Byzantine-Resilient Real-Time Reliable Broadcast on Partially Connected Topology Cases

Thom Breugelmans
Supervisor(s): Jérémie Dechouchant, 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*—Increasing digitalisation of society due to technical advancement has increased the appearance and size of cyber-physical systems. These systems require real-time reliable control, which comes with its challenges. These systems need reliable communication despite the presence of attacks or faulty processes and bad connections, which can disrupt the timeliness and correctness of these applications. Furthermore the scale of these systems are becoming increasingly larger as such the cost of connecting all processes that make up these systems become very high. This paper addresses the issues of reliable communication in these conditions while allowing for networks to be partially connected. We show that the protocol discussed can meet real-time requirements of these systems even under conditions of bad connections and low network connectivity.**

***Index Terms*—real-time distributed systems, reliable broadcast, byzantine resilience, partially connected networks**

I. Introduction

Many of our physical structures get monitored by automated systems or are being automated. These structures are called *cyber-physical systems* (CPS) [1] and often consist of sensors, actuators and computing devices. These devices are connected over networks, often wireless, and bring with them a fair share of problems. In this paper we will emphasise on the communication aspect of these CPS and focus on the problems of *reliable communication* (RC) over faulty networks and *real-time* communication.

Networked systems especially wireless systems bring uncertainties with them. Links connecting two nodes in a network can fail and as a result hamper timeliness of messages due to delay or corrupting/dropping the message. Another problem with networked systems is that they are liable to process faults of other nodes on the system, these faults can range from software errors or bugs to malicious intent from an unknown party.

Both of these problems have been subject of research in the past as they are often occurring problems. The *Byzantine Generals Problem* [2] models the problem of having reliable communication and reaching a *consensus* in a network despite the presence of adversaries. What we are looking for is not consensus, however, but broadcast. Previous research on RC gave solutions to the problems that networks bring with them. However these solutions either cannot perform under time constraints, cannot take adversaries in a network into account or simply have high network requirements by needing a fully connected network.

As such in this paper we will present a protocol that is capable of real-time performance and can guarantee the reliable delivery of messages in partially connected networks that are subject to faults (i.e., one third of the processes being Byzantine and high message loss rates). Specifically we extend upon the research of Kozhaya et al. namely their presented protocols, RT-ByzCast [3] and PISTIS [4]. We explore the network connectivity requirements these protocols, and show the performance implications of varying degrees of connectivity. Furthermore we propose an alteration to the protocol that reduces the bandwidth by lowering the amount of messages sent using probabilities.

The paper is structured as follows. Section II will elaborate on previous studies similar to this paper, what their contributions are and why they differ from our paper. Section III contains a model of our system. Section IV shows an overview of the underlying principle of the protocols of PISTIS and RT-ByzCast. Section V discusses how the protocols perform in partially connected networks. Section VI shows and discusses our proposed method for reducing bandwidth. Section VII contains the analysis of the results and how those results were gathered and realised. Section VIII we will discuss our ethics while working on this paper. Lastly section IX states some future work and improvements and in section X we will conclude.

II. Related Work

Reliable communication (RC) especially in networks with *byzantine* properties has been subject of research before. Evidence of this is the existence of the Byzantine Generals Problem (BGP) and the existence of protocols that solve this problem. Such as Bracha's [5] and Dolev's [6] protocols.

Both protocols solve BGP, however they do not fully conform to our specifications. Bracha states the requirement of a fully connected network to function [5], which conflicts with our requirements and goals. Dolev, however, has no need for a fully connected network [6]. The problem with Dolev being that it cannot adhere to timing constraints and thus can run for an undefined amount of time. As such, while both protocols solve BGP, they cannot be replaced by our protocol as they have specifications that conflict with our requirements.

RT-ByzCast [3] and PISTIS [4], both by Kozhaya et al., are two protocols that work very similarly and both solve BGP under real-time constraints. The main difference between the two is that PISTIS is an improvement on the former and is event-based in contrast to RT-ByzCast, which is round-based [4]. These protocols work similar to Bracha, however they aggregate signatures in the messages to verify the contents and that it has been received by said node. These protocols, however, state the necessity for a fully connected network.

