Reliable Inter-Blockchain Protocol for improving scalability

Keyang Liu, Yukio Ohsawa[1]

Abstract—**Scalability is an open question of the blockchain. Ongoing solutions, like Sharding and Side-chain, try to solve it within an independent blockchain system. Besides, a lot of systems try to used a blockchain beyond several blockchains to provide interoperability. By learning from these works, We propose a new inter blockchain communication framework which does not rely on a strong connection between blockchains. In this model, secure cross-chain operations are handled through a particular workflow and pegged to the blockchain. All blockchains' final statues are spread through the gossip channel so that any fork or reverse of the blockchain is denied by the network. Through managing interactions among blockchains, this framework can guarantee the interoperability and improve the scalability of all blockchains.**

1. INTRODUCTION

Blockchain, a solution to decentralized systems, can solve problems in fields like finance[Eyal 2017], supply chain[Abeyratne 2016], crowdsourcing [Li 2018]. Currently, blockchain can provide two functions. First, blockchain can work as a securely distributed ledger[Ren 2018] to storage data redundantly and correctly. Second, a blockchain can provide a reliable distributed calculating platform. With the help of smart contracts[Underwood 2016], All nodes of blockchain can execute task consistently and provide a robust service to all users with the same quality.

Generally, a decentralized system is more robust and trusted than a centralized one, in case of crash and byzantine faults. However, lack of scalabilities like latencies growth rate or superfluous messages caused by growing nodes limits the use of this framework. This weakness is a result of the consensus algorithm under different security assumptions[Karame 2016] and results in plenty of solutions trade-offs between scalability and security in the field of blockchain system designing. Sidechain shifts some payload into a quick response part for high frequent services[Back 2014] and confirmed the final result of these payloads in the future. Sharding tries to split payloads into several shardings for parallel processing and converge all final result into the mainchain[Luu 2016]. All these proposals sacrifice security or consistency for a part of the system for speed.

This work tries to solve the scalability problem through cooperative problem-solving. In this model, each blockchain system is an independent agent. Main contributions of this work are 1. A protocol for delivery versus payment(DVP) problem. 2. The framework of Blockchain's communication network.

2. RELATED WORKS

Sharding is an idea borrows from database designing that split the record into several shardings so they can be handled by different groups simultaneously. In a blockchain, sharding means requests are distributed to different groups for handling while the final result is merged by mainchain to keep consistency. Elastico[Luu 2016] and Rapid chain[Zamani 2018] are great implementations of this solution. In these systems, the randomness of each sharding limits the possibility of collusion and planned attacks. This idea shares some similarity with multiple blockchains system. However, its security largely relies on the randomness of shard members selection, which means collusion of one shard can affect the consistency of the whole system.

Chen et al[Chen 2017] and Kan et al[Kan 2018] had finished some works about the communication between different blockchains. They simulate Internet stack and TCP protocol to manage the Inter blockchain communication. Although these methods are useful, they are also fragile to possible collusions within or among blockchains. Besides, these works did not solve the Delivery versus Payment(DvP) problem between two blockchains which could be a target for the attacker.

Cosmos[Buchman 2016], Polkadot[Wood 2016] and other existed systems also provide some possible design for blockchains' interoperability. They both try to create the main blockchain, whose validators represent independent blockchains, to serve inter blockchain communications. For protecting the security, there can be some nodes outside of the system to monitor behaviors of validators. Although this function sounds quite applicable in the real world, it still remains an issue about the collusion of validators in the main blockchain.

In this work, we propose a framework, which adopts weak secure assumptions of each blockchain, to handle inter blockchain communications. By considering possible crash or byzantine faults in each blockchain, this work focus on restricting the result of attacks and keep availability of other systems after the attack. Our main contributions are 1. A blockchain to blockchain communication scheme solves DVP problem between two blockchain. 2. A blockchain network framework to control the result of possible failure in some blockchains so that they will not affect the availability of other blockchain.

3. PROBLEM CLARIFICATION

This part will clarify the problem model this paper focus on and the benchmark used to evaluate our result.

