Approaching Message Optimal Byzantine Reliable Broadcast using Routing

Dany Sluijk
Supervisor(s): Jérémie Decouchant, Bart Cox
EEMCS, Delft University of Technology, The Netherlands

**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*—In this paper we will consider the Byzantine Reliable Broadcast problem on partially connected networks. We introduce an routing algorithm for networks with a known topology. It will show that when this is combined with cryptographic signatures, we can use the routing algorithm to create an optimal amount of messages. We will introduce a few algorithms to estimate the routes required for this algorithm. Our simulations show that the amount of messages required for Byzantine Reliable Broadcast can be massively decreased by using this. We will also show that the method of choosing these paths has a huge impact, but come with their own trade-offs.**

I. Introduction

Distributed systems are older than the internet itself, but have always had their limitations compared to centralized systems. The use of these distributed systems has therefore been limited to some niche use cases. But recently there has been a shift away from centralized systems, by things like cryptocurrencies. This has created a need for reliable communication within these distributed systems.

In this paper we will limit ourselves to two problems within distributed systems: Reliable Communication (RC) and Byzantine Reliable Broadcast (BRB). Reliable Communication means that a node can verify that a received message has been send by one particular node in a partially connected network, giving the illusion that the network is fully connected. Byzantine Reliable Broadcast means that all correct nodes in a system will collectively accept or reject a broadcast, even in the presence of faulty nodes. Both these problems have been solved already, Reliable Communication by Bracha with Double Echo [1], and Byzantine Reliable Broadcast by Dolev [2]. It also has been shown that these two protocols can be combined in order to uphold both constraints with Bracha-Dolev from Bonomi et ai. [3].

While these protocols can guarantee reliability and fault tolerance, they are very expensive in communication and have severe constraints. For example, the connectivity of the system has to be strictly more than twice the amount of fault nodes ($c \geq 2f + 1$). This limits the amount of nodes in the system which are realistically possible, and therefore the possible practical applications of these protocols. It is therefore useful to reduce these constraints to improve the efficiency of the systems without impacting the fault-tolerant or reliability properties. There has been work done to reduce these limitations [4], [5], but it is expected that the efficiency can still be improved.

We explore possible optimizations if we assume to have knowledge of the topology of the system. By expanding on previous work done by Klabér [5] with regards to signed messages we can optimize the broadcast by using routing of messages. We will show that we can reduce the amount of required messages send, while still upholding the properties of Byzantine Reliable Broadcast.

The paper is structured as follows. In the first Section II we will explore the background where this paper is based on. We will then move on to the improvement which adds routing to the graph in Section III. After the routing on the graph, we will explain in Section IV a few methods to generate routing tables for this algorithm. We will then in Section V talk about the evaluation of these improvements, and what the results are and mean for this paper. The we will turn our attention to talk about responsible research in Section VI. Lastly we will close off the paper with the conclusions in Section VII, and talk about the main takeaways.

II. Background

In this section we will elaborate on the background surrounding the Byzantine Reliable Broadcast problem. First we will define Byzantine Reliable Broadcast and Reliable Communication more precisely. We will then discuss Dolev's protocol for Reliable Communication on partially connected graphs. Finally we will discuss an algorithm which uses signatures to improve the message complexity and connectivity requirement, and guarantees BRB. This last algorithm is where we will build upon later.

A. Byzantine Reliable Broadcast

Byzantine Reliable Broadcast (or BRB) is the guarantee that a broadcast will be agreed upon by all honest nodes in a network, even if there are byzantine nodes present. Byzantine nodes can behave arbitrarily, and the network should be able to tolerate any type of irregular behaviour. Agreement here means that all nodes will either accept (and thus process) the message, or reject (ignore) it. BRB algorithms do already exist, such as the Double Echo algorithm by Bracha [1]. Another example is Bracha-Dolev by Bonomi et ai. [3], which extend the Double Echo algorithm to work on partially connected networks. In order to say that an algorithm upholds as BRB, we need to guarantee a set of requirements:

- *Validity*: if a honest node n broadcasts an message m, then there is an honest node which will accept m eventually.
- *No duplication*: no honest node will accept a message m twice.

- *Integrity*: If a honest node accepts a message m from node n, then n has previously broadcast-ed m.
- *Agreement*: If a honest node accepts a message m, then all honest nodes will eventually accept m.

