

Design Issues in Permissioned Blockchains for Trusted Computing

Wei-Tek Tsai
National Key Laboratory of
Software Development Environment,
School of Computer Science and
Engineering, Beihang University,
Beijing, China
tsai7@yahoo.com

Xiaoying Bai
Department of Computer Science
and Technology, Tsinghua
University, Beijing, China
baixy@tsinghua.edu.cn,

Lian Yu
School of Software and
Microelectronics, Peking University,
Beijing, China
lianyu@ss.pku.edu.cn

Abstract

A permissioned blockchain (BC) is a secure distributed ledger maintained by a number of trusted validation nodes. However, a validator may become compromised and send inconsistent messages to different nodes. To counter the problem, consensus protocols like Practical Byzantine Fault Tolerance (PBFT) can be used. The paper presents a permissioned BC system and discusses various design issues including control messages, block size and window size of block creation time. Furthermore, it is necessary to identify those compromised nodes discovered during the consensus protocol. This paper then proposes a reputation system to track the trustworthiness of nodes as it is necessary to distinguish failed nodes from compromised nodes. A compromised node may send inconsistent messages to others or accuse other nodes compromised, and it should be removed as soon as it is identified and confirmed.

Keywords: Permissioned Blockchain, Byzantine Fault Tolerance, Reputation Systems

1. Introduction

Blockchain (BC) has received significant attentions recently as major banks are exploring its applications. While initially a BC is implemented as a public ledger where any node can join and participate in the decision process of block creation as it is done in Bitcoin [2], more recent implementations of BCs are private ledgers where new blocks are decided by trusted participating nodes. The former approach is called public BC, and the latter permissioned BC. Permissioned BCs have been proposed to be used in banks, insurance, copyright management, digital IDs, and computational law, thus it is important that the data stored should not be compromised easily as the data may address financial and legal aspects.

A permissioned BC needs trusted parties in creating new blocks and maintains the associated ledger by majority voting mechanisms. However, a validation node may be compromised, known as the Byzantine Generals Problem (BGP) [4]. A compromised node may send different messages to different validators thus preventing the system from reaching an agreement. Public BCs can use proof-of-work (POW) or similar mechanisms like proof-of-stake (PoS) to prevent attacks. In a permissioned BC, as only trusted parties will be involved in decision making, the number of participants need not be large comparing to public BC systems. This paper uses Practical Byzantine Fault Tolerance protocol [5] to vote that can sustain 33% of nodes compromised.

In addition to consensus agreements, as the number of participating nodes increase over time, it is necessary to identify and rule out the compromised nodes. One issue of identifying compromised nodes is that a node may just fail temporarily due to message lost, message delay, or delay to computation such as I/O operations, and these need to be distinguished from malicious nodes that send out inconsistent messages to compromise the whole operation.

A reputation system can identify those good parties from bad ones, and it has been used in social networks and online e-commerce websites. Various algorithms have been proposed to characterize different aspects of reputation. A reputation system can be applied to a permissioned BC to identify those malicious nodes as well as those nodes that had temporary problems. The paper proposes a schema for rating nodes' reputation. A node will lose reputation if it fails to send consistent messages to others and agrees with majority decisions. Reputation, as a sign of a node's dependability, can be used to rule out a node when its rating drops below certain threshold.

The paper discusses various design issues of a permissioned BC and reputation systems. This paper is organized as follows. Section 2 presents the

assumptions of the model; Section 3 discusses design issues of a permissioned BC; Section 4 proposes a reputation system; and Section 5 concludes this paper.

2. Assumptions and Model

This paper makes the following assumptions.

Trusted Validators: Only trusted validators or nodes can send messages in a permissioned BC, and each message may be encrypted with digital signatures. Those sources, while trusted, are still watched over carefully by all the nodes in the permissioned BC. Furthermore, depending on the application, each node has application-specific validation checker to ensure that the messages transmitted are valid. For example, if a BC is used to store lottery tickets, if two different sources sent in the same lottery ticket number, this indicates potentially one or more parties may engage in a fraud as each lottery ticket can be issued once only. While each node is a trusted party that can engage in BC activities, but the trust is not permanent and will be dynamically watched by other nodes.

Sufficient Computing Capability: Each node is a server with multiple CPUs and each is capable to store the entire BC and perform common BC operations efficiently.

