Applying Optimizations from Bracha-Dolev Broadcast Protocol to Bracha-CPA Broadcast Protocol

Qusay FantaziaE[1] , **Jérémie Decouchant**[1] ,
[1]TU Delft

Abstract

Broadcast protocols are a crucial building block for some Agreement protocols. These are protocols used to reach an agreement on common values, action or datum in a distributed system through sending it in a message for other processes to accept it [3]. Byzantine processes are processes that hinder the network from reaching the agreement by sending malicious (malicious processes) or faulty messages(faulty processes).

Many broadcast protocols for distributed systems have been presented, depending on the topology, synchronicity(the network is asynchronous or synchronous), etc. One of these protocols, we are going to refer to as Bracha-Dolev, has been presented in the paper Practical Byzantine Reliable Broadcast on Partially Connected Networks[1]. Bracha-Dolev protocol can be used to reach an agreement in at least 2f+1-connected, asynchronous network [1].
The second protocol we're going to come across in this paper is a protocol we're going to refer to as Bracha-CPA. Both Bracha-Dolev and Bracha-CPA are built by combining two protocols. Bracha-Dolev is built through combining the protocol presented by Gabriel Bracha [3] and the protocol presented by Danny Dolev et al [4]. We're going to refer to these protocols with Bracha's and Dolev's protocol. Bracha-CPA is built combining Bracha's broadcast protocol and the protocol presented by Chiu-Yuen [7] called Certified Propagation Algorithm.

There are two important metrics when talking about broadcast protocol the first one is the average message complexity which is the average number of messages until all processes accept the message. The second is the average delivery time, which is the average time until all the processes deliver the message. This paper demonstrates that by applying some of the optimizations applied to Bracha-Dolev [2], we can decrease the average message complexity of Bracha-CPA up to 20%. the average delivery time doesn't seem to decrease. The paper will also demonstrate that CPA has the highest probability of succeeding on a network with a k-diamond or k-pasted graph when we have a maximum number of Byzantine nodes.

1 Introduction

Agreement protocols are crucial to reach a consensus in the presence of f Byzantine processes in a distributed network, where f must satisfy the condition $f < n/3$ [3] and n is the number of processes in the distributed network. A lot of algorithms has been developed for this purpose [10; 8; 5; 6].

Bracha's Agreement protocol uses a broadcast protocol to reach an agreement in an asynchronous, fully-connected network[3].
Another protocol we want to address is Dolev's reliable broadcast protocol, the protocol works on at least a 2f+1-connected (any two processes have at least 2f+1 disjoint paths) usually synchronous network. One drawback of this protocol is that it is very computationally expensive because of checking for the f+1 disjoint paths. In section 2, we explain how Bracha's and Dole'v broadcast protocol work.

By Combining Bracha's and Dolev's protocols, we get state-of-the-art Bracha-Dolev broadcast protocol. Bracha-Dolev broadcast protocol that works on at least 2f+1-connected(f the number of Byzantine processes), asynchronous or synchronous network[1]. Bracha's broadcast protocol requires a fully connected network. That's why we use Dolev's protocol to provide the abstraction of a reliable point-to-point link[1]. Bracha-Dolev broadcast protocol is explained further in section 3(problem description).
A lot of optimizations have been applied to the Bracha-Doelv broadcast protocol to improve its latency and bandwidth consumption, etc[1]. A drawback of this protocol is that it is complex due to combining Bracha's and Dolev's protocols. A message is accepted if it was accepted by both protocols. Also, this protocol has to send a lot of messages before all processes accept the message and it is computationally expensive because Dolev's part of the protocol checks that it received the message from f+1 disjoint paths before it delivers it. There hasn't been any work to discuss the possibility of replacing Dolev's protocol with another protocol and if the resulting protocol will have better characteristics

We have two types of fault models: local and global. Global fault model means there are at most f Byzantine processes in the network, while local means there are at most f Byzantine processes amongst the neighbour of any

process. In all of the protocols mentioned above the number of byzantine processes is globally bounded. The certified propagation Algorithm protocol works on a network that is f-locally bounded [11]. One caveat is that the CPA protocol can't be applied to all networks, therefore a lot of criteria has been developed to check if a graph is f-locally bounded[11; 12; 9]. All of these papers lack any information about the percentage of graphs we can apply CPA on or if there are types of graphs CPA usually works on. in section 3, we discuss the criteria that have been mentioned in these papers and how CPA work.

With the complexity of Bracha-Dolev and its high average message complexity and high computational complexity due to checking for f+1 disjoint paths as mentioned above, we have the option of replacing Dolev with CPA to get Bracha-CPA. Replacing the Dolev broadcast protocol by CPA raises some interesting points to research:

1. From the optimizations applied to Bracha-Dolev by Silvia Bonomi et al [1], Can we apply any of them to Bracha-CPA?
2. What are the average message complexity and average delivery time of Bracha-CPA with some of the optimizations mentioned in [1] compared to plain Bracha-CPA and Bracha-Dolev?
3. what is the success probability of using CPA on some types of graphs? Are there any guidelines to know if we can apply CPA on a graph? [11; 12; 9]

In this paper, we will demonstrate that if we apply some of the optimizations applied to Bracha-Dolev to Bracha-CPA, we will have a protocol that needs up to 20% fewer messages than normal Bracha-CPA and that needs up to 60% fewer messages than Bracha-Dolev. We will also show that on some types of graphs we can use Bracha-CPA or CPA on them with high success probability for particular connectivity, number of nodes and number of Byzantine nodes. An example is when we have a maximum number of Byzantine nodes, the k-pasted graphs has the highest success probability, followed by k-diamond graphs, followed by k-regular graphs and finally generalized-wheel graphs and multi-partite-wheel graphs.

Section 2 presents the problem description, where we explain how Bracha's, Dolev's and Bracha-Dolev protocol work.

In section 3, we state the contribution of this paper, we first state the optimizations applied to Bracha-Dolev from the paper [1] and choose the ones that we can apply to Bracha-CPA. Afterwards, we summarize the criteria that have been developed to check if we can use CPA on a graph[11; 9; 12] and how CPA and Bracha-CPA work.

In section 4, We discuss our implementation of the algorithm F, L, R partitioning. We use it to draw some interesting conclusions about when we can use CPA on a graph and the success probability of using CPA on some types of graphs, like k-regular and k-diamond graphs. we also have a section about responsible research where we discuss the possible implication of this paper. Finally, we finish with some conclusions.

