Optimizing trust in networks through exchange of message paths

Luka Dubravica
Supervisors: Bart Cox, Dr. Jérémie Decouchant
EEMCS, Delft University of Technology, The Netherlands

June 23, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract—**Distributed systems are networks of nodes depending on each other. However, each network can have multiple faulty nodes, which are either malfunctioning or malicious. Bracha's algorithm allows correct nodes inside the network to agree on certain information, while tolerating a certain amount of faulty nodes. Nodes exchange Send, Echo and Ready message types to inform each other of receiving the message and agreeing on its trustworthiness. The focus of this paper is to present the functional differences of exchanging message paths that messages have crossed instead of the message types. In conclusion, the BMP algorithm showed potential to outperform original Bracha's algorithm in two general cases: networks that have a low probability of successfully transmitting a message, and networks where nodes have a system of trust established which allows them to determine trustworthiness of other nodes. Otherwise, in general usage, original Bracha's algorithms appears to be superior in comparison to the BMP algorithm.**

I. Introduction

Distributed systems require protocols that allow them to trust other nodes in the system and verify the messages received. Reliable Communication (RC) protocols allow nodes to authenticate messages in a partially connected network, while providing to each node the illusion that it is directly connected to all other nodes. Byzantine Reliable Broadcast (BRB) protocols allow all correct nodes inside a network to agree on accepting a certain value. The network is defined in terms of n nodes, among which there are f faulty ones, which are either malfunctioning or malicious.

An example of an RC protocol is Dolev's algorithm [1], which a node uses to verify that a message is true if it receives the message from at least f+1 disjoint paths inside a network. Every message carries a particular value, and the path of nodes that it has traveled through in the network. When a node x broadcasts a message with a value v, that message keeps the log of all the nodes it visits. Therefore, a node y in the network can trust the received message containing a value v under the condition that the node y has received the message from at least f+1 distinct message paths.

An example of a BRB protocol is Bracha's algorithm [2], which checks whether there is an agreement between a sender and a receiver through their exchange of three message types: Send/Initial, Echo and Ready. When a node x wants to broadcast a value v, it transmits a Send/Initial message to all other nodes, with a value v attached. When another node y receives that message directly from the node x, the node y transmits an Echo message to all other nodes. The nodes that are not directly connected to the original sender will send Echo message only once they have received enough Echo messages from other nodes. Lastly, when a node y receives enough Echo messages, indicating that enough nodes have heard of the broadcast, the node y will send a Ready message to all other nodes. Once a node y receives enough of Ready messages, it can conclude that the whole network is likely to accept the broadcasted value, so the node y accepts the value itself, implying the agreement.

The original Bracha's algorithm loses a lot of data about the network and the paths the messages have crossed by exchanging only the three mentioned message types. It could therefore benefit from applying principles of an RC protocol, Dolev's algorithm. By keeping track of the network paths that the message has covered, the way that Dolev's algorithm does, more data can be extracted and more deduction made, leading to a reduction in the required amount of messages exchanged. An immediate observation is that messages will become heavier by having to carry whole message paths instead of only two bits indicating the message types. This is a cost of having more data to work with.

However, the message paths give us at least as much functionality as the message types. This is due to the ability to deduce the hypothetical message type purely based on the message path. On top of that, message paths allow for further optimizing deductions from the newly acquired data. An example is that if a certain node is included in enough message paths that we have received, we do not need a direct message from that node as we can deduce its trustworthiness. The end goal is to use these functional gains to optimize a certain use case of Bracha's algorithm.

This research paper aims to present the functional benefits of Bracha's Message Paths (BMP) algorithm. The BMP algorithm is a variation on Bracha's BRB algorithm, which uses messages that carry the network path that the message has crossed instead of the message type.

The contributions that we have achieved while exploring this topic are the following:

- We have developed a theoretical BMP algorithm, a version of Bracha's algorithm that uses message paths instead of message types.
- Using the theoretical version, we have implemented a usable version of the BMP algorithm in C++ to test the algorithm in real case scenarios.
- Lastly, we compared the BMP algorithm's functionality to the original Bracha's algorithm functionality.

