Project Report

CS-598

Fault Tolerant Distributed Algorithms

Crash Fault Tolerant Broadcast with Linear Message Complexity

Authors:
Shivram Gowtham
Samarth Aggarwal

Advisor:
Dr. Ling Ren

Abstract

We learnt the Dolev-Strong algorithm for crash fault tolerant broadcast under lock-step synchrony in the lectures. Dolev-Strong has $O(n^2)$ message complexity, which is much worse than the theoretical lower bound of $\Omega(n)$. We explored algorithms that perform better than Dolev-Strong. An algorithm by Galil-Mayer-Yung (state-of-the-art) solves the problem with $O(n)$ message complexity. The Galil-Mayer-Yung paper proposes two algorithms based on the concept of diffusion trees with varying heights, resulting in $O(n + f * \sqrt{n})$ (2-height) and $O(n)$ ($log(n)$ height) messages respectively. We studied their algorithm(s) and implemented the 2-height diffusion tree variant. We tested our implementation for a wide range of n, f and used the aggregated results to compare Galil-Mayer-Yung with Dolev-Strong. We also perform a theoretical message-complexity analysis of the algorithm to understand the average case and derive a potential worst case. This helps us with explaining our results and to draw some interesting conclusions.

Contents

1 The Broadcast Problem

The broadcast problem is a classic problem in distributed systems, where a designated sender s among n nodes wishes to broadcast a value v to all other nodes. This problem has been solved for different timing and fault models. We assume that the timing model is lock step synchrony, where all nodes progress through the protocol in perfectly synchronized rounds. The protocols we look at tolerate upto f crash faults and guarantee the following:

- Liveness: All nodes terminate the protocol eventually and output a value.
- Safety: All alive nodes should output the same value.
- Validity: If the sender did not crash, all alive nodes should output the sender's value.

1.1 Dolev-Strong Algorithm

Dolev-Strong [1] proposed an algorithm that solves broadcast against byzantine faults. A simplification of that leads to an algorithm for crash fault tolerance, where each node executes the following pseudocode [6] for $f + 1$ rounds.

```
if round ≡ 1 then
    if self ≡ sender then
        value ← input()
        broadcast(value)
    else
        value ← default
    end if
end if
if receive(value) and round ≤ f and not broadcasted then
    broadcast(value)
    broadcasted ← True
end if
if round ≡ f + 1 then
    return value
end if
```

1.1.1 Message Complexity

Essentially, each party broadcasts the value it receives to everyone, and terminates after $f + 1$ rounds to tolerate f crash faults. This algorithm guarantees liveness, safety, and validity condition. However, since each node broadcasts the value to everyone, the worst case message complexity becomes quadratic in n.

Theoretical lower bound message complexity for broadcast problem is $\Omega(n)$ as each node should receive at-least one message to ensure safety, in case the sender does not crash. The Dolev-Strong algorithm is far from this bound. In the next section, we look at an algorithm due to Galil, Mayer, and Yung which achieves the theoretical lower bound of linear message complexity.

2 Galil-Mayer-Yung Algorithm

2.1 Diffusion Tree

Unlike Dolev-Strong, the Galil-Mayer-Yung(GMY) algorithm [2] assigns each node a different role. A node can be a "sender", "coordinator", or a "leaf". Role assignment is done by visualizing the n nodes as a tree, with the root as the sender. The children of the sender will be coordinators, whose children would be leaves. The role of any node is to "diffuse" the value it learnt from its parent to its children.

The GMY algorithm has two variants based on the height of the diffusion tree. The height depends on the degree of each node. Each node having $\Theta(\sqrt{N})$ children would result in a height 2 diffusion tree. In this case, the algorithm solves broadcast with $O(N + f * \sqrt{N})$ messages in $O(f + 1)$ rounds. Note that the round complexity matches the theoritical lower bound [5]. The other extreme is when each node has only two children, resulting in a tree of height $h = \Theta(log(n))$. Nodes on height $1 \leq i \leq h - 1$ are called $level_i$ coordinators. In this case, the algorithm solves broadcast with $O(n)$ messages in "almost linear" time - $O(n^{1+O(\frac{1}{h})})$ rounds.

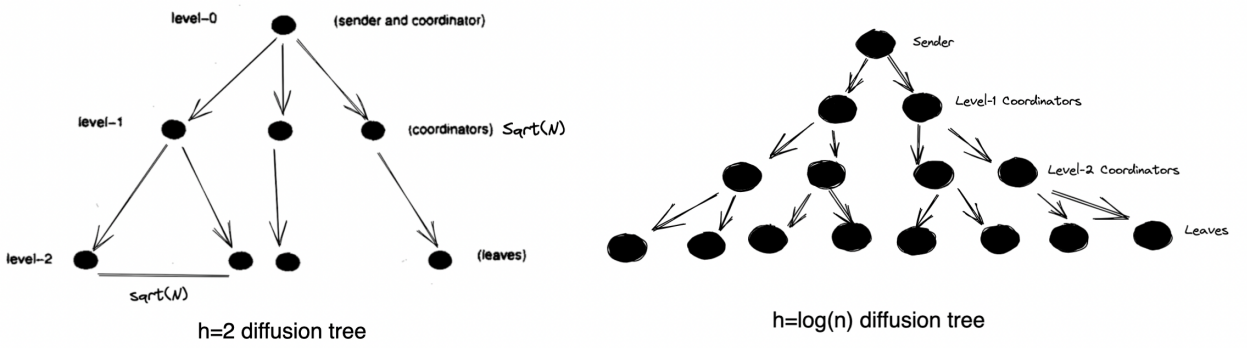


Figure 1: Diffusion trees

2.2 Checkpointing

Another basic building block of GMY algorithm is the Checkpointing problem. The Checkpointing routine is run at various stages of the GMY algorithm, where each alive process learns the list of all alive processes (called as result set E). It's also possible that the processes crash during the checkpoint routine execution. This is formalized by De Prisco, Mayer and Yung [3] as follows:

Let `START` be the set of live processes at the beginning of the execution of checkpointing and let `END` be the set of live processes when the protocol terminates. Then correctness of the result-set E should satisfy

- Agreement: E must the the same at each process.
- Validity: `END` $\subseteq E \subseteq$ `START`

2.3 GMY algorithm using $h = 2$ diffusion tree

In this section, we look at the algorithm with $O(N+f*\sqrt{N})$ using the height 2 diffusion tree. The algorithm assume that the initial sender has $ID = 1$ and the diffusion tree is constructed with increasing IDs along the level-order traversal (breadth-first traversal). The algorithm operates in multiple iterations of the following 5 phases. The diffusion tree at the beginning of the i^{th} iteration is constructed during phase 5 of $(i-1)^{th}$ iteration maintaining the invariant that the lowest `ID` alive node will be the sender and the next $\Theta(\sqrt{N})$ alive nodes will be the coordinators.

