First Impression on "Asynchronous Consensus-Free Transaction Systems"

Shoma Mori

October 11, 2019

1 Motivation

In existing blockchain systems, it is assumed that the consensus problem needs to be solved. Payment transacton systems, which are one of the dominant ways to utilize blockchain, are also required to get a consensus in order to avoid double-spending problems. However, making a consensus on transactions takes time, thus preventing scalability. Therefore, blockchain could get scalability if systems can be managed without making a consensus, and they would be used more in practice.

In fact, if some node makes transactions whose sending addresses are the same using the same UTXO (Unspent Transaction Output), it is technically easy to find that behavior (Fig. 1). Moreover, it is obvious for the sender that he/she made a double payment since they actually paid. Therefore, if systems have "validators", who verify transactions, they do not need a consensus on transactions in terms of solving double-spending problems.

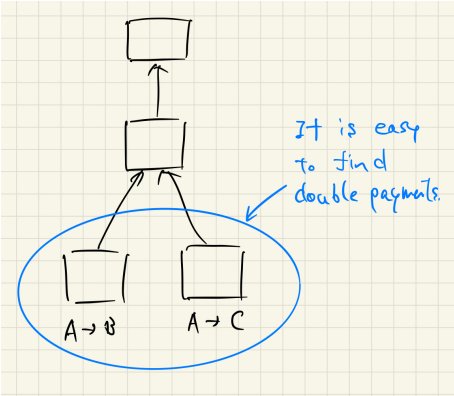

Figure 1: Double-spending

2 Transactions&Blocks

The following is the flow that describes how a transaction from node v_1 to node v_2 is added to blockchain in our systems without a consensus.

1. v_1 makes a transaction whose transfer destination is v_2'address using one UTXO linked to v_1's address.
2. v_1 broadcasts the transaction to its neighbors.
3. Every node including validators receives the transaction.
4. Validators verify whether the UTXO that is used as input in the transaction has not used yet by tracing blockchain.

5. If not, validators include the transaction in their creating blocks.

6. Validators broadcast their own new blocks when enough transactions are added to the blocks.

7. Each node who received new blocks verifies whether every transaction in the blocks is valid by tracing blockchain as well.

8. If all the transactions are valid, nodes add the blocks to their (local) blockchains.

Nodes may want to add transactions whose UTXOs for inputs are already used, i.e. double-spending. In order to resolve this problem, the systems deny transactions whose inputs are the same as a transaction in blockchain but their outputs are different. Since transactions are broadcasted to the system, every node eventually gets the set of transactions and can detect the incorrect situation (Fig. 2). When such transactions are detected, it is required to reject all the transactions which are related to the double payment because if one of them is made remained in blockchain, the system needs a consensus to choose the one.

There is a possibility that nodes receive blocks whose parents are the same node. Since the system does not make a consensus, it is not possible to choose leaders that connect to each parent. In other words, blockchain can fork, that is, it is possible that the same transaction can be included in different blocks and the blocks can be added to blockchain. Fig. 3 shows the case where two different blocks which include the same transaction are created and broadcasted to the network. In that case, the system can merge the two blocks into one block (Fig. 4).

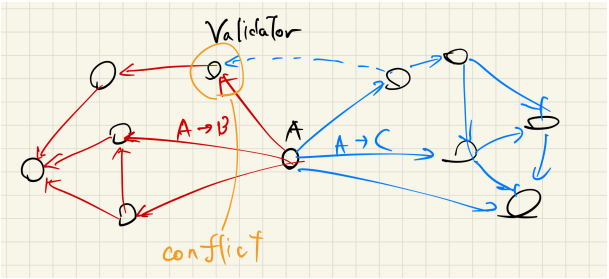

Figure 2: Node A tries to make a double payment (A→B and A→C), but a validator detects a conflict by tracing blockchain.

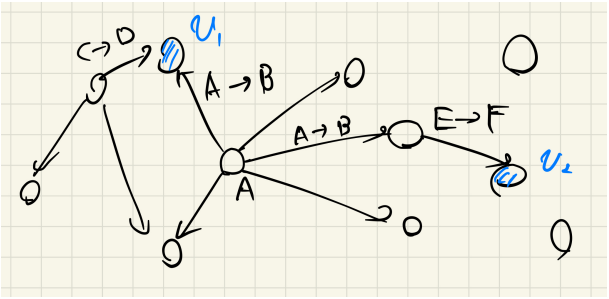

Figure 3: Two different blocks which include the same transaction (A→B) may be created because of the difference of the delivery order of transactions.

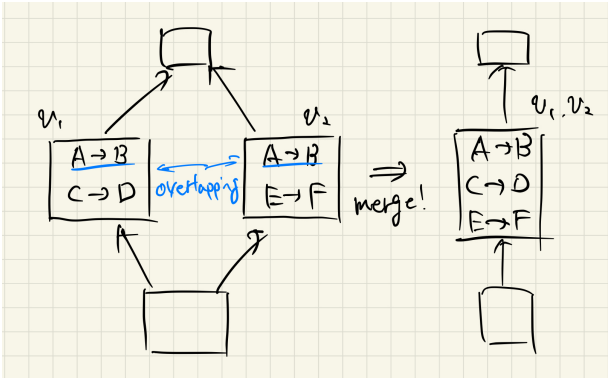

Figure 4: Blocks which have the same transactions can be merged into one block.

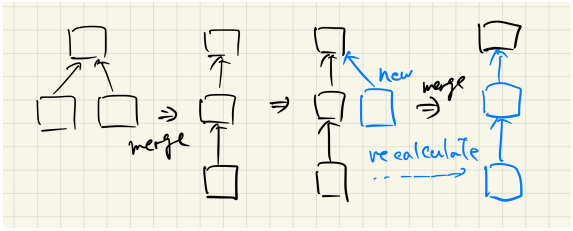

Figure 5: The case where a node received a block whose parent is the same as the merged block. If the system remerges the new block to merged block, its descendant must be recalculated.

It is also possible that only some nodes are merged and other nodes are not. This case can happen when a node receives a block whose parent is the same as the merged block. In this case, if the other nodes are also merged into one block again, it may need to recalculate the child, perhaps even grandchild and so on because child node may be added to the merged node.

3 Open Questions

In our systems, blocks can be added to blockchain without any workload or voting, which is different from blockchain systems with consensus algorithms. Therefore, every node can make forks in order to try to tamper the history of transactions. However, it can be avoided by rejecting double-spending transactions as described before. The question is why we need to use blochchain. I mean, we can also use an ordinary database to record transactions.

On the other hand, blockchain systems adopting PoW, for example, are not tampered easily since if someone tries to tamper some block, he/she has to tamper all blocks after the block, which needs tremendous workload. That is why the shape of "chain" is meaningful.

In our systems, however, since nothing is required to add blocks, it seems that recording in blockchain is essentially the same as recording in ordinary databases, such as RDB. What is the advantage of using blockchain in our systems? Or, in the first place, we do not assume to use blockchain?