StarReact: Detecting Important Network Changes in BFT Protocols with Star-Based Communication

Martin Nischwitz
Physikalisch-Technische Bundesanstalt
Berlin, Germany
martin.nischwitz@ptb.de

Marko Esche
Physikalisch-Technische Bundesanstalt
Berlin, Germany
marko.esche@ptb.de

Florian Tschorsch
Technische Universität Dresden
Dresden, Germany
florian.tschorsch@tu-dresden.de

2024 IEEE 49th Conference on Local Computer Networks (LCN) | 979-8-3503-8800-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/LCN60385.2024.10639767

***Abstract*—Threshold signatures have improved the scalability of BFT protocols by replacing all-to-all broadcast with star-based communication. On the flip side, this approach renders network re-organization and performance optimization more costly, because information can only be exchanged between the leader and all other nodes.**

We present StarReact, an extension for BFT protocols that relies on star-based communication to collect and disseminate quorum certificates. This extension allows all nodes of the system to measure and evaluate the network state by monitoring the messages disseminated by the leader. Each node decides independently if performance degradation justifies a leader change. By deploying a median filter, the measured commit latency of the system is evaluated by each node to determine if a network change has a lasting effect that warrants a change in leadership or should be ignored. We showcase how StarReact is able to identify important network changes for different deployment scenarios and explain how view change executions should be adapted to different environments to improve commit latency and potentially throughput for such systems.

***Index Terms*—distributed systems, fault tolerance, latency**

I. Introduction

The term Byzantine fault tolerant (BFT) describes the ability of a distributed system to tolerate a certain number of Byzantine faults, i.e., faults in which a process may behave arbitrarily. The motivation for new implementations of BFT protocols has been manifold, including applications in permissioned blockchains, cryptocurrency, etc.

While recent research has mainly focused on increasing the throughput and scalability of BFT algorithms, the system response time, or in other words the commit latency, is often of minor concern. In most evaluations, the network characteristics, including the latencies, are assumed to be static after some stabilisation time. While this may hold true in some theoretical scenarios, the actual end-to-end latency between nodes may change in practice due to, e.g., router reconfigurations, congestion or packet losses.

An approach to optimise the performance regarding the commit latency was presented with AWARE [12], which is designed to measure network latencies during runtime and find an optimal configuration. But since the latest generation of BFT protocols (e.g., SBFT [9], HotStuff [10] or Kauri [11]) have reduced their communication complexity by utilising threshold signatures, the measurement of all network latencies would once again increase the communication complexity.

The challenge considered for this paper was to design a virtually cost-free algorithm that is able to detect changes in the network and is tailored for the latest generation of BFT protocols. To this end, we present StarReact, an extension for BFT protocols with a star-based communication pattern, which is individually executed by each node, to detect dynamic network changes in real time to optimise performance of the executed BFT protocol regarding the overall latency of the system. Based on a quorum-specific network abstraction, StarReact measures and filters the commit latency of the system during runtime and enables a suitable reaction. StarReact was implemented and tested on top of HotStuff [10] and evaluated and optimised within the emulation/simulation environment Delphi-BFT [15].

In detail, our contributions are

- a quorum-specific network abstraction that simplifies the calculation of the quorum latency,
- a novel approach, based on that network abstraction, how to measure and assess the performance of the system and detect important network changes,
- the design of a suitable filtering algorithm to balance the detection of decreased quorum latencies while simultaneously not overreacting to short-term network anomalies and
- the application of that filtering algorithm in the hybrid simulation/emulation environment Delphi-BFT to showcase that network events that negatively affect the performance are successfully detected and acted upon.

II. Related Work

An approach for dynamic leader election based on performance characteristics was presented by Eischer et al. [25]. They proposed a scheme in which the clients of the system observe and evaluate the end-to-end latency of the system by probing the system with special messages. The replicas then reconfigure themselves to select the best suited leader.

