BBFT: a Hierarchical Byzantine Fault Tolerant Consensus Algorithm

ByStack Team

v1.0

1 Introduction

Consensus is the process applied by nodes to reach final agreement on network states. Byzantine fault tolerant (BFT) consensuses are considered well suited for permission-ed blockchains where semi-trust exists among node. BFT consensus can deliver high TPS (transactions per second) while guarantees safety when at most $\lfloor \frac{n-1}{3} \rfloor$ out of total n nodes are simultaneously faulty.

PBFT (practical byzantine fault tolerant) is one of the widely adopted BFT consensuses[6]. However, it does not scale well because of $O(n^2)$ communication complexity. In this report, we present a hierarchical BFT consensus algorithm - BBFT. It utilizes network topology to effectively distribute and aggregate messages among nodes and provides $O(n)$ communication complexity.

2 System Model

We define the system as an asynchronous distributed environment where individual nodes communicate via point-to-point network. The network may fail to deliver message, delay them (not indefinitely), or deliver them out of order. The network by itself is heterogeneous, meaning communication quality between nodes is not equal and does not remain constant between the same set of nodes.

Message authentication is guaranteed by asymptotic security through cryptographic signatures and hash functions[15, 17, 12].

We call a node a *consensus node* if it participates in the consensus process. Consensus node replicates the current state of the system. *Gateway node* is a consensus node that performs additional signature aggregation. *Leader node*

is a consensus node that proposes a block to be verified at the beginning of the consensus round. Every node possess a unique ID and is known by other nodes.

We call a node *byzantine node* if it behaves arbitrarily. It may delay communication, modify messages, replay messages and forge signatures. Byzantine nodes can even collude. We assume that the faulty nodes are computationally bound so that it is impossible to subvert the cryptographic techniques used for message security.

Nodes are organized as tree vertices with edges being communication paths as shown in figure 1. Non-leaf nodes are gateway nodes, and leaf nodes are consensus nodes. Top level gateway nodes are connected to each other. Leader node is always one of the top level gateway nodes.

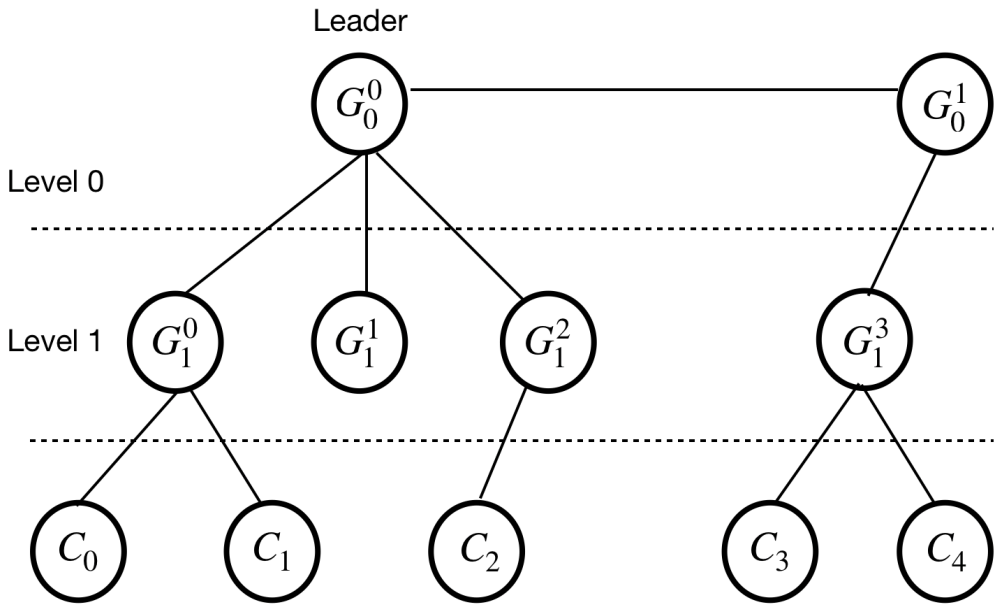

Figure 1: Example Nodes inter-connection Topology

The system is *Byzantine Fault Tolerant* with minimal number of nodes $n = 3f + 1$ when up to f nodes are faulty. This number provides a lower threshold because in worst case of f faulty nodes, a node must be able to proceed with $n - f$ responses, despite that the f unresponsive nodes are actually non-faulty. This requires $n - 2f > f$. Therefore $n > 3f$.