B. Reliable Communication

Reliable Communication (RC) protocols allow processes to authenticate received messages. This is useful on partially connected networks, where multiple hops might be required to deliver a message to a node. When a RC protocol is used, it gives the possibility to add the illusion of a fully connected network. The first proposal for an RC protocol was made by Dolev in his paper [2]. Improvements on this protocol have been proposed later, for example by Bonomi et al [6]. With most RC protocols in papers signatures are not used. This partly is the case because using signatures make authentication of messages trivial, and therefore not interesting for research. Within this paper signatures are used in order to guarantee RC. Reliable Communication has the same requirements as Byzantine Reliable Broadcast, with the exception of agreement. Within RC, agreement is only guaranteed if the sender node itself is honest. In our protocol RC is guaranteed, as it also guarantees the stronger Byzantine Reliable Broadcast.

C. Dolev

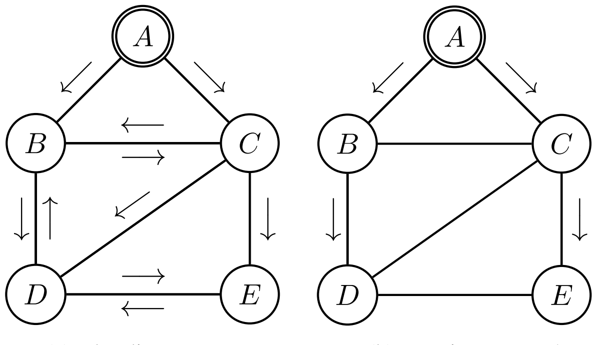
In Dolev's paper, an protocol for Reliable Communication (RC) is described [2]. This requires a connectivity of $2f+1$ in the network, but it does allow the network to be partially connected. When a node receives a Dolev message, it will forward that message to all neighbours, after adding itself to the path of that message. If a message has been received through $f+1$ disjoint paths, then the message can be delivered in the node.

D. Signatures

The previous algorithms all work without signatures to verify the authenticity of a message. Dolev does this by waiting for $f+1$ in order to establish their authenticated links. However, this is very inefficient in terms of message complexity. By adding signatures to the broadcast of a message, we can verify that a message was indeed send by a certain node. This allows us to instantly accept a message, given that there are other guarantees to uphold the agreement requirement of BRB. As observed by Klabér [5], this means that we only need to have a connectivity of $f+1$, instead of $2f+1$ with the other algorithms. This is also the lowest possible limit, as going any lower makes it possible to encircle one or more nodes and therefore cut them of from the rest of the graph. In the next section we will improve on this knowledge to decrease the message complexity even further.

III. Improving Broadcast with Routing

In the paper on using signed messages by Klabér [5] the message efficiency was improved greatly by requiring a message to be received $f+1$ at minimum, instead of $2f+1$ times as required by Dolev. This is essentially a flooding algorithm on the graph, which implies that there are a lot of duplicate messages send around. This is a result of the fact that all nodes blindly broadcast the message to all nodes, and the effect of this can be seen in Figure 1a.



(a) Flooding, $m = 10$. (b) Routing, $m = 4$.

Fig. 1: Broadcast using Flooding and using Routing ($f = 0$).

If the correct topology of the graph is know to the nodes, then further optimizations are possible by adding routing. A routing table can be build for each sender, which ensures that every node is routed from exactly disjoint $f+1$ paths. As the network is at least $f+1$ connected, such a routing table can always be build. One algorithm for building this table will be discussed in section IV. Removing any one message means that a node receives messages from strictly less than $f+1$ paths. This means that a property of BRB will be violated, implying that this routed algorithm is optimal. When a node receives a message it has not received before it, will broadcast that message only to the nodes which are in the routing table. As these routing tables can be completely pre-computed, the computational impact after the generation is finished is minimal. A re-computation of this table is not needed, unless the topology of the network changes. It is important that these routing tables are deterministic across all correct nodes, or $f+1$ connections cannot be guaranteed. Such a routing algorithm should take this into account. With the routing table known, it's easy to route incoming messages, as shown in Algorithm 1.

