Introduction to Distributed Computing (95.401)
Lecture Notes for 4 October 1999

Gary Branscombe 247538 David Brons 237123
James McAvoy 194358 Derek Tam 228762

18 October 1999

1 Introduction

This is a continuation of Classical Problems of Election and the minimum finding problem in rings. Previous lectures dealt with the very basic technique of sending messages all around the ring. The algorithms studied so far require an excessive amount of communication, up to n^2 messages in the worst case.

Consider for review LeLann's Algorithm. LeLann figured that every node can send their value "All the way Around" the ring. Each node would receive all messages and in the end each node would know who is the leader. This brute force method works but is less graceful. It requires an astounding worst case of n^2 number of messages. Cheng and Roberts improved on this by implementing stop checks where the message is stopped by node of smaller value. Only the smallest node gets its response back and by virtue of the fact that it got the response it knows it is the leader. This leader then transmits a notification telling all other nodes that it is the leader. This "As Far as I can" algorithm has a worst case of $\binom{2}{n}$ messages but on average it is $.69n \log n$.

Hirshberg and Sinclair improved on this by utilizing bidirectional messages of *Controlled Distances*, where the worst case was $8n \log n$ messages. Hirshberg and Sinclair worked only if the links were bidirectional

Franklin introduced a bidirectional algorithm that required only $2n \log n + O(n)$ messages. This algorithm can be transformed into a *unidirectional* algorithm, which is the purpose of this lecture.

1.1 Restrictions

We make the following restrictions on the ring:

1. Bidirectional links
2. Total reliability
3. G is connected
4. Distinct values for all entities for election purposes

2 Franklin's Algorithm

Consider a node connected by bidirectional links on a ring. This node would be configured so that it can receive in both directions and send in both directions. To perform leader election, each node on the ring broadcasts its value in both directions and then waits until it receives a message from both its neighbors in both directions. If the node receives a value less than itself then the node is defeated and just passes all subsequent messages along. If the node receives both values that are greater than itself then the node lives and goes on to the next stage. The next stage of the algorithm is the same as the first stage, only this time there are less players because many have been defeated. In every stage, the number of players have been reduced in half at least. Only defeated nodes pass the messages along. So if one node receives its own message then every other node is defeated, and this node must be the new leader.

2.1 Rules

Let x, y and z be nodes connected in a ring.

- Rule 1. if($x > y$ OR $x > z$) then x is defeated and is eliminated from the election campaign.
- Rule 2. if($x < y$ AND $x < z$) then y and z are defeated.
- Rule 3. if ($x = y$ AND $x = z$) then x is the leader.

Note the redundancy here. If $x > y$ and $x > z$, then both y and z kill x. This is somewhat redundant and will be explored later.

2.2 Complexity Analysis

To find the worst case, it is important to find the number of stages it takes to arrive at the minimum number of messages. As mentioned, the number of players in each stage are cut in half. There could be more than half the players cut, but there is at least a drop by half in every round. So in the worst case, in every stage, the number of players is cut only in half, no more. For each stage, the number of players keeps decreasing until there is only one player left. We need to find out how many stages the algorithm goes through before it reaches one, the leader. Therefore, the number stages before the algorithm terminates is $\log n$, where n is the number of nodes.

$$\log n = \begin{cases} n \\ \vdots \\ n_i \\ n_{i+1} \leq \frac{n_i}{2} \\ \vdots \\ 1 \end{cases}$$