III. System Model

Since we extend upon the work of the PISTIS and RT-ByzCast protocols our system model will be very similar to theirs.

Processes. The system is modelled as a distributed system where each entity of the system is a *process*. Processes in the system are denoted by p_i and the system as a whole as the set $\Pi = p_0, p_1, ..., p_N - 1$ of $N > 1$ processes. Processes are able to be uniquely identified and processes themselves are able to use signatures to sign messages. A signature of process p_i is denoted by σ_i. A process should have full knowledge of the topology of the network in which it resided, this means knowledge of (approximated) link failure, all processes in the network and which edges or links are connected to which processes. Processes can be faulty or byzantine as a result of which they violate timing constraints or modify messages of a broadcast.

Clock. Processes have access to local clocks that contain negligible drift from real time.

Communication. Processes are connected to other processes through the use of links. A pair of processes is connected through two uni-directional links, e.g. processes p_i and p_j are connected with a link from p_i to p_j and vice versa. These links have a probability $P(linkfailure)$ to correctly deliver the message from p_i to p_j within a maximum delay d (d is known to the process). A link fails to correctly deliver a message if the message is either delivered after the maximum delay d or if the message is altered in the process of delivering the message.

Network. A network is comprised of the set Π of processes and their links. The network has the ability to function with up to f byzantine nodes. In order to function the system must have a size $N \geq 3f + 1$ and each process must have at least $f + 1$ links.

IV. Protocol Overview

Both the RT-ByzCast and PISTIS protocols have very similar underlying working principles with the main difference being that RT-ByzCast is round-based [3] and PISTIS event-based [4]. Both protocols send messages at intervals, with RT-ByzCast being every round r and PISTIS after every delay d denoted by t. Since the principles are so similar we use the notation of r for the explanation of the protocols.

A. Overview

Processes in the network regularly send out heartbeats to indicate liveliness denoted as `HB` messages. These messages contain the heartbeat of the process itself and the heartbeats the process has gathered from its surroundings. If a process does not receive $2f + 1$ heartbeats in an interval of $\mathbb{R}$ rounds the process will move into the $DeadState$ as to not implicate the timeliness of the network. A process in the $DeadState$ does not send any messages.

When a process starts broadcasting a message or receives a broadcast it will start sending out `ECHO` messages containing the content of the broadcast, signatures of itself and the signatures of other processes of said broadcast it has received, and the heartbeats it has gathered. A process aggregates signatures from the `ECHO` messages it receives and passes it along in their next `ECHO`. A process will start sending `ECHO` messages every round when it has been made aware of a broadcast. As a result it will no longer send `HB` messages, however, the heartbeats will be included in the `ECHO` messages.

If a process has received $2f + 1$ signatures the process will start delivering the message by sending `DELIVER` messages. These messages also contain the content of the broadcast and all aggregated signatures, however, heartbeats are no longer present in the message. The process will deliver the message for $2\mathbb{R}$ rounds. The broadcast is successful if all correct processes in the network are delivering the message.

V. Extending to The Partial Topology

PISTIS and RT-ByzCast state a requirement of a fully connected network to function, e.g. in a system with N nodes each node has $N - 1$ links, however thanks to the use of signatures this requirement can be lowered to $f + 1$. In this section we will test innate ability of both protocols to perform in partial networks and what the performance implications are when the protocol executes in a partially connected network versus a fully connected network.

A. Explanation

Firstly to understand why the reduction is possible we first need to know the reason for reducing to $f + 1$ and not lower or higher. We know the system contains N processes where $N \geq 3f + 1$ and we know that the system can support up to f byzantine processes. What we want to achieve is a minimal network with the property that each process can still reliably send and receive messages and know if a message has been tampered with. As such each process must have at least a single path to another process that passes no byzantine processes. Thus each process in the network must have at least one neighbour that is not byzantine. This does not automatically result in the $f + 1$ as certain distributions of the byzantine processes in a network still allow for this constraint, however if we want to guarantee the ability to deliver a message a set of constraints must be given that allow for message delivery unrelated to the distribution of processes in a network. As a result the constraint must take into account the probability that all byzantine processes have a common neighbour and that neighbour must still have a link to a non-byzantine process, hence the $f + 1$.