Extensive research has been performed by Berger et al. to optimise the latency of BFT algorithms using adaptive voting weights to fasten the quorum aggregation process [12]. In the same vein followed the development of AWARE [21], a scheme to measure the global latencies of the system, utilising the all-to-all broadcast of the classic BFT protocols. An adaptation of AWARE was also published in [13], albeit

still requiring the same communication complexity as the default AWARE.

Another method to address dynamic changes is to switch the consensus algorithm altogether during execution by categorising different protocols according to their strengths and weaknesses [7] [23]. A similar study was undertaken that evaluated how protocols can be adapted during execution [24].

In comparison, the main benefit of StarReact is the virtual cost-free applicability to protocols with star-based communication.

III. Byzantine Fault Tolerance

BFT protocols are often times designed to implement state machine replication (SMR) given the possibility of byzantine faults, i.e., a subset of replicas may be under control of a malicious entity and may show arbitrary behaviour. PBFT [1] was the first BFT protocol that was implemented with the purpose to be practical and comprehensible. Subsequent protocols improved upon properties like optimistic responses (Zyzzyva [2]), increased robustness (Aardvark [3], RBFT [4]), hardware-related trust systems (CheapBFT [5]), modularity (BFT-SMaRt [6]), adaptive protocol switching (BFT selection [7]) and wireless application (RATCHETA [8]). Although BFT has been notoriously known for its limited scalability, recent publications have successfully addressed this issue (SBFT [9], HotStuff [10] and Kauri [11]) by reducing the communication complexity.

Two important properties to describe a BFT protocol are *safety* and *liveness* [1]. Safety states that the replicated system behaves like a single machine, i.e., all correct nodes execute their system state in the same order. Liveness guarantees that the system will eventually make progress and advance its system state. Both properties can only be ensured, if the system consists of at least $N = 3f + 1$ nodes for a maximum number of f faulty nodes [16]. In order to uphold the safety property, it follows that at least $2f + 1$ nodes are in a consistent state. This is often solved through the aggregation of quorums of at least that size, which can be used as proof for the system state. It is also possible to reduce the quorum size with, e.g., trusted hardware components (see CheapBFT), redistributing the voting power of the nodes [12] or backtracking commits with forensics [18] (see FlashConsensus [19]).

Another defining aspect that that has also evolved over time is the communication pattern. PBFT was designed based on an all-to-all broadcast, where two such phases constitute the main consensus algorithm (in addition to an initial dissemination phase). While effective in disseminating information, this pattern scales (at least) quadratic with the number of nodes [17] making it impractical for larger networks. To remedy that, SBFT, and later HotStuff, utilised threshold signatures to implement a star-based communication pattern and reduce the communication complexity to $O(n)$. While improving the scalability, the star-based pattern also has some disadvantages. Firstly, the overall latency increases, because aggregation and dissemination is separated in two phases. Secondly, the available information on the whole system in each node is reduced because a node is no longer required to exchange messages with every other node.

Regarding the performance, a lot of research has been conducted on improving the throughput and scalability of BFT protocols, while latency is only rarely of interest, which is surprising considering the fact that latency and throughput are probably of high interest in real-world SMR applications. BFT protocols incorporate a quorum aggregation process, i.e., responses from all nodes are being gathered in order to ensure safety. To reduce the latency of the overall algorithm, the latency for aggregating these quorums should be kept to a minimum.

IV. System Model

A. Processes

We assume a set of n nodes that will receive requests from an unspecified but limited number of clients. As usual for BFT protocols, the number of faulty nodes f is limited to $f \leq \left\lfloor \frac{n-1}{3} \right\rfloor$.

B. Network

In this paper, the network shall follow the partially synchronous system model [20], i.e., messages will be delivered within a fixed upper time bound after some unknown *global stabilisation time*. Each process shall have a reliable and authenticated peer-to-peer connection to all other processes. Reliable means that even though network distortions such as, e.g., packet loss, might occur, message retransmissions are covered by a suitable transport layer implementation, e.g., Transmission Control Potrocol (TCP), and should be delivered eventually. We consider the network properties to be dynamic regarding the round-trip times (RTTs) of the connections. While the connections might be stable for some time we are interested in the impact of, e.g., changing RTTs or packet loss bursts. This might also occur due to a changed network topology or because of a re-routing of the network connections. As such, we also consider highly dynamic networks as they might be found in mobile networks [8].