The rest of this paper is structured in the following manner. *Related Work* (II) gives an overview of the state-of-the-art algorithms in the field, and how is this paper building on top of that. *Methodology* (III) describes the path that was taken to achieve the listed contributions. *Bracha's Message Paths Algorithm* (IV) presents the main contribution of this paper, the BMP algorithm and the reasoning behind some of the design choices. *Evaluation and Discussion* (V) reflects on the BMP algorithm and how it compares with the original Bracha's algorithm, presenting us with the results. *Responsible Research* (VI) focuses on the ethical aspects of this research, as well as the reproducibility of the method. Lastly, *Conclusion and Future Work* (VII) summarizes the research and raises the questions of future research.

II. Related Work

There exists a lot of research in the fields of distributed systems communication, byzantine reliable broadcast and other related topics, but there is no paper that resembles the goal of this one. To the best of our knowledge, there was no research found that presented a way to modify Bracha's algorithm in the similar manner. However, there are nonetheless a lot of state-of-the-art modifications of Bracha's and Dolev's algorithms. These could offer inspiration and foundation for further research and improvements of the BMP algorithm, as this paper aims to present the BMP algorithm built on top of the core principles of Bracha's and Dolev's algorithms.

Primary focus of the literature study was studying Bracha's and Dolev's algorithms as they provided the basis for the development of the BMP algorithm. The secondary focus was researching state-of-the-art in the field dealing with byzantine fault. The primary focus was covered in section I, so the summary of highlights of secondary focus will be presented here.

Significant improvements have been achieved when it comes to both combining Bracha's and Dolev's algorithm, as well as to the two algorithms separately. Dolev's algorithm saw recent improvements when it comes to message complexity, even though it kept the worst-case complexity in asynchronous systems [3]. The improvements were also seen by the state-of-the-art BRB protocol, a combination of Bracha's and Dolev's algorithm. The recent paper presented how the previously mentioned improvements to Dolev's algorithm were applicable to the combination of Bracha's and Dolev's, as well as twelve more modifications, out of which some are cross-layer [5].

All of the noted improvements to the algorithms could be explored in the future research as possible modifications to the BMP algorithm.

III. Methodology

The process of achieving the presented results in this paper is broken down into three main steps. Preceding that is the literature study. The goal is to explore how Bracha's and Dolev's algorithms work, what are the applications for those algorithms, and is there a study achieving similar work.

After literature study, the first step is to develop the theoretical version of the BMP algorithm, the variation on Bracha's algorithm that exchanges the message paths instead of message types, with an aim to reach an agreement sooner. Using the research studied, the Bracha's algorithm will be readjusted to implement the characteristics of Dolev's algorithm, in a manner that allows the new modified algorithm to benefit from the newly acquired data. This part of the research is the core, as the new BMP algorithm presented is used in the rest of the research to present the functional gains of the BMP algorithm.

Following the theoretical implementation, it is important to implement the usable code version of the BMP algorithm for two reasons. First, the code implementation allows for the more in-depth study of the BMP algorithm, presenting space for improvement, as well as potentially pointing out wrongly structured aspects of the BMP algorithm. Second, the code implementation prepares the algorithm to be readily available in the real-life scenarios. The code version of the algorithm is implemented in the programming language C++.

Using the theoretical and practical versions of the BMP algorithm, the third step compares their functionality to the functionality of the original Bracha's algorithm. This allows for explicitly stating the differences and focusing on the functional improvements. Listing them out and analyzing them is important for answering the aim of this research paper.

IV. Bracha's Message Paths Algorithm

As the main contribution of this research, we have developed the new BMP algorithm, which builds upon the core ideas from Bracha's and Dolev's algorithms. Bracha's algorithm exchanges messages between the nodes inside the network to achieve agreement over information broadcasted. Dolev's algorithm uses the message paths to verify a particular message in the network. By substituting Bracha's idea of using message types to achieve agreement, and instead applying Dolev's idea of using message paths, we constructed the BMP algorithm. Algorithm 1 presents the theoretical pseudocode of the BMP algorithm.

A. Underlying components of the BMP algorithm

The new BMP algorithm required constructing the new message type that would optimally suit our purpose. Therefore, we constructed *BMPEcho(value, broadcasterId, linkSenderId, paths)*.