However simply needing $f + 1$ neighbours is not enough as processes still need to *verify* a quorum has received the message, which is realised through the use of signatures. The processes need a way to verify a quorum has received a message that cannot be tampered with. If there is a way to tamper with the verification process, then the byzantine processes can exploit this method and broadcast arbitrary messages. Signatures indicate a message has been sent and received by a process. If the signature method is *cryptographically* secure, then byzantine processes cannot tamper with the signature and the content.

As a result of the $f + 1$ neighbours and the signatures a process can then ultimately confirm a quorum has received the message.

B. Testing the Partial Topology

As discussed in V-A theoretically RT-ByzCast and PISTIS should support partial topologies and simulations confirm this hypothesis. Using the public simulation code of PISTIS [1] and our own code we tested the innate ability to work under partial topologies. We tested the protocols on various network sizes and with varying probabilities of link failure. What we found was that all processes in these networks were able to successfully

[1] https://github.com/vrahli/pistis

obtain $2f + 1$ signatures, confirming our hypothesis of the protocols being able to perform in a partial topology.

C. Implications

Failure to reach a quorum is the biggest implication of reducing the amount of links, however this can be mitigated by increasing the parameter $\mathbb{R}$ for RT-ByzCast and $\mathbb{T}$ for PISTIS. These parameters represent the amount of time or rounds that can pass before a process enters the so-called $DeadState$ [3, 4]. Doing this allows the processes a longer time frame to send messages to reach the specified $2f + 1$ signatures in the first $\mathbb{R}$ rounds or $\mathbb{T}$ time. If not enough signatures are received after this time, as specified by the protocols, a process should enter the $DeadState$ [3, 4]. The problem with this is that the performance of the protocol will suffer due to this as the time frames become longer. Both protocols state that, with negligible chance at failure, all non-Byzantine processes will reach a byzantine quorum within $3\mathbb{R}$ rounds for RT-ByzCast [3] and $3\mathbb{T}$ for PISTIS [4] and since we are extending this parameter as such the time within which the protocols state a quorum is reached is also lengthened.

In section VII these implications will be explored and discussed along with an analysis of the performance of the protocol on the partially connected topology versus the fully connected topology.

VI. Improving Performance

To increase performance we examine two possibilities, improve performance by reducing the time the protocol takes to attain a successful broadcast or by reducing the bandwidth requirement by decreasing the amount of messages sent or decreasing the size of the messages. Improving performance by reducing the time is the most optimal solution, the only way to achieve this, however, is to reduce the amount of intermediary processes between two processes. The reason being that with more connections less processes need to be traversed to go from A to B, ultimately lowering the minimal amount of rounds needed to succeed and thus lowering the time. This means that the performance can be improved by increasing the number of links a single process has, which is exactly the opposite of the goal of this paper. The next best option is to reduce the bandwidth the protocol needs by reducing the amount of messages that are being sent or reducing the size of the messages.

To reduce the bandwidth and thus improve performance a method of calculating the probability a process has received a message and signature will be used. With this information a decision will be made on whether to send a message or to not send a message, decreasing the amount of messages sent. This method will be discussed in VI-A and the implementation details of the method will be discussed in VI-B.

A. Overview

Our proposed improvement keeps track for each process whether it has received and signed the message of the broadcast. With this information a process can then calculate the probability a neighbour has received the message and whether it has received a specific signature. If a neighbour of a process has received all currently known signatures, and thus also the actual message, with a probability higher than some specified threshold d, then the process will stop sending $ECHO$ messages to that neighbour.

To calculate the probability a neighbour has received a signature we make use of the fact that the topology of the network is known as specified in Section III. As a result of the topology being known we can compute the shortest path from any process to any other process. If a process p_i knows that process p_k has signed the message at a time t_0, e.g. by having received the signature, then using the shortest distance we can compute the probability a neighbour p_j has received that signature from p_k. The equation would look as follows:

$$P_{received} = \begin{cases} 1 - P(linkfailure)^{t-t_0}, & \text{if } t - t_0 \geq dist \\ 0, & \text{otherwise} \end{cases}$$

where $P(linkfailure)$ is the probability of a link failing to adhere to the constraints of timeliness and correctness, t is the current time/round and $dist$ is the minimal distance between p_j and p_k.