3.1. Problem model

This part will clarify the problem model this paper focus on and the benchmark used to evaluate our result.First, $[N] = 1, 2, ...N$ represents the set of systems. Each system is a distributed network that uses blockchain $B[]$ with all terminated blocks list linearly to maintain consistency among nodes. The position of a block in the blockchain called its height. Nodes of each system run a consensus algorithm to provide their services to its users. A Terminated block means at least $f > 0.5$ nodes have confirmed and stored the block in their local copy of $B[]$. f is the parameter of each system's consensus algorithm (e.g. $2/3$ in $PBFT$).

Second, A block in each system's blockchain contains at least two parts: header and body. A block header contains at least the hash of the previous Block, metadata of the block body, and signatures of its creator. For convenience, all block body's contents, like transactions or database operations, are named as an abstracted class – log. Hence, each system uses its blockchain to maintain the order of logs within the system to provide services to users.

Third, we assume there is a reliable communication channel between every two systems, which means for a system A and system B, at least f_A part of A's nodes can send requests and get responses from f_B part of nodes of the system B. As a result, system A can make a consensus about the state of all systems with a limited error. This reliable channel is important for building reliable communication between two blockchains for inter blockchain communication. Besides, there should be a gossip network[Van 2008] based on these channels. The gossip network is used for transporting view update messages that do not affect systems' function. The detail of the gossip channel will be introduced later.

Last, we split the job of a system into two parts. One part is its local task that verifies and adds logs to its blockchain independent to other systems. The other part is a cross-chain task, which in addition requires verifying a particular $log2$ in another system's blockchain before committing $log1$. Each system should work independently for its local task at any time and support all legal cross-chain task through inter blockchain communication. There is no guarantee that a system will not tamper their record forever, which indicates the security of cross-chain tasks may be undermined by another system. Tampering of another system's blockchain may finally affect some local tasks which are based on the result of some cross-chain tasks. This kind of attack is an unavoidable risk accompany to interoperability. Any inter blockchain communication scheme should consider this risk and design the method to reduce the possible damage caused by a malicious system and protect the correctness of local and other unrelated cross-chain tasks.

3.2. Benchmark

The target of this framework is solving the scalability problem for a multi-blockchain system. Hence we want to evaluate the performance of blockchains under different parameters. We evaluate a blockchain through two aspects: 1. the cost of availability 2.the cost of security.

the cost of availability means to finish a cross-chain task, how many times of cross-chain communication are needed? as said before, a reliable channel between two blockchains can be a costly multicast network. The use of this channel will affect the efficiency of each system.

the cost of security is another cost related to the gossip channel which is not necessary when all systems are reliable. However, if some systems are malicious, this cost can give all system the power to protect itself.

this work will first proof that its setting can guarantee both availability and security of cross-chain task and show the growth of these cost under different setting which indicates the scalability of the system.

4. PROPOSAL

This section includes the detail of our framework. To begin with, we defined two extended functions for each participant of agents in our model.

4.1. Notations

First, each node of a system A will maintain a view list for all other systems in the network that system A can access to. One view contains the information of another system's terminated blockchain current states, which is a recursively calculated hash, and latest k terminated blocks' hash and sufficient proof(e.g. f_A signatures from the system). one view can be generated by the system itself according to Algorithm 1

Algorithm 1 View generation

```
Input: Blockchain $B[]$
Output: list of hash
 1: $Result = [""]$
 2: $l$ = length of $B[]$
 3: for $i$ in [0,l-k] do
 4:     $HeaderHash = Hash(B[i].header)$
 5:     $Result[0] = Hash(Result[0]||HeaderHash)$
 6: end for
 7: for $i$ in [l-k,l-1] do
 8:     $HeaderHash = Hash(B[i].header)$
 9:     $Result.append(HeaderHash)$
10: end for
11: return Result
```

However, for a node in another system, it only needs to maintain $O(n * (k + 1))$ Hash value to monitor the whole network's latest status and verify any updates that less than k blocks for a blockchain. Whenever a message that contains view information is received, a node will use this message to update the existed view list. A system will broadcast his view list in gossip channel[Van 2008] once his view list is updated. The detail of this gossip channel will be clarified later.