- Parameter *value* carries the message that the broadcasting node wishes to transmit to the rest of the network.
- Parameter *broadcasterId* carries the id of the node that has initialized the broadcast.
- Parameter *linkSenderId* carries the id of the node that was last to forward the message. In case the node x receives the message from node y, that was broadcasted by node z, *linkSenderId* will carry the id of node y. While the first two parameters stay the same as the message gets forwarded around, *linkSenderId* keeps changing. Similarly, parameter *paths* changes with every forwarding.
- Parameter *paths* carries all the paths through which the associated value was received. Therefore, if node y sends us a *BMPEcho* message, we will see its id stored in the parameter *linkSenderId*, and we will see all the paths through which it has received the associated value in the parameter *paths*. Once we decide to forward the message ourselves, we will append the id of node y to every of the paths in the parameter *paths*, to indicate that the message has travelled through all the existing paths, including the node from which we have received the node.

Since each node has to keep track of all the paths from which it has received the associated values, we introduced a map *valuesPaths* which maps a value key to the list of paths from which the value was received. Whenever a new *BMPEcho* message arrives, the node appends to all of them the id of the node from which it has received the message. Afterwards, it checks whether each path is already stored in *valuesPaths*, and if it is not, it inserts it into the map. Once we get to sending the message, we can just check if we have a list of paths associated with the value that we want to send. If we do, we can attach it. Otherwise, in case it is the initial broadcast, we can just attach the empty list.

Algorithm 1 *The BMP Pseudocode, at node p_i*

```
$valuesPaths \leftarrow []$
$acceptedValues \leftarrow []$

function SENDBMP(value, broadcasterId)
    # Construct the message and send it to every neighbour
    $paths \leftarrow valuesPaths[value]$
    $msg \leftarrow BMPEcho(value, broadcasterId, p_i.id, paths)$
    for i in neighbours do
        SENDMSG(msg, i)
    end for
end function

function HANDLEBMP(msg)
    for path in msg.paths do
        # Append the sender of the message to all received paths
        $path \leftarrow path + msg.linkSenderId$
        # Store every new path in $valuesPaths$
        if path not in valuesPaths[msg.value] then
            $newPaths \leftarrow valuesPaths[msg.value] + path$
            $valuesPaths[msg.value] \leftarrow newPaths$
        end if
    end for

    # If no paths received, create and store the initial path
    if msg.paths is empty then
        $valuesPaths[msg.value] \leftarrow [msg.linkSenderId]$
    end if

    $leadingNodes \leftarrow []$
    $echoNodes \leftarrow []$
    $forwardMessage \leftarrow false$
    # Traverse the list of stored paths
    for path in valuesPaths[msg.value] do
        # Check by what nodes was a node informed
        for node x in path do
            for node y in path, after node x do
                $leadingNodes[y] \leftarrow leadingNodes[y] + x$
            end for
        end for

        if not forwardMessage then
            # count from how many nodes the message was received
            for node x in path, after 0th node do
                $echoNodes \leftarrow echoNodes + x$
            end for
            # Check if the message was directly received
            $directCast \leftarrow path.size == 1$
            # Check if the message was received through $(n+f)/2$ nodes
            $enoughNodes \leftarrow echoNodes.size \geq (n+f)/2$
            $forwardMessage \leftarrow directCast \vee enoughNodes$
        end if
    end for

    $countReadyNodes \leftarrow 0$
    for pair in leadingNodes do
        # Check if a node received a message through (n+f)/2 nodes
        if pair.second.size $\geq$ (n+f)/2 then
            $countReadyNodes \leftarrow countReadyNodes + 1$
        end if
    end for

    $valueAccepted \leftarrow msg.value$ in $acceptedValues$
    $enoughReadyNodes \leftarrow countReadyNodes \geq 2f+1$
    $notAllReady \leftarrow countReadyNodes < (n-1)$
    # Check whether to accept the value
    if enoughReadyNodes $\wedge$ not valueAccepted then
        ACCEPT(msg.value)
        $acceptedValues \leftarrow acceptedValues + msg.value$
    end if
    # Check whether to forward the message
    if forwardMessage $\wedge$ notAllReady then
        SENDBMP(msg.value, msg.broadcasterId)
    end if
end function
```

B. Functions of the BMP algorithm

The code is structured into two functions, *sendBMP* and *handleBMP*, that utilize the mentioned message type *BMPEcho* and the mentioned data structure *valuesPaths*.