To find t_0 we could use the time process p_i received the signature, however that would not be a very accurate representation of t_0 and thus also widen the distance between the actual probability of p_j having received the signature of p_k and the computed probability. To accurately get t_0 we will need to store, in addition the the signature, the time a process has signed the message.

In addition to using the probability p_j has received the signature from p_k, we can also use the probability p_j has received the signature of p_k from p_i. In this case t_0 will become the time/round p_i has received the signature of p_k and $dist$ will become the shortest distance between p_i and p_j. Combining these two probabilities we have an accurate representation of the probability p_j has received the signature of p_k.

With this information process p_i can then compute the probabilities of receival for each of its neighbours for all known signatures. Then, if the probability of receival does not reach the required threshold d, p_i will retransmit the message to that specific neighbour, else it will hold off on that message.

Holding off on messages however is dangerous as that would also mean *heartbeats* are not sent. Since the underlying protocol is very reliant on the heartbeats we will transform the messages to be `HB` instead of `ECHO` or `DELIVER` as opposed to simply dropping the message. That way heartbeats are still passed around the network, preventing processes from unnecessarily moving into the $DeadState$. This means that we are not entirely dropping messages, but only reducing the size of the message. This is sub-optimal, however, since we are reducing the size, we are decreasing the bandwidth. If these heartbeats can also be dropped we can further lower the bandwidth requirements, which is a topic that is touched upon in the future work Section IX.

This method of calculating the probability of receival for a process p_j is not perfectly accurate. The current method does not take into account the other possible paths a message might take to reach a process, which would increase the probability process p_j has received the message. As such this method produces a lower probability than is actually the case in most situations. This point is further discussed in section IX.

B. The Algorithm

Algorithm 1 `should-send(`p_j`, Σ, t)` *at process* p_i

Parameters:
  p_j: the neighbour process to send to or not
  Σ: the aggregated signatures and their times as a tuple
  t: the current round

if $|\Sigma| > 2f$ **then**
  // the current process is delivering, thus only use the probability of p_j having received from the process itself
  $dist \leftarrow$ distances$[p_i][p_j$
  $P_{received} \leftarrow 1 - P(linkfailure)^{t - t_{start\_delivering}}$
  return $P_{received} < threshold$
else
  for $(\sigma_k, t_k) \in \Sigma$ **do**
    $P_{received} \leftarrow 0$
    // get the shortest distance from p_k (process of signature) to p_j (neighbour process)
    $dist \leftarrow$ distances$[p_k][p_j]$
    if $t - t_k < dist$ **then**
      $P_{received} \leftarrow 1 - P(linkfailure)^{t - t_k}$
    end if

    if $P_{received} < threshold$ **then**
      // a signature has not reached required threshold, so message should be sent
      return $TRUE$
    end if
  end for

  return $FALSE$
end if

Algorithm 2 `send(Msg, `p_j`, t)` *at process* p_i

Parameters:
  Msg: the message to send
  p_j: the neighbour process to send to
  t: the current round

if $Msg =$ `HB` **then**
  Send Msg to p_j
else
  if `should-send(`p_j`, Σ, t)` $= FALSE$ **then**
    // only send heartbeats
    Send `HB` to p_j
  else
    Send Msg to p_j
  end if
end if

The following algorithm, Algorithm 1, determines whether or not to send the message or change the message to be a heartbeat. The method of integrating it into the existing protocol is shown in Algorithm 2. This method will be called instead of the normal `Send` method and is called each time an attempt at sending a message is made.

