Faculty Electrical Engineering, Mathematics and Computer Science

TODO title

TODO subtitle

Kelong Cong

Supervisors:
Dr. J. Pouwelse
Dr. Unknown
Prof. Unknown

Delft, Month 2017

Abstract

THIS IS MY ABSTRACT

Preface

Please write all your preface text here. If you do so, don't forget to thank your supervisor, other committee members, your family, colleagues etc. etc.

Contents

Chapter 1

Introduction

Nothing here. . .

Chapter 2

Bottom-up Consensus

2.1 Requirements

- Permissionless
- Byzantine fault tolerant
- No PoW
- Works under churn
- Underlying data structure is TrustChain
- Detects forks or double-spends
- No step in the protocol blocks transactions
- Application independent

2.2 Assumptions

- The total number of valiators is N. The number of faulty validators is no more than f, where $n > 3f$ (this value depends on BFT consensus algorithm that we use).
- Validators have the complete history of the previously agreed set of transactions.
- Weak synchrony, where messages are guaranteed to be delivered after Δ (also depends on BFT consensus algorithm).

2.3 Protocol

The goal of the bottom-up consensus protocol is to validate a set of new transactions against the previously validated transactions, and then disseminate these transactions. For a set of transactions to be valid, they need to adhere to the TrustChain construction. Out solution addresses the issue of Proof of Work (wasteful) and classical BFT algorithms (does not scale).

To describe the protocol, we make two assumptions, later we show to these assumptions are removed.

1. We assume there exist a set of validator nodes for every consensus run and for every round in a consensus run that are selected in a fair way and is known to all nodes. We address this assumption in section 2.3.6 and section 2.3.7.

2. We assume there exist a set of valid transactions from the previous consensus run. Every node knows the Merkle hash of this set of transactions. Assumption addressed in section 2.3.8.

Figure 2.1 shows the state of the system. Before explaining the four phases of the protocol, we describe the concept of checkpointing.

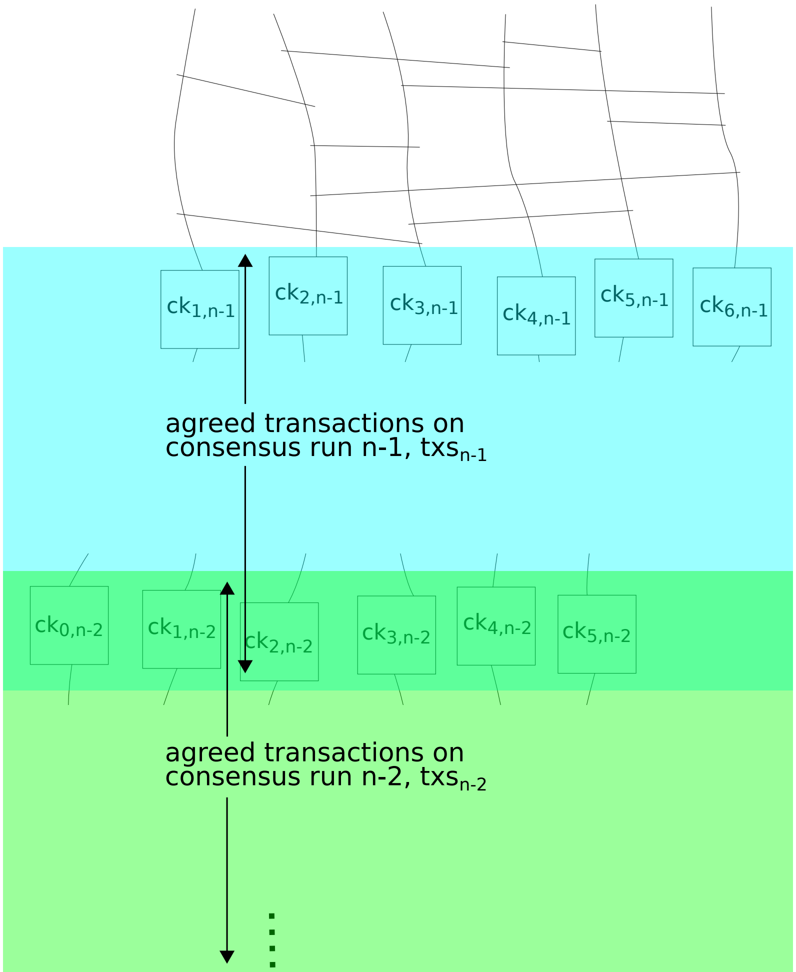
ck1,n-1
ck2,n-1
ck3,n-1
ck4,n-1
ck5,n-1
ck6,n-1
agreed transactions on consensus run n-1, txsn-1
ck0,n-2
ck1,n-2
ck2,n-2
ck3,n-2
ck4,n-2
ck5,n-2
agreed transactions on consensus run n-2, txsn-2