When a broadcast has to be initialized and a new message created, or a received message forwarded with new paths attached, we call *sendBMP*. As already explained, this function creates a new, or retrieves an existing list of paths associated with the value it plans to send. Using the available information, it constructs a new *BMPEcho* message. The new message carries the *value* and the *broadcasterId*, which are often copied from the received message. On top of that, the new message also carries the id of the node that is about to send the message, and the list of paths associated with the value. The function finishes by sending the newly constructed message to all of the node's neighbors.

When a message is received and it has to be processed, we call *handleBMP*. This function has three main purposes:

- *Update and process received message paths.* As it was explained earlier in this section, the received paths get updated with the node from which the message was received, and they get properly stored for future use.
- *Determine whether the node accepts the broadcasted value.* This depends on two conditions. First, has the node received enough messages from other nodes to conclude that $2f+1$ nodes are ready to accept the value. Second, has the node already accepted the value. If there are enough ready nodes and the value has not yet been accepted, the node will accept the broadcasted value.
- *Determine whether the node forwards the received message.* This also depends on two conditions. First, if the message has been received directly from the broadcasting node, or has been received from $(n+f)/2$ other nodes, directly or indirectly, then the node is allowed to forward the message. Second, if there are less than $n-1$ nodes that are not ready, there still might be a need for forwarding messages. Therefore, if the node is allowed to forward the message, and there is a need to do so, the node will forward the received message with the updated list of paths attached.

C. The Code Implementation

The version implemented for the practical testing is based on the theoretical version presented in Algorithm 1. It is implemented in C++ (code available online[1]). Developing the code required defining message types data structures, writing out functions and defined data structures in C++, and properly setting up serialization for asynchronous network communication. For achieving the simulated communication between the nodes, Salticidae[2] library was used, a C++ library for asynchronous network communication. The rest of the code heavily resembles the pseudocode presented.

[1]BMP GitLab Repository
[2]Salticidae GitHub Repository

V. Evaluation and Discussion

Following the theoretical evaluation, we observed that in comparison to the original Bracha's algorithm, Bracha's Message Paths algorithm achieves functional improvements, at the cost of other problems. Exchanging message paths instead of message types allows for a lot more data to be available to the nodes that are processing the messages, but the messages become significantly heavier. With the implementation of the BMP algorithm, we have noticed the following functional differences.

A. Lost messages in unstable networks

In the networks that have a high probability of losing a message, transmitting all the received paths allows the nodes to make conclusions about the network based on less messages. With a single trusted message, the network can deduce how many nodes decided to forward the message, and how informed the other nodes are. This however requires a trusted source, which related to the next listed benefit.

B. Single trusted message

Nodes require a connectivity of 2f+1 to be able to verify messages. This means that they have to be connected to at least 2f+1 other nodes in the network. If a node finds itself at a connectivity lower than required, but among the connected nodes it has a trusted node, it can still reach valid conclusions. The BMP algorithm uses BMPEcho message which in case of a trusted transmitter can inform the network with a weak connectivity about the situation in the rest of the network. This allows the weakly connected nodes to deduce which of the nodes in the network have forwarded/echoed the messages and which nodes can be considered ready to accept. With this information the node can decide to accept the value.

C. Trading latency for number of messages exchanged

If the latency is of less importance, but the networks can handle bigger chunks of data, the nodes could decide to wait before forwarding the messages. By waiting, the nodes could accumulate more paths from more received messages, instead of forwarding them all immediately. Then, when the specified time period elapses, the nodes could forward all the gathered paths at once. This is not part of the current pseudocode, but the framework offers foundations for a future improvement.

D. Deduce network trustworthiness

Depending on the network, the nodes could know its current topology. In such a case, the nodes could build a picture of the perceived topology through analyzing all the received message paths. If the perceived topology does not match the known topology, with sufficient information, nodes could deduce which nodes are faulty. This data could be useful in the field of collaborative artificial intelligence, where keeping track of trustworthiness of other nodes can be of importance. Similarly to the last point, this offers a potential for future improvement.

E. Heavier Messages

The mentioned pros unfortunately come at a cost of one significant drawback. More data to work with implies that more data has to be transferred. Compared to Bracha's original algorithm, the BMP algorithm exchanges messages that carry significantly more data. Original Bracha's messages carries one of three possible types: initial, echo, or ready. Two bits are enough to encode one of the three message types, in an optimal implementation. In comparison, our current implementation of BMP algorithm already requires at least 96 bits to just store empty list of paths, another 96 bits for each path inside the list, and 32 bits for each node in the path.[3] This implies that BMP's equivalent of optimal original Bracha's Initial message weighs 48 times more, and that is just in the case of the lightest message that gets sent once per broadcast. In all following messages, the size will be a lot larger. The size of average message will also scale with the size of the network, as larger networks require more paths and have more nodes.

