Brief Announcement: Communication-Efficient BFT Using Small Trusted Hardware to Tolerate Minority Corruption

Sravya Yandamuri
Duke University, Durham, NC, USA

Ittai Abraham
VMware Research, Herzliya, Israel

Kartik Nayak
Duke University, Durham, NC, USA

Michael Reiter
Duke University, Durham, NC, USA

Abstract

Small trusted hardware primitives can improve fault tolerance of Byzantine Fault Tolerant (BFT) protocols to one-half faults. However, existing works achieve this at the cost of increased communication complexity. In this work, we explore the design of communication-efficient BFT protocols that can boost fault tolerance to one-half without worsening communication complexity. Our results include a version of HotStuff that retains linear communication complexity in each view and a version of the VABA protocol with quadratic communication, both leveraging trusted hardware to tolerate a minority of corruptions. As a building block, we present communication-efficient provable broadcast, a core broadcast primitive with increased fault tolerance. Our results use expander graphs to achieve efficient communication in a manner that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases communication complexity, consensus, trusted hardware

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.62

Related Version *Full Version*: `https://eprint.iacr.org/2021/184.pdf`

1 Introduction

The number of faults tolerated by a Byzantine Fault Tolerant (BFT) protocol depends on the network assumptions between parties, the use of cryptography, and other assumptions. In particular, it is known that to maintain safety when the system is asynchronous, without additional assumptions, one cannot tolerate one-third or more Byzantine faults [8]. However, tolerating fewer than one-third Byzantine faults may not be enough for some applications. There are two known approaches to increase this fault threshold. The first approach is to give up safety in asynchrony by assuming synchrony and using some method to limit the ability of the adversary to simulate honest parties (e.g. assuming a PKI or proof-of-work) [2, 9, 11]. The second approach lets the adversary delay messages but limits its ability to corrupt by assuming the existence of a *trusted hardware*. The adversary cannot tamper with this hardware even if it fully controls the party. At a high-level, the hardware provides non-equivocation guarantees, essentially transforming Byzantine failures to omission failures and hence improving the fault tolerance threshold to one-half (e.g., [6, 8, 10]) in partial synchrony and asynchrony.

In this work, we focus on the use of *small trusted hardware* primitives to tolerate a minority Byzantine corruption and stay safe in asynchrony. Specifically, each node is equipped with hardware that implements the abstraction of an "append-only log," the contents to which it can attest using a *conventional* digital signature with a key that it holds. This capability is supported by numerous *existing* trusted add-ons (TPMs [1], YubiKeys [14], smartcards, etc.) and is far simpler to implement than secure enclaves for arbitrary computation, as Intel SGX [7] attempts to do – but arguably fails [5, 13, 15, 16, 17].

The use of such small trusted hardware to boost fault tolerance was explored in A2M [6] and TrInc [12], which specifically improved PBFT [4]. However, this came at the expense of an $O(n^3)$ communication complexity per view for consensus among n parties, measured as the (expected) number of words that all honest parties send. On the other hand, in the standard setting, we have recently seen considerable progress in improving communication complexity of consensus protocols. In particular, HotStuff [19] achieves $O(n)$ communication complexity per view under partial synchrony and VABA [3] achieves the optimal $O(n^2)$ communication complexity under asynchrony. A natural question is whether fault tolerance can be boosted (but communication costs retained) in these protocols using small trusted hardware. In this abstract, we answer these questions affirmatively for a corruption threshold $t \leq (\frac{1}{2} - \epsilon)n$ and $\epsilon > 0$.

2 Communication-Efficient BFT with Small Trusted Hardware

We first describe a communication-efficient provable broadcast primitive that will be used as a building block towards communication-efficient variants of HotStuff and VABA tolerating $t \leq (\frac{1}{2} - \epsilon)n$ Byzantine faults. We only describe the intuition here; for a detailed explanation, refer to the full version of the paper [18].

Communication-Efficient Provable Broadcast. In provable broadcast, a designated leader sends a value to all the parties and obtains a proof that a majority of the parties delivered this value. For safety, the leader should only be able to obtain a proof for one value, and for liveness, an honest leader should obtain this value even without participation from faulty parties. Our goal is to achieve this primitive with linear communication complexity while having an $O(1)$-sized proof.

A straightforward approach would be to use the non-equivocation property of the hardware, i.e., the hardware can only produce a *signed attestation* for one value at one position in the log. Thus, intuitively, if $\frac{n}{2} + 1$ parties attest to a value at a position, then no other value can have $\frac{n}{2} + 1$ attestations. However, while a party's attestation from its trusted hardware is sufficient for safety, receiving such proofs from $O(n)$ parties produces an $O(n)$-sized proof sent to the leader. This does not satisfy the $O(1)$-sized proof requirement. Our solution relies on parties diffusing the attestations to a constant number of other parties, called their *neighbors*, instead of sending the attestations directly to the leader. A party sends a *vote* to the leader if it receives attestations from a threshold of its neighbors. This vote can be a threshold signature share, which can eventually be combined by the leader into an $O(1)$-sized voting proof. Why does this work? We connect parties to each other using a constant-degree expander graph [18]. Informally, to send a (non-attested) vote, a party just needs to verify that a constant fraction of its neighbors have attested. The expander graph construction guarantees that even if an ϵn fraction of the parties vote, a majority of the parties must have attested (ensuring safety). Similarly, to guarantee liveness, the graph can be parameterized to ensure sufficiently many parties vote if all honest parties attest

Results. Our first result improves HotStuff to tolerate a $t \leq (\frac{1}{2} - \epsilon)n$ corruption while still retaining its linear communication complexity per view.