Second, Each system should support two operations: $verify$ and $check$. $verify(log)$ will return whether the log can be added to the next block. Taking Bitcoin's transactions as an example, the input should be a subset of unspent transaction output (UTXO), and the sum of inputs should be no less than the sum of outputs.$check(H(log))$ returns the position of one log in B_i. It will return -1 when it does not exist.Formally speaking, for any system i, if $\exists h that log \in B_i[h], check(log) = h$, otherwise $check(log) = -1$. Hence, the following properties hold:

- a.(Validity) for any blockchain with maximum height h, if $verify(log) == True$, $check(H(log)) == -1$.
- b.(Agreement) In a system A, if $check(H(log)) > 0$, log can be accessed from at least f_A part of nodes.

$check()$ function is also an API of a system that allows other systems to check the existence of a log. When a system sends or replies a $check()$ request or to another system, it will send the answer attached with its view.

Second, we define a contract log for system A as $clog1 = H(log1)||H(log2)||H(log1')||H(log2')||B||h_1||h_2$. This contract represents $log1$ is a cross chain operation related to $log2$ in system B, where $H()$ is a hash function. The expiration height for this condition log is h_1 and h_2 in respect system A and B. The logical behind this contract is like this: $log1$ and $log2$ should be verified before height h_1 and h_2 in respect blockchain, if not, $log1'$ will be added to this blockchain which will reverse the achieved result of $log1$. Hence, we can easily create a reverse contract log for system B as $clog2 == H(log2)||H(log1)||H(log2')||H(log2')||A||h_2||h_1$. $verify(clog1) = verify(log1)$

Third, we define $log- > log'$ represents log is a precondition of log'(e.g. $clog1- > log1$,$log1- > log1'$). Define a state for called locked for log. If log is locked, for any log' that $log- > log'$, $verify(log') == False$. Each node in a system should keep all contract logs, which are not yet expired, in his local waiting list. Whenever a contract is in waiting list, its related log is locked. Such waiting list is used to help check validation of a process and timeout process introduced later.

4.2. Workflow

the workflow of cross chain operations are following:

- 1. Register: A user submits $clog1, log1, log1'$ to at least one node of the system A. nodes check $verify(clog1)$. If it returns True, commit $clog1$ to the next block and add it to the waiting list when the block is committed.
- 2. Check: Once a contract has been added, whenever a new block is about to be created, the proposal will send a request to the system B for checking the states of their contract. If it system B did not commit $clog2$, system A will wait till system B send check request for $clog1$ and then resend its check for $clog2$.
- 3. Pre-commit: when system B committed $clog2$, System A will check $verify(log1)$. Then commit $log1$ and lock it as soon as possible.
- 4. Terminate:After the height of blockchain reaches $h1$ and the view of system B reaches $h2$, node expire $clog1$ from waiting list and request $check(H(log2))$ and $check(H(log2'))$ to determine whether to commit $log1'$ or not. If $log1'$ need to be committed, a proof of SystemB's Hash state need to be used as a proof.

The workflow of a successful cross-chain task is shown in Figure 1. In Figure 1, commit is a process until the

