transactions per block	2000
Number of transactions in a transaction pool (mempool)	4000
Bloom filter S size	2.434 KiB
IBLT I size	0.901 KiB
Bloom filter R size	1.453 KiB
IBLT J size	1.407 KiB
Probability of 1st decoding success (scenario 1)	100 %
Probability of 1st decoding success (scenario 2)	0 %
Probability of 2nd decoding success	100 %

TABLE V
PROPOSED PROTOCOL PARAMETERS.

Block generation notification size	7.272 KiB ($r = 0.1$) or 4.468 KiB ($r = 0.5$)
Block size	1.0 MiB
Methods of block propagation	CBR
Depth of prerequisite block	1, 2 or 3

since the decoding almost fails in scenario 2. The success probability of the second decoding, i.e. the decoding of IBLT J is assumed to always succeed, as it is almost 100 % according to Ozisik et al. [4].

3) Implementation of the Proposed Protocol: Table V shows the parameters of the proposed protocol implemented in this experiment. The size of a block generation notification is the sum of the size of a transaction filter described in Section III and the block header of Bitcoin. The prerequisite block indices are disregarded because they are very small. Parameter r is the ratio of the highest number of false positive transactions to the total number of transactions that have not been included in the block and should be available for mining. In this experiment, we set $r = 0.1$ and, 0.5 and compare. Mining is assumed to be a constant depth block on all nodes, and the compared depth is 1 to 3. Block bodies are propagated with CBR.

B. Results and Consideration

We describe the comparison of propagation times according to the parameters of the proposed protocol and those between each protocol. The end block height of the simulation is 100,000 blocks.

1) Comparison according to the parameters of the proposed protocol: Table VI shows the comparison of the propagation times of the block generation notification at $r = 0.1$ and 0.5. The depth of a prerequisite block is 1. The size of the block generation notification is 7.272 KiB when $r = 0.1$ and 4.468 KiB when $r = 0.5$; consequently, the propagation time is smaller for $r = 0.5$ than for $r = 0.1$.

Table VII compares the propagation times of the block generation notification and the number of requests for the prerequisite block when the depth of the prerequisite block is varied. The value of the parameter r is 0.1. The difference between the average propagation time of the block generation notification for depths 1 and 2 was 9 ms, while for depths 2 and 3 the average propagation times were equal. The larger the depth of a prerequisite block is, the fewer times the prerequisite block was requested, as the block body was spread out in the blockchain network, and the average propagation time was correspondingly shorter. In the real situation, the depth of a prerequisite block is changed by the spread of the block in the blockchain network. However, the impact of the depth of a prerequisite block body on the block propagation time is small.

TABLE VI
PROPAGATION TIME OF BLOCK GENERATION NOTIFICATION BY R.

	$r = 0.1$	$r = 0.5$
50 %ile propagation time [ms]	274	242
90 %ile propagation time [ms]	453	418
100 %ile propagation time [ms]	634	601
Average propagation delay [ms]	302	269

TABLE VII
PROPAGATION TIME OF BLOCK GENERATION NOTIFICATION BY DEPTH OF A DEPENDENT BLOCK BODY.

	Depth 1	Depth 2	Depth 3
50 %ile propagation time [ms]	274	267	267
90 %ile propagation time [ms]	453	443	443
100 %ile propagation time [ms]	634	564	570
Average propagation delay [ms]	302	293	293
Requests number for prerequisite blocks	3802710	133445	118951

2) Comparison between each protocol: Figure 3 and Table VIII show the propagation delay for each protocol. In the proposed protocol, the parameter $r = 0.1$ and the depth of a prerequisite block is 1 and the propagation time of the block generation notification is measured. The propagation delay for the proposed protocol was the smallest in all percentiles, followed by Graphene and the CBR, with the legacy protocol having the largest propagation time. Compared to Grapheene, the propagation time of the 50 %ile, 90 %tile, and 100 %tile for the proposed protocol is 41.1 %, 39.2 %, 27.8 %, respectively, and the average is 40.8 %.

C. Network communication amount

Figure 4 shows the expected value of the total message size for the transmission of a block or a block generation notification between two nodes for each protocol, where the parameters of each protocol are shown in Section IV-A. For the proposed protocol, two separate message sizes are presented: a message for the block generation notification only, and an other message for the block and block generation notification. The message size of the block generation notification is important in terms of the fork rate. The expected total size of the messages between two nodes per hop is 7.3 KiB, with the smallest block generation notification of the proposed protocol, followed by Graphene with 13 KiB and the CBR with 43 KiB. Legacy protocols were the largest with 1 MiB.

D. Fork rate

The fork rate is calculated as $F = T_W/T$,where F is the fork rate, T_W is the average block propagation time weighted by the hash rate, and T is the block generation interval [13]. Table IX shows T_W and the fork rate calculated by each T_W.

