Invited Paper: Planetary Scale Byzantine Consensus

Gauthier Voron
EPFL
Lausanne, Switzerland
gauthier.voron@epfl.ch

Vincent Gramoli
University of Sydney
Sydney, Australia
vincent.gramoli@sydney.edu.au

ABSTRACT

Byzantine fault tolerant consensus protocols are implemented with consecutive broadcasts but suffer from a low throughput at large geographical scale or *planetary scale*. A reason for this inefficiency is believed to be their all-to-all communication complexity, which led researchers to design new consensus protocols with more consecutive one-to-all broadcasts but cumulatively fewer messages.

We show, through a step-by-step evaluation, ranging from LAN/WAN broadcast benchmarks to a state machine replication (SMR) application, that this intuition can be misleading. In particular, we identify two underestimated factors that can impact consensus performance much more at a large scale: *(i)* the *goodput* of the broadcast as the rate at which bits are delivered to the application and *(ii)* the *hiccup* or waiting time between consecutive broadcast phases. Finally, we show that a leaderless SMR with $O(n^4)$ complexity can outperform a leader-based SMR with $O(n^3)$ complexity by 20×.

This work promotes a new family of byzantine consensus protocols exclusively based on all-to-all broadcasts that take into account these two factors. Our result promises to impact the design of blockchain systems that aim at performing well in WANs at a planetary scale.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**; • **Computing methodologies** → *Distributed algorithms.*

KEYWORDS

State Machine Replication, Blockchain, Large scale, Broadcast

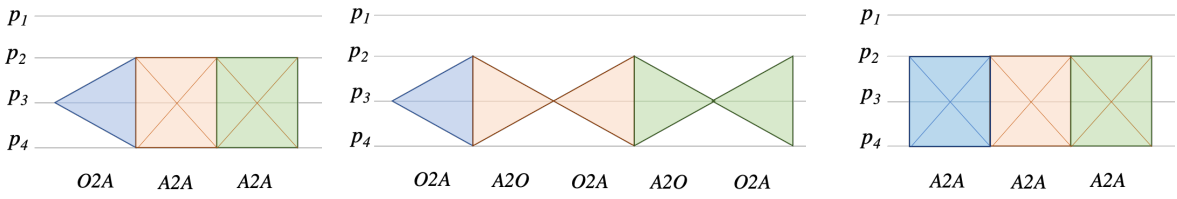
1 INTRODUCTION

There has been a recent resurgence of interest in byzantine consensus protocol proposals [16, 23, 35] largely driven by the need to obtain a blockchain system that performs efficiently at large geographical scale or *planetary scale*. It is well-known that byzantine consensus solutions are slow at a planetary scale because of their bandwidth usage [11, 30, 31, 33] but it is hard to pinpoint precisely their bottleneck. In fact, we know since 1985 that solving this problem requires n nodes to exchange a number of messages proportional to the number f of faulty nodes [14], hence leading to an inherent $\Omega(n^2)$ message complexity when f is a fraction of n. The difficulty to improve consensus performance even triggers an amusing amount of new proposals featuring very similar communication patterns [25].

The communication pattern [7, 9, 10, 27] of classic byzantine fault tolerant consensus protocols has three subsequent broadcasts: a *one-to-all (O2A)* broadcast followed by two consecutive *all-to-all (A2A)* broadcasts as depicted in Figure 1(a). As there could be $O(f)$ iterations of this pattern (with some additional steps to change the leader that initiates the O2A broadcast), these A2A broadcasts lead typically to a cubic message complexity. In order to scale, the trend is now to replace each A2A by O2A broadcasts (followed by its corresponding *all-to-one* (A2O) responses) to reduce the overall message complexity of the fast path by a linear multiplicative factor [17, 22, 33, 35] as depicted in Figure 1(b). The rationale behind this trend seems to be that by reducing the communication complexity, the limited bandwidth observed at large scale will no longer be the bottleneck.