Sufficient and Dedicated Communication Bandwidth: Each node is connected to every other node via a high-speed network. The network may be a broadcast model such as Ethernet where each attached node can listen to the broadcast message, or the network can be linked. To ensure security and safety, these communication channels are dedicated to communication in the permissioned BC only.

Secure Mechanisms: Each node as well as the communication channel will be equipped with security mechanisms including firewall, encryption, policy enforcement, and virus detection. All messages transmitted are encrypted with digital signatures with timestamps.

Asynchronous Message Arrivals from Multiple sources: The permissioned BC may receive data from multiple external sources, and messages may arrive in different order at these nodes at different time, and messages may be lost. For security, messages may be encrypted with digital signatures; for reliability, common network protocols such as Sliding Window protocols and checksum can be used to ensure that from a given pair of sources and destinations, messages

will arrive in the right order. However, as data may come from multiple sources, thus nodes may receive messages in different orders. One way to address this problem is to have a common reception node that receives all the messages from all the sources, and dispatch them to each node in the permissioned BC. However, this approach may not be desirable as this may become the single failure point for the permissioned BC.

Another solution is to have multiple reception nodes and each can receive messages and dispatch to each node in permissioned BC, however, this will require synchronization among these reception nodes. This assumption is needed so that permissioned BCs can be used for public trust. For example, if one uses a permissioned BC to validate lottery tickets, lottery ticket information can be transmitted from lottery machines everywhere to the BC, and as hundreds of thousand machines may be sending messages to the BC, each node will receive messages in different orders.

Independent Assumption and Constant Monitoring:

Each node behaves independently from other nodes in a permissioned BC, and each will watch over the behavior of other nodes constantly, and alerts the rest of nodes in case it identifies a node is behaving strangely.

Chinese Wall: All the people and systems involved in a node in a permissioned BC can communicate with other nodes in a strict manner. Specifically, only those specified in the permissioned BC protocols can be used, and no other communication such as emails or phones can be communicated among people operating on different nodes. This will prevent any propagation of damages or collaborative sabotage of the permissioned BC among participating nodes.

Cooperation: Other than those restrictions from the Chinese Wall and independent assumption, the nodes participate in a permissioned BC in a cooperative manner and engage in block creation, voting, dissemination of votes collected, and reputation computation. In case that a node has crashed temporarily and later recovered, other nodes can send out BC information to the recovered node so that it can resume operations as soon as possible. To ensure that the recovered node receives the right information, it may request the BC information from all working nodes, and verify the information before storing into its own databases.

3. Byzantine General Protocols

BGP protocols such as PBFT provide consensus mechanisms in permissioned BCs. A BGP protocol allows the network to detect the behavior of malicious nodes, and all working nodes can reach a consensus if the number of compromised node is less than one-third of the total number of nodes, or $n \geq 3f + 1$ where n is the number of nodes and f is the number of compromised nodes.

PBFT has three phases, and each node maintains its own view of the network. The protocol is initiated by a leader in the network. Even if the leader failed, other nodes take over the responsibility and resume the operation. A simplified 3-phase process is as follows:

- A leader sends out *pre-prepare* message to all the nodes;
- Each node received the message will produce a response, and send the response, i.e., *prepare*, to every node;
- After receiving sufficient such as $2/3$ prepare messages from the nodes, each can commit to the agreement, and send out the *commit* message to everyone.

Thus, to reach a consensus, there will be three rounds of messaging, one for each pre-prepare, prepare, and commit message. The PBFT is more complicated as it also handles the case when a leader is crashed.

In permissioned BCs, a Byzantine consensus will be needed only when a new block is to be created, and all working nodes need to agree with the contents of the new block. However, for an enterprise system, due to asynchronous message arrival and large volume of data, each node may receive different messages since the previous block creation, and thus it can be difficult for all the nodes to agree with all the contents of the new block even if the nodes are working properly. If the system has few transactions per second, this is not an issue, but this is an issue for an enterprise system. Thus, the goal of consensus protocol is modified as follows:

- 1) Most of working nodes can reach a consensus on most of items in a new block;
- 2) Detection of those failed nodes so that they can be safely ignored during consensus building;
- 3) Detection of those compromised nodes that sent out different messages to different nodes to disrupt the operation;
- 4) Those items that should be in the BC, but not voted in the current block, will be placed in the next block to be voted in the future;

- 5) Those items that should not be in the BC should be clearly identified and should not be placed in a queue to be considered in the next block.