In the following sections we describe the process of how nodes reach consensus. Then we discuss the two important components of the algorithm: how network is partitioned and how to aggregate signatures effectively.

3 Consensus Process

The consensus process resembles of a typical PBFT[6] with additional handling of signature aggregation at the gateway nodes. Under normal case, the algorithm consists of three stages: *distribute*, *aggregate*, and *finalize* as shown in figure 2.

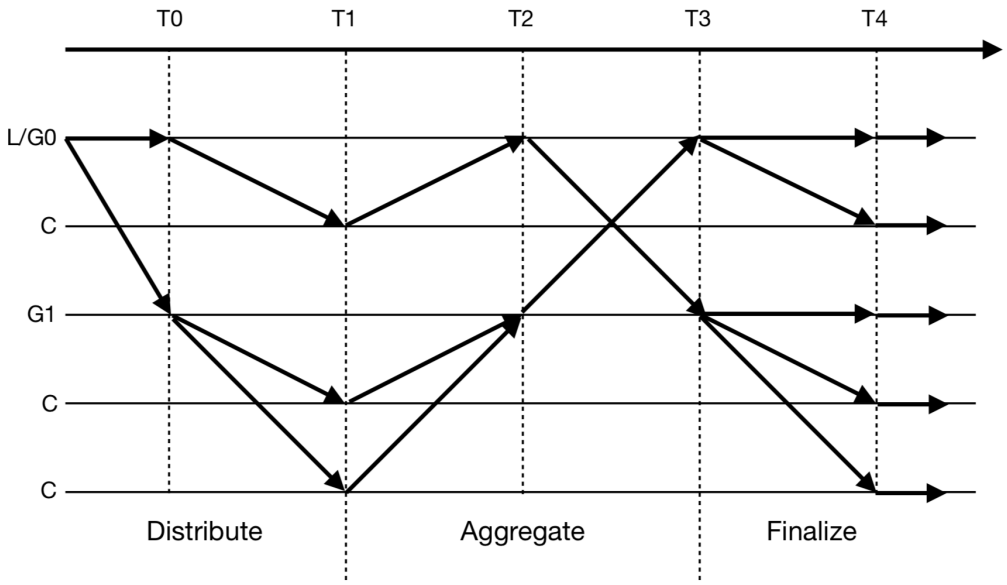

Figure 2: Example Consensus Process of Single Level Five Nodes Network

At the beginning of consensus round, the nodes have the same view number $v = 0$, and block height h. Every node has its unique id i, which is known by other nodes. A bitmap m is used during message passing to indicate which nodes' signatures are included in the aggregated signature.

1. The leader node p selects transactions from its transaction pool and packages them into the next proposed block b. The proposal is broadcasted to its peer gateway nodes at the same level as $\langle$DISTRIBUTE, $h, v, m, b, b_\sigma\rangle$.

2. Its peer gateway nodes receive the proposal, and start to pass it down along the topology tree edges. This concludes the *distribute* stage.

3. At the leaf level, upon receipt of the proposal, consensus node i verifies the block and signature, signs the block, and sends the result to its parent gateway node as $\langle$AGGREGATE, $h, v, m, b_\sigma\rangle$.

4. At every non-leaf level, gateway nodes verify the signatures received from its children and aggregate them together with its own incrementally, and pass it up along the topology tree edges.

5. When top level nodes receive the aggregated signatures, they do a full exchange of the message. This concludes the *aggregate* stage.

6. The fully aggregated signature is then passed down along the topology tree edges to reach the leaf nodes as $\langle \text{FINALIZE}, h, v, m, b_\sigma \rangle$.

7. Any consensus node, upon receipt of at least $n - f$ signatures, reaches consensus and logs the block to its ledger. This concludes the *finalize* stage.

A consensus node can initiate a *VIEW-CHANGE* proposal when it cannot reach consensus within t time interval, an invalid block has been proposed, or an invalid aggregated signature has been received. Here t is chosen to be the maximum consensus process time, and depends on the number of nodes and network characteristics. When any node received at least $n - f$ *VIEW-CHANGE* messages with the same view number from other nodes, current view number is updated and new consensus round starts.