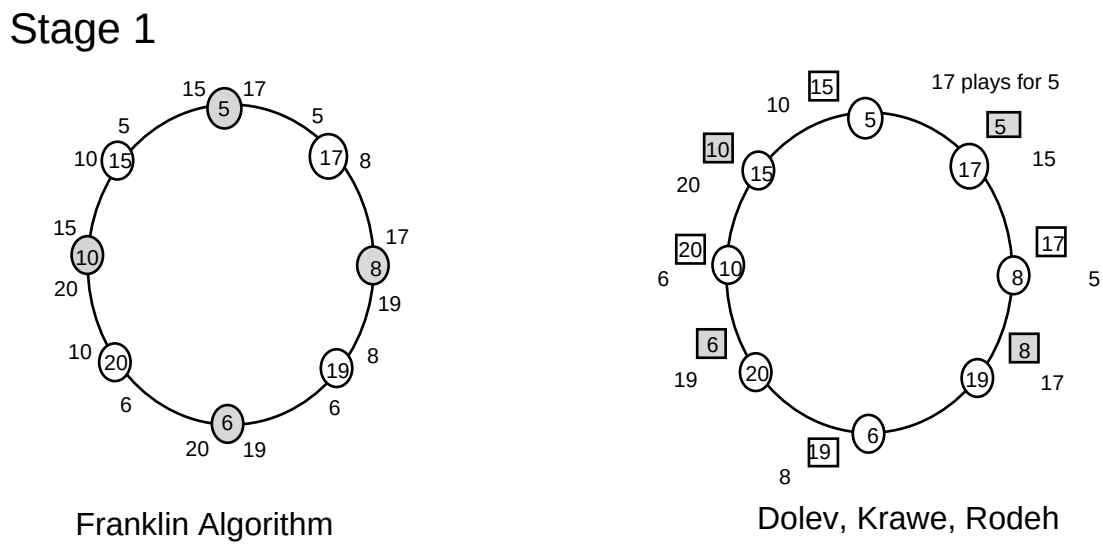


Figure 1: Stage 1 of the Franklin vs. Dolev et al algorithm

2.2.1 Message Complexity

$2n \log n + O(n)$, where $2n$ is the number of messages that each node sends in each stage, $\log n$ is the number of stages require to terminate the algorithm and $O(n)$ is the cost of notification.

3 Dolev, Klawe, Rodeh's Algorithm

Dolev, Klawe, and Rodeh converted the Franklin bidirectional model to a unidirectional. Figure 1 shows the two models: Franklins bidirectional model on the left and the unidirectional model of Dolev, Krawe, Rodeh. At the end of the the first stage, the Franklin model has sent two messages for each node in both directions. Node 5 makes a decision for himself because node 5 knows it is lower than 17 or 15 and decides to survive to the next stage. The point is nodes make decisions for themselves. The nodes 17, 15, 19, and 20 are eliminated. The unidirectional model also sends two message two complete the first stage. Each node sends a message in the only direction it knows, *right*. And then each node sends another message in the only direction it knows, *right*. In the second message, each node propagates the node it received in the first message forwards to the next node. The results are shown in the model on the right. Careful examination reveals that all the same information is there. It is just shifted right by one node. The same information a node has in the Franklin model (e.g. 5, 15, 17) is still captured within one node, only in the unidirectional model, it is within a different node. The same information to make a decision on the outcome of a