- Phase 1: In two rounds of lock-step synchrony, the sender first diffuses the value to the coordinators, who in turn diffuse it to their children(leaves). If the sender crashes during this stage, the coordinators who did not receive the value do nothing in phase 1.
- Phase 2: A checkpointing is executed among the coordinators (including sender), where each coordinator agrees upon E_1, the set of alive coordinators after phase 1. As a slight modification to the original algorithm proposed by [DMY94, 2.2], the checkpoint is also used to learn the senders value if it was sent to atleast one alive coordinator in phase 1. This can be done by making each coordinator send

the value it received in phase 1 (default if it didn't receive one) as part of the status message sent for checkpointing. If atleast one coordinator sends a non-default value, all coordinators in E_1 learn that value.

- Phase 3: Those coordinators that didn't diffuse the value in phase 1 but learnt it as a result of phase 2 now diffuse the value to their children.
- Phase 4: A second checkpoint among the coordinators is executed which results in E_2. The crucial observation here is that all level-1 coordinators not in E_2 potentially did not forward the value to their children, as they could have failed at any point of time in the previous phases. Each coordinator can compute this leaf set $L_{resend} = L - children(E2)$, where L is the set of all leaves in the current diffusion tree. L_{resend} are the set of leaves to which the value has to be diffused again.
- Phase 5: This is a recursive phase. If $|L_{resend}| = 0$, then all leaves have received the value, and the algorithm proceeds to termination. If $|L_{resend}| \leq \sqrt{N}$, we use the standard rotating coordinator technique [4] to solve broadcast for the remaining small subset of leaves. If $|L_{resend}| > \sqrt{N}$, a new $h = 2$ diffusion tree is constructed with leaves as L_{resend} and we restart from phase 1.

2.3.1 Synchronization and Recruitment

The paper does not explain in detail about synchronization, termination and reconstructing the diffusion tree in Phase 5. In this section, we look at a few key aspects of these problems and how the algorithm tackles them. Synchronizing all five phases over multiple iterations is crucial as all nodes should maintain a consistent view of the current diffusion tree and know their role. The nodes use the result of checkpointing and timeouts for maintaining consistency. For example, each leaf node sets a timeout (which is reset in case of message arrival) to identify if all lower ID nodes have failed, so that it can take over as the sender. One thing to note is that the node with lowest ID in E_2 will become the sender for the next diffusion tree. The coordinators use E_2 to compute their "distance" to this next sender, and set timeouts proportional to this distance for taking over as the sender. To terminate the algorithm after all nodes have received the value, the current sender diffuses "commit" and "good-bye" messages to everyone using the same five-phase recursive technique.

The lowest ID node in E_2 becomes the sender in the next diffusion tree. In case L_{resend} is large, it may not be possible to just use the current set of coordinators from E_2 to complete diffusion. When this happens, the new sender has to "recruit" some leaves as new coordinators during phase 5. In Figure 2, until phase 4, coordinators $\{1, 4, 5\}$ have crashed (marked in red). The leaves marked in green have received the value for sure. On the other hand, the ones in red, $L_{resend} = \{14, 15, ...21\}$ may not have received the value. Node 2 becomes the new sender and 3 stays as a level-1 coordinator, with a new set of children $\{14...17\}$. The leaf node 6 is recruited to be the new coordinator and the resulting diffusion tree is as shown in the figure.

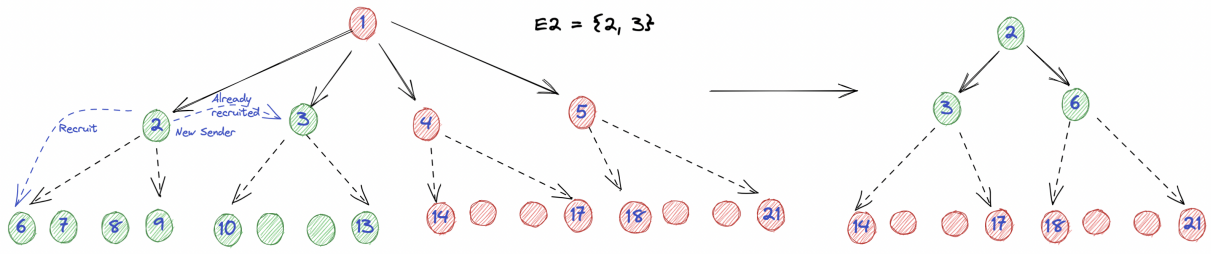


Figure 2: Recruitment and reconstruction of diffusion tree in phase 5

2.3.2 Message complexity

In one iteration of the five phases, the diffusion results in $O(N)$ messages, as atmost one message is sent across every edge. The intuition behind $O(N + f * \sqrt{N})$ messages is that for every coordinator that crashes,

we have to resend the value to its children(leaves), resulting in $O(\sqrt{N})$ messages. The paper claims that in worst case, when all coordinators fail in an iteration, the extra messages incurred is $O(\sqrt{N})$ when amortized per failure.

2.4 GMY algorithm outline using $h = log(n)$ diffusion tree

The iterative 5-phase algorithm described before is extended to a $log(n)$ height diffusion tree. In this tree, there are multiple levels of coordinators and only the nodes of the last level are defined as leaves. Phase 1 and 3 are same as before, where nodes perform diffusion along the tree edges. Here, the diffusion could take upto $O(log(n))$ rounds to complete. Phase 2 and 4 perform checkpointing of all coordinators (nodes upto height `h-1`). The checkpointing algorithm for height `h`, `CP(h)` is defined inductively using `CP(h-1)` with base case as `CP(1)` = [DMY94, 2.2]. In phase 5, based on the number of dead coordinators, (the height of) the next round diffusion tree is computed. The paper claims, without a detailed proof, that the $log(n)$ height diffusion tree variant of the algorithm achieves an asymptotic message complexity of $O(n)$ due to an amortized $O(1)$ extra messages for every failure.

3 Implementation

We implement the two-height diffusion tree algorithm as explained in section 2.3 and make our code available on GitHub[1]. Unlike the Dolev-Strong Algorithm, the Galil-Mayer-Yung algorithm does not give a formula to precisely count the number of messages passed between nodes. Hence, the implementation helps to count the messages passed in each run and aggregate these numbers to calculate the empirical statistics.

3.1 Specifications

We code our implementation in *Golang* in the interest of the ease of spawning multiple subroutines, runtime efficiency and high degree of parallelism support owing to lightweight go routines. We also assume the timing model of the algorithm to be *lockstep-synchrony* similar to Dolev-Strong to provide a fair comparison between the two algorithms.

3.2 Modules

The code is distributed across multiple modules for efficient organisation.

3.2.1 Main

The main function serves as the starting point of the implementation. It is responsible for spawning all the nodes and blackbox. Additionally during initialisation of the nodes, the main function chooses a random subset of atmost `f` nodes and sets the probability of failure for them as per the command line input. For all the other nodes, the probability of failure is set to `0`. Once the algorithm terminates, the main function also performs the liveness, safety and validity checks for broadcast. It also calculates the total messages exchanged in the run of the algorithm.