Issue 1 is a relaxed version of a common BC. Instead of having identical contents, one can allow each node to have slightly different contents in a new block yet to be voted on. This is to allow each node to receive different messages from different sources due to asynchronous messaging. However, as a part of the voting process, all working nodes must decide on the contents of the new block. In this way, after voting, all the nodes will have identical content in the newly created block.

Issue 2 is important in a permissioned BC as failed nodes should not be involved in future voting until they are fully recovered. The recovered node also needs to retrieve any missing data during the absent period.

Issue 3 is important as compromised nodes should not be involved as a compromised node can cause significant troubles for a BC. In fact, they should be excluded from the BC operation completely as soon as they are identified as the damage done by compromised nodes is far greater than the damage done by failed nodes.

Issue 4 is a new problem in a permissioned BC as each second numerous data may enter the BC, and those valid data must be stored to maintain the trust of the BC. For example, a lottery machine may send out ticket information to a permissioned BC. Due to message delays, some messages did not get into every node, and they were not included in the new block. As they are valid tickets, they should be stored in the next block. This scenario is considered as *false negative* as a valid message was excluded due to various reasons.

Issue 5 is a new problem due to Issue 4 as data not included in the previous block can be included in the next new block to be created, thus potentially an increasing number of data can be postponed and continued to be considered for inclusion. But some of these data may not be valid at all, and should be removed for good rather than to be re-considered in future blocks. This scenario is considered as *false positive* as invalid messages continued to be considered in block creation.

A. Communication Consideration

All messages originated by participating nodes in a permissioned BC are considered as control messages. A control message will have a priority higher than data messages, and it will be sent over the communication channel even if other data messages are already on the queue to be transmitted. Furthermore, each node will handle control messages first before processing data messages. Typical control messages are:

- All the voting-related messages;
- Alert messages to signal the identification of a compromised or failed node;
- Those messages sent from working nodes to a newly recovered node for missing data.

B. Determining the Contents of a New Block

As each node may receive different messages and a large number of messages may have arrived, it may be difficult for all the nodes to agree on the contents of a new block.

Assuming that all the working nodes agreed on the previous block, and each node will hash all the messages collected with identical hash function. During the voting, each node broadcast the hash values of all the content of the new block. Once received all the results from every node, assuming more than $3f+1$ working nodes agreed on $X\%$ of the contents, where X is a high number between 95 and 100, each working node can select only those contents that are covered by all the working node in the new block. Those contents that are not selected will be queued to be included in a future block.

C. New Block Creation Time

Many proposals have been made regarding to the block creation time. For example, in Bitcoin, each new block will be created every 15 minutes; other cryptocurrencies have other block creation time. Some of these have fixed creation time.

In general, if the time between block creation is large, potentially more data will be entered into a new block and thus the block size needs to be larger. But it will take more computing to process a large block. On the contrary, if the time is short, the block size can be smaller. However, the system will spend proportionally more time in communication to engage in voting and alerting. Thus, this is a tradeoff between computation and communication. To have a long time in-between two block creations, the system will need to have

larger blocks, and spend more time in computing. To have a short time in-between, the system will spend comparatively more time in control messaging, but each block will be smaller.

For permissioned BCs, as a large number of data messages may be generated, but messages may arrive at different rates at different time. For example, for a lottery system, messages may arrive at a high rate during a sport event or hours before drawing of a lottery, and then the rate will drop after these events. Furthermore, certain hours in a day will have more messages than those quiet periods.

Thus, one possible solution is to have an adaptive window for block creation as well as an adaptive block size. During a high period, the window size for in-between block creation will be smaller and the block size can be increased. But during a quiet period, the window size can be increased to wait for more messages, and the block size can be smaller.

One can design such a system by changing one parameter, e.g., changing the window size without changing the block size as the node may have optimized algorithms to handle blocks of certain size, and thus not suitable to process blocks of different sizes.

Other considerations are also possible. For example, a BC system may prefer low delay rather than high throughput, and depend on load balancing mechanisms to handle large throughput. Such a load balancing mechanism is possible with ABC/TBC structure [8].

4. Reputation Systems

The goal of a reputation system is to estimate the trust of each node, and the trust is determined by the behavior of each node during voting. The reputation system can be performed as a part of the consensus protocol, and it is carried out at each participating node. The reputation is computed independently, and can be synchronized from time to time together with data messages.