There are two important metrics when talking about broadcast protocol the first one is the average message complexity which is the average number of messages until all processes accept the message. The second is the average delivery time, which is the average time until all the processes deliver the message. This paper demonstrates that by applying some of the optimizations applied to Bracha-Dolev [2], we can decrease the average message complexity of Bracha-CPA up to 20%. the average delivery time doesn't seem to decrease. The paper will also demonstrate that CPA has the highest probability of succeeding on a network with a k-diamond or k-pasted graph when we have a maximum number of Byzantine processes.

2 Replacing Dolev with CPA in Bracha-Dolev and CPA's Limitations

Bracha-Dolev broadcast protocol

Bracha's broadcast protocol uses three types of messages: (initial, message),(echo, message) and (ready, message). The broadcaster sends an (initial message) to all processes in the network, a process sends its echo when it is certain that a broadcaster sent the message (receives an initial message from the broadcaster, receives (n+f)/2 echo messages, or f + 1 ready messages), ready is sent when a process believes that a message is the only message sent by the broadcaster(receives (n+f)/2 echo messages, or f+1 ready message and has sent its echo message), finally when receiving 2f+1 ready messages the process delivers(accept) the message.
In the Appendix Figure 2, there is a picture that depicts how Bracha's protocol works. the broadcaster process with id 0 sends the initial message, all the process receives the initial message and send their echo. Next, a process that receives n+f/2 echo can send its ready and wait for 2f+1 ready to accept (deliver) the message.

Dolev's broadcast Protocol works on at least a 2f+1 connected synchronous network. Here are very simple steps on how it works without any optimizations:

1. the source sends the message to all its neighbour processes
2. A process relays the message to its neighbour processes not included in the path appending the id of the sender process to the path.
3. A message is delivered if there exits f+1 disjoints path related to it.[2]

In the appendix figure 3, there is a picture that depicts how Dolev's broadcast protocol works. The broadcaster process 0 send a message with an empty path. Process 2 and 3 receive the message from the source so they relay it to their neighbours appending 0 to the path. process 4 will deliver the message because it received it through two paths [02] and [03] and relay the message again to process 2 and 3. process 2 and 3 will deliver the message next because they both received it from through source and process 4.

Now it is time to discuss Bracha-Dolev and its disadvantages further. we'll keep the talk about the optimizations applied to Bracha Dolev to section 3.
Bracha-Dolev's broadcasting protocol uses Dolev's protocol to achieve the abstraction of point-to-point reliable

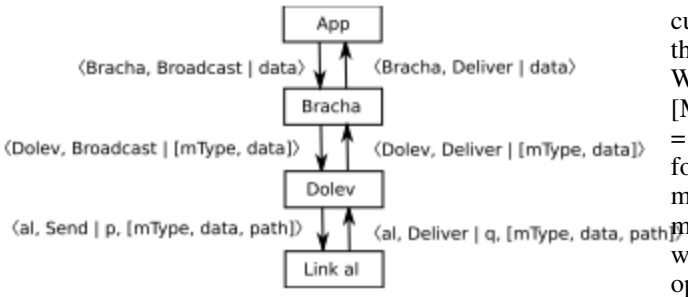


Figure 1: the protocol stack of Bracha-Dolev

communication[1].
The Protocol's stack can be seen in figure 1. The protocol uses the same messages used by Bracha's protocol: (initial, message), (echo, message), (ready, message). The protocol works as follows: when a message reaches a process, the Dolev part of the protocol relays it to its neighbours. The Dolev Part of the protocol forwards to the Bracha part if it received through f+1 disjoint paths. Bracha part of the protocol works as mentioned in the section above. When the Bracha part of the protocol receives an initial message, n+f/2 echo or f+1 ready message and it has not sent an echo message the process sends its echo message, the ready message is sent if a process sent its echo and it receives n+f/2 echo or f+1 ready messages when a process receives 2f+1 ready messages it delivers(accepts) the message. Bracha-Dolev protocol delivers the message when Bracha part of the protocol does[1].

Bracha-CPA and its limitation

We realize that the protocol has a high message complexity because the Dolev part of the protocol relays a message to other processes and the Bracha part generate its messages. Also, the protocol has high computational complexity because of checking for the disjoint paths. We believe that replacing Dolev with CPA will decrease the number of messages and the computation complexity associated with checking for f+1-disjoint paths. One drawback is that we can not apply CPA and thus Bracha-CPA on all graphs. In the next section, we present some of the algorithms to check if we can apply CPA on a graph and discuss their message complexity. We are also going to discuss our implementation of the algorithm developed in the paper [12] and talk about the experiments we run.

3 Bracha-CPA with optimizations from Bracha-Dolev and checking if we can apply CPA on graph

3.1 Selecting optimizations from Bracha-Dolev to apply to Bracha-CPA

The optimizations applied to Bracha-Dolev protocol in the paper Practical Byzantine Reliable Broadcast On Partially Connected Networks can be classified into 4 main categories, each of them contains sub-optimizations [1]. Through this chapter we are going through the optimizations applied to Bracha-Dolev, if any of them is applicable, we are gonna discuss it, and if not, we will briefly say why not and proceed to the next one.
We use the same message template used in [1] which is [MessageType, Pi, (S, Pid), M, Path] where MessageType = SEND, ECHO, READY; Pi the id of the process that forwarded the message; S is the broadcasterid; Pid is the message-id(this is used in case the process send the message multiple times); M is the message; the path is the path through which the message went to reach the process. Next to the optimization name, we will add the optimization abbreviation used in the paper [MBD.Number], for example MBD.1 is used for the first optimization in the paper.
Bracha-CPA has the following Message template [Messagetype, LinkSenderId, EchoOrReadySenderId, BroadcasterId] where MessageType=Echo or Ready; LinkSenderId is the id of the process who sent the message; EchoOrReadySenderId is the id of the process that generated the message, EchoOrReadySenderId is different from LinkSenderId when the message is relayed; BroadcasterId is the id of the process who started he broadcast.