Figure 2.1: The initial state of the bottom-up consensus protocol. The thin lines are hash pointers to transactions and $ck_{x,n}$ represents a checkpoint block, where x is the node ID and n is the consensus run.

2.3.1 Checkpointing

A checkpoint block is a self-transaction where its input and output only involve the node itself. Node x creates the checkpoint block using

$$ck_{x,n} = H(pk_x || H(txs_{n-1})),$$

where H is a cryptographically secure hash function, pk_x is public key of x and txs_{n-1} is the validated set of transaction in consensus run $n - 1$.

Transactions between two checkpoints form a chain which may or may not be validated. For example, the chain between $ck_{1,n-2}$ and $ck_{1,n-1}$ in fig. 2.1 is validated, any chain after $ck_{1,n-1}$ is not validated. Only one checkpoint block is allowed for every agreed set of transactions txs_n.

The goal of every honest node is to validate their unvalidated chain. They do this by sending it to the validators. The validators then performs the actual validation with other validators and reach consensus. We describe these in detail next.

2.3.2 Setup phase

To prepare a chain on node x for validation in consensus run n, x first communicates to the validators of consensus run $n - 1$ for the Merkle hash of the valid set of blocks (recall that we assume every node knows the validators), and then generates a new checkpoint block (fig. 2.2). This completes the setup phase.

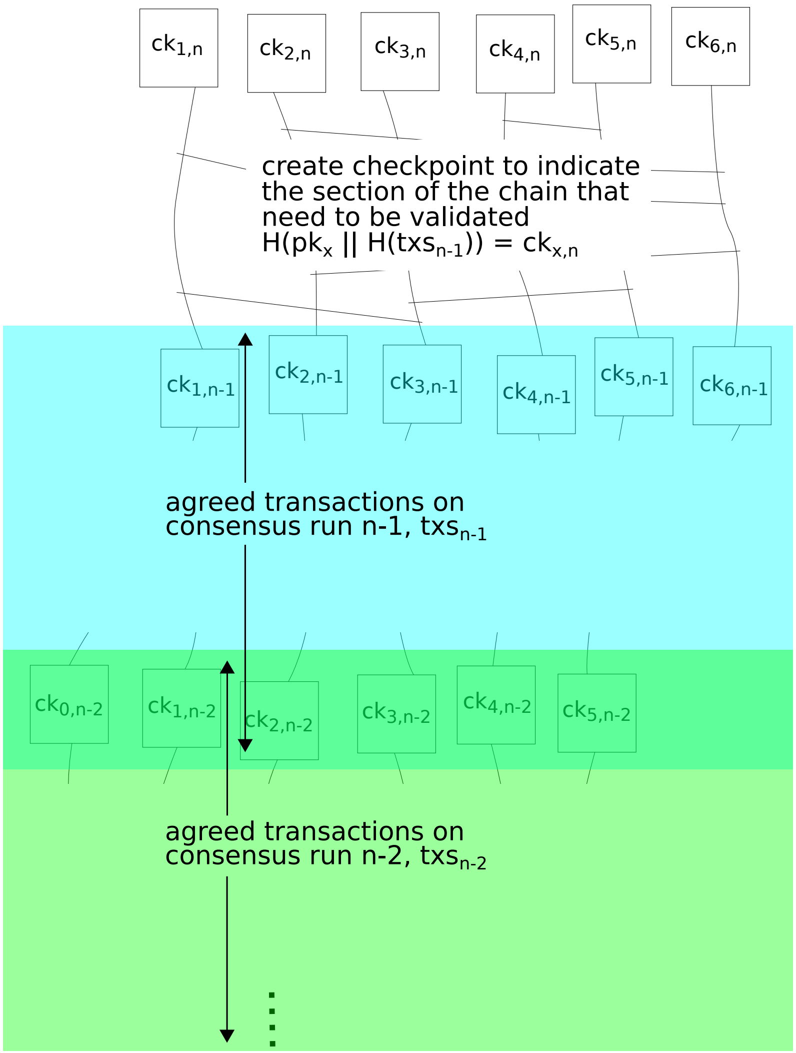
ck1,n
ck2,n
ck3,n
ck4,n
ck5,n
ck6,n
create checkpoint to indicate the section of the chain that need to be validated
H(pkx || H(txsn-1)) = ckx,n
ck1,n-1
ck2,n-1
ck3,n-1
ck4,n-1
ck5,n-1
ck6,n-1
agreed transactions on consensus run n-1, txsn-1
ck0,n-2
ck1,n-2
ck2,n-2
ck3,n-2
ck4,n-2
ck5,n-2
agreed transactions on consensus run n-2, txsn-2