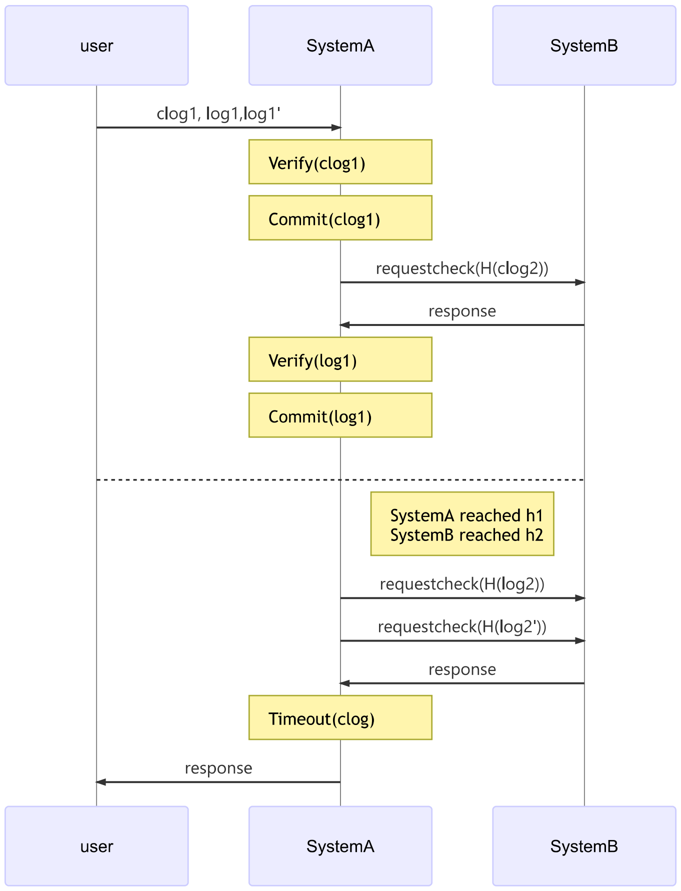

Figure 1: The work flow of one cross chain operation.

required log has been added to a terminated block. Since the cross-chain operation is independent of the local task, such a workflow can last for some time, to wait for the result to be verifiable. The timeout process is described in Algorithm 2

Algorithm 2 Function Timeout

```
Input: $clog$; $log1'$; $hc$ (result of $check(H(log2))$); $hc'$ (result of $check(H(log2'))$)
Output: None
1:  remove $clog1$ from waiting list
2:  $h = check(H(log1))$
3:  if $h < 0$ then
4:      return
5:  end if
6:  release Lock of $log1$
7:  if $hc < 0$ or $hc > clog[5]$ or $hc' > 0$ then
8:      $commit(log1')$ //attachs proof
9:      return
10: end if
```

Timeout is used to unlock $log1$ and finished the contract. Timeout will always be executed when both expired times reached. the correctness of this scheme is in the next section.

Till now, we have clarified the main steps of normal cross-chain operation. After the cross-chain task is finished (all timeout process for this height is finished) one system will broadcast its latest view list to all systems through the gossip channel. When a system receives such a gossip channel, it will find out all its local views that need to be updated. If the view list is updated, it will broadcast its newest view again through the gossip channel. if one node finds a system contains multiple different block headers for one height, it will broadcast the proof header to all systems. The system with conflict heights will be treated as two different branches, and other systems will only admit one branch to inherit the identity of the system. The rule of selecting branch based on which branch has the most cross-chain task according to existed contracts.

4.3. Communication rule

Two things related to the proposal need to be clarified further, one is how to do communication between two blockchains while the other one is how to make use of gossip channel. Assume the case is one system A sends message Msg to another system B with a size of nodes s_A and s_B. For security reason, the easiest method is a full connection between two systems' all nodes. That is to say, all nodes in system A need to send Msg to all nodes in systems B until system B's at least f_B nodes received a $f_A * s_A$ identical copies of Msg for confirmation. This method cost at least $2(f_A * f_B * s_A * s_B)$ and at most $2(s_A * s_B)$ times node to node communications for one-time request and response. As an improvement, send a request does not need that much confirmation. Hence sending a request can just cost $(1-f_B)*s_B+1$ times to make sure the request will be received by at least one perfect works nodes. On the other hand, both sending messages and receiving messages can be done by one node of system A because all messages can be verified through signatures. For say, when node A can be the proposer of the next consensus, the proposer of the current block, or even a node chosen by the user can send requests to the nodes in the system B and distribute received results to peers in the system A. Hence the minimum times of node to node communication are $s_B + 1$ and at most $2s_B$. Furthermore, if any node in system A is linked to system B's all nodes or its links are not hijacked by crashed or byzantine nodes, the communication times can be further decreased. For a given small possibility p, a node only needs to see m_B same confirmations without objection to believing such result has been confirmed by all honest participants. m_B should satisfied $(1 - f_B)^{m_B} < p$. Hence, $m_A \geq \frac{\ln(p)}{\ln(1-f_B)}$ is necessary for believing such response is not forged. Hence the minimum required result can be decreased to $(1 - f_B) * s_B + 1 + \frac{\ln(p)}{\ln(1-f_B)}$.