1. Limiting payload transmission [MBD.1]: When a process Pi receives the message with an unknown payload for the first time it chooses a local id to associate with the payload and sends it along with the payload when relaying the message. Later, when the same process pi wants to send the same message, it can send the local id instead of the whole payload. This optimization can be applied by making each process generate a unique id for each message he receives. When a process receives a local id for a payload he can generate his own or use the received one as his local id without any problems. Processes should keep track of to whom they've sent their local id.
   This optimization has not been applied because it is hard to implement and takes a long time to debug[1].
2. Bracha phase transitions
   (a) Single-hop Send messages [MBD.2]: when a process Dolev-delivers a send message it relays it to its neighbours, the process also sends an echo message because of the Bracha part of the protocol. This optimization will make the process only send an echo message, which other processes can use to derive the send message using this rule : [Echo, pi, (s, bid), m, path] $-$ $>$ [SEND, (s, bid), m, path]. We applied this optimization to Bracha-CPA as follows: In our implementation, we don't have a send message sent from the broadcaster instead we have an echo, but it works the same because when the processes receive the echo from the broadcaster it sends its echo. Every process that receives an echo message from a process pi [echo, pj, pi, broadcaster] also handles the echo from the broadcaster implicitly included in the message [echo, pj, broadcaster, broadcaster]. This optimization has been applied to Bracha-CPA.
   (b) Echo to echo transitions [MBD.3]: When a process

Dolev-delivers an echo message, it forwards the echo message to all of its neighbours. As a result of Dolev-delivering, the Bracha-part of the protocol might also want to send an echo message(after receiving a send a message or (N + f)/2 + 1 echo message). To reduce the number of messages the process sends, we send an echo-echo message instead of sending the two echo messages. We applied this optimization to Bracha-CPA as follows: when a process pi CPA-validates echo message from process pj (received the echo message from the Echo-Sender pj or from f + 1 processes). The Bracha-part of the protocol might want to send its echo message that (received validated echo from the broadcaster or has received f + 1 validated echo messages). The process will then send [echo_echo, pi, pj, BroadcasterId], the process that receives the message from pi can obtain the two echo message:[echo, pi, pi, broadcasterId] and [echo, pi, pj, broadcasterId].

(c) Echo to Ready transitions[MBD.4]: when a process Dolev-delivers an echo message, it relays it to all of its neighbours. As a result of the Dolev-delivering of the echo message, the Bracha-part of the protocol might also want to send a ready message (after having Dolev-delivered 2f + 1 Echo messages). To reduce the number of messages the process send, we send an echo-ready message instead of sending the the echo and the ready message separately. We applied this optimization to Bracha-CPA as follows: when a process pi CPA-validates echo message from process pj (received it from Echo-Sender or f + 1 processes), the process might want to send its Ready message(has (N+f)/2 + 1 validated echos or f+1 validated ready message). The process will then send [echo_ready, pi, pj, BroadcasterId], the process that receives the message from pi can obtain the two echoes message:[Ready, pi, pi, broadcasterId] and [echo, pi, pj, broadcasterId].

3. Optimized messages[MBD.5]:

(a) Send messages: in Bracha-Dolev, Send messages are single-hop messages, which means they are only sent by the source. With This optimization, the message doesn't have to contain the source id because the links are reliable. the message will have the following template [MessageType, (S, Pid), M, Path] without Pi the sender id.
This optimization can be applied to Bracha-CPA by not sending the linkSender id and even the echo-sender id in the echo message sent by the source to its neighbour. This optimization has not been applied due to a lack of time and limited value.

(b) Optional fields

i. PayloadBit: payload bit is a bit in the message to tell the process that receives the message if the message contains the payload or not(it is related to the optimization Limiting payload Transition). This optimization can be applied to Bracha-CPA. This optimization hasn't been applied to Bracha-CPA because it takes a long time to debug the code.

ii. SenderID: As we've said Send messages are single-hop messages, we don't include the sender in the message. The Sender bit is included in the message to tell the process that receives the message if the source is included in the message. This optimization can not be applied to Bracha-CPA. This optimization hasn't been applied to Bracha-CPA due to lack of time and it is limited value.

4. Handling asynchrony:

(a) Ignore Echos received after Dolev-delivering the corresponding Ready [MBD.6]: Ignore Echos received after the Dolev-delivering the corresponding Ready of a process because the Echo reflects an old state of the process.
This can be applied to Bracha-CPA because a process might receive a ready message from a process and after that, an echo relayed through a longer path from the same process. This optimization has been applied to Bracha-CPA.

(b) Ignore Echos received after delivering the content [MBD.7]: Ignore Echos received after delivering the content because to Bracha-deliver a message a process needs 2f+1 ready. These processes that send their ready will exchange the ready and all other processes will deliver the message.
This also can be applied to Bracha-CPA because for a process to deliver a message it has also to wait for the 2f+1 ready from the 2f+1 processes from which only f processes might be byzantine, while the others will relay and send their ready messages until every process delivers the message. This optimization has been applied to Bracha-CPA.

(c) Receiving Readys before transmitting Echos[MBD.8]: If a node pi has Dolev-delivered the Ready message of its neighbour pj, it can avoid sending any future Echo message it receives to pj.
This optimization can be applied to Bracha-CPA when CPA in a process pi validates a ready message from a process pj, then there is no need to send any echo messages to pj. This optimization has not been applied to Bracha-CPA because it takes a long time to debug.

(d) Avoiding neighbours that delivered[MBD.9]: If a process pi has received 2f+1 Readys (generated by 2f+1 different processes) with empty paths that are related to the same content from a neighbour pj, then pi can avoid sending any message related to that content to pj in the future. The reason why we do not have to forward any messages is that process pj will deliver the message (received 2f+1 echo messages) so sending a message to the process is also not necessary.
This can be applied to Bracha-CPA, the same way it is applied to Bracha-CPA when a process received 2f+1 ready from one of its neighbours then it can

avoid sending any message to that process. This optimization has not been applied because it takes a long time to debug.

(e) Ignore messages whose path is a superpath of a message [MBD.10]: This does not apply to Bracha-CPA because we do not use paths in Brach-CPA, the message template for Bracha-CPA is [Message-type, LinkSenderId, EchoOrReadySenderId, BroadcasterId] as mentioned earlier.

5. Non-tight cases. These optimizations are used when we have more 3f+1 processes in the network. this optimization has two types:
   (a) Reduced number of messages in Bracha [MBD.11]
   (b) Send messages in Bracha-Dolev. [MBD.12]

   both optimizations can be applied to Bracha-CPA. Both optimizations haven't been applied because they take a long time to debug.