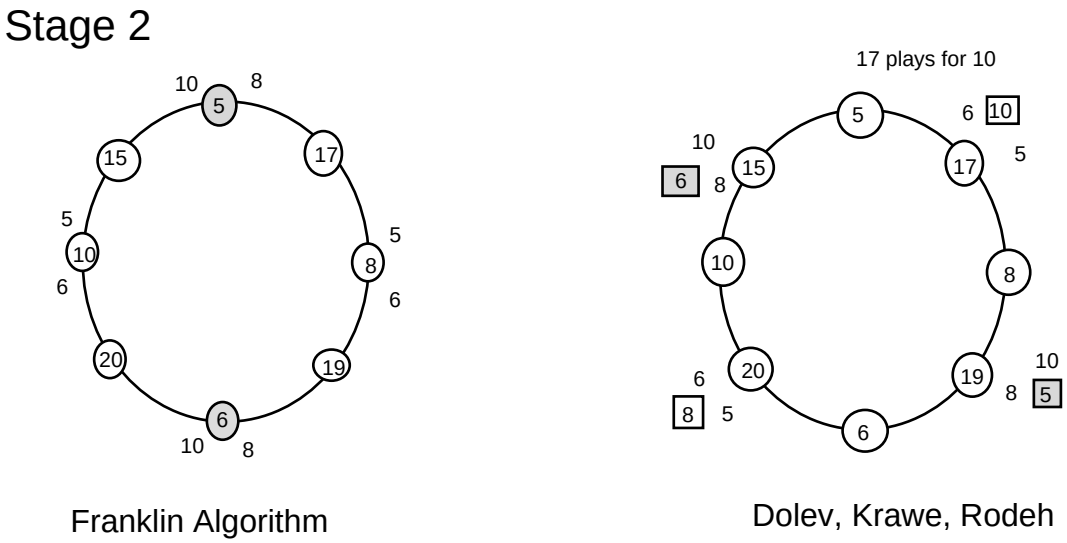


Figure 2: Stage 2 of the Franklin vs. Dolev et al algorithm

node (5, 15, 17) is still available except it is not within node 5 but within node 17. Node 17 now must act on behalf of node 5 and makes a decision. Node 17 makes the decision and indicates that node 5 survives. Instead of making a decision for itself every node makes a decision for a node, where the node is not itself. Node 5 is making a decision on behalf of node 15 and it dies. Node 17 is making a decision on behalf of node 5 and it survives. You can have the smallest value but still be out.

In stage two of the Franklin model, those that survive send their values in both directions. Node 5 sends a value 5 to node 10 and node 8. In the unidirectional model, Node 15 is playing on behalf of 10 so it sends a 10. Node 17 is playing on behalf of 5 so it sends a 5 and so on. The second step of this stage is to send what was received, so node 19 received 5 and 19 sends a 5, node 17 sends a 10. The node always plays on behalf of the first message it receives. So node 17 plays on behalf of 10, node 19 plays on behalf of 5, 20 plays for 8 and 15 plays for 6. In the first stage, node 17 was making a decision on behalf of 5, so it stuck around to the next round, but it must remember it was playing for 5. Now in the second round, Node 17 is playing for node 10 and must make a decision on behalf of node 10 and decide between 5 and 6. Since 10 is greater than 5 or 6, 10 loses and therefore 17 dies. It lived on behalf of 5 and died on behalf of 10. The first message 19 received was 5, so 19 is now making decisions on behalf of 5 and must decide between 8 and 10. Referring to the Franklin model you will note that the same decision is being made: 5 is deciding between 8 and 10. 5 is smaller than 8 or 10 and therefore 19 survives on behalf of 5. The third stage progresses in the same fashion as the second.

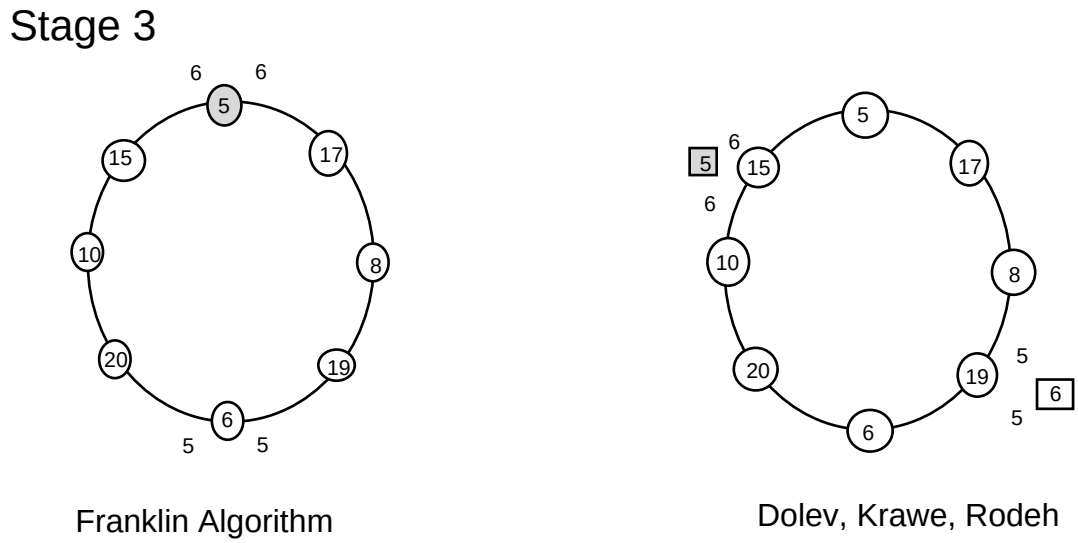


Figure 3: Stage 3 of the Franklin vs. Dolev et al algorithm

3.1 Complexity Analysis

There is an exact model duplication between the two models, but every time who is playing for whom changes. Every time it shifts by one. There is an exact translation between unidirectional and bidirectional. Surprisingly, the cost is identical between the two models:

$$2n \log n + O(n)$$

Where every node sends two messages, there is $\log n$ stages and $O(n)$ cost for notification. This means that the cost $n \log n$ is achievable in a unidirectional ring. It is important because it shows that there is no difference in the computing power between half duplex (unidirectional) and full duplex (bidirectional) lines.