Gossip network is another thing really important in the system. It used to transfer view updates message so any possible fork will be detected during handling the cross-chain task. The quality of the gossip channel affects the design of the view list. Gossip network is a relatively low-cost network based on a reliable network. This is because the message goes through the gossip network has the lowest priority compared to any other messages. For a gossip channel between two systems, each time message sending is a kind of broadcast to a subset of systems from the systems that this system can be achieved. By using k, we measure the maximum gap of a view's height between the received message and local view lists. This gap is important due to the latency of $gossip$ channel and the size of the view update message. Once the gap of view's height is above k, a big inconsistency is detected in the network. The node of a system who is in advanced should immediately broadcast its view list to other systems and other systems who received these new view lists need to communicate with the source system of the view through a reliable channel for details of missed views and verified them. Similarly, systems can require the temporal leader of consensus ta send the view update messages to gossip channel. As later proof shows, this method makes send view update messages faster and significantly reduce the cost of exchanging messages with the cost of some security. Besides, view updates only need to happen when a new block is created, and the new view list just broadcasted within the system. Because these update messages do not require to be a real-time nor they need a response, so the chosen leader only need to send the messages to a limited number of random nodes while the intrasystem transportation of these messages can be done at the same time of making consensus. Hence the cost of one time sending a view update message to system A through gossip channel is at most $(1 - f_A) * s_A + 1$.

5. ANALYSIS

This section tends to provide the correctness and security proof of our method.

5.1. Function analysis

In the beginning, a user needs to send $clog1, log1, log1'$ to system A which are the logs may be committed by the system. By checking $log1$ and $log1'$'s hash and verify the validity which is the precondition of $clog$, system A can directly commit $clog1$. Second, After committing $clog1$, system A will send a check request to B, B has not committed $clog2$, system A will wait till system B, it will wait till system B send such check request and do check request again. Hence, By three times request, system A and system B can make sure they are all registering the contract to their blockchain. Then, they can verify and commit the $log1$ and $log2$ to their blockchain and locked till confirmation. contract log must be commit before committing normal log because contract log can guarantee both system's nodes understand $log1$ and $log2$ are special log for cross-chain communication and they must be locked till expiration time. Contract log also define the condition of committing $log1'$ and $log2'$ to reverse the result of $log1$ and $log2$. If expiration happened before committing $log1$. The contract will expire so the system stops handling that contract. Hence this step guarantees $clog1->log1$, $clog1->log2$, $clog2->log1$, $clog2->log2$.

After committing $log1$, the system only needs to wait until $log2$'s committing. When both expiration time is reached, which can be checked by view list, system A will begin to check the if system B has finished the contract. Since both systems have reached the expiration time, so the result of the contract cannot be changed anymore. According to the contract, if $log2$ exist and $log2'$ is not committed, system B's responsibility has been finished so system A can finish the contract by unlocking $log1$. Otherwise, $log1'$ will be committed to reverse the result of the contract. Because all blocks before $h1$ and $h2$ are all being terminated, the result of the contract is determined. In normal case, $log1'$ should not be committed before time out. However, system B is a BlackBox for system A which means system A has no guarantee about the logic of system B. System A can only believe the terminated blockchain will not be changed and the result of check function with enough proof. Hence, $log2'$ should be checked during the time out process and the view of system B that can prove such a result will be broadcast too all other systems.

As a conclusion, this algorithm guarantees the following conditions: 1. $clog1$ and $clog2$ is committed to the respect terminated blockchain before committing the real log. 2. $log1$ and $log2$, if committed, will be locked before both blockchains reach the committing time. 3. After both systems' expiration time, the final result of the contract will be determined and the contract will be completed by removing the lock of $log1$ and $log2$. If necessary, the commit of $log1'$ and $log2'$ claim the failure of this contract. 5. In normal case, only 4 logs will be committed for two systems. The success case (e.g. $clog1$,$clog2$,$log1$,$log2$) and fail case (e.g. $clog1$,$clog2$,$log1$,$log1'$)