▶ **Theorem 1** (HotStuff-M, Informal). *For any $\epsilon > 0$, there exists a primary-backup based BFT consensus protocol with $O(n)$ communication complexity per view consisting of n parties, each having a small trusted hardware, such that $t \leq (\frac{1}{2} - \epsilon)n$ of the parties are Byzantine.*

HotStuff is a primary-backup protocol that progresses in a sequence of views, each having a designated leader (primary) and consisting of three rounds. HotStuff routes all messages (votes) through the leader in each of these rounds while keeping the message size $O(1)$ for a total of $O(n)$ communication per view. Abstractly, this can be viewed as a sequence of provable broadcasts (though with additional $O(1)$-sized messages). HotStuff crucially relies on threshold signatures to aggregate votes of individual parties into an $O(1)$-sized message; these signatures act as a proof for parties in subsequent rounds/views to determine whether they should vote in that round.

To increase the fault tolerance of HotStuff, we replace the steady state of its protocol with three sequential provable broadcasts led by the leader of the view. However, this alone does not suffice for safety across views. In particular, while our trusted hardware provides us with an abstraction of an append-only log that disallows appending different values at the same position (equivocation), a party can potentially present only selected (older) entries of the log during a view change. This can potentially result in a safety or liveness violation. Of course, this could be fixed by always presenting the entire log each time, but the communication complexity would grow (unbounded) with the number of views. Instead, we use a combination of techniques including: multiple logs (though only $O(1)$), one for each phase of the protocol; tying log positions to view numbers; and using one attestation to present the end state of all logs.

Our second result improves the VABA protocol of Abraham et al. [3] to tolerate minority corruption while retaining its $O(n^2)$ communication complexity. We show the following:

▶ **Theorem 2** (VABA-M, informal). *For any $\epsilon > 0$, there exists a validated asynchronous Byzantine Agreement protocol with $O(n^2)$ communication complexity consisting of n parties, each having a small trusted hardware, such that $t \leq (1/2 - \epsilon)n$ parties are Byzantine.*

In each view of VABA, in parallel, each party attempts to drive progress by acting as a leader in a "proposal promotion" (similar to a view of HotStuff). After $n - t$ proposal promotions complete, the parties elect one leader randomly and adopt the progress from the leader's proposal promotion instance during the view-change step. Depending on whether the leader completed its proposal promotion, parties decide at the end of the view or try again in another view.

There are two key challenges in augmenting VABA to tolerate a minority corruption. The challenges relate to the amount of storage in the small trusted hardware and maintaining the communication complexity of the VABA protocol. First, in HotStuff-M, only $O(1)$ logs were used. A straightforward translation would require $O(n)$ logs. If reduced to $O(1)$ logs, each party needs to send $O(n)$ attestations to other parties during a single round of the protocol. The challenge relates to the existence of arbitrary message ordering across proposal promotion instances and the fact that the trusted hardware only supports an append-only log. Our solution crucially relies on the fact that the parties have the same neighbors across proposal promotion instances, and thus, even if values from proposal promotion instances are appended arbitrarily, parties can perform the necessary validation across instances. Second, the view-change step requires every party to share "progress" from the elected leader's proposal promotion instance to all parties. However, due to the concern described earlier, only a party's neighbors can validate whether it used the trusted hardware correctly. To make matters worse, a party or its neighbors can be Byzantine. Fortunately, since parties are connected using an expander graph, we can bound the number of parties with a majority of Byzantine neighbors. By careful analysis, we ensure the delivery of the latest state of the elected leader's proposal promotion instance to all parties. We explain formal details of these results in the full version of the paper [18].

References

1 Trusted computing group. URL: `https://trustedcomputinggroup.org/`.

2 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

3 Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

4 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

5 Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. In *IEEE European Symposium on Security and Privacy*, 2019.

6 Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. *ACM SIGOPS Operating Systems Review*, 41(6):189–204, 2007.

7 Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

8 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. `doi:10.1145/42282.42283`.

9 Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

10 Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: Resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 295–308, 2012.

11 Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.

12 Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *NSDI*, volume 9, pages 1–14, 2009.

13 Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on intel sgx. *arXiv preprint*, 2020. `arXiv:2006.13598`.

14 Suresh Thiru, Shamalee Deshpande, and Stina Ehrensvard. Yubikey strong two factor authentication, January 2021. URL: `https://www.yubico.com/`.

15 Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018. See also technical report Foreshadow-NG [17].

16 Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM Conference on Computer and Communications Security*, 2017.

17 Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. *Technical report*, 2018. See also USENIX Security paper Foreshadow [15].

18 Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael K Reiter. Communication-efficient bft protocols using small trusted hardware to tolerate minority corruption. *IACR Cryptol. ePrint Arch.*, 2021:184, 2021.

19 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain. *arXiv preprint*, 2018. `arXiv:1803.05069`.