# Memory-Efficient Byzantine Fault-Tolerant Replication for Highly Resource-Constrained Systems

Harald Böhm
hboehm@cs.fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg

Tobias Distler
distler@cs.fau.de
Friedrich-Alexander-Universität Erlangen-Nürnberg

## ABSTRACT

Recently, possible application scenarios of Byzantine fault tolerant (BFT) state-machine replication have been expanded to include highly resource-constrained nodes, with RAM of 1MB of or less. As memory is a scarce resource, using it efficiently is of utmost importance. In this paper, we show that the upper bound for memory demand of the PBFT protocol reported by previous work can be improved even further, by applying additional optimizations.

## 1 INTRODUCTION

Byzantine fault tolerant (BFT) state-machine replication is a powerful tool, offering resilience against arbitrary faults. Traditionally, research has been focused on server-grade hardware [1, 3, 6, 5, 8, 9]. While there exists prior work exploring replication on embedded systems, most of it has targeted larger general-purpose platforms like the Raspberry Pi [4, 11]. More recently, however, we have made efforts to expand use-cases of this technology to tiny devices, such as system-on-chip platforms. The result of this work is TinyBFT [7], a state-machine replication library for highly resource-constrained systems based on the PBFT protocol.

The term *highly resource-constrained* here refers to replicated systems consisting of a number of nodes that only have a very limited amount of RAM, typically 1 MB or less. Possible examples include the ESP32-C3 with a 160MHz MCU and 400KB RAM [10]. In order to run the consensus protocol, nodes are equipped with transceivers, allowing them to communicate with each other over a partially synchronous network.

Employing BFT state-machine replication on these tiny devices offers the benefit of improved resilience directly where data is created in computing infrastructures of the Internet-of-Things (IoT). This opens up a number of new use-case scenarios, including self-sufficient control applications based on distributed key-value stores or blockchain-based recording of sensor data [7].

## 2 PROBLEM STATEMENT

Improving memory efficiency of BFT state-machine solutions without degrading their performance is crucial when referring to highly resource-constrained devices. In this context, as a first step it is necessary to significantly shrink down the memory footprint of replication protocols. Traditional implementations have been running on hardware with RAM orders of magnitude larger than on the devices targeted here (e.g., [3, 4, 6, 9, 11]). Hence, memory demand has been a negligible factor in their design.

However, merely reducing memory demand to the point were a replication library is small enough to fit into the hardware's RAM is not sufficient in practice. Further improvements can have beneficial effects on system performance and energy consumption, because the already scarce amount of available RAM has to be shared with a number of other software components. This can make certain trade-offs necessary. In particular, device drivers for transceivers (e.g. WiFi) typically store incoming and outgoing packets to a fixed amount of buffers in RAM. If all available buffers are occupied, packets are dropped, making expensive retransmissions necessary. Hence, reducing the memory footprint of the replication library frees up space that can be used for additional packet buffers, leading to fewer dropped packets and, consequently, fewer retransmissions. Conversely, over-approximating the maximum memory demand of the replication library can leave less space for other parts of the system and, therefore, lead to degraded performance.

While [7] has layed a solid foundation by providing an upper bound for PBFT's [9] memory demand and implementing TinyBFT, we show that there is still room for further improvement. More specifically, the memory demand of certificates can be reduced. Thus, our paper makes the following contribution: we outline two additional optimizations related to certificates that reduce TinyBFT's maximum memory demand.

## 3 APPROACH

As our approach aims at reducing memory demand of *certificates*, we give a brief description of their structure and purpose in PBFT, before moving on to our improvements.

### 3.1 Certificates in PBFT

In Castro's PBFT protocol, *certificates* consist of a number of matching messages sent by nodes. Since all messages are cryptographically signed by their respective sender, they cannot be forged. As a result, replicas can use certificates to prove to each other that a sufficient number of correct replicas agree on a specific piece of information. With regard to memory demand, this means that replicas will store most incoming messages *as-is* in a corresponding certificate, provided they deem the message valid.

## 3.2 Compressed Certificates

Certificates, by definition, are a set of *matching* messages from nodes and, therefore, they are in parts identical. More specifically, matching messages all share the information $v$ that replicas want to agree on. As a result, it is possible to compress certificates by storing matching values transported in messages only once, instead of storing every message in its entirety. Since there are at most $f$ faulty replicas, there can be up to $f + 1$ different values $v$ of size $|v|$, before nodes agree on one of them. Additionally, for each node $i$ agreeing on a value, the message's signature $\sigma_i$ has to be persisted. However, a replica has to receive at most $f + 2$ different messages, before it knows which value $v$ it should consider correct. Therefore, it can then drop all signatures not related to that value. We define $|\sigma|$ to be the size of a signature in bytes.