4 Peterson "Alternate"

There are two versions of the Peterson algorithm. We will look at the bidirectional first and then the unidirectional version of this algorithm next.

4.1 Peterson Bidirectional

Peterson was able to improve on the bidirectional Franklin algorithm by sending messages one direction at a time, rather than passing messages to both neighbours at the same time. This would eliminate unnecessary message passing in cases where a node may be defeated by

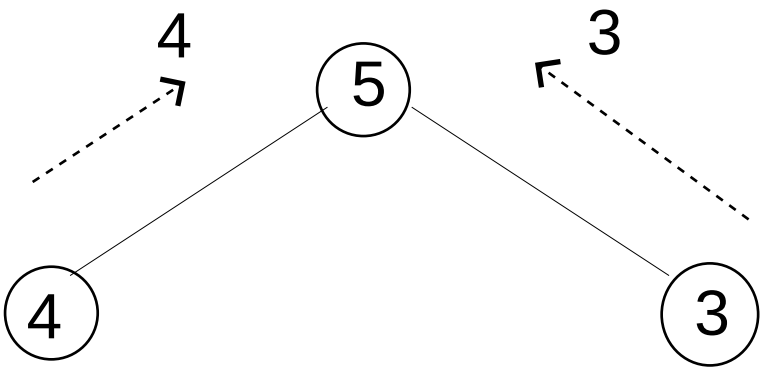


Figure 4: Key observation about passing messages to both neighbours

both messages received from its neighbours on either side. All that's needed is one message of lower value to be defeated. In the above figure 4, the entity with the value of 5 will be defeated by both its neighbors who have a value of 3 and 4.

Peterson's bidirectional algorithm involves the passing of messages from each node to a single neighbour, with each node passing a message in the same direction. At the termination of a stage, a decision is made as to which nodes advance to the next stage by comparing the value of the received message to that of its own. Nodes which have advanced then send their message in the opposite direction, bypassing those nodes which have been previously defeated. In other words, odd numbered stages proceed in the clockwise direction and even numbered stages proceed in the counter-clockwise direction. This procedure continues until we have a final victor. Peterson's method of "alternation" has a message complexity of $1.44n \log n + O(n)$.

If we assume that Peterson's algorithm commences in a clockwise manner, by the end of the first stage, each node has sent a message to its right-most neighbour. At this time, we shall look at the rules of the game. Let us denote x as the value of the node and y is the value of the passed message.

4.1.1 Rule

player$\times$ RECEIVE(y) $\longrightarrow$
if ($y < x$) BECOME *defeated*
else if ($y > x$) next stage flip direction
else if ($y = x$) BECOME *leader*

Perhaps this will be easier to follow with a short example. At the end of the first stage, each node must make a decision based on the received message as to whether or not it continues. By applying the above rules, we can see stage 1 in figure 5 that nodes 14, 17, and 20 have been defeated since the value of the received message is less than its own. On the other hand, nodes 5, 8, and 14 were victorious and continue onto the second stage.