VI. Responsible Research

The BMP algorithm presented in this paper is intended to be used for increasing trust in the networks that consist of n nodes and tolerate up to f faulty nodes. Therefore, while the BMP algorithm contributes to promoting safety in the distributed systems, it is our responsibility to present the algorithm that indeed tolerates f faulty nodes. Each study read should always be studied with care, but it is at the same time the responsibility of the writers to present the most correct possible information, reducing the possibility of a mistake in the future.

The method in this paper is easily reproducible. The theoretical BMP algorithm is built on top of the two mentioned existing algorithms. The constructed pseudocode is available in the paper as Algorithm 1, and the C++ implementation is available online[4]. The results of the comparison are performed using the presented pseudocode and the implemented C++ code. We believe that the reader will be capable of deducing similarities between the BMP algorithm, and its sources: Bracha's and Dolev's algorithm. Therefore, all the resources are also available to the reader to perform the theoretical evaluation of the BMP algorithm in comparison to the other two mentioned algorithms.

VII. Conclusion and Future Work

In this paper we presented the functional differences of using Bracha's Message Paths algorithm in comparison to original Bracha's algorithm. To achieve this goal, we made the following contributions: we developed a theoretical version of the BMP algorithm; we implemented the BMP algorithm in C++ for further analysis; and we evaluated the functional differences of the BMP algorithm in comparisson to the standard Bracha's algorithm. Each of these contributions is respectively covered in the section IV, the subsection IV-C, and the section V.

[3]Stack Overflow Calculation
[4]BMP GitLab Repository

Our initial hypothesis was that exchanging whole paths instead of message types might turn out to be suboptimal in general application, but it might have a promising application in specific cases. That turned out to be confirmed by the evaluation. There are specific promising cases of applying the BMP algorithm. As we presented in the section V, such promising cases include networks that have lower probabilities of messages getting through, networks where nodes have trusted connections inside the system, and networks where nodes aim to keep track of each other's trustworthiness. In general usage, original Bracha's algorithm appears to be superior in comparisson to the BMP algorithm. The primary reason behind this is the weight of the BMP messages.

This paper raised many questions worth of further research, spanning from possible applications, to possible modifications. Regarding possible applications, the following fields should be studied for potential cases: networks that implement mechanisms of trust (such as collaborative artificial intelligence), and networks that have lower probabilities of successful message delivery. Studying such fields could reveal a potential usage case for the BMP algorithm.

The BMP algorithm could also benefit from possible modifications. These modifications include already mentioned ones such as: ability to trade-off the duration of the message exchange for the total size of messages exchanged; and build the topology map of the network; as well as others. The BMP algorithm should also be tested beyond just theoretical stage. The code should be evaluated as that would allow us to spot more possible improvements. We expect that some stated numbers of required nodes might not be optimally configured. One example would be the parameter *notAllReady*, which might be reducable to *n-f-1*, instead of being at *n-1*. Some additional AB testing could also be performed to further understand the nature of the algorithm. For example, does in practical application transmitting messages with only one path beat transmitting messages with multiple paths.

References

[1] Dolev, D. (1981, October). *Unanimity in an unknown and unreliable environment*. In 22nd Annual Symposium on Foundations of Computer Science (sfcs 1981) (pp. 159-168). IEEE.

[2] Bracha, G. (1987). *Asynchronous Byzantine agreement protocols*. Information and Computation, 75(2), 130-143.

[3] Bonomi, S., Farina, G., Tixeuil, S. (2019). *Multi-hop Byzantine reliable broadcast with honest dealer made practical*. Journal of the Brazilian Computer Society, 25(1), 1-23.

[4] Yin, T. (2022, February 4). *Salticidae: minimal C++ asynchronous network library*. GitHub. https://github.com/Determinant/salticidae

[5] Bonomi, S., Decouchant, J., Farina, G., Rahli, V., Tixeuil, S. (2021, July). *Practical Byzantine Reliable Broadcast on Partially Connected Networks*. In 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS) (pp. 506-516). IEEE.