5.2. Security analysis

This part provides some brief proofs of security for two kinds of attacks: the byzantine system and byzantine nodes' cross-system collusion. For a byzantine system Eve, since it can provide arbitrary messages to the system, we can assume system A will commit $log1$ and finish contract as Eve wish. This can be done by two ways: first, the Eve admit the contract and related log into its blockchain, after the contract is finished, Eve create another branch and claim the previous branch is controlled by minority nodes that cannot represent the consensus of Eve. This is possible when System B believe $\frac{\ln(p)}{\ln(1-f_{Eve})}$ can represent the system while they are all collapsed. However, because gossip channel will send the new view back to Eve's at least one up-to-date node, conflict should be revealed once an up-to-date received it and the node will directly broadcast the conflicts to all nodes that are connected to as soon as possible. Assume there are np systems linked to system Eve, and rate of sending view update message is q', that is saying a system will send view update messages to q' part system it can link to. Since gossip channel will eventually update all system's view list, hence view update messages will be sent to $(1 - f_{Eve}) * s_{Eve} + 1$ node of the system Eve by np' system for at least twice chance. Hence, Eve's claim has a chance of $(1 - q')^(2np')$ to be true. Given a small number λ, the minimum q' is $1 - \lambda^{\frac{1}{2np'}}$. p' evaluates the expected distribution of gossip channel between systems, hence we can say q' represents the effort systems need to put for guarantee the security at level λ. It is reasonable to say if more connections are built among systems, fewer view update messages are necessary for each miner to send, while the malicious system cannot deny its misbehavior.

Second, Eve may create a branch in advance and add the cross-chain task to that branch while keeping the main chain updates for other views. This requires system A should not update its views with other systems. Since we assume the connection between systems are not collapsed, system A should be able to receive and send view list update messages with other systems which will show the fork of Eve's blockchain and Eve cannot deny its misbehavior. Hence, we can say if gossip channel is secure enough, a malicious system cannot deny its malicious behavior and its new branch that violate the cross-chain rule will be not accepted to inherit its identity by all other systems.

Another possible risk is system Eve make collusion with some nodes in system A. Other than the previous case, this time, system A may cooperate with Eve in some degree for say not sending view lists update during its operation. By carefully controlling the system A, we can assume system A cannot send view update messages to other systems. However, because system A's list node view is old, they can only update their view list by receiving view update messages from other nodes. If the control of the gossip channel is long enough, it is possible for Eve to finish the contract and let other systems to admit the result of another branch of Eve. Assume Eve use branch b_1 for cross-chain task, and b_2 is the branch used to replace b_1. If b_2 is reveled in some view update messages, before the finish of cross-chain task at b_1. The view of b_2 has a possibility to be sent to honest nodes of system A which will help conflicts to be detected and found by all other systems. Hence, it is important to keep that b_2 a secret before the finish of cross-chain task on b_1.

However, it is possible that b_2 is revealed immediately after a cross-chain task time out at b_1. Once the conflict is detected, all other systems need to determine which branch can inherit the identity of Eve. If there is no cross-chain task on b_2, b_1 should be admitted as the real branch of Eve, and all systems reject the view of b_2. However, it is possible that another $clog3$ is registered before the fork of b_1 and b_2. This case makes the reveal of b_2 can trigger the time out of $clog3$ as soon as b_2 is revealed. To solve this problem, the expiration time of a contract needs to be set carefully. If more than $1-f_A$ part of nodes in system A join the collusion, system A can also be treated as a malicious system that the result of the branch doesn't matter to other systems. Hence, the expiration time for a contract can be long enough so that an honest node can send the gossip channel to other systems before the contract finished. However, this method will significantly reduce the availability of the system. Another way is to use the property of the gossip channel. The view list of one node can be updated if a system's node finds received view list messages of the system itself is too old, such node can directly send gossip channel to other systems by assuming the leader nodes' gossip channel is unavailable. We can find the consistency of view list among all systems is very important to detect attacks of malicious systems or collusions.