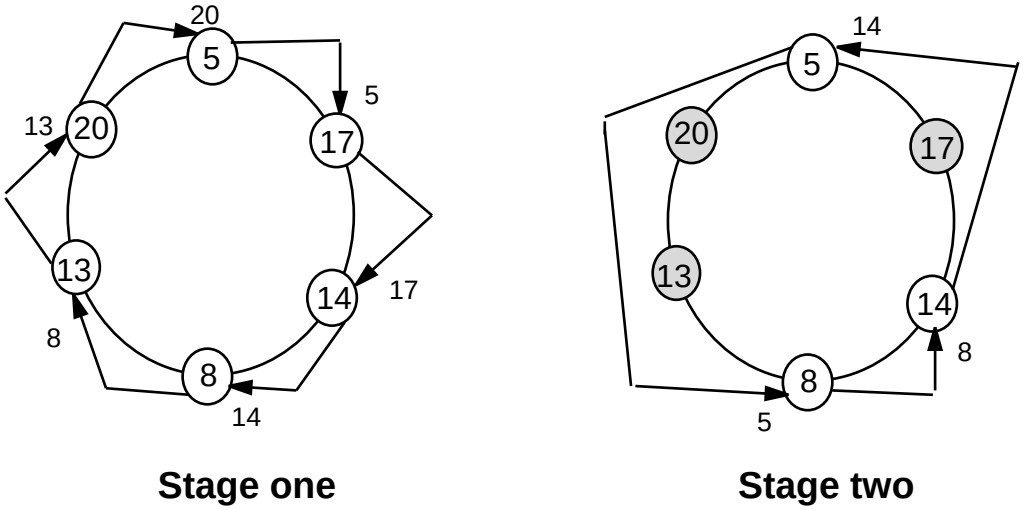


Figure 5: Stage 1 and 2 of the Peterson bidirectional algorithm.

According to Peterson's bidirectional algorithm, the next stage continues with messages being passed in the counter-clockwise direction. Nodes pass messages to the left, skipping over nodes which had been previously defeated (denoted in figure 5 by dark shading). We observe that 5 passes to 8, 8 passes to 14 and 14 to 5.

Proceeding to stage 3, we see that in the previous stage nodes 8 and 14 were defeated as a result of receiving messages with a lesser value, since $5 < 8$ and $8 < 14$ respectively. Since this is an odd-numbered stage, messages are sent to the right. Because 5 is the only remaining node, it sends a message to its right, skipping over defeated nodes and receives its own message, and is thus declared the final winner.

We have just demonstrated the use of the Peterson bidirectional algorithm for the purpose of determining the minimum value of a ring. We can observe that at any given stage, no more than n messages will be passed, where n is the number of nodes in the graph. However, we must be careful to note that we are guaranteed to eliminate only one node in the first stage, for example if we had a ring of 6 integers increasing in a counter-clockwise manner.

4.2 Algorithm Description (Unidirectional)

The bidirectional algorithm can be simulated on a unidirectional ring. Initially all nodes are active. Every active node with value x stores a current value x'. Initially $x' = x$. As described in its bidirectional version, the algorithm proceeds in stages. In a given stage, a node will receive a value y.

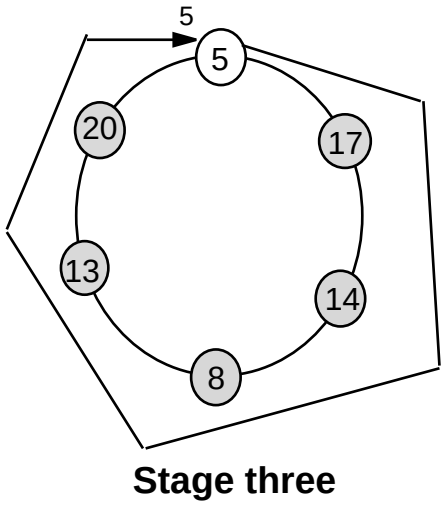


Figure 6: Stage 3 of the Peterson bidirectional algorithm.

4.2.1 Rules

If the direction of the protocol = physical link direction (i.e. odd stages) play for the current value x'

$y < x'$ defeated
$y > x'$ go to the next stage ('flip')
$y = x'$ leader

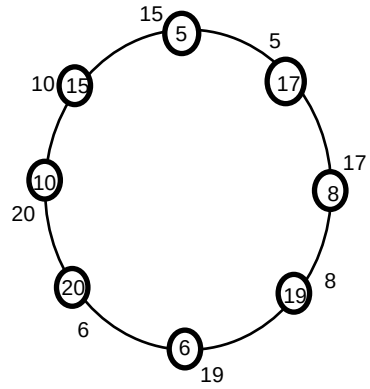
If the direction of the protocol not equal physical link direction (i.e. even stages) play for the received value y