3.2.2 Blackbox

```
type Blackbox struct {
    Id          int
    NumNodes    int
    MyCh        chan types.Msg
    NodeCh      []chan types.Msg
    OutputCh    chan types.Msg
    CurrentTree types.DiffusionTree
```

[1] https://github.com/samarthaggarwal/Fault-Tolerant-Broadcast

```
    DeadNodes    []int
}
```

Some parts of the algorithm are not fully described in the paper such as the checkpointing algorithm and recruitment of coordinators. We introduce the blackbox module to keep the algorithm going despite the absence of details of certain parts. Hence, the onus of Checkpointing in *Phase-2* and *Phase-4* of the algorithm lies on the blackbox module. It acts as an oracle that provides the nodes with the result of the Checkpointing. Additionally, the blackbox module repairs or recontructs the diffusion tree after *Phase-4*. In doing so, the blackbox module also does the recruitment of the coordinators for the next round of the algorithm.

The blackbox module is also responsible for maintaining the *lockstep-synchrony* timing model by sending a `START_PHASE` message to each alive node at the start of next phase. Likewise, it also detects the termination of the algorithm and informs all alive nodes about the same.

3.2.3 Node

```
type Node struct {
    Id          int
    NumNodes    int
    Tree        types.DiffusionTree
    NodeCh      []chan types.Msg
    BlackboxCh  chan types.Msg
    OutputCh    chan types.Msg
    Value       int
    MsgCount    int
    FP          float64
}
```

Each node acts as the leader, coordinator, or leaf depending on the current state of the diffusion tree in the algorithm. Additionally, nodes with a non-zero probability of failure fail at a random time during the run of algorithm. Our implementation keeps track of the failed nodes by necessitating each failing node to send a message to *Blackbox* before failing.

3.3 Experiment Framework

This implementation was tested on MacOS and Linux. For every run of the algorithm, we store the following quantities:

(a) `numNodes`: Number of nodes

(b) `maxFaults`: Maximum number of faults that can take place

(c) `failProb`: Probability of Failure for failing nodes

(d) `realFaults`: Number of faults that actually occured during the run

(e) `honestMsgCount`: Total number of messages sent by honest (alive) nodes

(f) `failedMsgCount`: Total number of messages sent by crashed (failed) nodes

(g) `totalMsgCount`: Total number of messages sent by all nodes

(h) `multiplier`: Ratio of total messages exchanged to the theoretical upper bound

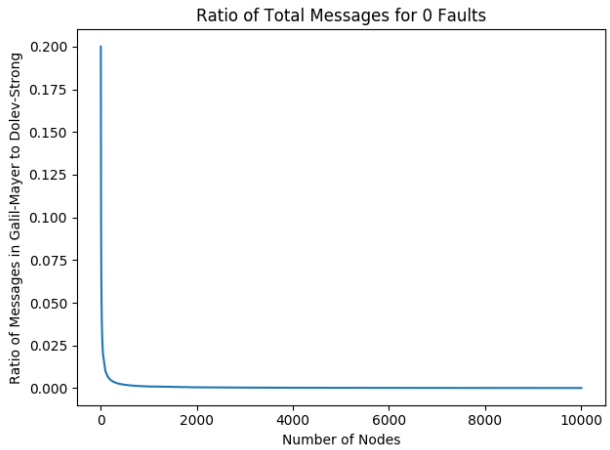
We develop a experiment script to facilitate running a large number of experiments. In the same, we vary `numNodes` from 5 to 10,000. For each value of `numNodes`, we vary `failProb` from 0 to 0.99 with a gap of 0.1. For each combination of `numNodes` and `failProb`, we set `maxFaults` to 0, 1, 2, $\frac{n}{4}$, $\frac{n}{2}$, $\frac{3n}{4}$, and n-1. Additionally, we run each of these experiments 20 times to ensure sufficient variance in the results for the average values to be representative of the general case.

4 Results

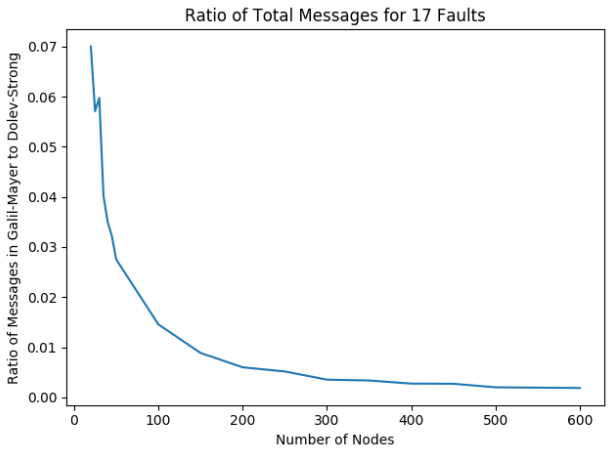
4.1 Comparison against Dolev-Strong

To compare the message complexity of Galil-Mayer-Yung algorithm against the Dolev-Strong algorithm, we plot the ratio of total messages exchanged for a constant number of faults as the number of nodes vary.

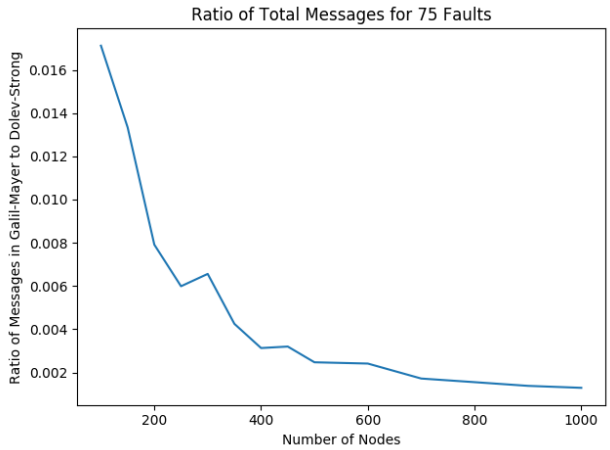
Let $GMY(n, f)$ and $DS(n, f) = n * (n - 1)$ be the number of messages exchanged in Galil-Mayer-Yung algorithm and Dolev-Strong algorithm in case of n nodes and f faults. Hence, we plot $X = n$, $Y = \frac{GMY(n,f)}{DS(n,f)}$ for a constant number of faults as the number of nodes vary. We observe that the plot resembles the plot of $\frac{1}{n}$ for small number of faults. Please refer to Appendix A.1 for more plots.