V. Aggregating Quorums

An elementary part of most consensus protocols, is the aggregation of quorums to validate the current system state. In order to aggregate a quorum in BFT protocols, a node has to wait for $2f + 1$ votes from $3f + 1$ other nodes. The time needed to commit a state change in the protocol is therefore dominated by this process.

In classic BFT protocols, which are based on all-to-all broadcast messages, $2f + 1$ nodes need to aggregate quorums and disseminate their results afterwards. With a star-based communication pattern, only a single node (the so-called leader) has to perform that task.

A. Most Significant Link

In order to reduce the commit latency, only nodes that are able to quickly aggregate quorums should be tasked to do so. For protocols with all-to-all broadcast communication,

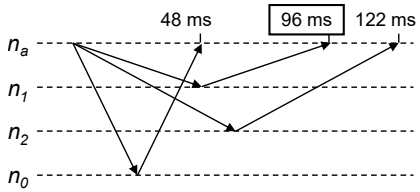

Fig. 1: Visualisation for quorum aggregation latency in a typical BFT system with $n = 4$, $f = 1$ nodes. Node n_a will aggregate the quorum 96 ms after dissemination, with reception of the second message from n_1.

there is only limited room for optimisation because the system has to wait for $2f + 1$ nodes to aggregate their individual quorums. AWARE [21] implemented a prediction scheme, in combination with voting weight distribution, to select a specific set of nodes that aggregate quorums and reduce the commit latency. This prediction scheme utilises the all-to-all broadcast scheme of BFT-SMaRt to measure the latencies between all nodes and establish a latency matrix of the whole system in each node. Based on this matrix, AWARE selects the optimal protocol configuration.

The only knowledge required to predict the latency for aggregating a quorum (from now on referred to as quorum latency) are the connection latencies of each connected node. Considering in a network of $n = 3f + 1$ nodes, a node n_a needs to aggregate a quorum $q = 2f + 1$ votes. If n_a has free processing power, i.e., it can immediately process any received message, the quorum latency is equal to the connection latency between n_a and the node that sent out vote number $2f$. The process is visualised in Figure 1. From now on, we will refer to this connection as the *most significant link (MSL)*.

B. Observing Latency

Determining the MSL is simple if all latencies in the system can be measured during execution. If, on the other hand, star-based communication takes place, those measurements are no longer possible because there is no regular message exchange between all nodes. An alternative to direct measurements is to observe the commit latency or quorum latency within the system.

With star-based communication, each node will receive messages from the leader exactly once between the collection of quorums. Consequently, each node is able to timestamp the received messages and create a time series t_i with i denoting the index of the quorum round. Based on those measurements, the nodes can further create a differential time series

$$d_i = t_i - t_{i-1}, \tag{1}$$

i.e., a series of values denoting the time between successive quorum rounds. As long as the network is stable the quorum time, and therefore d_i, will remain constant.

By calculating and monitoring d_i, each node is able to make assumptions regarding its environment and can take appropriate action. In order to give a better understanding how the nodes can and should react to changes in the measured quorum time, the following section will explain how different cases of network scenarios will affect the quorum time and with it d_i.

TABLE I: Setup for showcasing the effect of different network changes on the measured quorum time d_i.

Node	n_0	n_1	n_2	n_3
RTT to n_0 in ms	-	23	109	297

C. Differentiation of Observable Cases

In order to better visualise the observable changes to both time series t_i and d_i, we have simulated changes to the RTT of selected links in a network consisting of four nodes deployed in Delphi-BFT. A detailed overview of the setup is given in Table I: node n_0 is the designated leader for all experiments and the connection between node n_0 and node n_2 is the MSL in the starting configuration.