Figure 2.2: In the setup phase, nodes obtains the new set of validated transaction from the validators and generate checkpoint blocks (top of the figure).

2.3.3 Group consensus phase

Every node (validator or not) belong to a *consensus group* (again we assume this is known). Validators that belong to the same consensus group is called a *validator group*.

Nodes send their unvalidated chain to their respective validator groups. The validator collect those chains for a fixed amount of time and then rejects any new chains. Next, the validator groups perform a BFT consensus algorithm (e.g. PBFT [4]) and reach consensus on a set of valid transactions. At the end of this phase, the validators in every validator group should have a set of transactions that has reached group consensus.

If the BFT consensus algorithm is sufficient when there is only one consensus group, then there is no need to perform the group merge phase. But most algorithms do not scale well [7].

2.3.4 Group merge phase

Given a set of valid transaction for every validator group, we need to merge them to achieve global consensus. We do this using a bottom-up approach, similar to divide and conquer algorithms such as merge sort. The algorithm works in rounds, but it does not assume a synchronous environment.

Every validator group merges with another validator group (assuming the total number of groups is even). We assume the pairing is known just like how every node knows the identities of the validators, but we remove the assumption later. Validators from both groups share their validated transactions and run a BFT algorithm to agree on the final result, which is simply the union of the two previously validated transactions. By this point the group is merged with a new set of validators.

The group merge phase proceeds to the next round by repeating itself until there is only a single validator group. Groups may not finish the same round at the same time, thus groups that finish early need to wait until its pair is ready (by exchanging messages).

2.3.5 Dissemination phase

The validator in the final round publishes the set of validated transactions (possibly using BitTorrent) and the Merkle hash of the validated transactions (possibly in a hierarchical way). They also create a self-transaction in their own chain that contains the Merkle hash.

2.3.6 Consensus group formation

Until now we always assumed the consensus groups and validator groups are known, in this and the subsequent sections we describe a novel technique to perform group formation.

Group formation is a hard problem in distributed systems because it is essentially a consensus problem—every node need to know the group membership information of every other node. Even worse, running a BFT algorithm may not be possible in this case because we have dynamic group sizes.

Our novel idea is to piggyback on the latest consensus result. Every node knows the latest consensus result txs_{n-1} (if they don't they can ask the validators) or the Merkle hash of it $H(txs_{n-1})$, then they perform some deterministic computation to find the luck value

$$l_{x,n-1} = H(H(txs_{n-1})||pk_x)$$

for every node that participated in txs_{n-1}. The consensus group for x in consensus run n on round k (initially at 0) is then

$$g_{x,n,k} = l_{x,n-1} \mod 2^{m-k},$$

where m can be for example 8, it depends on the estimated number of users and the target consensus group size.