Here is a table of the optimizations applied to Bracha-Dolev in the paper [1] and if they have been implemented in Bracha-CPA(Yes or No). In case the optimization is very hard or can not be implemented, there will be Doesn't apply next to it. The table can also be found in the appendix.

Optimization	Implemented
Limiting payload transmissions Bracha	No
Single-hop Send messages	Yes
Echo to Echo transitions	Yes
Echo to Ready transitions	Yes
Optional field(payload bit)	No
Optional field(Sender bit)	No
Ignore Echos received after Dolev-delivering the corresponding Ready	Yes
Ignore Echos received after delivering the content	Yes
Receiving Readys before transmitting Echos	No
Avoiding neighbors that delivered	No
Ignore messages whose path is a superpath of a message	Does not apply
Reduced number of messages in Bracha	No
Send messages in Bracha-Dolev	No

3.2 Checking if we can apply CPA on a graph

A network is f-locally bounded means that we can apply the CPA algorithm on it with each process having at most f Byzantine processes. The algorithm runs in this way. The broadcaster (source) sends the message to all of its neighbours and stop. A process accepts the message and relays it to its neighbours if it received it from f+1 processes or the broadcaster.

Next, we want to discuss some of the papers that developed some criteria to check if we can apply CPA to a graph.
The first paper I want to address that tries to answer if a graph is f-locally bounded, is the paper by Andrzej Pelc [11]. The paper introduces two parameters X(G) and LPC(G).
To compute X(G), we go over all the processes looking for the process with the least number of neighbours that are closer to s (the source) than to itself,

$$X(G) = min\{X(v,s)|v,s \in V, (v,s) \notin E\} \quad (1)$$

where X(v,s) denote the number of neighbours of a node that are closer to s than v including v. if $f < X(G)/2$ then the graph is f-locally bounded. This condition is sufficient for a Graph to be f-locally bounded, but not necessary because there are f-locally bounded graphs that do not satisfy the condition $f < X(G)/2$.
To compute LPC(G), we look into a cut in the graph, a set of nodes whose removal will disconnect the graph into two parts, such that the two parts induced by this cut are f-local pair cut, LPC(G) is the smallest f such that we have f-local pair cut. If $f \geq LPC(G)$ then the graph of the network is not f-locally bounded. For f values between LPC(G) and X(G), the paper doesn't present any criteria to check if the graph network is f-locally bounded or not[11].
The pseudo-code to compute X(G) can be found in Figure2. looking at the pseudo code we can estimate the time complexity of computing X(G). The time to compute the shortest path between two nodes is O($Nodes^2$) (line 5 and 6). We also have two for loops on lines 3 and 4. the first one goes over all nodes which means it is O($Nodes$). The second goes over the neighbours of a node which takes constant time if we have an adjacency list presentation of the graph. To sum this up, the complexity is O($Nodes^3$).

Another paper I want to mention is the paper by Chris Litsas [9], the paper introduces three parameters K(G,D), M(G,D,T) and T(G,D). K(G,D) is the max f such that the network graph has a minimum f-level ordering. The following induction gives us a definition of minimum f-level ordering

$$L_1 = N(s) \quad (2)$$

$$\{L_i = v \in V \setminus \cup_{j<=i-1} L_j : |N(v) \cap \cup_{j<=i-1}| >= f\} \quad (3)$$

, where N is a function that returns the neighbours of a process. Any network graph with $f < K(G,D)/2$ is f-locally bounded (sufficient, not necessary condition) and for any graph, the fmax has to satisfy

$$K(G,D)/2 - 1 <= fmax < K(G,D)$$

The time complexity to compute K(G,D) is O($E \times \delta$) according to [9].
Another parameter that is presented in this paper is M(G,D,T).

$$\min_{T:f-localset} K(G_T, D) \quad (4)$$

In the formula above T is f-local set and $G_t = G \setminus T$(the graph induced by removing T from the set of nodes). For a graph with M(G, D, T) $\geq$ f + 1 the graph is f-locally bounded (necessary and sufficient condition).
Last but not least, if we want to compute Max f for CPA we can use the formula

$$T(G, D) = max_{f \in N} M(G, D, t) \geq f + 1 \qquad (5)$$

, where N is all positive integer number.
the time complexity to compute M(G, D, t) is exponential because it requires coming up with all possible f-local fault set(a set where each process had at most f Byzantine processes) [9]. A pseudo-code to compute M(G, D, t) can be found in Algorithm2.

The last paper I want to address is the paper by Lewis Tseng [12]. In the paper, it is stated that if a graph G(V, E) can be split into 3 parts F, L and R such that the source is in L, R is not empty and F is f-local fault set. if L and R satisfy that $L => R$ (R has a node that has at least f+1 incoming neighbours from L) or R has a node that is neighbour of the source, then the graph is f-locally bounded. For the set F to be f-local fault set, it has to satisfy the following condition, for every node in V that is not F, F has hast at most f distinct incoming neighbors[12].
The algorithm has exponential running time because we have to consider all possible set of F, L and R. A pseudo-code is in (Figure 4).

In the appendix(C), there is a table that summarizes all the parameters we mentioned from the papers [11; 12; 9], if the condition related to them are sufficient or necessary and their time complexity.

Algorithm 1 The pseudo-code to compute X(G)

nodes $\leftarrow$ *nodes in the graph*
$minimum \leftarrow MAX_INTEGER$
for $node1 \leftarrow nodes$ **do**
  for $node2 \leftarrow neighbor(node1)$ **do**
    distanceToS $\leftarrow Distance(node2, source)$
    numCloserToS $\leftarrow 0$
    distance12 $\leftarrow Distance(node1, node2)$
    if $distanceToSource < distancebetweentwo$ **then**
      $numCloserToS \leftarrow umCloserToS + 1$
    end if
  end for
  if $numNodesCloserToSource < minimum$ **then**
    $minimum \leftarrow numNodesCloserToSource$
  end if
end for
return $minimum$

Algorithm 2 The pseudo-code to compute M(G,D,T)