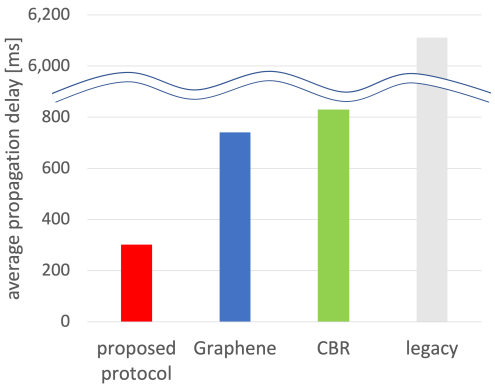

Fig. 3. Propagation time for each percentile.

TABLE VIII
PROPAGATION TIME.

	Proposed protocol	Graphene	CBR	legacy
50 %ile propagation time [ms]	274	666	746	6182
90 %ile propagation time [ms]	453	1154	1241	8633
100 %ile propagation time [ms]	634	2282	2367	15326
Average propagation time [ms]	302	741	828	6478

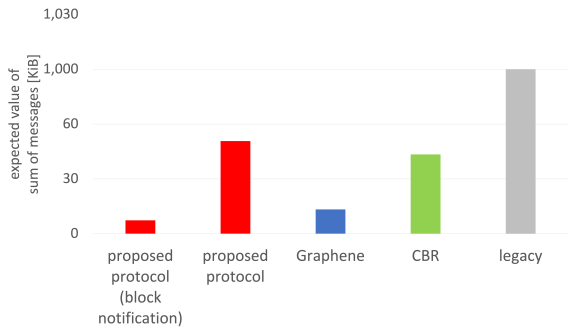

Fig. 4. Expected message size between two nodes per hop.

Figure 5 shows the fork rate. We measure the propagation time of the block generation notification for the proposed protocol. These results show that the proposed method had the best fork rate, which is 40.8 % of that of Graphene.

V. CONCLUSION

In this paper, we proposed a protocol for quickly propagating block generation notification using a Bloom filter in the blockchain network to reduce the fork rate. In experiments in which a simulator is used, the propagation time of the 50 %ile was 41.1 % of an existing protocol, that of the 90 %ile was 39.2 % of the existing protocol, and the fork rate calculated from the average propagation time is 40.8 % of the existing protocol. Although shortening the block generation interval increases the fork rate [2], it can be offset by suppressing the fork rate using the proposed protocol, so that the block generation interval can be shortened while maintaining the fork rate. Arakawa et al. [14] proposed such a method.

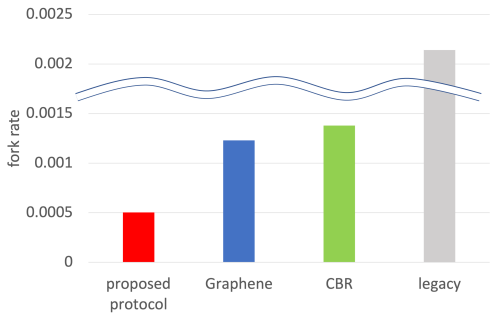

Fig. 5. Fork rate.

TABLE IX
FORK RATE

	Proposed protocol	Graphene	CBR	legacy
T_W [ms]	302	741	828	6478
F	0.000503	0.00123	0.00138	0.0107

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP21H04872.

REFERENCES

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.
[2] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*, pages 507–527. Springer, 2015.
[3] Matt Corallo. Bip 152: Compact block relay, 2016. https://github.com/Bitcoin/bips/blob/master/bip-0152.mediawiki.
[4] A Pinar Ozisik, Gavin Andresen, Brian N Levine, Darren Tapp, George Bissias, and Sunny Katkuri. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 303–317. 2019.
[5] Blockchain.com, (Accessed on 10/12/2022). https://www.blockchain.com/explorer/charts.
[6] Ryunosuke Nagayama, Ryohei Banno, and Kazuyuki Shudo. Identifying impacts of protocol and internet development on the bitcoin network. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2020.
[7] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[8] Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.
[9] Bitcoin visuals, (Accessed on 10/12/2022). https://Bitcoinvisuals.com/chain-input-count-tx.
[10] Kaylash C Chaudhary, Vishal Chand, and Ansgar Fehnker. Double-spending analysis of bitcoin. In *Pacific Asia conference on information systems*. Association for Information Systems, 2020.
[11] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. Simblock: A blockchain network simulator. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 325–329. IEEE, 2019.
[12] Ryohei Banno and Kazuyuki Shudo. Simulating a blockchain network with simblock. In *2019 IEEE international conference on blockchain and cryptocurrency (ICBC)*, pages 3–4. IEEE, 2019.
[13] Akira Sakurai and Kazuyuki Shudo. Impact of the hash rate on the theoretical fork rate of blockchain. In *2023 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4. IEEE, 2023.
[14] Masumi Arakawa and Kazuyuki Shudo. Block interval adjustment based on block propagation time in a blockchain. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 202–207. IEEE, 2022.