In this paper, we show that this intuition can be misleading, instead, we promote a higher bandwidth consumption but balanced across the many disjoint routes of a WAN, as depicted in Figure 1(c). It follows from a simple observation: at large scale, saving the bandwidth at one end of the network does not necessarily compensate for the high consumption at the other end. In particular, we identify two underestimated factors that impact the performance—these are the *hiccup* or waiting times between consecutive broadcasts and the *goodput* or the payload transfer rate—and demonstrate them with broadcasting benchmarks and a state machine replication application.

First, the observed hiccup shows an inherent asynchrony among nodes preventing their sequence of broadcasts from progressing in-sync. It lowers the broadcast performance by up to 20× in a WAN (Figure 3). Second, the goodput of the consensus protocol that is expressed as the number of proposals that end up being decided per consensus instance has a crucial effect on throughput. We show empirically that in a WAN, the throughput does not even depend on n when



(a) classic broadcast pattern (b) modern broadcast pattern (c) recommended broadcast pattern

Figure 1: The typical broadcast patterns of various byzantine consensus protocols with $n = 4$ and $f = 1$ and where p_1 is ignored as it is byzantine — triangles indicate O2A and A2O communication steps whereas squares indicate A2A communication steps

nodes send and receive the same data and the message header sizes are negligible compared to the payload sizes. Note that our planetary scale study does not cover RDMA consensus protocols [1, 20] well suited for datacenters.

Finally, we illustrate this phenomenon by evaluating experimentally the performance of two Byzantine fault tolerant SMRs, HotStuff [33] that is being implemented in Facebook Libra/Diem blockchain [4] and DBFT [12] as implemented in Red Belly Blockchain [13, 29], showing a consensus algorithm with communication complexity $O(n^4)$ can outperform another with complexity $O(n^3)$ by up to 20× in a WAN of up to 150 AWS machines (Figure 7).[1]

In the remainder, we use c5.xlarge AWS instances with 4 vCPUs and 8 GiB of memory each and report the average values over three runs where error bars depict min and max. We explain the importance of broadcast in consensus (§2) and introduce the goodput (§3) before comparing the scalability of broadcasts in local area networks (LANs) and wide area networks (WANs) (§4), and under hiccups (§5). Finally, we apply these observations to consensus and state machine replication protocols (§6) and conclude (§7).

2 BROADCAST

The *broadcast* paradigm plays a key role in fault tolerant distributed protocols, as it allows a node to send some information to all nodes in a *one-to-all* fashion, or *O2A* for short. The reason why consensus relies on the broadcast paradigm is because it aims at tolerating failures. If $f < n$ nodes can fail, then it is necessary for a node to send its message to $f + 1$ nodes for at least one non-faulty or correct node to learn about it. In order to tolerate a significant portion of all the nodes failing, it is thus necessary to send the messages to at least $\Omega(n)$ nodes, and the broadcast appears ideal.

It is thus not surprising to see that the influential Practical Byzantine Fault Tolerant (PBFT) consensus protocol [10] starts with a leader sending a proposal to the other nodes with a O2A broadcast. For the nodes to learn fast whether a proposal is voted upon by sufficiently many nodes, all nodes can broadcast their vote, hence leading to a linear number of O2A parallel broadcasts, that we altogether call an *all-to-all* broadcast, or *A2A* for short. Indeed, PBFT then runs A2A broadcasts for all its nodes to learn the votes and decide as depicted in Figure 1(a). Interestingly, although there exist variants [3, 21] with different speculative executions and fast paths, this pattern is at the core of a large family of byzantine consensus protocols [7, 9, 27].

Today, however, the communication complexity is believed to play such an important role in the lack of performance of byzantine consensus at a planetary scale, that various research groups are redirecting their efforts to eradicate these costly A2A broadcasts to reduce the communication complexity at all costs [17, 22, 33, 35]: increasing the number of O2A broadcast phases, increasing the workload of a few (but mostly only one) nodes, adding corresponding A2O response steps that increase latency, etc. Figure 1(b) presents the resulting sequence of O2A broadcasts followed by their response. This is the precise broadcast pattern of SBFT [17] that aims at achieving high performance at planet scale. This pattern is shared by HotStuff [33] that is itself at the core of Pompē [35].