Let n be the number of consensus nodes, and m be the number of top level gateway nodes, the total number of messages communicated in the consensus process is $\lambda = m^2 - 2m + 3n - 2$. When $1 \leq m \leq \sqrt{n} \Rightarrow \lambda \leq 4n - 2\sqrt{n} - 2$, the communication complexity is $O(n)$. Multi-level gateway nodes do not impact overall complexity, because only the top level gateway nodes perform full message exchange. BBFT exhibits its flexibility when m changes. As two special cases of BBFT, when $m = 1$, BBFT becomes FBFT[16], and when $m = n$, BBFT becomes PBFT.

4 Network Zoning

We define *topology graph* as the logical view of inter-connection among nodes. Nodes are vertices in the graph, and undirected edges are the communication paths between nodes, with its weight being communication latency. We define *topology tree* as the minimal spanning tree (MST) that connects all the nodes together, without any cycles and with the minimal possible total latency. We consider the topology tree unique given the fact that it is highly impossible for the latency to be exact the same between any two pairs of connected nodes in real use case.

Finding MST of an undirected graph is a well studied area[14, 7, 10]. Highly optimal linear complexity algorithms exist[8, 13]. Linear complexity

distributed solutions also exist, where nodes are fully disconnected and only communicate by message passing[9, 1]. To construct the MST in BBFT, we can either deploy a dedicated service that requests latency numbers from nodes and returns the MST back to nodes, or distributed algorithm can be applied. Note that even with the centralized approach, the service internal implementation can be transparent to the nodes. The detail of the algorithm is out of the scope of this report.

Once the MST is built, the tree is rotated such that the elected leader node is at top level, and the number of the top level gateway nodes does not exceed $\sqrt{n}$. Given the dynamic of network inter-connection state, the topology tree is re-generated periodically. Re-generation frequency depends on the use case and is coordinated with leader election.

5 Signature Aggregation

When gateway node receives responses from it children consensus nodes, it performs signature aggregation before routing the message. A naive aggregation would simply concatenate all signatures together, which may lead to big message size and requires recipient to verify all signatures. We call a key aggregation *effective* if

1. The aggregated key size should remain relatively constant regardless of number of participants.

2. The verification time of the key should remain relatively constant regardless of number of participants.

Here the term "*key*" may refer to any types of primitives, such as secret keys, public keys, signatures, etc.

In BBFT, we use BLS (Boneh–Lynn–Shacham) multi-signature scheme[4, 5, 3] to effectively aggregate signatures. With BLS, it's possible to aggregate all types of primitives and the result is always another valid primitive. Primitives which were already aggregated can also be further aggregated incrementally, independent from the order in which it happens. With this property, the aggregated signature size remains constant regardless of number of participating signatures. Another unique property of BLS is that it can be done in single round, unlike Schnorr signature scheme[11], where a pre-commitment setup is needed, which leads to multiple rounds of communication. This feature makes it ideal for distributed and trust-less environment.

Let n be the number of participants, m be the message communicated among them. The BLS signature aggregation scheme in BBFT works as follows:

- Setup: We choose an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$[3], where $\mathbb{G}_0$, $\mathbb{G}_1$, and $\mathbb{G}_T$ are groups of prime order q. Let g_0 and g_1 be the generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively. We also choose two hash functions $H_0 : M \rightarrow \mathbb{G}_0$, a mapping from message space to $\mathbb{G}_0$, and $H_1 : \mathbb{G}_1^n \rightarrow \mathbb{Z}_q$.

- KeyGen(): Every participant picks a secret key $sk \in \mathbb{Z}_q$, and shares its public key $pk = g_1^{sk \cdot H_1(g_1^{sk})}$ with each other. $pk \in \mathbb{G}_1$.

- Sign(sk, m): Participant with secret key sk signs m and outputs signature $\sigma = H_0(m)^{sk \cdot H_1(g_1^{sk})}$. $\sigma \in \mathbb{G}_0$.