As stated before in addition to storing the signatures the messages also need to contain the time of signing for the method to work. Alongside this the process also needs to have knowledge of the round it started delivering the message, this variable is denoted by $t_{start\_delivering}$ in method `should-send` in algorithm 1. As visible the `should-send` method in algorithm 1 returns a truth value iff the process should send the message to its neighbouring process p_j. The method it computes whether it should send or not differs slightly if the process is echoing or delivering. When echoing it uses the probability of p_j having received from itself and from the process that has signed the message. When process p_i is delivering it uses only the probability p_j has received the `DELIVER` message from itself. If the computed value has not reached a specified threshold, then the process receives a $TRUE$ to indicate it should send a message.

VII. Evaluation

In this section we evaluate the performance implications the protocol has when executing in partial topologies, we also examine the proposed method for reducing the amount of messages.

To test the protocols and to generate data we made use of the OMNeT++ 5.5.1 network simulator and our own implementation of the protocols.

All our data points are gathered using $\mathbb{R}/\mathbb{T} = 10t$, except for the information shown in figure 1 as there we wanted to know the minimal required value of $\mathbb{R}/\mathbb{T}$. The data presented in section VII-A are the average results of 10 or more simulations. In section VII-B they are the average results of 1000 simulations.

A. Performance in Partial Topologies

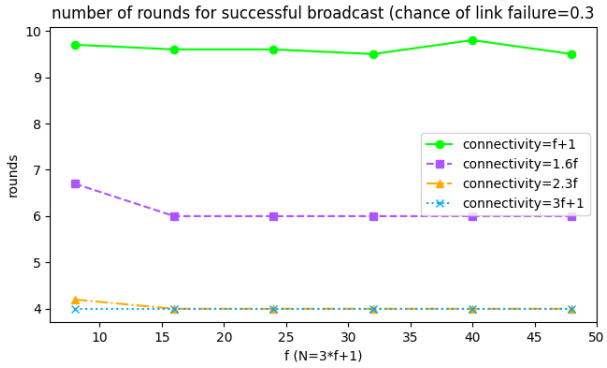

Fig. 1. Performance of the protocol on different levels of connectivity on increasing network sizes (N)

First of all, simulations show that the protocol works on networks with connectivities as low as $f+1$. If no recovery from the $DeadState$ is allowed the protocol becomes dependent on how low the $\mathbb{R}/\mathbb{T}$ value is. Figure 1 shows that with $P(linkfailure) = 0.3$ they need to be at least 10, this value minimal value decreases as the connectivity goes up, visible in figure 3, but lowers again when $P(linkfailure)$ increases

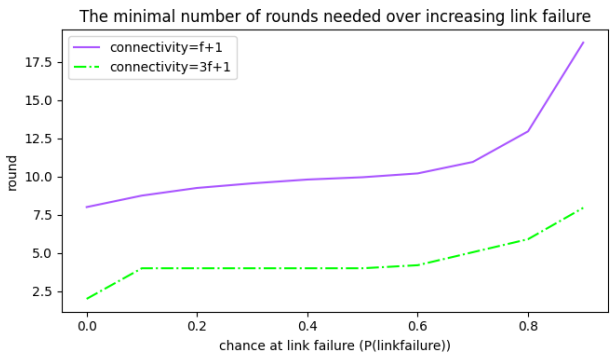

Fig. 2. The minimal number $\mathbb{R}/\mathbb{T}$ should be per $P(linkfailure)$, with $f = 24$

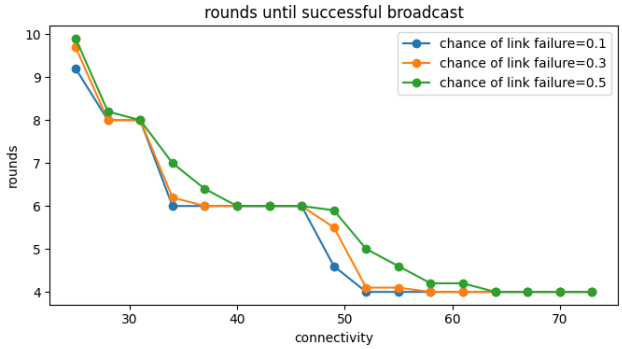

Fig. 3. The number of rounds the protocol needs until a successful broadcast, $f = 48$

as visible in figure 2. In figure 2 we can also see that for probabilities ≤ 0.7 the minimal number of rounds stays under or around 10, increasing the probability more we see a spike in the minimal number of rounds needed. The curve for fully connected systems is similar to the curve for systems with $connectivity = f + 1$, albeit flattened and lower.