A new type of consensus communication pattern follows exclusively an A2A message pattern (Figure 1(c)), hence allowing all nodes to collaboratively exchange their input data in what we will call the cumulative A2A or CA2A (Section 3). This pattern has been used in DBFT [12], Dispel [30] and Lyra [34].

3 GOODPUT

As we aim at improving the performance of the consensus protocols at a planetary scale we focus on the broadcast *goodput*, the amount of payload the broadcast can convey to the overlying application per unit of time. (Note that the payload typically excludes metadata information like the

[1] HotStuff has an $\Theta(n^3)$ communication complexity because it needs $\Omega(n)$ synchronizations each of $\Omega(n^2)$ bits for enough participants to reach the same good view [33].

identity of the sender or the information stored in the header of the message.) To this end, let us consider the goodput that an A2A broadcast can deliver depending on two ways it is used during the execution of a consensus protocol:

(1) **redundant all-to-all broadcast (RA2A):** all the nodes broadcast the same data. This is the case when nodes of the classic pattern send a constant-size confirmation message to all other nodes and wait to receive this confirmation from a minimum number $(2f + 1)$ of nodes to commit (cf. last A2A of Figure 1(a)).
(2) **cumulative all-to-all broadcast (CA2A):** all the nodes broadcast distinct data. This is the case when each node of the recommended pattern sends a distinct proposal message to all other nodes and waits to receive $(2f + 1)$ proposals from other nodes (cf. first A2A of Figure 1(c)).

As a result, the goodput of the RA2A broadcast is $O(1)$ whereas the goodput of the CA2A broadcast is $\Omega(n)$. As the CA2A goodput grows with n while the RA2A goodput does not, it seems intuitively that CA2A could be better suited than RA2A to increase the performance as the system size increases, a property commonly known as *scalability*. Note that this observation has already been quantified in [18]. This observation could also explain why crash fault tolerant consensus leveraging multiple leaders [26] as well as byzantine fault tolerant leaderless algorithms [2, 12] were efficient at large scale. Let us try to confirm this scalability observation empirically.

4 SCALABILITY

We refer to *scalability* as the ability for a distributed system to treat more requests per time unit as its size increases. More precisely, we consider the system size as the number of nodes that runs its protocol (e.g., a consensus protocol). There exist, however, other definitions for distributed system scalability [28] and an equally important one consider the size as the distance between machines. This distance is often expressed in terms of network latency: the distance between two nodes increases proportional to the network latency and, sometimes, inversely proportional to the bandwidth. To take this distance into account, we measure the scalability of the goodput downloaded per node in a LAN setup where nodes are close to each other, and in a WAN setup where nodes are far from each other, by having each node continuously streaming bytes on all its TCP connections in parallel. Note that streaming is used here for simplicity, we revisit this streaming behavior by introducing hiccups in §5.

4.1 LAN broadcast

To compare the performance of O2A, CA2A and RA2A broadcasts in a LAN, we measure their goodput as the payload

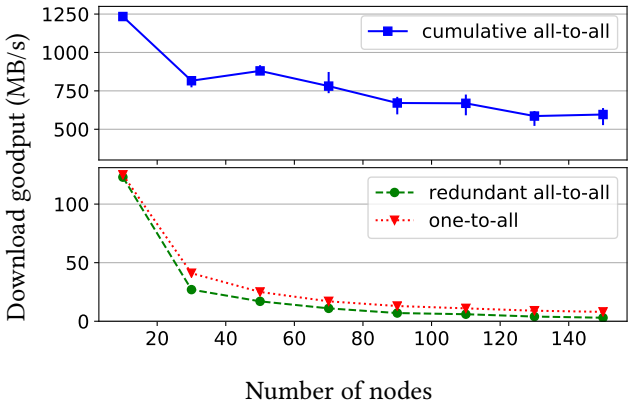


Figure 2: Download goodput in a LAN setup of the one-to-all (O2A), redundant and cumulative all-to-all (RA2A and CA2A) broadcasts

throughput obtained without counting the TCP/IP headers. The LAN consists of up to 150 AWS c5.xlarge virtual machines communicating with a latency lower than 1 ms and a 10 Gbps bandwidth.