Input t value , G (the graph)

nodes $\leftarrow$ *nodes in the G*
$MConstat \leftarrow MAXINTEGER$
for $localFaultSet \leftarrow combinations(nodes)$ **do**
  $isFlocal \leftarrow checkisflocal(nodeCombination)$
  if $isTlocal$ **then**
    $G = G \setminus localFaultSet$
    $KParamter \leftarrow computeK(G, t)$
    if $KParamter \leq MConstat$ **then**
      $MConstat \leftarrow KParamter$
    end if
  end if
end for
return $MConstat$

Algorithm 3 The pseudo-code to compute if graph is f-locally bounded using the F,L,R set from Lewis Tseng paper

Input f value , G (the graph)

```
NSet ← nodes in G
for source ← nodes do
  NWithoutSource ← NSet \ source
  for LWithoutSource ⊂ NWithoutSource do
    LSet = nodesinLWithouSource ∩ source
    for RFSet = nodes \ LSet do
      for FSet ⊂ RFNodes do
        if checkFLOCALFAULTSET(FNodes)
        then
          RSet ← FRSET \ FSet
          RSetIsNeighbourofSource = False
          RSethasf + 1NeighInL = False
          for RNode ∈ RSet do
            if source ⊆ Neighbour(RNode) then
              RSetIsNeighbourofSource = True
            end if
            if size(Neighbour(RNode)) ∩ LSet ≥
            f + 1 then
              RSethasf + 1NeighInL = True
            end if
            if not(RSetIsNeighbourofSource) &
            not(RSethasf + 1NeighInL) then
              return false
            end if
          end for
        end if
      end for
    end for
  end for
end for
return true
```

3.3 Experimental work

After deciding which optimizations from Silvia Bonomi [1] we want to apply to Bracha-CPA(section 3.1). In section 4, we compare Bracha-CPA, with optimizations, with both Bracha-CPA(without optimizations) and Bracha-Dolev. All these algorithms have been explained in the previous sections.

We also use our implementation of the partitioning into F, L, R algorithm to give some statistics about the percentage of graphs on which we can use CPA[12]. We will use a different type of graphs: k-regular graphs, Generalized-wheel graphs, multi-partite-wheel graphs, k-diamond graphs and k-pasted graphs.

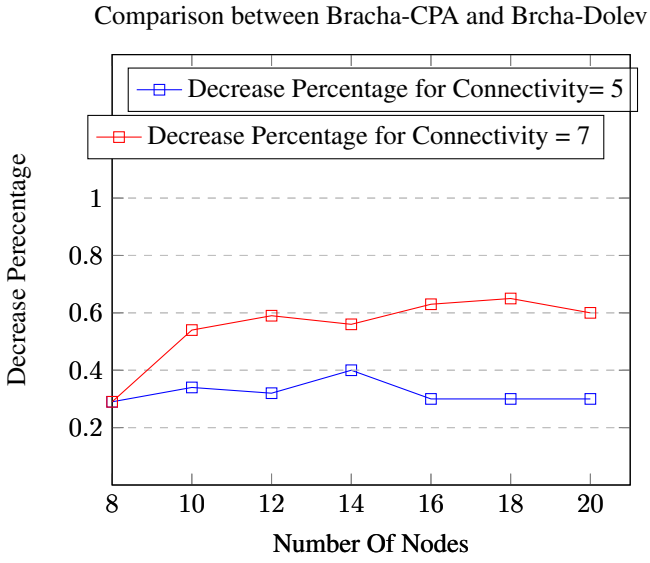
3.4 Improvement of an idea

The reason we decided to apply the optimizations from Bracha-Dolev to Bracha-CPA is that they are similar and in both protocols, some protocol provides a reliable point-to-point link abstraction for Bracha's protocol to work. In addition, CPA's message size is smaller because it doesn't contain the path and CPA has less computational complexity because it doesn't check for f+1 disjoint paths.

4 Experimental Setup and Results

4.1 Comparing Bracha-CPA and Bracha-Doelv

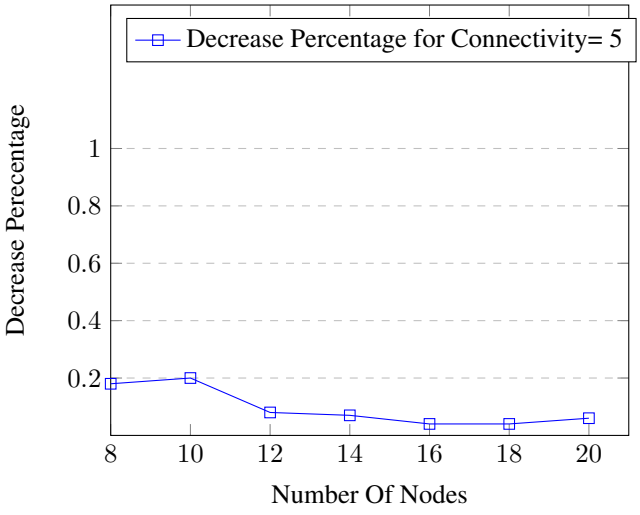
To compare Bracha-CPA with Bracha-Dolev we made a set of networks with k-regular graphs varying the number of nodes between 8, 10, 12, 14, 16, 18 and 20 and the connectivity between 5 and 7. We used our implementation of F, L, R partitioning Lewis Tseng [12] to check that we can apply CPA on the graphs. In the figure beneath, we can observe that the decrease percentage for average message complexity with the connectivity



4.2 Comparing Bracha-CPA with optimizations with Bracha-CPA without optimizations

We use the same type of regular networks with the same number of processes and connectivities we used to compare Bracha-CPA with Bracha-Dolev.In the figure beneath, We realize a decrease of around 20 % for connectivity of around half of the number of nodes.

Figure 5: Comparison between Bracha-CPA with and without optimizations



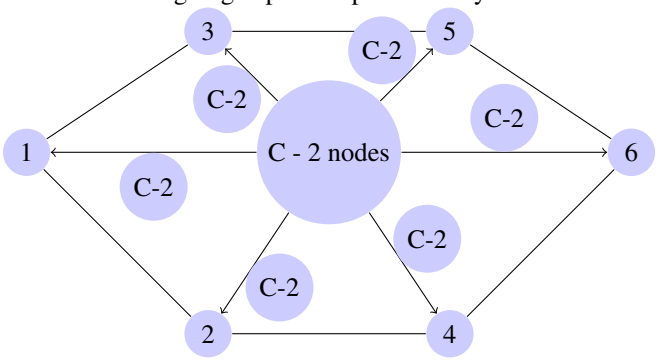
4.3 Success Percentage of CPA on different Graph types

To test this, we made our implementation of the algorithm F, L, R partitioning mentioned in the paper [12] by Lewis Tseng.