Algorithm 1 Routing incoming messages

```
upon event $< Init|top, id, faulty >$ do
    $N \leftarrow GetNodes(top)$
    $E \leftarrow GetEdges(top)$
    $routes \leftarrow BuildRoutes(id, faulty, N, E)$
upon event $< Message|sender, from, msg >$ do
    if $verify(msg, sender) = false$ then
        return
    end if
    $deliver(sender, msg)$
    $destinations \leftarrow routes[s]$
    for $dest$ in $destinations$ do
        $sendMessage(sender, id, msg)$
    end for
```

IV. Route Building

In the last section we discussed how an routed algorithm would work. What we deliberately skipped over is how to actually build these routes.

It's easy to check that these paths exist using Menger's Theorem [7] and a Max-Flow algorithm. It follows that it's possible to check if a set of routes can guarantee Byzantine Reliable Broadcast. This is because the routes can be converted into a directed graph of all nodes, on which the same Max-Flow algorithm can be applied to.

Actually computing these paths is a way harder task. It has to be deterministic, as this routing table is not generated in a centralized place. Determinism is therefore required in order for all nodes to calculate the same routing table independently. It's possible that generating these paths is a NP-Complete problem, and therefore might not have an guaranteed and optimal polynomial time solution. In this section we will consider three different algorithms with different positive and negative traits. Later in Section V-B we will evaluate the Fast and Path-finding algorithms.

A. Brute-Force Algorithm

The simplest algorithm is, as it often is, a brute-force algorithm. This is done by enumerating over all possible combinations of routes, and return a valid set of routes. This can either be the first match, or the one with the least amount of required links. The first one is the fastest and correct as it can return early, but probably won't return the optimal solution. The second one is correct and optimal, but requires you to iterate over all possible solutions making it slower. Due to the high computational complexity of $\mathcal{O}(2^{E}NEf)$, this algorithm is very slow. An algorithm for the first version is shown in Algorithm 2.

Algorithm 2 Brute-Forced route building

```
procedure BUILDROUTES($s, f, N, E$)
    return $BuildRoutesRec(s, f, [\,], N, E)$
end procedure
procedure BUILDROUTESREC($s, f, R, N, E$)
    $e \leftarrow$ pop $E$
    $RE \leftarrow E - \{e\}$
    $Exc \leftarrow BuildRoutesRec(s, f, R, N, RE)$
    if $IsValidRouting(s, f, Exc, N)$ then
        return $Exc$
    end if
    return $BuildRoutesRec(s, f, R \cup \{e\}, N, RE)$
end procedure
procedure ISVALIDROUTING($s, f, R, N$)
    for $t$ in $N$ do
        if $s = t$ then
            continue
        end if
        if $MaxFlow(s, t, R) \leq f$ then
            return false
        end if
    end for
    return true
end procedure
```

B. Fast Algorithm

The previous algorithm always returns a correct answer, but is very slow for anything but the smallest amount of edges. To make this more viable for larger networks, another algorithm is needed. With this algorithm, paths are created from the sender s, and growing outward. In one cycle, all paths grow with a length of one, splitting up when paths diverge. A node accepts a path when it has not accepted enough, and if the proposed path is independent of all already accepted paths. If, in one cycle, there are more possible paths to add to a node than there is space, then a conflict resolution algorithm is used. For this see Section IV-B1. This algorithm has an polynomial time complexity, and is therefore relatively fast. As a disadvantage it does not guarantee an optimal solution, or even a correct one (as shown in Section V-D).

1) Conflicting Paths: It is possible for multiple paths to be candidates to be added to a node, while less paths are required. This can happen when there are more than $f+1$ connections to one node. To resolve these conflicts you can use a cryptographic hash, like SHA [8], on the candidate paths. Then accept the path with the higher hash value. As there is no inherent value on the hash value, it does not matter what method is for deciding