Figure 2 shows the download goodput of the broadcasts depending on the number of nodes. We observe that for the O2A, RA2A and CA2A broadcasts, the download goodput decreases following an hyperbolic curve. Indeed, in a broadcast, each sender shares its bandwidth between a growing number of connections. To transmit a constant number of bits to n receivers, each sender thus sends $O(n)$ bits over the network, leading to a throughput inversely proportional in n. Additionally, in the A2A broadcasts, all n senders compete in using the network, in this case, the LAN switches, resulting in a total of $O(n^2)$ exchanged bits. However, despite the difference in communication complexity, the difference of throughput between the O2A and the RA2A broadcasts is relatively small. This is because the wire between each sender to the network switch has a 10 Gbps bandwidth, smaller than the transfer speed of the switches (100 Gbps). Consequently, each sender reaches the bottleneck of its own wire before saturating the network switches. This limits the competition between the senders in the A2A broadcasts.

The CA2A broadcast solves a problem that differs from the one solved by the O2A and RA2A broadcasts, as the receiver of the CA2A receives a number of bits that grows with the system size. As a result, it is not surprising that the goodput of the cumulative CA2A broadcast is significantly larger than the two other broadcasts. Despite this difference, an A2A broadcast used as a procedure in a consensus protocol is accounted the same by a communication complexity analysis whether it is redundant or cumulative. The similar shape

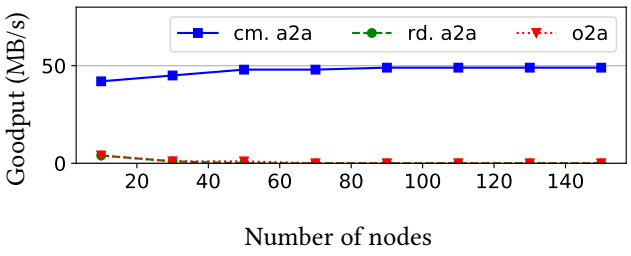


Figure 3: Download goodput in a WAN setup of one-to-all (O2A), redundant and cumulative all-to-all (RA2A and CA2A) broadcasts

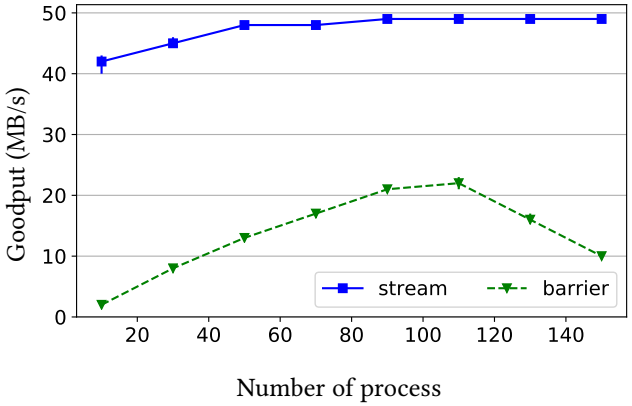


Figure 4: Download goodput in a WAN setup of a cumulative all-to-all (CA2A) broadcast with hiccup and stream when the number of participant changes

between all curves explains why the classic pattern (Figure 1(a)) seemed well-suited in LANs (e.g., for implementing NFS [10]).

4.2 WAN broadcast

In the second experiment we compare the throughput of the O2A, CA2A and RA2A broadcast in a WAN with 200 ms round-trip-time latency and a 400 Mpbs bandwidth, set with the linux `tc` command, on up to 150 AWS c5.xlarge virtual machines.