- Aggregate(σ_1, σ_2,...,σ_n): Upon receipt of multiple signatures on message m, the aggregated signature $\sigma = \prod_{i=1}^{n} \sigma_i$ is computed as the product of individual signatures σ_i. $\sigma \in \mathbb{G}_0$.

- Verify(σ, m): Participant verifies an aggregated signature on m by first computing aggregated public key $\kappa = \prod_{i=1}^{n} pk_i$, as the product of individual public key pk_i. $\kappa \in \mathbb{G}_1$. Then verification is done by the two-pairing check $e(\sigma, g_1) = e(H_0(m), \kappa)$.

Pairing operation is known to be expensive, it is order of magnitude slower than typical ECDSA, such as Schnorr signature[2, 11]. However, in the case of same message, the number of pairing is limited to 2 for every aggregated signature verified regardless of number of input signatures.

6 Conclusion

We present BBFT, an efficient byzantine fault tolerant consensus algorithm. The model achieves $O(n)$ communication complexity with certain limit on number of gateway nodes. It is worth noting that as a highly flexible and reconfigurable model, zoning and aggregation techniques proposed here are not necessarily the only choices. Depending on the real use case, other techniques can be easily integrated into the consensus model.

References

[1] B. Awerbuch. "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: ACM, 1987, pp. 230–240. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28421. URL: http://doi.acm.org/10.1145/28395.28421.

[2] Alexander Block. *BLS: Is it really that slow?* URL: https://blog.dash.org/bls-is-it-really-that-slow-4ca8c1fcd38e. (accessed: 05.21.2019).

[3] Dan Boneh, Manu Drijvers, and Gregory Neven. "Compact Multi-signatures for Smaller Blockchains". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 435–464. ISBN: 978-3-030-03329-3.

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*. ASIACRYPT '01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 514–532. ISBN: 3-540-42987-5. URL: http://dl.acm.org/citation.cfm?id=647097.717005.

[5] Dan Boneh et al. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps". In: *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT'03. Warsaw, Poland: Springer-Verlag, 2003, pp. 416–432. ISBN: 3-540-14039-5. URL: http://dl.acm.org/citation.cfm?id=1766171.1766207.

[6] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 173–186. ISBN: 1-880446-39-1. URL: http://dl.acm.org/citation.cfm?id=296806.296824.

[7] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0945-3245. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[8] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In: *J. ACM* 34.3 (July 1987), pp. 596–615. ISSN: 0004-5411. DOI: 10.1145/28869.28874. URL: http://doi.acm.org/10.1145/28869.28874.

[9] R. G. Gallager, P. A. Humblet, and P. M. Spira. "A Distributed Algorithm for Minimum-Weight Spanning Trees". In: *ACM Trans. Program. Lang. Syst.* 5.1 (Jan. 1983), pp. 66–77. ISSN: 0164-0925. DOI: 10.1145/357195.357200. URL: http://doi.acm.org/10.1145/357195.357200.

[10] Joseph B. Kruskal. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem". In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2033241.

[11] Gregory Maxwell et al. "Simple Schnorr multi-signatures with applications to Bitcoin". In: *Designs, Codes and Cryptography* (Feb. 2019). DOI: 10.1007/s10623-019-00608-x.

[12] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Norwell, MA, USA: Kluwer Academic Publishers, 1994. ISBN: 0792393686.

[13] Seth Pettie and Vijaya Ramachandran. "An Optimal Minimum Spanning Tree Algorithm". In: *J. ACM* 49.1 (Jan. 2002), pp. 16–34. ISSN: 0004-5411. DOI: 10.1145/505241.505243. URL: http://doi.acm.org/10.1145/505241.505243.

[14] R. C. Prim. "Shortest connection networks and some generalizations". In: *The Bell System Technical Journal* 36.6 (Nov. 1957), pp. 1389–1401. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1957.tb01515.x.

[15] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[16] Harmony Team. *Harmony Technical Whitepaper*. URL: https://harmony.one/whitepaper.pdf. (accessed: 05.21.2019).

[17] Gene Tsudik. "Message Authentication with One-way Hash Functions". In: *SIGCOMM Comput. Commun. Rev.* 22.5 (Oct. 1992), pp. 29–38. ISSN: 0146-4833. DOI: 10.1145/141809.141812. URL: http://doi.acm.org/10.1145/141809.141812.