$y < x'$ go to next stage ('flip') $x' := y$
$y > x'$ defeated
y=x' leader

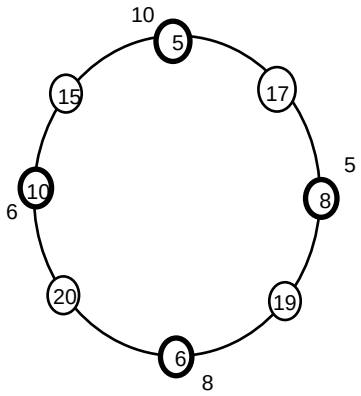
We keep on alternating. Instead of sending in both directions we only send in one direction. In order to simulate the bidirectional algorithm, we play for the message we have and then on the next turn we play for the message we receive.

4.3 Cost Analysis of Peterson Algorithm

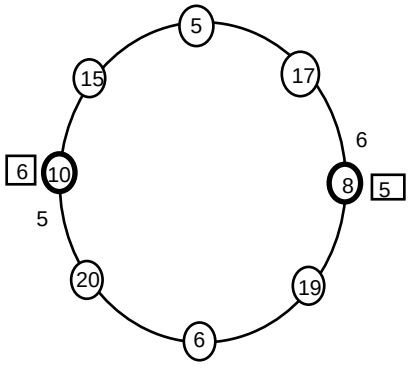
We cannot guarantee half will die after each stage of the algorithm in the case of the Franklin algorithm. We only can guarantee only one will die after each stage in the worst case. Thus, there are n messages per stage but how many stages are there in the worst case?



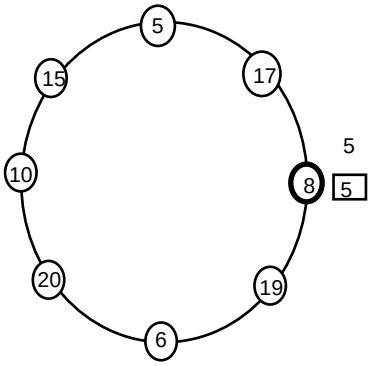
Stage 1: Play for current value



Stage 2: Play for received value



Stage 3: Play for current value



Stage 4: Play for received value

Figure 7: Stages of the Peterson unidirectional algorithm.

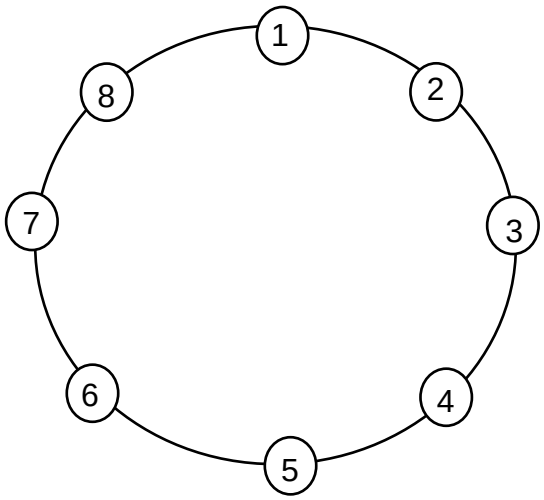


Figure 8: A sample ring with nodes that are order.

In figure 8, if we start the algorithm clockwise everyone will be defeated except for 1. In the counter-clockwise version only 8 will die. A single round will not guarantee consistent results.

In this algorithm, n messages are sent per a stage. To determine the total cost, we need to find out the total number of stages. Lets take the worst case and increase n (the number of nodes) as low as possible.