Figure 3 shows the download goodput of the cumulative all-to-all (CA2A) broadcast depending on the number of nodes. We observe that the goodput slightly grows until 50 nodes and then stabilizes. This differs from the CA2A behavior we observed in a LAN setup (see Figure 2) where the goodput decreases when the number of nodes increases. In a WAN setup, each process is connected to the network through a unique dedicated connection, called local loop (or "last mile"), typically provided by the ISP. The bandwidth of the local loop is small compared to the bandwidth between autonomous system routers.

Additionally, as long as nodes are geographically spread enough, it is unlikely for two routes between any pair of processes to share a common link or a router, except in local loops. As a result, there is almost no competition between the senders to use the network in a WAN setup. Consequently, for n nodes, assuming each process has the same constant local loop bandwidth of B, any given node receives B/n distinct bits per second from n senders, resulting in a download throughput of B. This is why in this WAN setup, *the performance does not even depend on the number n of nodes.*

5 HICCUP

A consensus execution is more complex than a broadcast execution, as it comprises a sequence of broadcast executions such that a node starts a second broadcast only after delivering messages from at least $\min(f+1, n-f)$ distinct nodes of the previous broadcast or responses as depicted in Figures 1(a) and 1(b). Each node executes the consensus protocol by interleaving broadcast phases with periods during which it actually waits for messages. To evaluate the impact of this waiting time, we add it to the previously tested broadcasts in the following experiments.

5.1 Hiccup vs. stream

We refer to a *hiccup* as the interleaving of broadcasts and waiting periods in each consensus execution because each node alternates between active periods where it sends messages and passive periods where it waits to receive messages. We refer to each waiting period as a *barrier*. We now modify the broadcast benchmark to mimic the hiccup execution. More precisely, each sender sends a byte sequence of a fixed size, called *batch*, to all the other nodes then waits to receive a batch from every other nodes before sending the next batch.

Figure 4 depicts the effect of such a hiccup on the goodput of a CA2A broadcast execution in the WAN setup with 100 KB batches. We observe that the goodput measured in the stream execution (§2) is significantly higher than in the hiccup execution for any system size. This can be explained by the barriers that slow down the hiccup execution and that are absent from the stream execution. Moreover, as the system size increases, the effect of the barriers is more pronounced and the goodput of the hiccup execution starts dropping while it remains stable in the stream execution. This observation raises a question: What are the causes of slow down in the hiccup execution in a WAN?

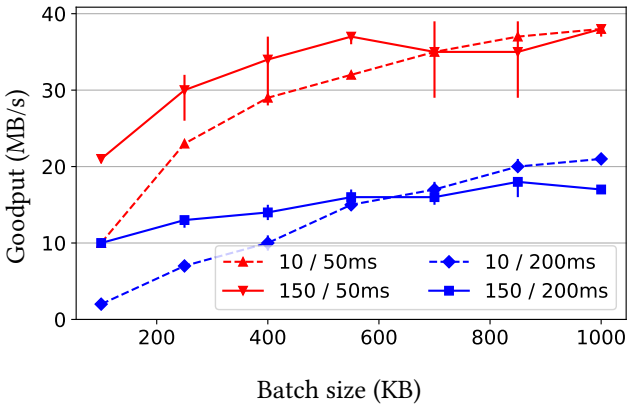


Figure 5: Download goodput in a WAN setup of a cumulative all-to-all broadcast with barriers for different number of participants and RTT when the batch size changes

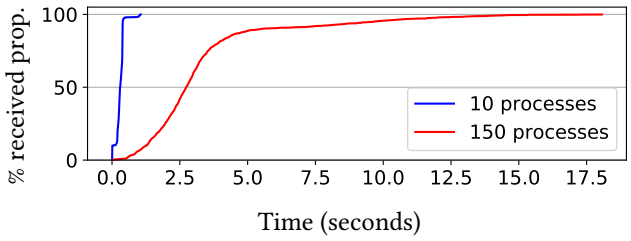


Figure 6: Cumulative distribution of the number of received batches by one node running the consensus over time for 10-node and 150-node systems in a WAN

5.2 Impact of the batch size on the hiccup

Finally, to better understand how the batch size and the distance between nodes impact the performance of the hiccup CA2A broadcast, we vary the batch size, the number of nodes and round trip time latency between all pairs of nodes.