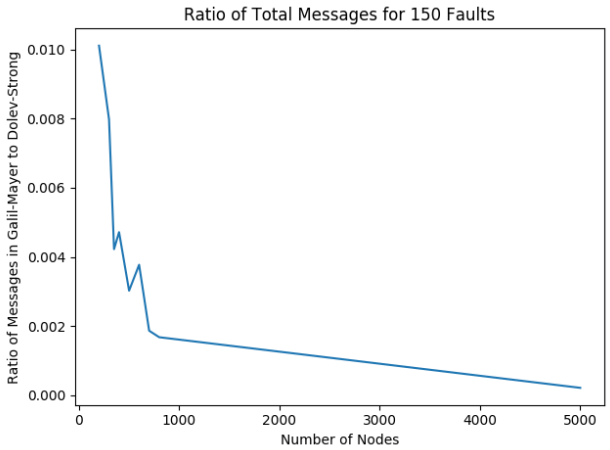
(i) $f = 0$



(ii) $f = 17$



(iii) $f = 75$



(iv) $f = 150$

Figure 3: Ratio of Total Number of Messages in Galil-Mayer-Yung to Dolev-Strong

4.2 Comparison against Theoretical Upper Bound

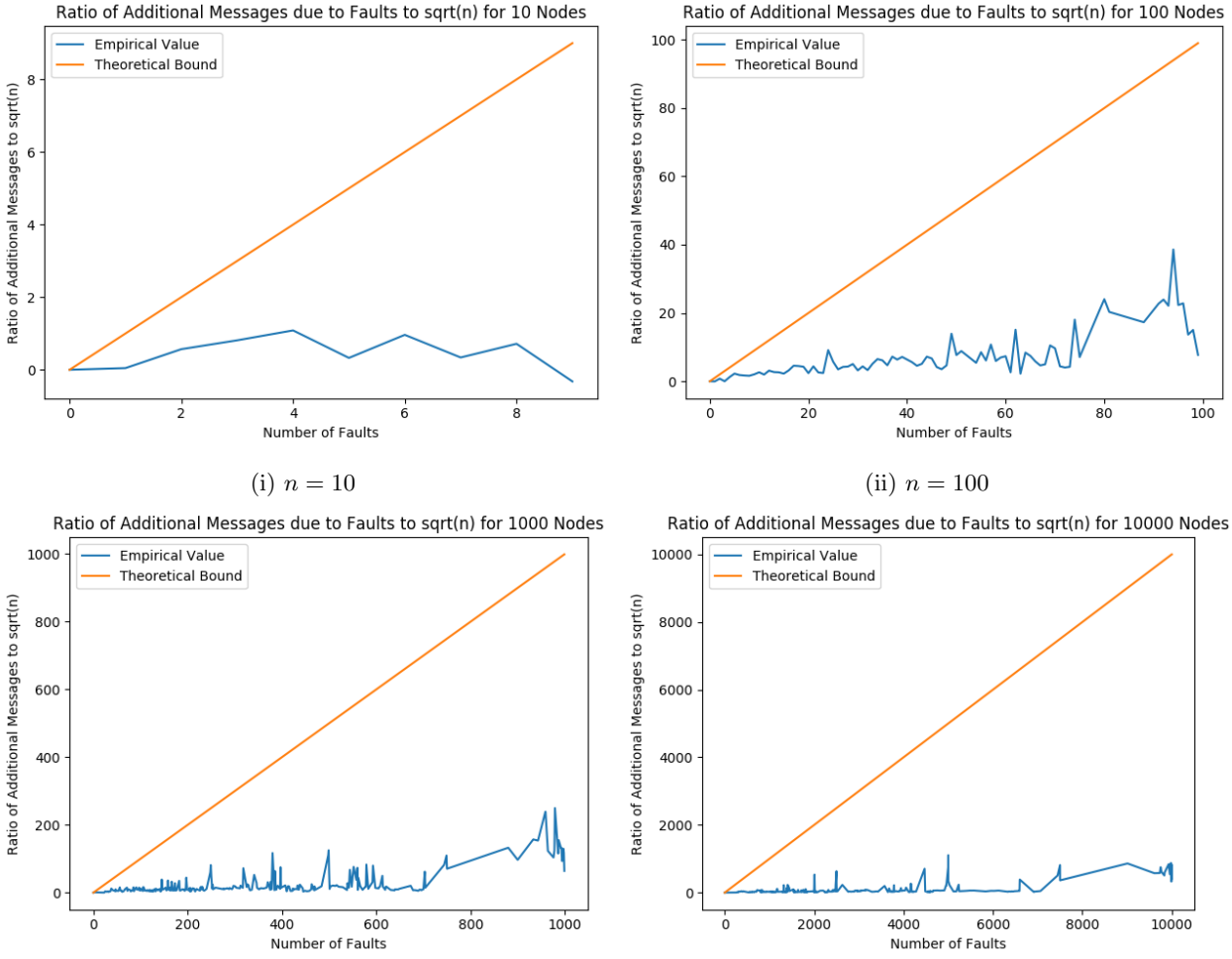
To compare the empirical message complexity of Galil-Mayer-Yung algorithm against the theoretical upper bound, we plot the ratio of additional messages exchanged due to faults over $\sqrt{n}$.

Let GMY(n,f) be the number of messages exchanged in case of n nodes and f faults.

Hence, we plot $X = f$, $Y = \frac{GMY(n,f)-GMY(n,0)}{\sqrt{n}}$ for a constant number of nodes as the number of faults vary.

Since GMY(n,0) = n-1 and $GMY(n, f) \leq n + f\sqrt{n}$, hence $\frac{GMY(n,f)-GMY(n,0)}{\sqrt{n}} \leq f$.

We observe that the empirical message complexity is much lower than the theoretical upper bound. Please refer to Appendix A.2 for more plots.



(i) $n = 10$

(ii) $n = 100$

(iii) $n = 1000$

(iv) $n = 10000$

Figure 4: Ratio of Additional Messages due to faults to $\sqrt{n}$

4.3 Potential Worst-Case

The Galil-Mayer-Yung algorithm does not have a proven worst case. We suggest a candidate for worst-case message complexity as the case where all $level_1$ coordinators fail in every round of the algorithm.

In our theoretical analysis (Section 5.2), we derive the lower bound of message complexity in this case to be $\frac{n(\sqrt{n}+1)}{2} - 1$. This lower bound asymptotically matches the theoretical upper bound on message complexity.

In order to test this worst-case, in each iteration, we failed all coordinators after they have diffused the value to their children. As per our experiments, this case does indeed achieve a message complexity that is very

close to the theoretical upper bound. This confirms that Galil-Mayer-Yung algorithm can indeed achieve $O(n + f\sqrt{n})$ message complexity although it performs much better in the average case.