When a recovery from $DeadState$ is allowed, however, the protocol will work on networks with lower values for $\mathbb{R}/\mathbb{T}$.

From this we can conclude that the protocol performs in networks with $connectivity \geq f+1$. From figure 2 we can conclude the protocol needs approximately twice as many rounds to succeed when working in networks where $connectivity = f+1$ as opposed to networks where $connectivity = 3f + 1$. This difference will stay the same when the probability of link failure ($P(linkfailure)$) stays within acceptable ranges, below approximately 80% chance of failure. Although this is a drastic decrease in performance, from figure 3 we can see the required amount of rounds stays approximately the same as the size of the network increases. This means that even if the network grows the latency until a successful broadcast stays roughly the same. From these conclusions we can say that, although slower, the protocol can meet the timing constraints of CPS applications with lower connectivity. This means that a trade-off can be made between the cost of network connectivity versus the latency of delivering the message.

B. Decreasing Retransmissions

TABLE I
Performance of the proposed method, $f = 49$ and $threshold = 0.1$

	$P(linkfailure)$	messages sent	#rounds
non-optimised	0.0	32450.00	8.00
	0.3	34034.00	8.41
	0.6	40462.00	9.91
optimised	0.0	20600.00	8.00
	0.3	28342.10	8.60
	0.6	40216.00	9.87

TABLE II
Average probability of successful broadcast for increasing $threshold$ with $f = 49$

	$threshold$				
$P(linkfailure)$	0.0	0.2	0.4	0.6	0.8
0.0	1.0	1.0	1.0	1.0	1.0
0.2	1.0	0.0	0.0	0.0	0.0
0.4	1.0	1.0	0.0	0.0	0.0
0.6	1.0	1.0	1.0	0.0	0.0
0.8	1.0	0.97	0.99	0.96	0.0

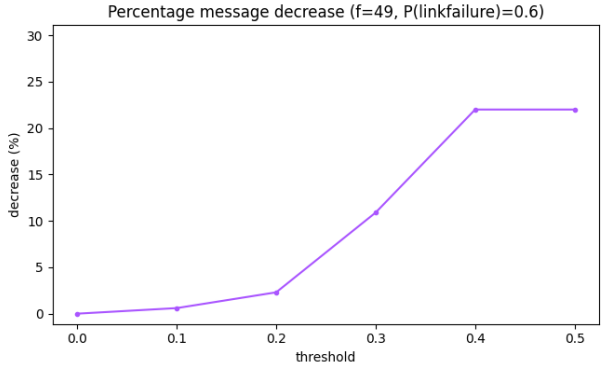

Fig. 4. Percentile message decrease for increasing $threshold$ with $P(linkfailure) = 0.6$ and $f = 49$

As visible in the table I there is a decrease in the messages that are sent. The reported numbers of messages sent omit the `HB` type messages as each time the `should-send` method returns $TRUE$ the message will be transformed in a `HB` message, as such, if they were included, no difference in messages sent would be found.

In table I we see that for $P(linkfailure) = 0.0$ there is an approximately 36.5% decrease in the number of messages sent. This value decreases when $P(linkfailure)$ increases, that is because the protocol requires 90% certainty ($threshold = 0.1$ that a neighbouring process has received the message. Therefore to reach the necessary level of certainty the process will have to retransmit the message more times. There is a drastic drop when $P(linkfailure)$ increases, when $P(linkfailure) = 0.6$ the decrease in messages only amounts to 0.6%, which is too low to be of any significance.

However, figure II shows that even when lowering the certainty value the protocol will still have approximately a 100% chance at success. The reasoning behind this is because the certainty is calculated using an heuristic that is less accurate than the actual value of process p_j having received the message, as such the certainty can be lowered to achieve better results. Figure 4 shows that increasing the $threshold$ also increases the percentage with which the messages decrease.

We can conclude that our proposed method decreases the messages sent by $\geq 20\%$ depending on the chosen $threshold$ with high certainty of success.