Figure 5 shows four curves, with a mix of 50 ms and 200 ms latencies and 10-node and 150-node system sizes as we increase the batch size from 100 KB to 1 MB. First, we observe that, as the batch size enlarges, the performance increases. This indicates that fewer barriers is beneficial to performance and confirms that as the batch size enlarges the hiccup execution gets more similar to the stream execution we measured in Figure 4. Second, the communication latency between pairs of nodes impacts dramatically the performance. This is due to longer latencies inducing a larger asynchrony between nodes that increases the length of the barriers.

6 CONSENSUS

We finally compare recent byzantine state machine replications (SMRs). Note that we are aware of crash fault tolerant SMRs that partially order commands [15, 24, 26] but we focus on byzantine and totally ordering SMRs for now: We use HotStuff [33] that is at the heart of the Facebook Libra blockchain and the Diem blockchain [4] and DBFT [12] that is at heart of the Red Belly blockchain [13] and the Smart Redbelly blockchain [29] and reused the source code from the authors of HotStuff [33] and the Dispel-seq baseline [30]. HotStuff is the typical consensus protocol building upon O2A-based broadcasts (Figure 1(b)) to lower the communication complexity of PBFT from $O(n^4)$ to $O(n^3)$, and decides one *proposal* (one of the initial batches). DBFT stands for the Democratic Byzantine Fault Tolerant consensus algorithm, which is the first formally verified blockchain consensus protocol [5, 6]. It adopts the message pattern of Figure 1(c) in that all nodes propose a batch by reliably broadcasting [8] to others and spawn a binary consensus instance with input value 1 for each rapidly reliably delivered proposal. It can decide upon all $O(n)$ proposals [30] but both its message and communication complexities are $O(n^4)$.

6.1 Impact of the scale on the hiccup

To confirm that the hiccup effect increases with the number of nodes also in the consensus protocols, we now measure the time it takes for a node to receive the batch messages. Each node logs the reception time of batches from every node in each DBFT consensus instance, using its own clock. The first received batch is considered as $T = 0$ and other reception times adjusted accordingly.

Figure 6 depicts the cumulative distribution function (CDF) of the time taken to receive n batches on two system sizes, $n = 10$ and $n = 150$ in the WAN setup. As the size and the number of batches to wait for increases, we observe that the time spent waiting at a barrier increases. The difference between the two curves outlines the magnitude of the hiccup phenomenon in WAN networks: the differences in the time batches take to reach their destination prevents nodes to proceed in-sync as opposed to what was previously observed with in LANs [32].

6.2 Comparing HotStuff and DBFT

Figure 7 compares the performance of DBFT and HotStuff in a WAN setup of 10 nodes. DBFT offers a lower latency and a higher throughput than HotStuff and its throughput improvement increases with the batch sizes. This is expected given that the CA2A broadcasts outperform significantly the O2A broadcasts at $n = 10$ in a WAN setup (cf. Figure 3). Just like the contrast between the CA2A and the RA2A, DBFT may decide $O(n)$ proposals, due to its initial A2A (Figure 1(c))

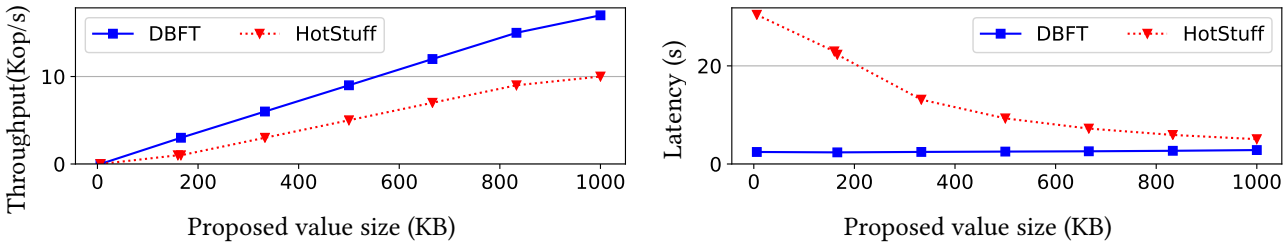


Figure 7: Throughput and latency of HotStuff and DBFT with a system size of $n = 10$ nodes in a WAN setup when the proposed batch size changes