Computing the cryptographic hash for every node in the set may become expensive, and poor clients may fail to do it in time. We can work around this by having the validators send the membership information to all members of their consensus group.

2.3.7 Validator group formation

Validators all belong to a consensus group, but only lucky nodes are allowed to be validators. To find the luckiness, we use the same luck value $l_{x,n-1}$, and check for the inequality

$$l_{x,n-1} < D,$$

where D is the difficulty value and a system parameter (undecided). Nodes that satisfies the inequality are considered lucky.

Recall that validator groups merge and create a new set of valid transactions. So the validator groups need to answer the following two questions.

1. Which other validator group to pair up (this information need to be symmetrical)?

2. Who are the new validators in the merged validator group?

The answer for (1) is encoded in $g_{x,n,k}$, namely $g_{x,n,k} = g_{x,n,k-1} \mod 2^{m-k}$. The information for (2) is the first c validators ordered by their luck, where c is a system parameter (undecided).

In essence, every node runs a deterministic algorithm on the latest consensus result to determine the consensus group, validator group, etc. for the next consensus run. With this, we circumvent the permissionless problem with classical BFT algorithms.

2.3.8 Bootstrapping

The protocol need to be bootstrapped, in this section we provide a possible method.

From the genesis block up until some consensus run n, there would be no validator group formation, all validation are performed by servers run by the developer. This property can be implemented in the client software (similar to how Bitcoin performs protocol upgrades). From consensus run $n + 1$ and onwards, the validator groups begin to form and the system begins to run the full bottom-up consensus protocol.

2.4 Choosing the right BFT consensus algorithm

PBFT [4] assumes weak synchrony. It uses a primary node to initiate the algorithm which may not fit nicely with our model. But it is somewhat successful in Tendermint and Hyperledger (citation and verification needed).

HoneyBadgerBFT [5] and Cachin's secure broadcast protocol [3] may suite our needs more, but further investigation is needed. The advantage of these two protocols over PBFT is that they work in the asynchronous environment.

The Swirlds [1] algorithm is a departure from the classical BFT algorithms. Nodes use "gossip about gossip" to determine whether a majority of the population has heard about a particular transaction. So agreement is achieved not after one execution of the algorithm but after the nodes have eventually exchanged enough messages—we call this "eventual consensus". The downside of Swirlds is that they assume static membership, thus it is not permissionless. Swirlds may be applicable to our design where we can use our group formation technique to select the validators and run Swirlds on top (assuming Swirlds scales well). This eleminates the need for the recursive (bottom-up) phase.

2.5 Limitations, Discussion and Questions

- What's the mechanisms for publishing double-spends?

- Doesn't prevent spam, so DoS is possible. But we can work around this by ignoring spammy nodes.

- If there is stake in the system, we can use it to elect validator, this would prevent the sybil attack. But I argue sybil attack is application dependent.
- No incentive in this system, but again it's application specific.
- If a node is offline for some time, it will not be in any consensus group. To join a consensus group, it must make a transaction to a node that is in a consensus group.
- Not all the validators are online (chrun), but we can model it using the Sleep consensus model [2].
- This model does not require a single connected component.
- I personally don't see a lot of benefit with the idea of spontaneously growing the set of valid transaction by traversing the DAG, in the end everything must be validated anyway, and I don't see any major advantages of that technique. It also must assume single connected component, where as this approach does not.

2.5.1 Merging consensus groups does not guarantee BFT with a probability of 1

Our assumption is that $n > 3f$, this can be alternatively written as $n = 3f + 1$, for the set of all validators. Suppose the validators are put into c groups and each group has $\frac{n}{c}$ validators. The bottom-up consensus protocol requires every validator group to reach consensus, thus if a group has $\frac{n}{3c}$ or more Byzantine nodes, then the group cannot reach consensus. We claim if Byzantine nodes can freely choose validator groups, then they can prevent consensus for $c - 1$ groups.