Algorithm 3 Fast route building

```
procedure BuildRoutes($N, f, s, n$)
    $paths \leftarrow [[s]]$
    $queue \leftarrow [[s]]$
    while $queue \neq \emptyset$ do
        $possiblePaths \leftarrow []$
        $changed \leftarrow queue$
        $queue \leftarrow []$
        for $path$ in $changed$ do
            $neighbours \leftarrow N$[last $path$]
            for $neighbour \in neighbours$ do
                if $neighbour \notin path$ then
                    $newPath \leftarrow path + neighbour$
                    $possiblePaths \leftarrow newPath$
                end if
            end for
        end for
        for $node$ in $N$ do
            $nodePaths \leftarrow possiblePaths$
            $nodePaths$ filter $last == node$
            $nodePaths$ sort on route hash
            $accepted \leftarrow$ count $paths$ with $node$
            $missing \leftarrow f + 1 - accepted$
            $newPaths \leftarrow nodePaths$ limit $missing$
            $paths \leftarrow newPaths$
            $queue \leftarrow newPaths$
        end for
    end while
    $routes \leftarrow []$
    for $path$ in $paths$ do
        if $path$ includes $n$ then
            $routes \leftarrow path[n + 1]$
        end if
    end for
    return $routes$
end procedure
```

which hash is higher. The only requirement here is that it's consistent across the entire network. As the paths are equal, and hashing is deterministic, the output of this is repeatable on every node. You need this step, instead of just using the first compatible path, to remove any bias towards certain nodes. This will make the routing graph more balanced.

C. Path-finding Algorithm

As we will see during the evaluation in Section V-B, the previous fast algorithm does improve the message complexity compared to flooding. But the issue is that the solution given is often still far from ideal. This is where we introduce a path-finding algorithm which adds the shortest possible routes with the least amount of additional links required.

It starts by finding the closest node to s with the most links still required, call this t. It will then try to find a possible paths between s and t. These are ranked on the following, using the next route iff the value is equal:

1) Is the shortest path.
2) Removes the least connectivity between s and t.
3) Has the least amount of overlap with other potential routes.
4) Adds the least amount of links to the graph.

This route is then added, and the intermediate nodes are marked unusable by t. This will repeat until all nodes have sufficient connectivity with s, using the stored routes.

This algorithm performs significantly better than the fast algorithm, but is also a lot slower. This mostly comes from the large amount of calls to the MaxFlow algorithm. It is still significantly faster than then brute force algorithm. In Algorithm 4 pseudo-code is shown.

V. Evaluation

In this section we will elaborate on the method we used to demonstrate the optimisations proposed in this paper. We will compare the both the fast and the path finding algorithms for route building to the flooding algorithm. The brute-force algorithm is not considered, as it is too slow. After this we will consider the time it takes for these routes to be build. Finally we will talk about the failure rate of the route building.

A. Methodology

For the test setup we created a virtual network with virtual nodes and links. This is implemented in the Rust programming language, with Tokio as an async runtime with green threads. Network links are simulated by adding a random delay on the $N(75, 25)$ distribution to the delivery of the messages. This is done to introduce some kind of randomness, as the performance of the algorithms should not be affected by this. Other than that, other properties like bandwidth is not restricted. The code used for the test setup can be found in the GitLab Repository.

Before simulations begin, random graphs are generated using the method from Steger [9]. However, these graphs are not guaranteed to be k-connected yet. To check this, we used a modified version of the method from Esfahanian [10] to calculate the connectivity of the graph. The graph will then be accepted if the connectivity is at least the c currently used, otherwise it will try the

Algorithm 4 Path-finding route building

```
procedure BUILDROUTES($N, f, s, n$)
    $paths \leftarrow []$
    $missing \leftarrow [\{..N\} -> f + 1]$
    $used \leftarrow [\{..N\} -> []]$
    while $m \neq 0 \in missing$ do
        $t \leftarrow FindT(s, N, missing)$
        $path \leftarrow FindPath(s, t, N, used, paths)$
        $paths = paths \cup \{path\}$
        $used[t] = used[t] \cup path$
        $missing[t] -= 1$
    end while
    return $paths$
end procedure
procedure FINDT($s, N, missing$)
    $queue \leftarrow [s]$
    while $queue \neq \emptyset$ do
        $considering \leftarrow queue$
        $queue \leftarrow []$
        $lowest \leftarrow None$
        for $i$ in $considering$ do
            $queue \leftarrow \{queue \cup N[i]\}$
            if $missing[i] < missing[lowest]$ then
                $lowest \leftarrow i$
            end if
        end for
        if $lowest \neq None$ then
            return $lowest$
        end if
    end while
end procedure
procedure FINDPATH($s, t, N, used, P$)
    $potential \leftarrow$ paths in $N$ from $s$ to $t$
    $l \leftarrow$ length of shortest path in $potential$
    $paths \leftarrow potential|$ filter length of $i == l$
    $conn \leftarrow$ [connectivity of $\{N - path\}$]
    $paths \leftarrow paths|$ filter $conn[path] ==$ max $conn$
    $over \leftarrow$ [amount of overlapping $path$ in $paths$]
    $paths \leftarrow paths|$ filter $over[path] ==$ min $over$
    return $path$ where size $(paths \cap P)$ maximum
end procedure
```