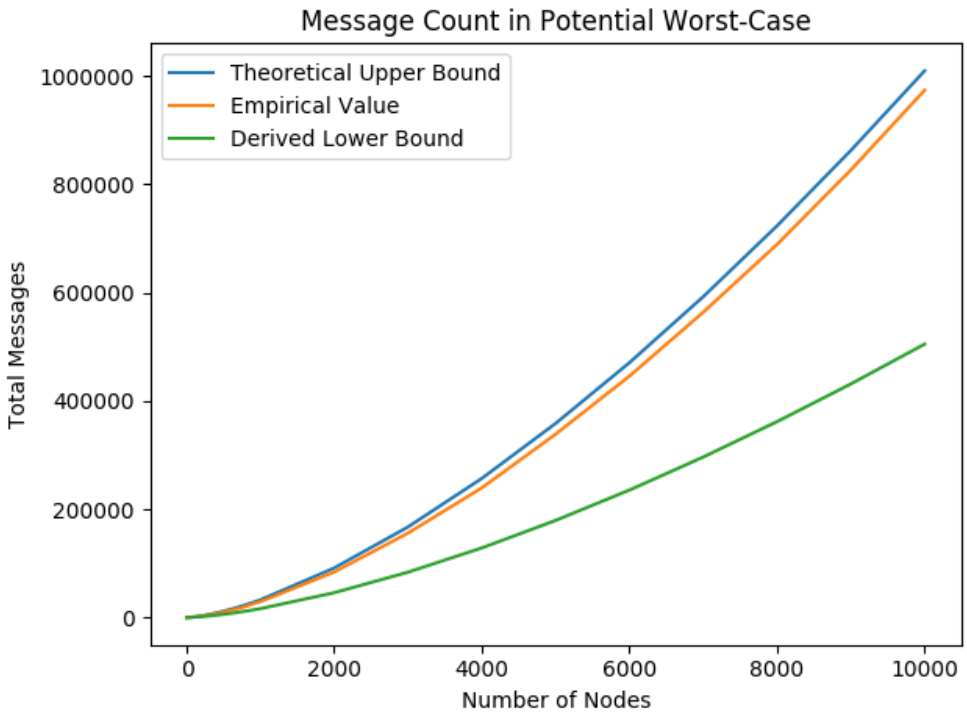


Figure 5: Comparison of Potential Worst-Case with Theoretical Bounds

5 Theoretical Analysis

5.1 Recurrence Relation for Message Complexity

In this section, we establish a recurrence relation for the total number of messages exchanged as per the Galil-Mayer-Yung algorithm.

Notation:
GMY(n,f): message complexity in case of n nodes and f failures
$level_i$: Nodes at depth i in the diffusion tree
f_i: failures in $level_i$ in the first round

Phase 1 and phase 3 together have n-1 messages. We ignore the messages in checkpointing (Phase 2 and Phase 4).

$$GMY(n, f) = n - 1 + \max_{f_0, f_1, f_2} GMY(\min(1 + f_1 + f_1\sqrt{n}, n - f_0 - f_1 - f_2), f - f_0 - f_1 - f_2) \text{ Equation(6.1)}$$

Explanation:
$(f_0 + f_1 + f_2)$ are the total number of failures in this round. Hence, $(f - f_0 + f_1 + f_2)$ failures are yet to occur in the subsequent rounds. $(n - f_0 + f_1 + f_2)$ is the number of remaining alive nodes after Phase-4. Hence, this is the maximum number of nodes that can be present in the diffusion tree formed in the next round. However, the number of nodes required to be present in the subsequent diffusion tree may be lesser depending on the number of $level_1$ coordinators that failed. Essentially, the children of f_1 nodes are to be given the

value in subsequent rounds. Each of the f_1 failed nodes had $\sqrt{n}$ children and the tree requires 1 root node at $level_0$. Hence, the subsequent diffusion tree will need atmost $(1 + f_1 + f_1\sqrt{n})$ nodes. Applying both these constraints, the subsequent diffusion tree will contain $\min(1 + f_1 + f_1\sqrt{n}, n - f_0 - f_1 - f_2)$ nodes. Since f_0, f_1 and f_2 can vary subject to the constraints given below, we take a max over them in the recursive call to GMY.

By definition, $f < n$, hence
$GMY(n, f) = GMY(n, min(n - 1, f))$

Constraints:
$f_0 \in \{0, 1\}$
$f_1 \in \{0, 1, 2, ..., \sqrt{n} - 1\}$
$f_2 \in \{0, 1, 2, ..., n - \sqrt{n} - 1\}$
$0 <= f_0 + f_1 + f_2 <= f$

Base Cases:
$GMY(n, 0) = n - 1$
In case of no failure, the diffusion is successful in Phase-1 itself. One message is sent along each edge in the diffusion tree comprising of n-1 edges. In total, n-1 messages are exchanged.

5.2 Potential Worst Case Analysis

To the best of our knowledge, there does not exist any proven worst case for the Galil-Mayer-Yung algorithm. We propose a case that is a potential candidate for the worst case. We also derive a lower bound to the message complexity in that case which is asymptotically equal to the worst case message complexity.

Intuition: The failure of the root ($level_0$) or leaf ($level_2$) does not trigger any additional messages. However, when a coordinator ($level_1$) fails, the algorithm sends a message to all its children even if the failed coordinator had already done so. This means that the cost of a failure, in terms of additional messages exchanged, is highest if that failure is in $level_1$. Extending this forward, we can achieve the maximum number of additional messages if all the coordinators fail in each round of the algorithm. Also, to maximize the failures in $level_1$ coordinators, no other node should fail.
Hence, the potential worst case is achieved when
$f_0 = 0$
$f_1 = \sqrt{n} - 1$
$f_2 = 0$
$f = n - 1$ (sum of failures across all rounds)

Lower Bound of Messages Exchanged
$GMY(n, f)$
$= n - 1 + GMY(\min(1 + f_1 + f_1\sqrt{n}, n - f_0 - f_1 - f_2), f - f_0 - f_1 - f_2)$
$= n - 1 + GMY(\min(1 + (\sqrt{n} - 1) + (\sqrt{n} - 1)\sqrt{n}, n - (\sqrt{n} - 1)), (n - 1) - 0 - (\sqrt{n} - 1) - 0)$
$= n - 1 + GMY(\min(n, n - \sqrt{n} + 1), n - \sqrt{n})$
$= n - 1 + GMY(n - \sqrt{n} + 1, n - \sqrt{n})$
$= n - 1 + (n - \sqrt{n}) + GMY(n - \sqrt{n} - \sqrt{n - \sqrt{n}} + 1, n - \sqrt{n} - \sqrt{n - \sqrt{n}})$

Reducing n will reduce number of messages
Since, $n - \sqrt{n} - \sqrt{n - \sqrt{n}} > n - 2\sqrt{n}$,
Therefore, $GMY(n - \sqrt{n} - \sqrt{n - \sqrt{n}} + 1, n - \sqrt{n} - \sqrt{n - \sqrt{n}}) > GMY(n - 2\sqrt{n} + 1, n - \sqrt{n} - \sqrt{n - \sqrt{n}})$

$GMY(n, f)$
$> n - 1 + (n - \sqrt{n}) + GMY(n - 2\sqrt{n} + 1, n - \sqrt{n} - \sqrt{n - \sqrt{n}})$
$> n - 1 + (n - \sqrt{n}) + GMY(n - 2\sqrt{n} + 1, min(n - 2\sqrt{n}, n - \sqrt{n} - \sqrt{n - \sqrt{n}}))$
$> n - 1 + (n - \sqrt{n}) + GMY(n - 2\sqrt{n} + 1, n - 2\sqrt{n})$

$> n - 1 + (n - \sqrt{n}) + (n - 2\sqrt{n}) + GMY(n - 3\sqrt{n} + 1, n - 3\sqrt{n})$
$> n - 1 + (n - \sqrt{n}) + (n - 2\sqrt{n}) + ... + (n - \sqrt{n}\sqrt{n})$
$= n(\sqrt{n} + 1) - 1 - \sqrt{n}\sqrt{n}(\sqrt{n} + 1)/2$
$= n(\sqrt{n} + 1)/2 - 1$

Asymptotically, lower bound $= n(\sqrt{n} + 1)/2 - 1 \in O(n\sqrt{n})$
Also, upper bound $= O(n + f\sqrt{n}) \in O(n\sqrt{n})$ since f = n-1
Since, the upper and lower bounds converge asymptotically, this case indeed achieves the worst-case message complexity. This is also confirmed by Figure 5 where we see that the empirical message complexity is very close to the theoretical upper bound.