Other than byzantine players, there are some network attacks like Sybil attacks and DDOS attack among systems. Sybil attack means the attacker create several systems in the model to arrange attacks. However, these bot agents need to spend enough resources to convince users of other systems for one round attack. Hence, this attack is not profitable if users can manage their risk. Another way is isolating one system like Man in the Middle(MITM) attack to block gossips as said before. This attack is very costly when the target is a distributed network. Fixed and reliable channels can also resolve this attack. In a word, the resilience against manufactured identities depends on the value of these systems for other systems. Lastly, DDOS attacks also worth considering. Attackers can create many contracts to delay the process of making consensus and increase the burden of the gossip channel as well as the reliable channel. The solution can be charging an additional fee for registering cross-chain tasks. Indeed, cross-chain operations require extra payment for stronger termination and complex procedures. The most significant problem is redundancy information spread through the gossip channel which is important and costly. Hence, the gossip channel needs some rules for filtering received packages and improve broadcast strategy. systems can require a signature for each message, limit the frequency of view update messages for each system, or do these updates periodically.

6. EXPERIMENT

This work intended to improve the scalability of one agent through the cross-chain operation. The experiment

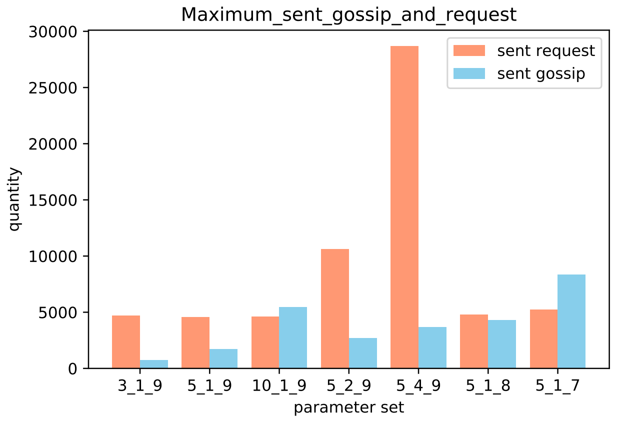

Figure 2: The graph of requests and gossips sent by a system, compare the change of scale, cross chain task rate and rate of sending a gossip.

evaluated the average gap of view lists to real cases and the maximum gossip and request($check()$) to different networks. In this experiment, we assume all systems create blocks with a speed of 0.1 seconds per block, and each system will create 10000 blocks. The setting parameter including three digits: the scale of system(n), rate of cross-chain tasks for a block, and rate of not send a view update message($1-q'$). For example, 3_1_9 represents the network with three nodes and 10% blocks contains a contract log while view update message will be sent to each node with possibility 1%. Gossip channel is simulated by a broadcast channel follow the rule of gossip algorithm. As previous analyses had shown, the function of our framework relies on request on the reliable channel and the security of our work is based on messages of gossip channel and the gap of view. Hence, we evaluate the effects of change of parameters to evaluate the scalability. With a lower gap of view, the system will be more secure while lower request and view update messages will reduce the cost of this framework.

Figure 2 shows the result of sent requests that related to cross-chain tasks. Generally, one success cross-chain task requires at least 4 times cross-chain requests. However, due to network latency and failed requests, it is possible that some requests failed. In our simulations, the network latency is dismissed, hence such result can only be the problem of the system itself, that is saying too many cross-chain tasks will delay the system and may cause more requests failed. In the experiment, the redundant messages rate increase from 25% to 50%.

As for the gossip channel, the scale will cause a big change to the required view update messages number that is growing faster than a linear function. The rate of cross-chain tasks per block, which leads to more view updates messages, only increase a slight part of updates messages. This result indicates the increasing sending view update messages will not significantly affect the volume of view update messages. Lastly, an obvious result is that increasing rate of sending view update messages will significantly increase the volume of view update messages which follow a linear relation. Figure 3 reflects the distribution of gap between view lists

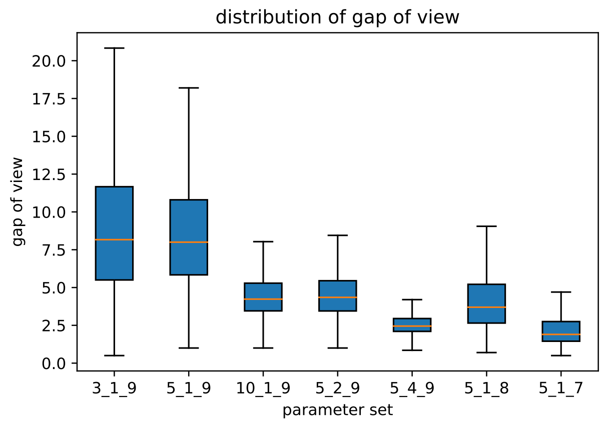

Figure 3: A distribution of gap of views for systems, compare the change of scale, cross chain task rate and rate of sending a gossip.