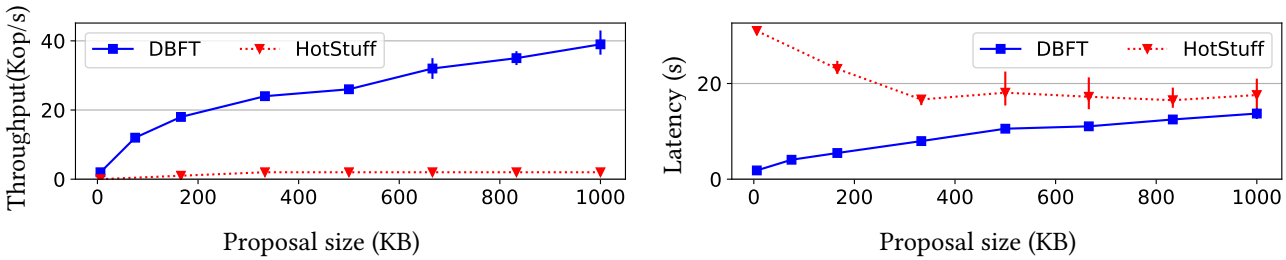


Figure 8: Throughput and latency of HotStuff and DBFT with a system size of $n = 150$ nodes in a WAN setup when the proposed batch size changes

whereas HotStuff decides $O(1)$ of them due to its initial O2A (Figure 1(b)). Finally, DBFT's performance increases with the batch size as expected, given that a higher batch size reduces the number of barriers in the hiccup of the C2A2 broadcast (cf. Figure 5).

Figure 8 compares the performance of DBFT and HotStuff in a WAN setup of 150 nodes. The performance improvement of DBFT over HotStuff is significantly higher than we observed at the smaller system size (Figure 7). This is expected given that, as the scale enlarges, the O2A broadcast of HotStuff slows down whereas the CA2A of DBFT speeds up (cf. Figure 3). As a result, the protocol with the $O(n^4)$ communication complexity is up to 20× faster than the protocol with the $O(n^3)$ communication complexity. Note that these complexities are obtained in the worst case in the presence of byzantine failures whereas none of the experiments included the byzantine behaviors leading to the worst-case execution. That said, even in the good case executions experimented here, the communication complexity of HotStuff is lower than the DBFT complexity.

7 CONCLUSION

In this work, we showed that the folklore belief that communication complexity limits performance of byzantine consensus in a WAN is unjustified. In particular, we identify two underestimated factors that can impact consensus performance much more at a large scale: (i) the goodput of the broadcast and (ii) the hiccup or waiting time between consecutive broadcast phases. We showed that a leaderless SMR, whose consensus has been recently formally verified, with $O(n^4)$ complexity outperforms a leader-based SMR with $O(n^3)$ complexity by 20×.

A concomitant work included the application of these findings to the development of the Redbelly Blockchain [13]. These efforts recently demonstrated on realistic workloads [19] superior performances to classic blockchains that typically make use of leader-based consensus protocols [29].

ACKNOWLEDGEMENTS

This work is supported in part by the Australian Research Council Future Fellowship funding scheme (#180100496) entitled "The Red Belly Blockchain: A Scalable Blockchain for Internet of Things". Vincent Gramoli is a Principal Investigator of the Algorand Centre of Excellence on Sustainability Informatics for the Pacific managed by the Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

REFERENCES

[1] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. Ubft: Microsecond-scale bft using disaggregated memory. In *ASPLOS*, page 862?877, 2023.