Each change of the RTT of a link was simulated by Delphi-BFT and the measured values of t_i and d_i were recorded by each node. To keep a uniform description in all of the following scenarios, the network change is always considered to be effective starting with timestamp t_e (and accordingly the derived timestamp d_e). A differentiation of the network changes has been done, depending on which connection is affected by the change. The first category encompasses all instances of network changes that do *not* affect the MSL. The second category covers changes that, whether directly or indirectly, impact the MSL. Lastly, the third case is a special subset of the second category and addresses the observations made by the node that represents the MSL when the RTT of that link is changing.

All plots referenced in the following scenarios will depict two signals, the RTT of the link that is currently changing is recorded on the left hand side while the differential time series d_i, that is recorded by the node affected by the change, is recorded on the right hand side.

1) Regular link: This category covers all network changes of links between nodes that do not directly affect the MSL. A change affecting the link between two non-leader nodes will have no detectable impact on the system.. If, on the other hand, the link between the leader and another node is affected, there can be two further outcomes, depending on the nature of the change.

The affected link might indirectly influence the MSL of the current configuration. This can happen if the new RTT of the changed link has turned it into the new MSL. In that case, the previous MSL will be replaced and the quorum time of the system will change accordingly. This category of network changes is further described in the next section.

Otherwise, if the MSL is not affected, the quorum time of the system will remain as is and only the two nodes connected by this link will be able to detect any changes. Before the network change, all timestamps t_i with $i < e$

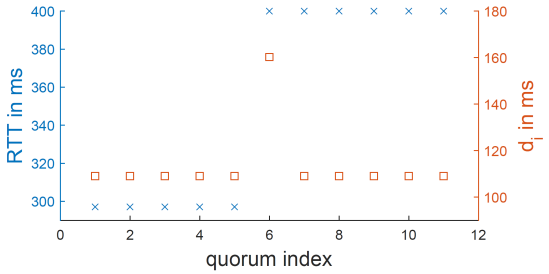

Fig. 2: Observed values of d_i by node n_3 in case the RTT between node n_3 and n_0 (leader) is increased (not the MSL).

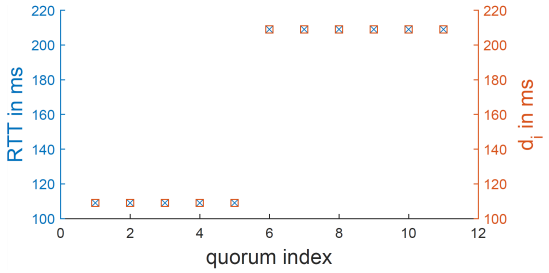

Fig. 3: Observed values of d_i by node n_2 in case the RTT between node n_2 and n_0 (leader) is increased (MSL).

are received in constant intervals, determined by the current quorum time of the system. Once the RTT between the leader and a node has changed, effective with the reception of t_e, the time between t_{e-1} and t_e will be lower or higher depending on whether the RTT has increased or decreased, respectively. Following that event, all subsequent timestamps t_i with $i > e$ will once again arrive with the same interval as before the change. The quorum time remains unchanged and the leader will disseminate messages with the same frequency as before. Consequently, d_i is constant for $i \neq e$ and shows only a single peak for d_e, indicating the increased or decreased value of the RTT.

A network change of that nature was simulated for the system described in Table I by decreasing and increasing the RTT of the link between nodes n_0 (the leader) and node n_3. Figure 2 depicts the RTT for the connection between both those nodes and shows an increase from the starting 297 ms to 400 ms. The previously described peak in the differential time series can be seen the moment the delay changes (t_e). A decrease in the delay will show a mirrored behaviour.