We can write

$$f = \frac{n-1}{3} = x\frac{n}{3c},$$

where x is the number validator groups that the Byzantine group can infiltrate. Suppose $x = c$, then

$$f = \frac{n-1}{3} < c\frac{n}{3c} = \frac{n}{3},$$

this implies that the number Byzantine nodes is insufficient to infiltrate all c groups. Now suppose $x = c - 1$, then we claim

$$f > (c-1)\frac{n}{3c}$$

for $n > c$. This is easy to see

$$\begin{aligned} f = \frac{n-1}{3} &> (c-1)\frac{n}{3c} \\ n - 1 &> n\frac{c-1}{c} \\ \frac{n-1}{n} &> \frac{c-1}{c} \\ n &> c. \end{aligned}$$

This result implies that there is a sufficient number of Byzantine nodes to infiltrate $c - 1$ groups.

While this disheartening, there may be a few ways to improve on it or look at it in a different light.

- The analysis assumes that the Byzantine nodes can choose which group to join, but in our protocol they cannot. Thus the aforementioned result is the worst case scenario.
- Perhaps we do not aim BFT with a probability of 1, but try to achieve BFT with a sufficiently high probability. Randomisation of group membership helps here.

- Does our approach break down when a consensus run is infiltrated by Byzantine nodes? Can we recover afterwards? My intuition says yes, because honest nodes will always detect malicious transactions due to the TrustChain structure. Thus they will not accept the "validated" transactions from the Byzantine nodes, and they can publish a proof of malicious transaction in future consensus runs.
- There is always one consensus group that is not infiltrated, can it do something about the situation?
- The bottom-up approach is aimed to make BFT consensus algorithms scale. In some applications it may be sufficient even without the bottom-up approach (HoneyBadgerBFT can handle thousands of transactions on 64 nodes [5]).
- The comparison (theoretical and practical) between bottom-up and non-bottom-up would make an interesting thesis topic.

2.5.2 Tail bounding the number of Byzantine nodes in a consensus group

We want to answer the following question. *Suppose there is an urn that has N balls, $M = (2N+1)/3$ of them are white and $N - M$ of them are black. N/c balls are drawn without replacement. What is the probability that the number of white balls drawn is lower than the expected number of white balls?* This question maps directly to our problem of Byzantine nodes, where the urn is the set of all validators and they form c groups each of size N/c. The white balls represent honest nodes and the black balls represent Byzantine nodes.

The problem is modelled by the hypergeometric distribution[1] and we can solve it using tail inequalities.

Using the notation and definition in [6], the tail inequality (not necessarily tight) is given by

$$\Pr[i \leq E[i] - tn] \leq e^{-2t^2 n},$$

where i is the random variable (the number of white balls), $E[i] = n\frac{M}{N}$ is the expected value and $n = N/c$ is the number of balls drawn. For our problem, we are interested in the probability that the number of white balls is lower than the expected value. Thus we can write $\Pr[i \leq E[i] - 1]$, and then

$$\begin{aligned} tn &= 1 \\ tN/c &= 1 \\ t &= c/N. \end{aligned}$$

Finally we get

$$\begin{aligned} \Pr[i \leq E[i] - 1] &\leq e^{-2(c/N)^2(N/c)} \\ \Pr[i \leq E[i] - 1] &\leq e^{-2(c/N)} \end{aligned}$$

To minimise the probability, we must maximise c/N. But $c/N < 1$, so parameter tuning cannot help us to reach a vanishingly small probability. This may be the nail in the coffin for the current algorithm.

2.5.3 How to set the initial validator group size?

This can be modelled as the occupancy problem.

[1] This is a variation of the geometric distribution where the balls are picked without replacement.

Bibliography

[1] LEEMON BAIRD. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. 2016. 7

[2] Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. Cryptology ePrint Archive, Report 2016/918, 2016. http://eprint.iacr.org/2016/918. 8

[3] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001. 7

[4] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999. 6, 7

[5] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016. 7, 9

[6] M. Skala. Hypergeometric tail inequalities: ending the insanity. *ArXiv e-prints*, November 2013. 9

[7] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015. 6