To add a new block, validator nodes in a permissioned BC vote following the PBFT protocol. While voting, each node broadcasts its decision and a bitmap is built at each node to record the received decisions from others. The voting records are broadcasted as well. The reputation system compares the records from each node against each other and rates the reputation of each node following the scoring schema.

For example, if the 3-phase PBFT consensus protocol is used, reputation system will be incorporated from Phase 2 during the voting process.

Phase 2: Each node will broadcast its own voting decision, and each node collects the votes from every other nodes.

In this phase, consensus can be reached assuming that each needs 2/3 majority to reach a consensus. Assuming less than 1/3 of nodes were compromised or failed, then all working nodes will reach a consensus at this stage. Furthermore, all the nodes that either did not respond or returned a different answer can now be labeled.

- Those that failed to deliver the message may be caused by the delay in communication, message lost, or the concerned node has crashed temporarily;
- Those that provided different answers to different nodes are compromised, and they cannot participate in decision making until they have been manually determined to behave correctly.

Phase 3: Each broadcasts its results collected from every other nodes. Those working nodes can exchange their results to see how nonworking nodes might have behaved:

- If none of the working nodes received a message from a certain node, the node is labeled crashed or compromised. The node may be able to return to normal operation it passes more tests;
- If some of working nodes received a message from that node, while others have not, and the message contents are the same. The node may be still operating but message was lost, or it was compromised;
- If working nodes received different decisions from a certain node, it is compromised.

A reputation system can be specified as follows to capture the essence of the discussions.

Let $R_i(t)$ be the reputation of a validator node N_i in the permissioned BC after the t_{th} round of voting. Then

$$R_i(t+1) = \begin{cases} 0 & \text{if the node sent out inconsistent data to different nodes.} \\ xR_i(t) & \text{if it disagreed with the majority.} \\ \frac{y}{m}R_i(t) & \text{if it sent consistent messages to some nodes only, and agreed with the majority.} \\ (1-z)R_i(t) + \frac{n}{n+1}z & \text{if the node agreed with the majority.} \end{cases}$$

where

- $0 < x < 1$
- $0 < x < y < 1$
- m is the integer of consecutive rounds of missing messages, $m \geq 1$
- n is the last sequence number of consecutive agreements, $n \geq 1$
- $0 < z < 0.05$

Losing reputation: A node can lose its reputation in three ways

- It will lose rating if its messages were lost;
- It will drop significantly if it disagrees with the majority; and
- It will lose all of its reputation if it sent out different messages to different nodes.

Gaining reputation: $R_i(t)$ grows slowly if it agrees with majority. For a large z , a node can regain its reputation quickly, but a small z , it can regain its reputation slowly. Additional mechanism may be installed. For example, for a node compromised, an explicit manual instruction from the security officer will be needed to start to regain its reputation.

Initially, for all i , $R_i(0) = 0.01$; and the algorithm will update $R_i(t)$ only if $R_i(t) > 0$. Thus, each node needs to earn its reputation via majority voting and it will be trustworthy if it reaches 99.99% reputation, anything below 60 means that it may have been compromised, and anything below 99.99% means it is not trustworthy.

Identifying Compromised Nodes

Previous discussions need a mechanism to identify a compromised node, and this subsection addresses this issue. The issue is complicated because a compromised node may behave in an opportunistic manner, and making identification difficult. However, one can still distinguish three different kinds of nodes: 1) working nodes; 2) failed nodes; 3) compromised nodes.

Working nodes will continue to work correctly until they fail or get compromised; Failed nodes will stop working until they are repaired; Compromised nodes will lie from time to time in an opportunistic manner. A compromised node cannot return to normal operation until the system is thoroughly checked manually. These are stated as follows:

Working Node Assumption (WNA): A working node will produce correct messages, even though messages sent may be delayed or lost, but the content is correct. A working node may become a failed node or a compromised node at any time.

Failed Node Assumption (FNA): A failed node will stop working and will not generate any messages. A failed node does not generate random messages to different nodes, if so, it is considered as a compromised node instead. A failed node can return to operation after it has been checked for sanity.

Compromised Node Assumption (CNA): A compromised node may lie from time to time and may behave correctly for a long time until it decides to strike. Furthermore, a compromised node can issue inconsistent message during any phase of PBFT. Also, a compromised node may strike in only one phase of PBFT, while behave correctly in other phases. A compromised node cannot return to operation easily.

Furthermore, each message is sent with digital signature proving the ID of the sending node. Thus, identities of sending nodes can be known.