2) Most significant link: A shift of the RTT of the MSL can be caused by multiple reasons but two basic differentiations can be made: either a) the MSL itself changes its RTT but the link remains the MSL or b) the network change causes a reordering of at least two links, including the previous MSL. In the latter case, the connection that was previously the MSL will no longer be the MSL and another connection will take over that role. As explained in Section V-A, the MSL is directly related to the quorum time of the system and whenever the RTT of the MSL changes, the quorum time will mirror that change. All nodes will be able to register such an event since

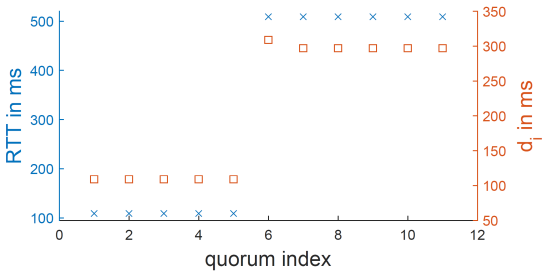

Fig. 4: Observable effect for an increased RTT of the link between node and leader which is simultaneously the MSL.

the differential time series d_i for $i > e-1$ has now increased or decreased according to the new value of the RTT of the MSL of the system. Barring any further network changes, the quorum time will remain constant for t_i with $i > e$ and so will d_i.

To validate the described behaviour, a shift of the RTT was simulated in the system from Table I for the link between the leader n_0 and node n_2, i.e., the MSL of that system. Figure 3 depicts the impact of increasing the RTT of that link by 100 ms. The measured values for d_i follow the configured RTT of the MSL perfectly, as expected. It should be noted that the plotted values for d_i in both figures are measured identically by all nodes of the system (as long as no other network changes overlap with the MSL changes). The time series d_i remains constant for $i > e$. As before, a decrease in the delay will show a mirrored behaviour.

3) Observer of the most significant link: Finally, we consider the merged case of the two previous scenarios. If the MSL of the system switches to another connection, the node that previously held the MSL with the leader will make a special observation. We simulated this scenario in our system by raising the RTT between node n_0 and node n_2 above that between n_0 and n_3. Consequently, the MSL will shift to n_0 and n_3. The process is visualised in Figure 4. The RTT between node n_0 and node n_2 was increased by 400 ms, setting the MSL to 297 ms, the RTT between node n_0 and node n_3. To explain the recorded values of the time series d_i of node n_2, two effects have to be considered. Firstly, d_i will show a singular increase that results from the delayed message reception due to the increased RTT to the leader. Since the RTT was increased by 400 ms, the first message at d_e will arrive 200 ms later. Secondly, the MSL has changed from 109 ms to 297 ms, an increase of 188 ms. This will lead to a constant increase of d_i to 297 ms for $i > e$. In the experiment the 200 ms spike (at quorum index 6) occurred before the quorum time adapted to the new MSL (quorum index 7). Depending on the timing, both effects might overlap leading to even higher spikes.

VI. StarReact

Given that each node records the differential time series d_i as described in Section V-B, it is now possible to formulate how d_i needs to be processed in order to obtain an indication

in case the network connections have changed. The analysis in Section V-C has shown that two kinds of effects have an impact on the value of d_i. While the quorum time of the system will dominate the base value of d_i, the analysis in Section V-C has shown that two kind of effects have a lasting impact on it. In addition, d_i will be affected by noise caused by the network or processing times as well as singular outliers (as shown in Figure 2). The goal is to process d_i in a manner to detect increased quorum times while ignoring the noise.

A. Removing the Noise

According to our network model, the latencies between nodes are considered dynamic, i.e., while the delay has a "base value" there might be stochastic variations as they are common for, e.g., a mobile setting. A simple filter to reduce noise and remove outliers is the median filter.

The order of the median filter determines the overall size of the sliding window that is used to determine the median. A higher order filter will be more robust against noise and outliers while at the same time requires more values to be accumulated before a response is formulated. In order to evaluate the value of d_i with a median filter of order m, at least $\lfloor m/2 \rfloor$ values after d_i, i.e., up until $d_{i+\lfloor m/2 \rfloor}$, need to be available to apply the filter without any padding, assuming that all past values of d_i, up to $d_{i-\lfloor m/2 \rfloor}$, are stored. In summary, a higher order offers more stability but will also slow down the response of the median filter.