We tested on k-regular Graphs, Generalized-wheel Graph, Multi-Partite-Wheel Graphs, k-diamond Graphs and k-pasted Graphs. We varied the size of the networks(N), the connectivity(C) and the number of Byzantine processes. For some types of graphs, there are some slight changes in the connectivity and the number of nodes because the number of nodes and the connectivity has to satisfy some conditions. Through these experiments, we used a sample size of 10 for each number of nodes and connectivity. Unfortunately, we were not able to test for a large number of processes (18 and above) because for large number of processes and a sample size of 10 the code takes hours to finish(more than 14 hours). One more important thing to note is that the maximum number of Byzantine processes f = C-1/2 because C has to satisfy C $\geq$ 2f + 1.

For Regular graphs, we varied the number of processes (N) between 8, 10, 12, 14, 16 and the connectivity (C) between 5 and 7. For the number of Byzantine processes, we choose (c-1)/2 the maximum number of Byzantine processes and c/2 - 1(less than the maximum number of Byzantine by 1 for odd connectivity).
Regular Graphs have a very low success probability when the number of Byzantine nodes is the maximum (c-1)/2 less than 1%. when the number of Byzantine nodes is less than or equal to c/2 - 1 the success probability is above 80%, the success probability decreases when the number of nodes increases and the Connectivity stays the same .

For Generalized-wheel graphs, we used the same number of processes, the same connectivity and the same number of Byzantine processes.
we realize that we have 100% success probability when the number of Byzantine processes is less than or equal to c/2 - 1. When the number of Byzantine processes is equal to the maximum (c-1)/2 the success probability is 0%. Here we're going to provide proof of why is that the case.



The generalized-wheel graph with N number of nodes and C connectivity looks something like the graph above. where the centre has C-2 processes and the rest of the processes surround the centre. Each process outside the centre is connected to the centre with C-2 edges. if the number of Byzantine processes B is (C-1)/2 and the Byzantine process are positioned at the centre, it means there are C-2 - (C-1)/2 = (C-3)/2correct processes in the centre. a process that is adjacent to the source and in the centre will deliver because they will receive the message from the source while the processes that are not can at most receive the message from the correct process in the centre plus one (C-3)/2 + 1 (C-1)/2 ¡ f + 1 = (C+1)/2 so they will not deliver the message.

Multi-partite-wheel graphs have to satisfy the condition C is not odd and $C \leq N/2$. We also did not want to have the number of Byzantine nodes equal to 0, which impacted the number of nodes N and the connectivity C we choose. For multi-partite-wheel graphs we choose the number of processes equals 12, 14, 16 and used connectivity of 6. The number of Byzantine nodes is equal is 2(maximum) or 1 . we have a 100% success probability when the number of Byzantine nodes is 1 and 33% success probability when the number of Byzantine nodes is 2.

K-diamond graph has to satisfy the condition the $C \leq N/2$. We also did not want to have the number of Byzantine nodes equal to 0, which impacted the number of nodes N and the connectivity C we choose. That is why we use number nodes equal to 12, 14 or 16 and the connectivity equals N/2 or N/2 - 1. For the number of Byzantine nodes, we used (C-1)/2 and C/2 - 1 for odd connectivity and C/2 - 1 and c/2 - 2 for even connectivity.
No matter the number of Byzantine node, the success probability was 100%.

The data we used to draw these conclusions call all be found in the appendix(D to F).

Regarding k-pasted graphs, we choose the same number of nodes, connectivity and number of Byzantine processes as we did with k-diamond graphs. The k-pasted graphs have a high success probability of 100% for any number of Byzantine nodes less than or equal to the maximum.

5 Responsible Research

Through this section, we're going to explain how the results we obtained in section 4 Experimental Setup and Results can be reproduced.

The first result we presented in section 4 was the decrease percentage in the average message complexity between Bracha-CPA and Bracha-Dolev. To produce the same result, the user has to run the ini files using omnet++ in the folder /rp21-group31-2-fantazia/BRB-partially-connected-networks-main/BroadcastSign/simulation/randomGraphs/regularGraphs with Bracha-CPA with optimizations and Bracha-Dolev. To change the protocol the simulation uses, the user can change that from base.h in rp21-group31-2-fantazia/BRB-partially-connected-networks-main/BroadcastSign/src. The result will be logged to the stats folder in the folder rp21-group31-2-fantazia/BRB-partially-connected-networks-main/BroadcastSign. Now all the user has to do is subtract the number of messages

for Bracha-CPA with optimizations from the number of messages for Bracha-Dolev, divide that by the number for messages for Bracha-Dolev and finally plot that for each number of nodes and connectivity.
The second result presented in Section 4 was the decrease percentage in the average message complexity between Bracha-CPA with optimizations and Bracha-CPA without optimizations. we can flow the same steps we used for Bracha-CPA and Bracha-Dolev. Next, we subtract the number of messages for Bracha-CPA with optimizations from Bracha-CPA without optimizations, divide by the number of messages for Bracha-CPA without optimizations and plot for the different number of nodes and connectivity.
The last result we presented was regarding when we can apply CPA on a different type of graphs and the success probability. To obtain the same result, all we have to do is run main.py in the folder /rp21-group31-2-fantazia/GenerateGraphs. the results will be logged to the stats folder in the same directory. Now all one has to do is do some simple arithmetic to obtain the results.

Distributed systems are present in a lot of fields. I'd recommend testing the code further with networks for a bigger size and be cautious when using the code for real-world applications.

6 Conclusions and Future Work

The first point we wanted to research was,from the optimizations applied to Bracha-Dolev by Silvia Bonomi et al [1], Can we apply any of them to Bracha-CPA? the optimizations that we can apply are [MBD.2], [MBD.3], [MBD.4], [MBD.5] and [MBD.7].