generation again. We will focus on a very dense amount of nodes, including any number of nodes between 1 and 20 inclusive. An exception on this is the timing graph, which goes up to 30 nodes. For connectivity we have generated a graph for every possible value. So a graph with $n = 4$, we will have a connectivity c of 1 to $n - 1$. Then we will generate 5 different graphs with these two parameters, which then generate more data points.

For the simulation all algorithms are run on all the generated graphs. With each graph we will explore every possible count of faulty nodes. This means that a graph with a connectivity of c will be tested with f from 0 to $c - 1$. A single message is send in the graph, as multiple messages can water down the significance of the amount of messages. Metrics are collected after the simulation has finished. First it verifies that all honest nodes have received all broadcasts correctly. It then counts the amount of messages exchanged, and the time it took to do this. The result of this can be seen in section V-B.

All tests are run on an Dell XPS 9550, which has an Intel Core i7-6700HQ and 16 gigabytes of system memory. The only evaluating test affected by this machine is the the timing of the route building. Other machines should have results of similar magnitude, but might differ slightly in actual numbers.

B. Message Complexity

The goal with the routed algorithm is to reduce the amount of messages required to achieve Byzantine Reliable Broadcast. While in theory a routed algorithm result in the most optimal amount of messages send, this heavily depends on the routing table. Getting the optimal routing table is computationally very difficult (as shown in Section IV). This is shown clearly with the fast algorithm in Figure 2. The lines in the figure represent different values of f, with the lower values starting more to the left.

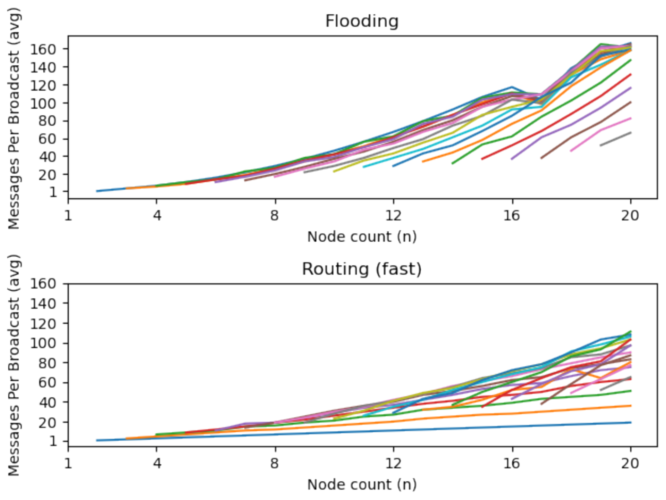


Fig. 2: Messages send of the fast algorithm.

You can clearly see that the routed algorithm is less affected by the growing amount of nodes n. This is explained by the fact that the routed algorithm tries to reduce the amount of excessive deliveries. Optimally the amount of additional messages per added node is $f+1$, instead of one message per edge of the node.

However the fast algorithm is still far from optimal, and the path-find algorithm is better in this regard. This can be seen in Figure 3. Here we see even less growth in message complexity for any added nodes. It is also interesting to note that the difference between values of f is also reduced. From this graph is seems clear that the path-finding is significantly better, but in Section V-C we will show that is has a big disadvantage.

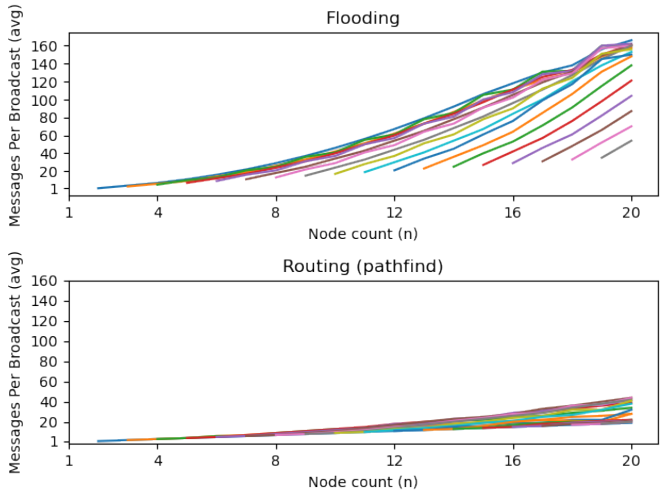


Fig. 3: Messages send of the path-finding algorithm.