B. Preventing frequent leader changes

The purpose of StarReact is to trigger if a degradation of the MSL is detected. In order to control the trigger happiness, a threshold parameter is added to limit the number of reconfigurations in the system (a similar approach was also implemented in [21] and [13]). In order to pick an appropriate value for that threshold, the deployed system model needs to be considered.

In wide-area networks in which the RTTs might vary drastically depending on the distance, yet remain rather stable over time, the filter order and the threshold value can both have lower values. If, on the other hand, the network is more similar to that of a mobile network with multiple connections having similar RTTs that also vary over time, the filter order and threshold value should be increased accordingly.

C. Implementation

StarReact implements a filtering function consisting of a median filter and a threshold parameter to react to the observed quorum time of the system. Given that each node measures the differential time series d_i presented in Section V-B, the decision on whether to initiate a view change in the system is based on whether the following expression holds true:

$$f(d_i) - f(d_{i-1}) > \Theta. \tag{2}$$

with

$$f(d) = \text{median}_m(d). \tag{3}$$

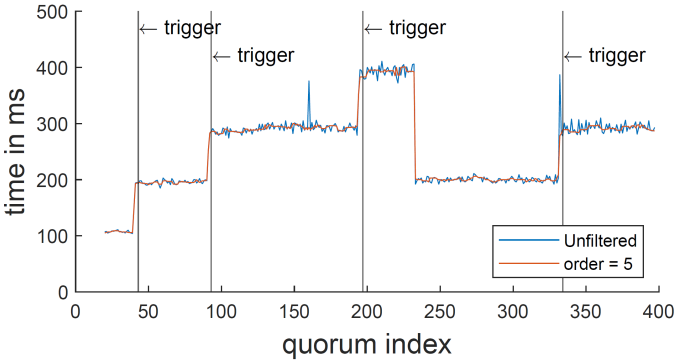

Fig. 5: StarReact response visualisation for noisy signals.

In which m denotes the order of the median filter and Θ being the threshold value that determines by what margin the measured median quorum time has to decrease in order for the node to initiate a view change.

It should also be noted that the filter order will determine at least how many values of d_i have to be below the chosen threshold value. The median filter can only output a value that is greater than $f(d_{i-1}) + \Theta$ if at least $\lfloor m/2 \rfloor$ of the values within the sliding window around d_i have been greater than $f(d_{i-1}) + \Theta$.

In summary, the order m of the median filter describes the resiliency against multiple outliers in close proximity and predicts the actual quorum time of the system while the threshold value Θ dictates by what margin the median quorum time has to decrease before a view change is initiated.

VII. Evaluation

In order to validate the observations and statements presented in this paper the hybrid simulation and emulation tool Delphi-BFT [15] was deployed. Regarding the efficacy, Berger et al. evaluated that the deviations of the measured performance, namely throughput and latency, between Delphi-BFT and their real-world experiments were below 3.5 % [15]. While this might pose a problem if accurate measurements for the performance of implementations are required, it should be acceptable for validating the functional behaviour of algorithms, as is done in this paper. We also added noise and dynamic latency changes to the network connections to better evaluate the viability of StarReact.

The effectiveness of StarReact was evaluated for different deployment scenarios in Delphi-BFT [15]. First, we simulated a network with four nodes with heavy noise on the connections. For this scenario, we simulated multiple crucial RTT changes and tested the robustness of StarReact regarding false positives, i.e., unnecessary leader changes.

Second, to measure the performance of StarReact, we simulated two networks consisting of four and ten nodes, spread across the globe with the help of the built-in latency configuration in Delphi-BFT according to Cloudping[1]. Each simulation was conducted for an hour. In order to simulate

[1]See https://www.cloudping.co/

TABLE II: Impact of StarReact during simulation with global node distribution. The two columns on the right list the number of committed blocks during 60 minute trials.

nodes	Continents	blocks (HotStuff)	blocks (StarReact)
4	EU	57261	73262
4	US, EU, AP, AF	12855	13120
10	US, EU, AP, AF, SA, CA	10909	13279

dynamic network events, a Gilbert-Elliott (GE) based delay model was implemented. Each connection was modelled with a good and a bad state. In the bad state, the delay was increased by 100 ms. The transition probability from good to bad state was $p_{g\to b} = 0.002$ and from bad to good state was $p_{b\to g} = 0.004$, respectively. The default HotStuff configuration is oblivious to the delay changes and keeps a fixed node as leader during each experiment. While this kind of setup is more appropriate for wireless networks [22], the model is fitting to showcase the viability of StarReact.