The second point we wanted to research was, what are the average message complexity and average delivery time of Bracha-CPA with of the optimizations we applied from [1]compared to plain Bracha-CPA and Bracha-Dolev? Bracha-CPA with the optimizations has up to 20% smaller message complexity compared to Bracha-CPA and up to 60% smaller massage complexity compared to Bracha Dolev. The average delivery time doesn't seem to decrease after applying the optimizations.
The last point we wanted to research is the success probability of using CPA on a graph. we also wanted to research if there are any guidelines to know if we can apply CPA on a graph. we realized that for a maximum number of Byzantine nodes, this is the order of the types of graphs starting for the smallest: Generalized-wheel, k-regular, Multi-partite-wheel, k-diamond and k-pasted. For a number of Byzantine equals maximum -1, the k-regular has the smallest success probability while all the others types of graphs have 100% success probability. Note that these conclusions are based on a network size between 8 and 16. Regarding guidelines, we proved that a generalized wheel graph has a 0% success probability if we have a maximum number of Byzantine processes. We can also prove the success probability would be 100% if we had a number of Byzantine processes smaller than the maximum.

Future work can focus on looking into applying the optimizations from the paper[1] that have not been applied in this paper and applying optimizations and other optimization.
Future work should also focus on implementing the condition presented in [9] to check if we can CPA to a graph and see if it has better or worse running time than the implementation we made using the paper[12]. It is probably also a good idea to find approve why some graphs have some success probability for some connectivity and number of Byzantine of nodes.

References

[1] Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. Practical byzantine reliable broadcast on partially connected networks. *arXiv preprint arXiv:2104.03673*, 2021.

[2] Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Multi-hop byzantine reliable broadcast with honest dealer made practical. *Journal of the Brazilian Computer Society*, 25(1):1–23, 2019.

[3] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[4] Danny Dolev, Michael J Fischer, Rob Fowler, Nancy A Lynch, and H Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[5] Danny Dolev and H Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 401–407, 1982.

[6] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[7] Chiu-Yuen Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 275–282, 2004.

[8] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(2):254–280, 1984.

[9] Chris Litsas, Aris Pagourtzis, and Dimitris Sakavalas. A graph parameter that matches the resilience of the certified propagation algorithm. In *International Conference on Ad-Hoc Networks and Wireless*, pages 269–280. Springer, 2013.

[10] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[11] Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, 2005.

[12] Lewis Tseng, Nitin Vaidya, and Vartika Bhandari. Broadcast using certified propagation algorithm in pres-

ence of byzantine faults. *Information Processing Letters*, 115(4):512–514, 2015.

Appendices

A How Bracha's protocol work

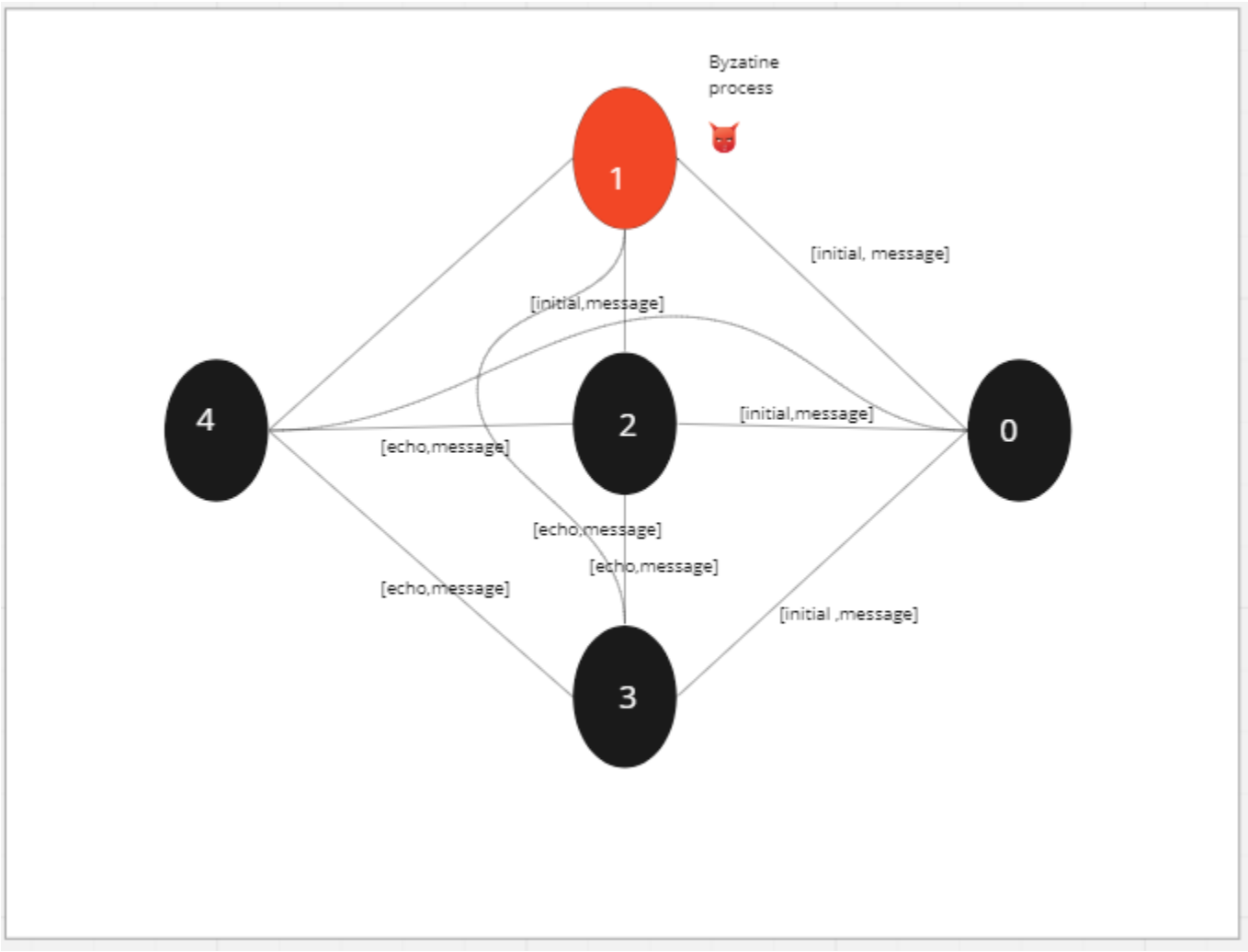


Figure 2: how Bracha's protocol works in a network with 5 processes and 1 Byzantine process