and real cases. According to the result, we can find the gap decreased as the scale goes larger which is generally the scale will increase the transporting view update messages in the gossip channel. As the network becomes larger, the gossip channel is easier to synchronize all systems' view lists. One interesting result is the increase of cross-chain tasks results in both the expectation but also the variance of gap decrease. This is because each cross chain task will broadcast more up-to-date information than a forward view update messages.

By comparing Figure 2 and Figure 3, we can find even a very limited view update messages are sending for each system the average gap of view can be very slow, while such gap shows how to determine k for view update message and the length of expiration of the system. In the simulation result, with less than if system increase the rate of sending view update messages like sending on average 1.2 systems each time, with less than one message per block(5_1_7 case), the expected gap will be less than 3 and k can be set to 5 for the worst case.

7. CONCLUSION

This work proposes a framework for improving the scalability of multi-blockchain systems through Inter blockchain communication. Under our framework, cross-chain tasks can create externality of each system and spread their security guarantees through the gossip channel. The security of cross-chain tasks can partially rely on other systems so one system can pay more attention to achieve agreements for normal workflows. On the other hand, cross-chain operations extend the ability of one system and allow a more flexible exchange between different systems.

This work can be optimized in many ways. Contract log and view update messages can be compressed to reduce latency and burden of gossip channel. A checking protocol based on gossip channel like heartbeat can be revised to give a better solution for detecting forks. The future works will focus on the consensus design under this framework, and optimization of the gossip channel as well as related protocols.

8. ACKNOWLEDGMENT

This work was funded by JSPS KAKENHI, JP16H01836, JP16K12428, and industrial collaborators.

References

[Eyal 2017] Eyal I. Blockchain technology: Transforming libertarian cryptocurrency dreams to finance and banking realities[J]. Computer, 2017, 50(9): 38-49.

[Abeyratne 2016] Abeyratne S A, Monfared R P. Blockchain ready manufacturing supply chain using distributed ledger[J]. 2016.

[Li 2018] Li M, Weng J, Yang A, et al. Crowdbc: A blockchain-based decentralized framework for crowdsourcing[J]. IEEE Transactions on Parallel and Distributed Systems, 2018.

[Karame 2016] Karame G. On the security and scalability of bitcoin's blockchain[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 1861-1862.

[Back 2014] Back A, Corallo M, Dashjr L, et al. Enabling blockchain innovations with pegged sidechains[J]. URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains, 2014.

[Luu 2016] Luu L, Narayanan V, Zheng C, et al. A secure sharding protocol for open blockchains[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 17-30

[Ren 2018] Ren Z, Cong K, Aerts T, et al. A scale-out blockchain for value transfer with spontaneous sharding[C]//2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, 2018: 1-10.

[Chen 2017] CHEN Z, Zhuo Y U, DUAN Z, et al. Inter-Blockchain Communication[J]. DEStech Transactions on Computer Science and Engineering, 2017 (cst).

[Buchman 2016] Buchman E. Tendermint: Byzantine fault tolerance in the age of blockchains[D]. , 2016.

[Wood 2016] Wood G. Polkadot: Vision for a heterogeneous multi-chain framework[J]. White Paper, 2016.

[Kan 2018] Kan L, Wei Y, Muhammad A H, et al. A Multiple Blockchains Architecture on Inter-Blockchain Communication[C]//2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2018: 139-145.

[Zamani 2018] Zamani M, Movahedi M, Raykova M. RapidChain: scaling blockchain via full sharding[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2018: 931-948.

[Underwood 2016] Underwood S. Blockchain beyond bitcoin[J]. Communications of the ACM, 2016, 59(11): 15-17.

[Van 2008] Van Renesse R, Dumitriu D, Gough V, et al. Efficient reconciliation and flow control for anti-entropy protocols[C]//proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. ACM, 2008: 6.