A. Noise robustness

For a better overview, the view change that would normally be triggered by StarReact was deactivated for the validation in this section. The latencies between nodes were overlapped with a log-normal distribution fitting to the mean latency. Figure 5 visualises the response of StarReact to noisy signals when with crucial changes in the RTT. The variance in Figure 5 was chosen rather low as it is more commonly found in stable networks between servers. For those scenarios, the filter and threshold can be configured low as well. A median filter of the order $m = 5$ in combination with a threshold of $\Theta = 15$ ms was sufficient to remove all outliers encountered in multiple hours of experiments.

In order to challenge StarReact we also deployed heavy noise on the connections within the same experiments as before. In order to compensate for the increased noise, the order of the filter had to be increased to $m = 9$ and the threshold to $\Theta = 20$ ms which suppressed all outliers even for very noisy latencies.

In summary, with appropriate configuration, StarReact can be deployed even with highly dynamic network connections featuring heavy noise on the signals.

B. Global Simulation

In order to compare the performance of StarReact to the default implementation of HotStuff, multiple test runs were conducted with nodes in different AWS regions, see also Table II. An exemplary visualisation of the algorithm in action is depicted in Figure 6.

As can be seen in Table II, the implementation of StarReact was particularly beneficial if all nodes are in the same region and for the larger network in multiple regions. If all nodes are in the same region, i.e., a low latency network, the bad states of the GE network model will significantly slow down HotStuff and each reconfiguration of StarReact that prevents

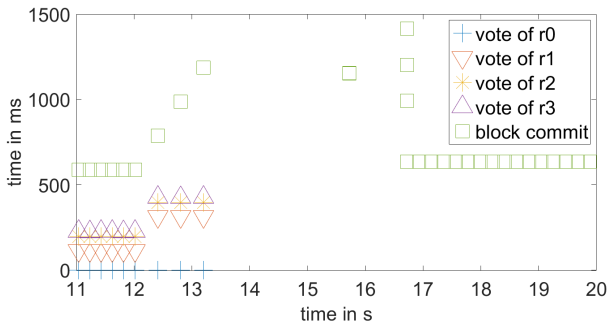

Fig. 6: Rotation triggered by StarReact (filter order 5) due to an increased commit latency for the leader node. At $t = 12\,s$ the latency of incoming packages is increased and the received votes are delayed. StarReact triggers at $t = 13.2\,s$ the rotation to a new leader, which completes at $t = 16.7\,s$.

a slow leader will keep the performance high. StarReact can only improve the performance if the setup offers sufficient room for alternative configurations. The scenario with four nodes on different continents often has inherently high delays and only little potential to find leaders with fast commit latencies. The ten nodes spread on six regions, though, contain redundant nodes that might have better connections compared to temporarily slowed down nodes. Those nodes are eventually elected by StarReact which increases the performance.

C. When to stop?

Although Table II depicts that StarReact already increases the latency of HotStuff for the scenario of dynamic links, the extension presented in this paper is not primarily focussed on performance optimisation. The purpose of StarReact is to detect when a leader rotation is warranted without increasing the communication complexity. In order to optimise the performance, the selection of the new leader needs to be considered.

It would be possible to use StarReact as a trigger for another auxiliary system, e.g., AWARE, to initiate a search for a new leader. Even though the execution of AWARE would increase communication complexity, the number of executions would be limited to network changes that trigger StarReact.