B How Dolev's protocol work

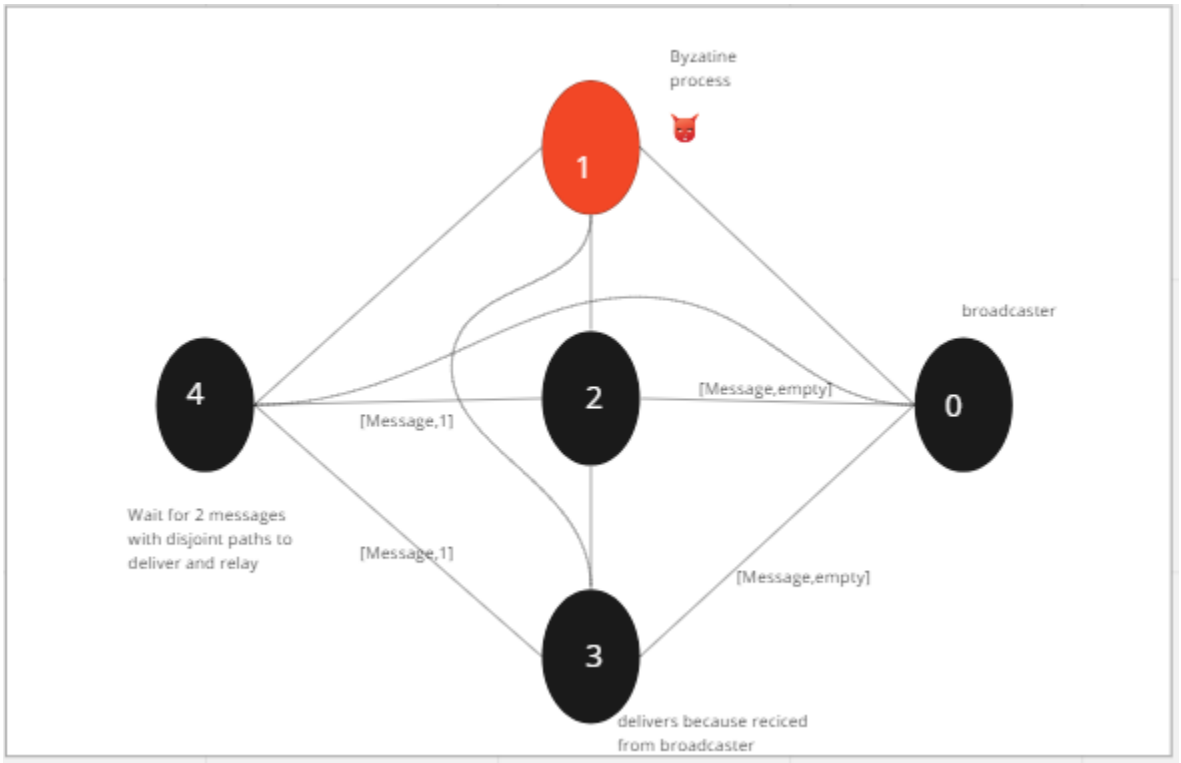


Figure 3: how Dolev's protocol works in a network with 5 processes and 1 Byzantine process

C Checking if we can apply CPA on graph

Theoretical parameter	Condition	Time complexity	Sufficient	Necessary
X(G)[11]	Graph is f-local $f < X(G/2)$	$O(nodes^3)$	Yes	No
LPC(G)[11]	Graph is not f-local $f \geq LPC(G)$	Exponential	Yes	Yes
K(G,D) [9]	Graph is f-local $f < k(G,D)/2$	$O(E \times \log \delta)$	Yes	No
M(G,D,t) [9]	Graph is f-local $f+1 \leq M(G,D,T)$	Exponential	Yes	Yes
T(G,D) [9]	Graph is f-local$f+1 \leq T(G,D)$	Exponential	Yes	Yes
Partitioning into F, L, R [12]	$L => R$ or R has a neighbour of the source	Exponential	Yes	Yes

Table 1: A table for different theoretic parameters introduced in this paper to check if we can apply CPA on a graph, their complexity and if the conditions that build on them are sufficient, necessary ors both

D Tables of success probability of CPA on k-regular graphs

number of nodes (N)	maximum Byzantine processes	maximum -1
8	1	10
10	0	10
12	0	10
14	0	9
16	0	9

Figure 4: A table that shows the number of successes when using CPA on a network with a k-regular graph network with connectivity **5**

number of nodes (N)	maximum Byzantine processes	maximum -1
8	10	10
10	0	10
12	0	10
14	0	8
16	0	5

Figure 5: A table that shows the number of successes when using CPA on a network with a k-regular graph network with connectivity **7**

E Tables of success probability of CPA on generlized-wheel graphs

number of nodes (N)	maximum Byzantine processes	maximum -1
8	0	10
10	0	10
12	0	10
14	0	10
16	0	10

Figure 6: A table that shows the number of successes when using CPA on a network with a generalized-wheel graph network with connectivity **5**

number of nodes (N)	maximum Byzantine processes	maximum -1
8	10	10
10	0	10
12	0	10
14	0	10
16	0	10

Figure 7: A table that shows the number of successes when using CPA on a network with a generalized-wheel graph network with connectivity **7**

F Tables of success probability of CPA on multi-partite-wheel graphs

number of nodes (N)	maximum Byzantine processes	maximum -1
12	10	10
14	0	10
16	0	10

Figure 8: A table that shows the number of successes when using CPA on a network with a multi-partite-wheel graph network with connectivity **6**

G Tables of success probability of CPA on k-diamond graphs

number of nodes (N)	maximum Byzantine processes	maximum -1
12	10	10
14	10	10
16	10	10

Figure 9: A table that shows the number of successes when using CPA on a network with a k-diamond graph network with connectivity **N/2**

number of nodes (N)	maximum Byzantine processes	maximum -1
12	10	10
14	10	10
16	10	10

Figure 10: A table that shows the number of successes when using CPA on a network with a k-diamond graph network with connectivity **N/2 - 1**

H Tables of success probability of CPA on k-pasted graphs

number of nodes (N)	maximum Byzantine processes	maximum -1
12	10	10
14	10	10
16	10	10

Figure 11: A table that shows the number of successes when using CPA on a network with a k-pasted graph network with connectivity **N/2**

number of nodes (N)	maximum Byzantine processes	maximum -1
12	10	10
14	10	10
16	10	10

Figure 12: A table that shows the number of successes when using CPA on a network with a k-pasted graph network with connectivity **N/2 - 1**