Let $C$ be a *quorum certificate*, that is, a certificate requiring $2f + 1$ matching messages in order to be considered complete. Then the worst-case memory demand of $C$ and its compressed form $C_{compressed}$ can be expressed as follows:

$$C = (2f + 1) \cdot |v| + (2f + 1) \cdot |\sigma| \quad (1)$$

$$C_{compressed} = (f + 1) \cdot |v| + (2f + 1) \cdot |\sigma| \quad (2)$$

This shows that the optimization can reduce memory demand by up to $f \cdot |v|$ bytes. In actual implementations, however, this difference can be even larger, because messages typically also have additional meta-data associated with them. In TinyBFT's reference implementation for example, the size of a commit message excluding its signature is 32 bytes. However, its payload size $|v|$, consisting of a sequence number and a view number, is only 16 bytes.

## 3.3 Replacing Commit Certifictes

At the end of PBFT's three-stage agreement protocol, replicas gather $2f + 1$ matching commit messages of a fixed size $\mathcal{M}_{commit}$ to form a *commit certificate.* After receiving enough matching messages, nodes are allowed to execute the corresponding request. Hence, the original memory demand model in [7] describes a commit certificate's memory demand as:

$$C_{commit} = (2f + 1) \cdot \mathcal{M}_{commit} \quad (3)$$

However, when taking a closer look at the protocol, it becomes clear that commit certificates gathered by nodes are never propagated in order to prove to other replicas that a request proposal has been committed. Hence, it is unnecessary for replicas to keep commit messages in their log. Instead, the relevant information is: (1) the value $v$ associated with the message, that is, sequence number and view of the corresponding pre-prepare message. (2) the number of matching messages received, and (3) which replica already has a message in the certificate, in order to prevent faulty replicas from trying to be counted more than once.

Using this information, we can now construct an alternative data structure to replace commit certificates entirely. The message's value is stored as is. Determining the number of matching messages requires one counter per possible value stored. The minimum size of it depends on the number of replicas. If RAM is byte-addressable, the minimum size for this counter has to be $\left\lceil \frac{log_2(3f+1)+1}{8} \right\rceil$. Finally, in order to track replicas that already responded, one bit of information per replica is sufficient. Hence, the minimum size is $\left\lceil \frac{3f+1}{8} \right\rceil$. Since there can be up to $f$ faulty replicas and commit messages may arrive before the replica has received the corresponding pre-prepare, we have to assume that there can be up to $f + 1$ different values that have to be cached. Putting all of this together, the new data structure reduces the memory demand of $C_{commit}$ as follows:

$$\left\lceil \frac{3f + 1}{8} \right\rceil + (f + 1)(|v| + \left\lceil \frac{log_2(3f + 1) + 1}{8} \right\rceil) \quad (4)$$

As already mentioned briefly in Section 3.2, a commit message in TinyBFT's reference implementation is 32 bytes large and its payload $|v| = 16$ bytes. Assuming that signatures are generated using RSA [12] with key sizes of 2048-bit, which is the minimum size recommended by NIST [2], then $\mathcal{M}_{commit} = 288$ bytes. Inserting these values into Equation 3, we get $C_{commit} = 864$. Using Equation 4 we see that the alternative data structure only requires 35 bytes to store the same information, an improvement of 96%.

## 4 CONCLUSION

This paper has described two optimizations that can be used to reduce the memory demand of certificates in PBFT. Using them, we were able to reduce the upper bound for memory demand originally shown in [7].

## REFERENCES

[1] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. 2020. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *Proceedings of the 41st Symposium on Security and Privacy (SP '20)*, 106–118.

[2] E. Barker and Q. Dang. 2001. Recommendation for Key Management. Part 3: Application-Specific Key Management Guidance. Tech. rep. National Institute of Standards and Technology, Washington, D.C. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf.

[3] J. Behl, T. Distler, and R. Kapitza. 2015. Consensus-oriented parallelization: how to earn your first million. In *Proceedings of the 16th Middleware Conference (Middleware '15)*, 173–184.

[4] C. Berger, H. P. Reiser, F. J. Hauck, F. Held, and J. Domaschka. 2022. Automatic integration of BFT state-machine replication into IoT systems. In *Proceedings of the 18th European Dependable Computing Conference (EDCC '22)*, 1–8.

[5] A. Bessani, E. Alchieri, J. Sousa, A. Oliveira, and F. Pedone. 2020. From Byzantine replication to blockchain: consensus is only the beginning. In *Proceedings of the 50th International Conference on Dependable Systems and Networks (DSN '20)*, 424–436.

[6] A. Bessani, J. Sousa, and E. E. P. Alchieri. 2014. State machine replication for the masses with BFT-SMaRt. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN '14)*, 355–362.

[7] H. Böhm, T. Distler, and P. Wägemann. 2024. TinyBFT: Byzantine Fault-Tolerant Replication for Highly Resource-Constrained Embedded Systems. In *Proceedings of the 30th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'24)*.

[8] E. Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. PhD thesis. University of Guelph.

[9] M. Castro and B. Liskov. 1999. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*, 173–186.

[10] Espressif Systems. 2023. *ESP32-C3 Series Datasheet*. Version 1.4.

[11] A. Loveless, R. Dreslinski, B. Kasikci, and Linh Thi Xuan Phan. 2021. IGOR: accelerating Byzantine fault tolerance for real-time systems with eager execution. In *Proceedings of the 27th Real-Time and Embedded Technology and Applications Symposium (RTAS '21)*, 360–373.

[12] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21, 2, 120–126.