C. Route Generation

One of the additional requirements of the routed algorithm is a routing table. The methods to compute this are discussed in more details in Section IV. The computation can take a significantly varying amount of time, depending on the method. In order to evaluate this timing, we tracked the time it took for a full routing table to be generated. The result of this is shown in Figure 4. The different lines correspond the different values of f. This correlation of n to the amount of time is very strong, so this graph is shown with a logarithmic scale. Do note that the routing of the flooding algorithm is just broadcasting to all other nodes, and therefore has a trivial routing table and not included in the graph. The brute force algorithm is also not included, as it gets computationally infeasible too quickly to be significant.

It is clear that the fast algorithm is indeed significantly faster, and can therefore scale better than the other two algorithms. While the path-finding algorithm finishes in about ten seconds for 20 nodes, the fast algorithm is still sub-second. It does still grow significantly for either algorithm, so it takes a non-negligible amount of time for either algorithm.

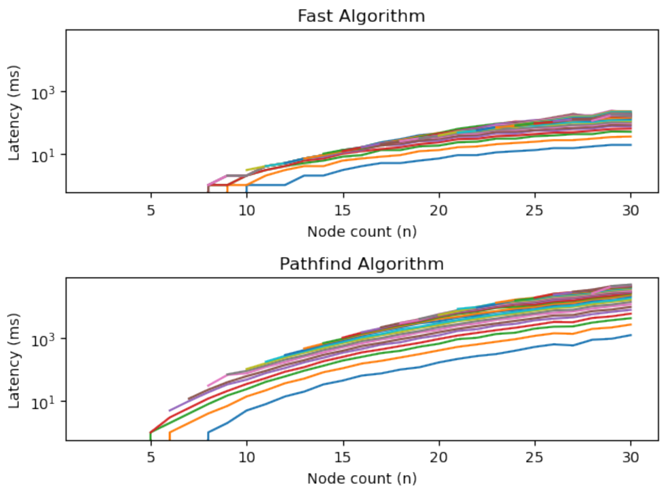


Fig. 4: Duration of route generation.

D. Error Rate

While in theory the routing algorithms can be completely error free, both algorithms still experience failure. This is because in the end of the day, they are still estimations of the optimal routing tables. In Figure 5 the failures in routing is shown. Currently it is not clear why these faults appear, this can be a starting point for further research. In the figure you can see the failures, but they seem significantly more commonplace than in reality. The fast algorithm resulted in 11 failing graphs, and the path-find algorithm in 8. This is out of the 4382 different considered graphs. This gives an error rate of $0,0025$ and $0,0018$ respectively. It means that the current route building algorithms cannot be used reliably for BRB or RC in their current state.

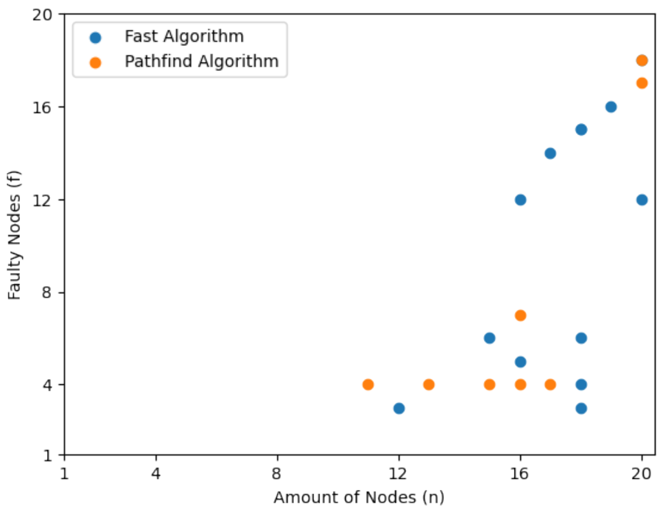


Fig. 5: Failures of the route building algorithms.