[2] Karlos Antoniadis, Julien Benhaim, Antoine Desjardins, Poroma Elias, Vincent Gramoli, Rachid Guerraoui, Gauthier Voron, and Igor Zablotchi. Leaderless consensus. *Journal of Parallel and Distributed Computing*, 176:95–113, 2023.

[3] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 BFT protocols. *ACM Trans. Comput. Syst.*, 32(4):12:1–12:45, January 2015.

[4] Shehar Bano, Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain, 2019. https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain.pdf (accessed 10/2019).

[5] Nathalie Bertrand, Vincent Gramoli, Marijana Lazić, Igor Konnov, Pierre Tholoniat, and Josef Widder. Brief announcement: Holistic verification of blockchain consensus. In *Proceedings of the ACM Symposium on Distributed Computing (PODC)*, 2022.

[6] Nathalie Bertrand, Vincent Gramoli, Marijana Lazić, Igor Konnov, Pierre Tholoniat, and Josef Widder. Holistic verification of blockchain consensus. In *Proceedings of the 36th International Symposium on Distributed Computing (DISC)*, 2022.

[7] Alyson Bessani, João Sousa, and Eduardo E. P. Alchieri. State machine replication for the masses with BFT-Smart. In *DSN*, pages 355–362, 2014.

[8] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, October 1985.

[9] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. Master's thesis, University of Guelph, 2016.

[10] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[11] Paulo R. Coelho, Tarcisio Ceolin Junior, Alysson Bessani, Fernando Luís Dotti, and Fernando Pedone. Byzantine fault-tolerant atomic multicast. In *DSN*, pages 39–50, 2018.

[12] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *NCA*, pages 1–8, 2018.

[13] Tyler Crain, Chris Natoli, and Vincent Gramoli. Red belly: A secure, fair and scalable open blockchain. In *IEEE Symposium on Security and Privacy (SP)*, pages 466–483, 2021.

[14] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, pages 191–204, January 1985.

[15] Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, and Pierre Sutra. State-machine replication for planet-scale systems. In *EuroSys*, 2020.

[16] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[17] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *DSN*, pages 568–580, 2019.

[18] Vincent Gramoli. *Blockchain Scalability and its Foundations in Distributed Systems*. Springer, 2022.

[19] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. Diablo: A benchmark suite for blockchains. In *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys)*, pages 540–556, 2023.

[20] Sagar Jha, Jonathan Behrens, Theo Gkountouvas, Mae Milano, Weijia Song, Edward Tremel, Robbert Van Renesse, Sydney Zink, and Kenneth P. Birman. Derecho: Fast state machine replication for cloud services. *TOCS*, 36(2), 2019.

[21] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 45–58, 2007.

[22] Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Trans. Computers*, 68(1):139–151, 2019.

[23] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with stellar. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 80–96, 2019.

[24] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Mencius: Building efficient replicated state machines for WANs. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 369–384, 2008.

[25] James Mickens. The saddest moment, 2013. https://scholar.harvard.edu/files/mickens/files/thesaddestmoment.pdf.

[26] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 358–372, 2013.

[27] João Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *SRDS*, pages 146–155, 2015.

[28] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms, 2nd Edition*. Pearson Education, 2007.

[29] Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. Smart Redbelly blockchain: Reducing congestion for Web3. In *Proceedings of the 37th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2023.

[30] Gauthier Voron and Vincent Gramoli. Dispel: Byzantine SMR with distributed pipelining. Technical Report 1912.10367, arXiv, 2019.

[31] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *IFIP WG 11.4 International Workshop on Open Problems in Network Security*, pages 112–125, 2015.

[32] Tian Yang, Robert Gifford, Andreas Haeberlen, and Linh Thi Xuan Phan. The synchronous data center. In *HotOS XVI*, pages 142–148, 2019.

[33] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the ACM Symposium on Distributed Computing (PODC)*, pages 347–356, 2019.

[34] Pouriya Zarbafian and Vincent Gramoli. Lyra: Fast and scalable resilience to reordering attacks in blockchains. In *Proceedings of the 37th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2023.

[35] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 633–649, 2020.