5.3 Average Case Analysis

Our results in Section 4.1 and 4.2 show that for randomized failures, the message complexity of 2-Height Diffusion Tree variant is close to $O(n)$, which is much better than the theoretical upper bound of $O(n+f\sqrt{n})$. This is because, when we randomize faults, majority of them are going to be leaves, as the range of f_2 is almost $\sqrt{n}$ times the range of f_1. Failure of leaves does not lead to re-diffusion (as per Equation 5.1, low f_1 and high f_2 will be a good case for the recursion), hence the algorithm performs much better than $O(n+f\sqrt{n})$ messages.

6 Conclusion

We studied Galil-Mayer-Yung algorithm for crash fault tolerant broadcast with sub-quadratic message complexity. Based on our implementation and results, we conclude the following.

- In the average case, the empirical message complexity of the 2-Height Diffusion Tree variant of the Galil-Mayer-Yung algorithm is much lower than its theoretic upper bound i.e. $O(n + f\sqrt{n})$. In fact, it is very close to O(n). This is backed up by our theoretical analysis, which shows that randomized failures is a good case for the algorithm, in terms of message complexity.
- We also encounter a case where the empirical message complexity is very close to the theoretical upper bound and asymptotically converges to it. We verified our analysis for this case by testing our implementation with the required failure pattern.
- Within the extent of our experiments, the Galil-Mayer-Yung algorithm *always* out-performs Dolev-Strong algorithm in terms of message complexity.

References

[1] Dolev, Danny and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement." SIAM J. Comput. 12 (1983): 656-666.

[2] Z. Galil, A. Mayer and Moti Yung, "Resolving message complexity of Byzantine Agreement and beyond," Proceedings of IEEE 36th Annual Foundations of Computer Science, 1995, pp. 724-733.

[3] Roberto De Prisco, Alain Mayer, and Moti Yung. 1994. Time-optimal message-efficient work performance in the presence of faults. In Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing (PODC '94). Association for Computing Machinery, New York, NY, USA, 161–172.

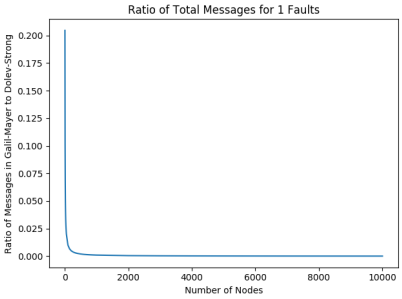
[4] T.D. Chandra and S. Toueg, Time and Message Efficient Reliable Broadcast. Proc. Int. Workshop on Distributed Algorithms 15'90, LNCS Springer-Verlag, 289- 303.

[5] Marcos Kawazoe Aguilera, Sam Toueg, A simple bivalency proof that t-resilient consensus requires t+1 rounds, Information Processing Letters, Volume 71, Issues 3–4,1999.

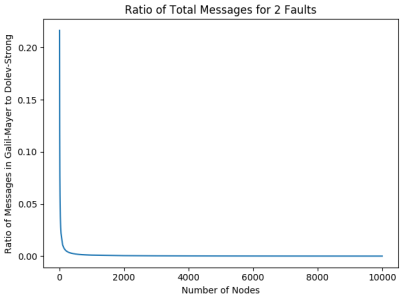
[6] Decentralized Thoughts, Dolev-Strong Authenticated Broadcast, Written by Ittai Abraham, Kartik Nayak

A Appendix

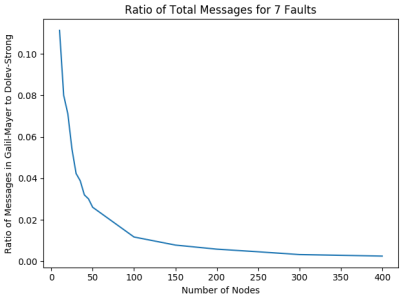
A.1 Plots for Comparison against Dolev-Strong