PBFT has three phases, but the detection of compromised nodes will occur only in Phase 3. A compromised node may act in the following manner:

- 1) As a **Compromised Sending Node (CSN)** by sending out different voting results to different nodes during Phase 2. A CSN initiated the deception.
- 2) As a **Compromised Receiving Node (CRN)** by reporting modified content during Phase 3. In other words, a CRN wrongly accuse a working node lied. It may correctly identify a CSN at the same time as it may choose to lie in only parts of messages.

Thus, one has four scenarios to consider. A compromised node can lie either in Phase 2 and/or Phase 3. For example, a compromised node can act as a CSN in Phase 2, but act normal in Phase 3 as in case 3 in the table below. At the same time, it can act normal in both Phase 2 and Phase 3 in a specific run of

PBFT; or it can act as a CSN in Phase 2 and a CRN in Phase 3 in the same run.

Scenarios	1	2	3	4
Phase 2	Normal	Normal	CSN	CSN
Phase 3	Normal	CRN	Normal	CRN

But, a working node will never be a CSN or CRN at any time until it is compromised. During a PBFT run, one has three cases to consider:

Case 1: Only one or more CSN appeared in the system, no CRN appeared. In this case, a CSN sends out incorrect messages, but all the nodes including CSNs report the correct voting results received in Phase 3. In other words, this case has scenario 3 only. In this case, the CSNs can be easily identified, as reported results are correct in this case. Furthermore, a working node votes with 2/3 of nodes in both Phases 2 and 3 due to WNA.

Case 2: There is no CSN appeared in the system, but one or more CRN appeared. In this case, all voting results are correct, but CRNs sends out incorrect collected results. In other words, this case has scenarios 2 only. But CRNs can be easily detected as the voting result contained in the message will be encrypted with private key of the voting node, thus when a message is received, the voting results can be verified by using public key of the voting node. Once a node detects that a CRN, the node broadcasts the identify of the CRN with evidence, such as message received to remove the CRN from the system. Other nodes can also verify the identity of CRN by re-checking the message.

Case 3: One or more CSN as well as one or more CRN appeared in the system, and they may be the same or different nodes. This case has scenarios 4 only. As long as the system has more than 2/3 nodes working, these working nodes will get a consensus from messages sent in Phase 2 and 3. As any CRNs can be identified by public key of voting nodes as in Case 2, and thus all CRNs can be immediately removed, once the CRNs are removed, CSNs can be identified as in Case 1.

All the remaining working nodes still need to reach a consensus among remaining working nodes. Another phase can be added to PBFT to reach this consensus. But this additional phase will not be needed if the current run is scenario 1. If any node has been identified as a CSN or CRN, a new phase will be added to ensure that every node knows who are valid members in the system.

In summary, PBFT can be updated as follows: normally it runs three phases as before; in case of CSNs and/or CRNs, another phase is added after Phase 3 to reach a consensus on remaining working nodes.

5. Conclusion

A permissioned BC relies on a limited set of trusted validators, rather than open validation in public BC. It is promising to provide more efficient and secured data storage and maintenance services. One goal is to develop an enterprise version of permissioned BC where hundreds of thousand transactions can be conducted within seconds. There are still open issues in the design of consensus protocols, malicious nodes detection algorithm, and scalable architecture.

Acknowledgements

This work is supported by National Key Laboratory of Software Environment at Beihang University, and National Foundation of Science China (No. 61472197).

References

- [1] “Blockchain: The Next Big Thing,” Available at: <http://www.economist.com/>, May 2015.
- [2] “The Trust Machine: The Technology behind Bitcoin could Transform how the Economy Works,” Available at: <http://www.economist.com/>, Dec. 2015.
- [3] V. Buterin, “On public and Private Blockchains,” Available at: <http://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>, Aug. 2015.
- [4] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.
- [5] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in *Proc. OSDI*, vol. 99, Feb. 1999, pp. 173–186.
- [6] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman, “Reputation Systems,” *Communications of the ACM*, vol. 43, no. 12, pp. 45–48, 2000.
- [7] L. Lamport, “Generalized Consensus and Paxos,” Microsoft Research, Technical Report MSR-TR-2005-33, 2005.
- [8] W. T. Tsai, R. Blower, Y. Zhu, and L. Yu, “A System View of Financial Blockchains,” *Proc. of IEEE SOSE*, March 2016.