VIII. Summary

StarReact is a lightweight extension that allows BFT protocols with star-based communication to react to network changes that have a lasting impact on the quorum time. Using a simplistic example, it was shown that StarReact is able to isolate and identify when the quorum time of the system changes due to worse network connections. The evaluation has shown that the algorithm can be adapted to different scenarios by changing the filter order or threshold value accordingly.

REFERENCES

[1] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[2] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine Fault Tolerance," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 45–58, Oct. 2007.

[3] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 153–168.

[4] P. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant Byzantine Fault Tolerance," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*, Jul. 2013, pp. 297–306.

[5] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine Fault Tolerance," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 295–308.

[6] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State Machine Replication for the Masses with BFT-SMART," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2014, pp. 355–362.

[7] A. Shoker and J.-P. Bahsoun, "BFT Selection," in *Networked Systems*, V. Gramoli and R. Guerraoui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–262.

[8] W. Xu and R. Kapitza, "RATCHETA: Memory-Bounded Hybrid Byzantine Consensus for Cooperative Embedded Systems," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2018, pp. 103–112.

[9] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: A scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 568–580.

[10] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus in the lens of blockchain," 2018. [Online]. Available: https://arxiv.org/abs/1803.05069

[11] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 35–48. [Online]. Available: https://doi.org/10.1145/3477132.3483584

[12] C. Berger, H. P. Reiser, J. Sousa, and A. Bessani, "Resilient wide-area byzantine consensus using adaptive weighted replication," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, Oct 2019, pp. 183–18 309.

[13] M. Nischwitz, M. Esche, and F. Tschorsch, "Raising the awareness of bft protocols for soaring network delays," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, 2022, pp. 387–390.

[14] O. Naor, M. Baudet, D. Malkhi, and A. Spiegelman, "Cogsworth: Byzantine View Synchronization," *Cryptoeconomic Systems*, vol. 1, no. 2, oct 22 2021, https://cryptoeconomicsystems.pubpub.org/pub/naor-cogsworth-synchronization.

[15] C. Berger, S. B. Toumia, and H. P. Reiser, "Does my bft protocol implementation scale?" in *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, ser. DICG '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 19–24. [Online]. Available: https://doi.org/10.1145/3565383.3566109

[16] G. Bracha and S. Toueg, "Asynchronous Consensus and Broadcast Protocols," *J. ACM*, vol. 32, no. 4, pp. 824–840, Oct. 1985.

[17] G. Zhang, F. Pan, Y. Mao, S. Tijanic, M. Dang'ana, S. Motepalli, S. Zhang, and H.-A. Jacobsen, "Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms," *ACM Comput. Surv.*, vol. 56, no. 5, jan 2024. [Online]. Available: https://doi.org/10.1145/3636553

[18] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "Bft protocol forensics," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1722–1743. [Online]. Available: https://doi.org/10.1145/3460120.3484566

[19] C. Berger, L. Rodrigues, H. P. Reiser, V. Cogo, and A. Bessani, "Chasing the speed of light: Low-latency planetary-scale adaptive byzantine consensus," 2023.

[20] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.

[21] C. Berger, H. P. Reiser, J. Sousa, and A. Bessani, "Aware: Adaptive wide-area replication for fast and resilient byzantine consensus," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1605–1620, 2022.

[22] A. Bildea, O. Alphand, F. Rousseau, and A. Duda, "Link quality estimation with the gilbert-elliot model for wireless sensor networks," in *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015, pp. 2049–2054.

[23] J. Bahsoun, R. Guerraoui, and A. Shoker, "Making BFT Protocols Really Adaptive," in *2015 IEEE International Parallel and Distributed Processing Symposium*, May 2015, pp. 904–913.

[24] C. Carvalho, D. Porto, L. Rodrigues, M. Bravo, and A. Bessani, "Dynamic Adaptation of Byzantine Consensus Protocols," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: ACM, 2018, pp. 411–418.

[25] M. Eischer and T. Distler, "Latency-aware leader selection for geo-replicated byzantine fault-tolerant systems," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, pp. 140–145.