VIII. Responsible Research

Work ethics are very important to take into account, especially with research. If not, then the integrity of the research falls as the research might not be reproducible. To ensure integrity and reproducibility data gathered must be of sufficient volume and must not be tampered with by omitting data or modifying data. In this section we present our efforts to ensure the integrity of our research.

A. Integrity

The experiments performed in this research were created and thought up so that the results of said experiments can be reproduced. No data gathered and shown is ambiguous or highly specific. All methods and code used for this paper is publicly available such that it can be examined and used to reproduce the presented data[2].

All data presented in this paper is the result of multiple experiments, as previously stated in section VII, to assure that no bias is present in the data. This is especially important when working with probabilistic systems such as the links in our networks. Each data point presented is the average achieved by at least ten iterations of experiments in order to reduce the bias that might be present.

IX. Future Work

Although the research we presented has solid results improvements can be made. Specifically the method of lowering the number of messages sent can be improved.

Right now the method makes use of the probability a neighbour has received a signature from itself and from the owner of the signature. The resulting probability is lower that the actual probability since it does not take the different paths a message can take into account. A way to improve this accuracy is to keep track of all processes that you know have received the signature and use all those processes to compute the probability. This way you take more paths into account and as more processes are known to have received the signature the accuracy will increase. The biggest hurdle is to find a way to get information on which process has received what signature and in what round.

The method of reducing messages right now also simply transforms a message to a `HB` message when the probability of receival is high enough. As such heartbeats are always sent and forwarded. If we can reduce the amount of times heartbeats are sent the number of messages will drastically decrease and so will the bandwidth. This could probably be achieved with probabilities similar to the current method but it would require some modifications as a heartbeat is only valid for $\mathbb{R}$ rounds and a process needs at least $2f + 1$ valid heartbeats within $\mathbb{R}$ rounds.

[2]https://gitlab.tudelft.nl/cse3000-2022-reliable-communications/thom-rtbrb-in-partially-connected-networks

X. Conclusion

In this paper we studied algorithms displaying byzantine reliable broadcast. We examined how real-time byzantine reliable broadcast protocols, RT-ByzCast and PISTIS, performed in partially connected topologies and how that affects the reliablility of the protocols and the performance. We showed that even if we decrease the connectivity of the network to $f + 1$ the protocols are still capable of obtaining a successful broadcast. We found that, although the drop in performance is significant when lowering connectivity to $f + 1$ (50% slower with probabilities of link failure $\leq 80\%$), the performance is roughly constant with increasing network size. We concluded that while the protocol might not meet certain timing restraints when the connectivity is $f+1$ a trade-off can be made between higher connectivity and higher performance.

Furthermore we presented a method to improve the bandwidth of the protocols. This improvement, with the right parameters, is able to reduce the messages sent by approximately 20%. We also showed that this improvement is reliable and that the protocol using the improvement succeeds with roughly 100% certainty.

References

[1] J. R. Moyne and D. M. Tilbury. "The Emergence of Industrial Control Networks for Manufacturing Control, Diagnostics, and Safety Data". In: *Proceedings of the IEEE* 95.1 (2007), pp. 29–47.

[2] L. Lamport, R. Shostak, and M. Pease. "The Byzantine Generals Problem". In: *Concurrency: The Works of Leslie Lamport*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 203–226.

[3] D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo. "RT-ByzCast: Byzantine-Resilient Real-Time Reliable Broadcast". In: *IEEE Transactions on Computers* 68 (3 Mar. 2019), pp. 440–454.

[4] D. Kozhaya, J. Decouchant, V. Rahli, and P. Esteves-Verissimo. "PISTIS: An Event-Triggered Real-Time Byzantine-Resilient Protocol Suite". In: *IEEE Transactions on Parallel and Distributed Systems* 32 (9 Sept. 2021), pp. 2277–2290.

[5] G. Bracha. "Asynchronous Byzantine agreement protocols". In: *Information and Computation* 75 (2 Nov. 1987), pp. 130–143.

[6] D. Dolev. "Unanimity in an unknown and unreliable environment". In: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*. 1981, pp. 159–168.