VI. Responsible research

In Section V-A we discussed in depth on how we set up the experimentation, but not exactly why it was responsible. This is something we will dive into in this section. With this research we have tried to remove as many sources of bias as possible. This is done by testing a small amount of graphs very extensively, to put the proposed algorithms to the test in all varying circumstances. In this paper we have also shown explicitly where there the faults of the algorithms lie.

With this paper it is reasonably possible to reproduce our results. The code is available to see at the TUDelft GitLab server to verify our specific test setup. The raw output for our tests are also available there, even including the specific topologies used in the *topogies* folder. The *dissyssym* folder contains the source code to build the binaries in order to run the simulation. The *dissyssym-lib* folder contains the meat of the code, which can be build separately and loaded as a Shared Object in a different project with minimal changes. In the *scripts* folder you will find some Python scripts which can generate the graphs used in this paper. It is not possible to reproduce the results exactly, as it depends on random sources and race conditions. This difference should however be minimal, and does not affect the outcome of the paper in any significant way. As it's shown, we have done any reasonable accommodation to remove any bias from the experiment setup.

VII. Conclusion

In this paper we introduced a routed algorithm with signatures for Byzantine Reliable Broadcast. We showed that this improvement theoretically decreased the amount of messages required to an optimal level. This makes the routed algorithm very interesting and a promising method to achieve BRB in large networks. Next we introduced several algorithms to compute the routes needed for the algorithm to work. All of them have their advantages and disadvantages, and in the end none of them are easy to recommend. In the evaluation section we discussed the performance of the algorithm in comparison to a flooding algorithm. There we also show that pre-computing the routing table takes a significant amount of time, but this has no impact later on. We also show that the current route building algorithms aren't completely reliable, with an average failure rate of $0,22\%$. While this is a small error, this does mean it still needs work before it can be used for Byzantine Reliable Broadcast.

A. Future Work

While routed algorithms are very promising to minimize the message complexity, the computation of these routes is still an unsolved problem. Before this can be used in BRB networks, a better method to generate the routing table has to be created. The main things to improve from the algorithms in this paper is the unreliability issue and the expensive computation. When these have been solved, then it can be interesting to look into reducing the amount of required links even further. It is still unclear if there exists an algorithm in P which can give an optimal solution, which is also an interesting angle.

References

[1] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987, ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(87)90054-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/089054018790054X.

[2] D. Dolev, "Unanimity in an unknown and unreliable environment," pp. 159–168, 1981. DOI: 10.1109/SFCS.1981.53.

[3] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, and S. Tixeuil, "Practical byzantine reliable broadcast on partially connected networks," pp. 506–516, 2021. DOI: 10.1109/ICDCS51616.2021.00055.

[4] T. Anema, "Message efficient byzantine reliable broadcast protocols on known topologies," 2021. [Online]. Available: http://resolver.tudelft.nl/uuid:44c79878-454d-4313-80f9-37d7e2e84431.

[5] R. Klabér, "Byzantine reliable broadcast on partially connected networks with signatures," 2021. [Online]. Available: https://resolver.tudelft.nl/uuid:c847d0e7-d85c-438e-97e6-ee917bb9f094.

[6] S. Bonomi, G. Farina, and S. Tixeuil, "Multi-hop byzantine reliable broadcast with honest dealer made practical," *Journal of the Brazilian Computer Society*, vol. 25, no. 1, pp. 1–23, 2019.

[7] K. Menger, "Zur allgemeinen kurventheorie," *Fundamenta Mathematicae*, vol. 10, no. 1, pp. 96–115, 1927.

[8] T. Hansen and D. E. E. 3rd, *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, RFC 6234, May 2011. DOI: 10.17487/RFC6234. [Online]. Available: https://www.rfc-editor.org/info/rfc6234.

[9] A. Steger and N. C. Wormald, “Generating random regular graphs quickly,” *Combinatorics, Probability and Computing*, vol. 8, no. 4, pp. 377–396, 1999.

[10] A.-.-H. Esfahanian, “Connectivity algorithms,” in *Topics in structural graph theory*, Cambridge University Press Cambridge, 2013, pp. 268–281.