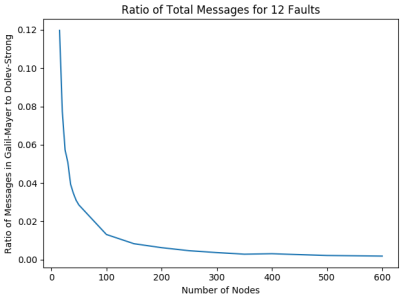
(i) $f = 1$



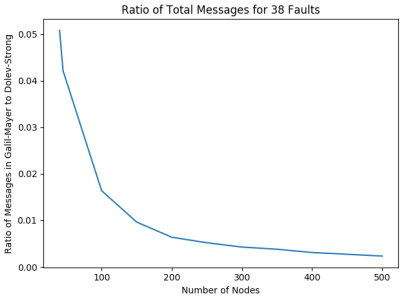
(ii) $f = 2$



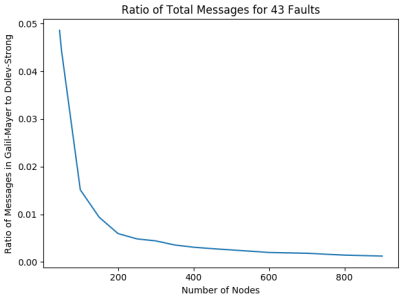
(iii) $f = 7$



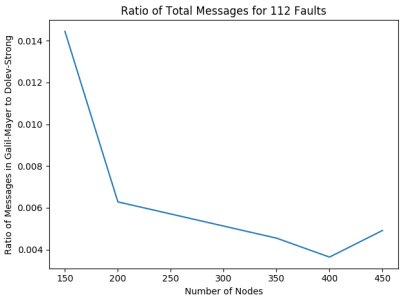
(iv) $f = 12$



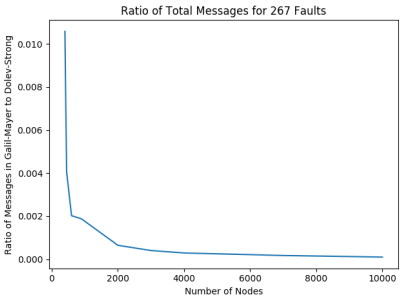
(v) $f = 38$



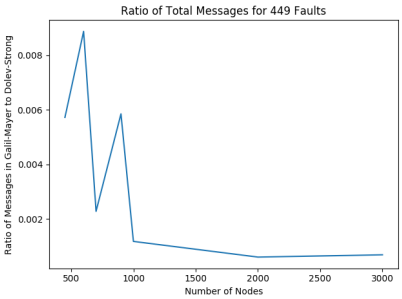
(vi) $f = 43$



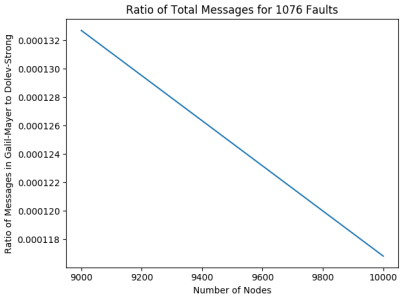
(vii) $f = 112$



(viii) $f = 267$



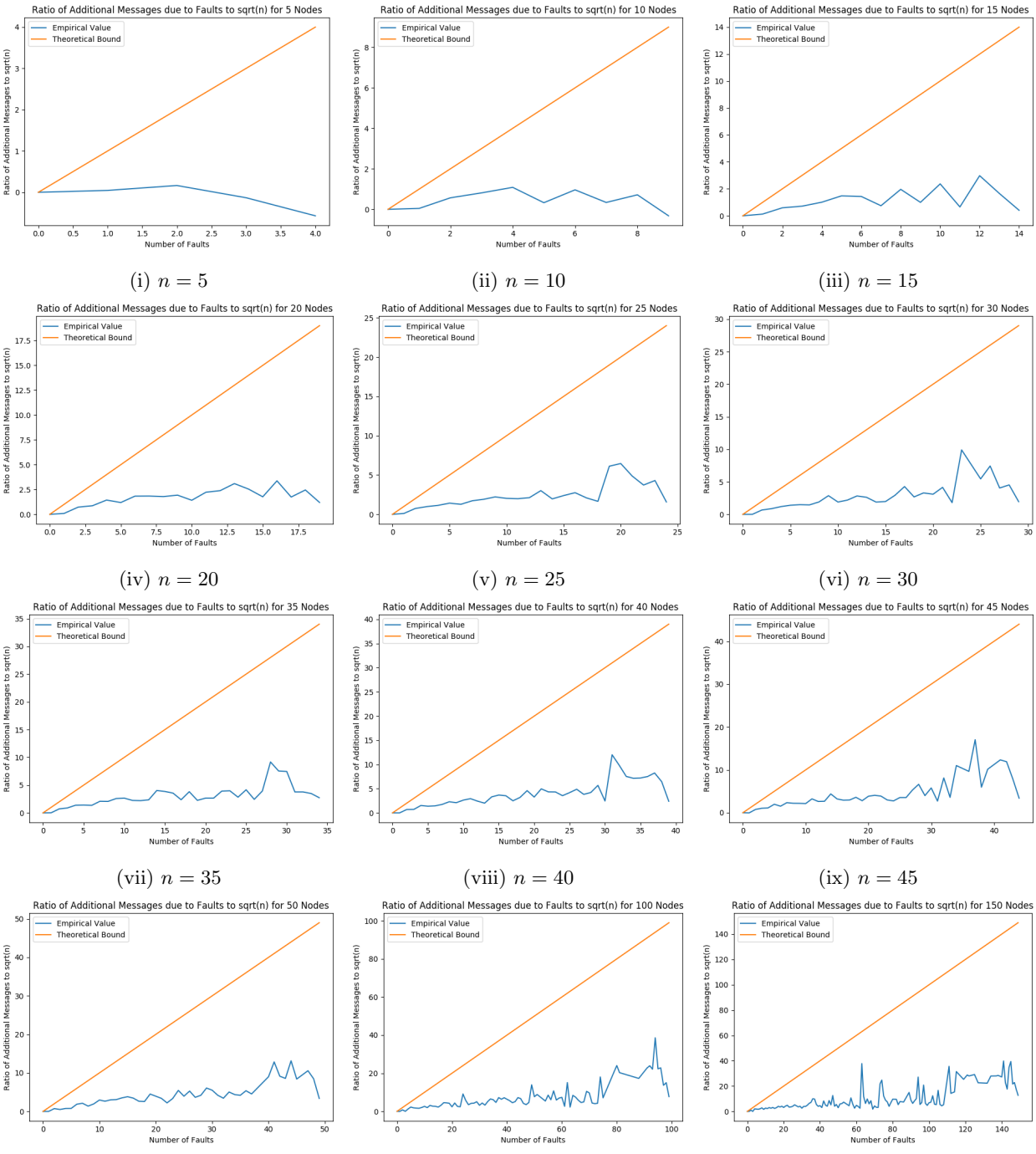
(ix) $f = 449$



(x) $f = 1076$

Figure 6: Ratio of Total Number of Messages in Galil-Mayer-Yung to Dolev-Strong

A.2 Plots for Comparison against Theoretical Upper Bound



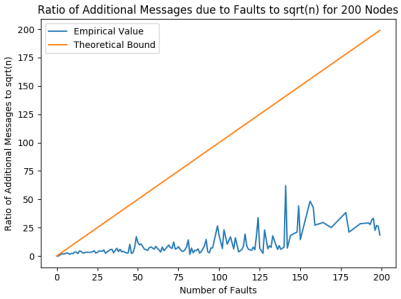
(i) $n = 5$ (ii) $n = 10$ (iii) $n = 15$

(iv) $n = 20$ (v) $n = 25$ (vi) $n = 30$

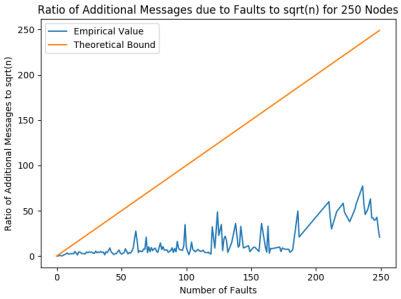
(vii) $n = 35$ (viii) $n = 40$ (ix) $n = 45$

(x) $n = 50$ (xi) $n = 100$ (xii) $n = 150$

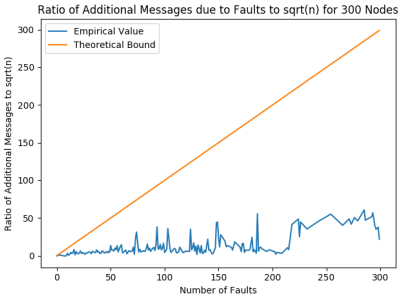
Figure 7: Ratio of Additional Messages due to faults to $\sqrt{n}$