STAGE	n_i	COMMENTS
k	1	only one node left, he is the winner
$k-1$	2	second last stage. Worst case is only one more node.
$k-2$	3	
$k-3$	5	
$k-4$	8	

To get there, in the last stage there was only one node 1. In the second last stage, $k-1$, the message was traveling in a clockwise direction so in the worst case there was only 2 nodes, number 1 and 2. In the $k-2$ stage, the direction of message changed to counter clockwise. In this direction and stage, we need to protect node 2 from 1, otherwise node 1 will kill node 2 and we don't want that because node 2 needs to survive to the $k-1$ stage. To save 2, we must protect it with node 3 and let node 1 kill node 3. We have 3 nodes in this stage: 1, 2, 3. In the $k-3$ stage, message direction is reverses back to a clockwise direction. This time we need to protect 2 from 1 again. Only this time it is from a different flank, so node 4 is added between 1 and 2. And we need to protect node 3 from node 2 so we add node 5 between them. In the $k-4$ stage, switch direction again and this time we are counter clockwise. We need to add node 6 to protect node 3 from 1, we add node 7 to

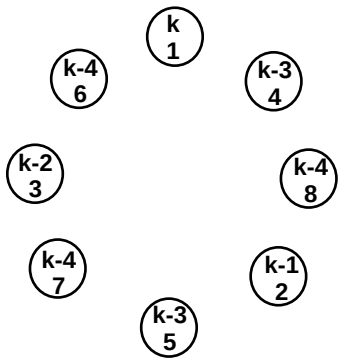


Figure 9: This figure shows the node's stage and value during the execution of the Peterson algorithm.

protect 5 from 3 and we add node 8 to protect node 4 from 2. We have a created a ring of the worst possible combination so that the number of stages needed to find the leader is maximized. Lets work forward to verify the ring is indeed correct:

Stage	n_i	Direction	Notes
4	8	$\Leftarrow$	1 kills 6, 3 kills 7, 2 kills, 8
3	5	$\Rightarrow$	1 kills 4, 2 kills 5
2	3	$\Leftarrow$	1 kills 3
1	2	$\Rightarrow$	1 kills 2
0	1	$\Leftarrow$	1 is the leader.

The sequence in column n_i is the number of nodes alive at that stage. It also represents the number of messages required to get to that stage. This series of numbers is very well known as the Fibonacci sequence. The ith term in the Fibonacci sequence can determine how many nodes are playing at the kth stage.

Stage	n_i	Fibonacci
k	1	f_1
$k-1$	2	f_2
$k-2$	3	f_3
$k-3$	5	f_4
$k-4$	8	f_5
$k-5$	13	f_6

The final result is $1.44 \log n$ if you add in the notification messages, you end up with $1.44n \log n + O(n)$.

4.3.1 Message Complexity

$1.44n \log n + O(n)$

5 Conclusion

Look at the cost results from other algorithms studied:

Algorithm	Cost	Link Type
Dolev, Klawe and Rodeh	$2n \log n + O(n)$	unidirectional
Peterson	$1.44n \log n + O(n)$	unidirectional
Higham	$1.26n \log n + O(n)$	unidirectional

Higham's code is complicated and awkward protocol but mathematically it is possible.

Patchel, Roach and Rotem stated that in the absolute worst case, the best you can do is $n \log n$ and in an average situation, the best average massaging cost is $0.69n \log n$.

Theorem 5.1 (Pachl, Korach, Rotem '84). *Any election algorithm in a unidirectional ring requires* $\Omega(n \log n)$ *messages.*

Even on average, we cannot do better than this. The only improve is in the constant. It turns out that Chang and Roberts very simple algorithm with an average case of $0.69n \log n$ messages is the best solution for unidirectional rings. Burns makes that following theorem,

Theorem 5.2 (Burns '80). *Any election algorithm in a bidirectional ring can be no lower than* $0.5n \log n$ *messages.*

To archive this low number, will require to add an additional requirement on the ring. The ring must have a *Sense of Direction.* Consider the following table:

UNIDIRECTIONAL	BIDIRECTIONAL	
n^2	n^2	
↓	↓	
$\binom{n}{2}$	↓	
↓	↓	
$2n \log n$	$2n \log n$	
↓	↓	
$1.44n \log n$	$1.89n \log n$	
↓	↓	
$1.25n \log n$	$1.44n \log n$	
↓	↓	
?	?	
↓	↓	
$0.69n \log n$	$0.5n \log n$	Theoretical lower boundary

What this means is that although the theoretical boundary is lower on the bidirectional links, than in unidirectional links, the gap between where the state of the art in massaging is greater in the bidirectional camp than it is in the unidirectional camp.