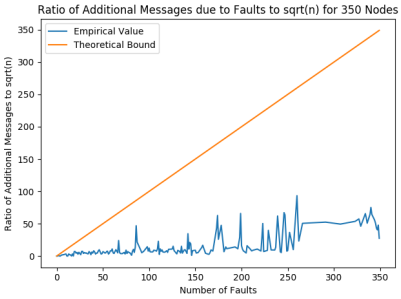
(xiii) $n = 200$



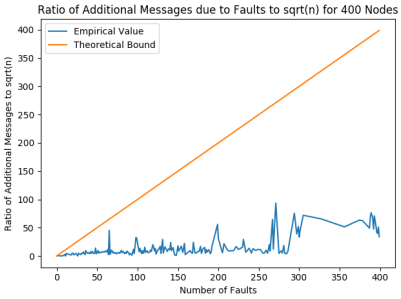
(xiv) $n = 250$



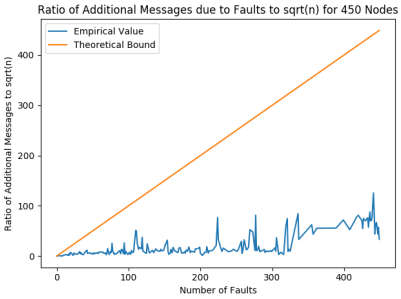
(xv) $n = 300$



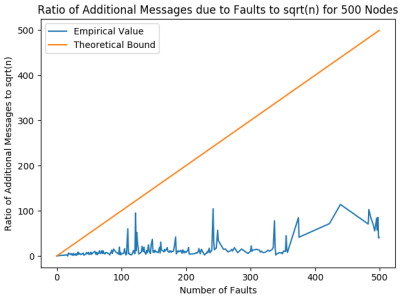
(xvi) $n = 350$



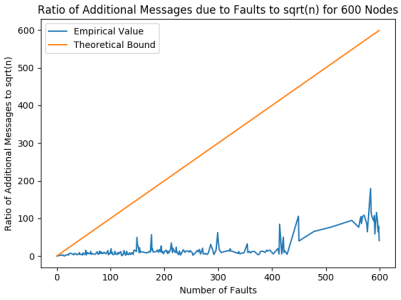
(xvii) $n = 400$



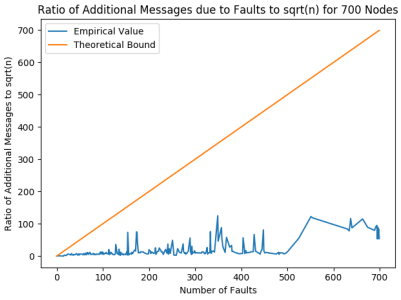
(xviii) $n = 450$



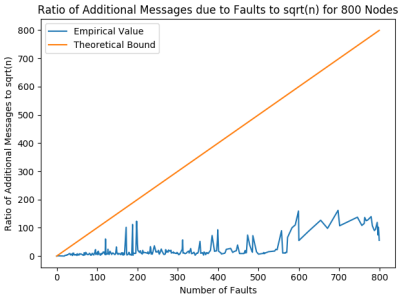
(xix) $n = 500$



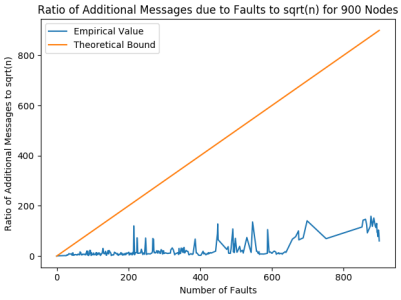
(xx) $n = 600$



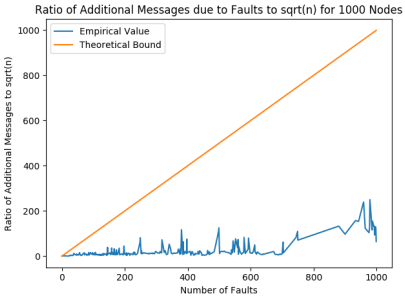
(xxi) $n = 700$



(xxii) $n = 800$

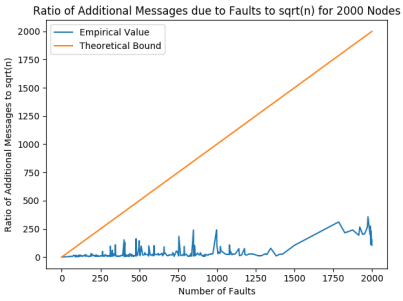


(xxiii) $n = 900$

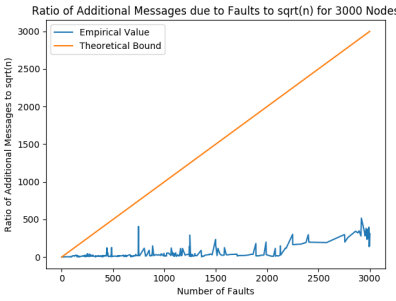


(xxiv) $n = 1000$

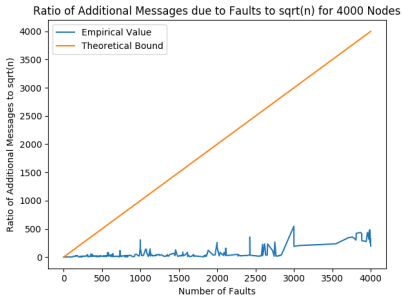
Figure 7: Ratio of Additional Messages due to faults to $\sqrt{n}$ (cont...)



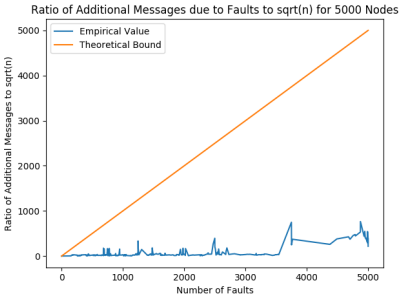
(xxv) $n = 2000$



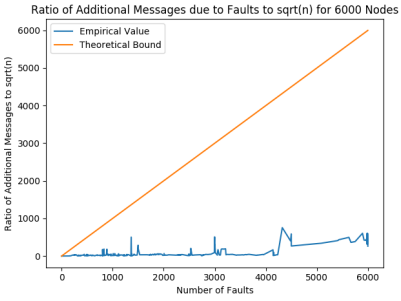
(xxvi) $n = 3000$



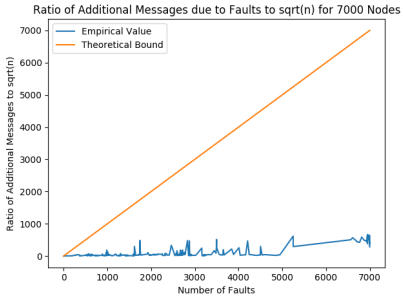
(xxvii) $n = 4000$



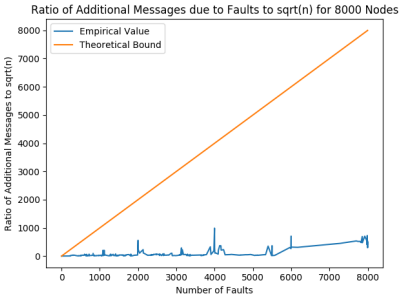
(xxviii) $n = 5000$



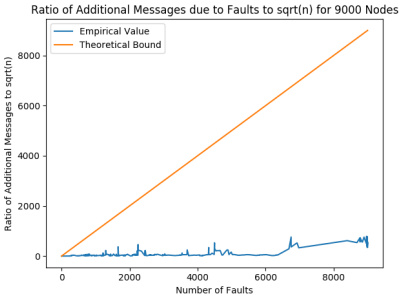
(xxix) $n = 6000$



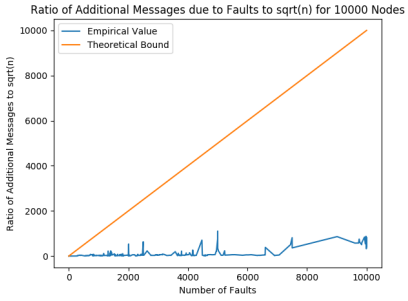
(xxx) $n = 7000$



(xxxi) $n = 8000$



(xxxii) $n = 9000$



(xxxiii) $n = 10000$

Figure 7: Ratio of Additional Messages due to faults to $\sqrt{n}$ (cont...)