SafeNet for Computer Network EDA387

Group 24

November 5, 2025

Problem Consider an asynchronous computer network with a distributed but fair scheduler, i.e., not a central daemon. The network has n nodes, where $n < N$ is a finite number that is unknown to the algorithm, and N is an upper bound on n, such that only the value of $\lceil \log_2 N \rceil$ is known. The network is based on shared memory where each processor can write to a single register with up to $O(\lceil \log_2 N \rceil)$ bits. Recall that each register can be divided into multiple fields, say, one field per neighbor. The processors do not have globally unique identifiers, and all nodes run the same program without the presence of a distinguished processor.
Your task is to design a self-stabilizing algorithm that operates within an asynchronous network with a graph topology, which we specify below. The algorithm is required to achieve an accurate node count. After a period of recovery from the occurrence of the last transient fault, each processor should output, using the operation `print(x)`, the total number x of nodes in the system.
Please present a comprehensive explanation of your solution along with your exact assumptions.

1. Provide a well-structured pseudo-code for your algorithm.
2. Define the set of legal executions.
3. Present a proof of correctness, complete with all necessary arguments to convincingly demonstrate the algorithm's accuracy and self-stabilization. If needed, separate between the convergence and the closure proofs.
4. What is the stabilization time of the proposed algorithm?
5. Does a self-stabilizing naming algorithm exist for the system described? If such an algorithm exists, provide a detailed description and proof of its correctness. If not, provide proof of the impossibility of the result.

Question 1.

Suppose the network topology is an oriented bidirectional path graph P_n. Specifically, each register is divided into two fields. That is, processor p_i can read its neighbors' registers, left_j or right_k, while writing/reading only to its own left_i and right_i fields in its register, where p_j and p_k are p_i's neighbors.
The path is oriented from left to right in the sense that each node has a right and/or left neighbor. That is, starting from the leftmost node (which has no left neighbor, i.e., $\text{left}_i = \perp$) and taking exactly n hops to the right brings us to the rightmost node (which has no right neighbor, i.e., $\text{right}_i = \perp$).

Proof. Because of each processor has its own register, and in the register, it can be divided into fields, we will use this to store the important information as in the pseudo-code below.

```
00 do forever
01  if not hasLeft:
02      left_i := 0
03  else
04      left_i := min(left_j + 1, N)
05  if not hasRight:
06      right_i := left_i + 1
07  else
08      right_i := right_k
```

Let consider the left field in the register is for the relative distance from the processor to the leftmost, while the right field is for the number of the processors. In this problem, this code uses 2 floods to update the correct value for a processors. One from the leftmost, carries the relative distance from leftmost processor to P_i. The other comes from rightmost, carries the information about the total number of processors in this set.

- **Legal execution and legal configuration** The legal configuration for this system is, in the register s_i, the left field is for relative distance to the leftmost and the right field is for the total number of processors in this system. Within the $do forever$, these processors will keep the same values for these 2 fields, hence, guaranted the stabilization. From this state, only legal executions can be started.

- **Correctness** this algorithm will flood the value from the leftmost to the rightmost. Because it takes exactly n steps from leftmost to rightmost, at the end of block $01-04$ (the left flood), each processor will know its relative distance to the leftmost, and this will remain unchanged (because the value is the same after the stabilization). As the same with block $05-08$, the rightmost will assign the value of n and will flood it to the leftmost. Because these 2 processes begin from different directions and write to different fields, it is guarantee that the values are correct.

- **Lemma for left convergence** if a processor P_i has a value $left_i$ after several asynchronuous rounds. Because the leftmost always set its $left$ to 0, it will be stabilized after at most n rounds. Hence, the complexity of this algorithm is $O(n)$ for the left convergence.
- **Lemma for right convergence** because the rightmost keeps setting its $right$ to the value of its $left$, if a register $right_i$ has transient fault, it will eventually adopt the value n after at most n rounds. Same as left convergence, complexity is $O(n)$.
- **Time** at most n cycles, all these processors will be in the safe configuration state.

For the question: *Can a naming self-stabilization algorithm exist on such a system*
The answer is yes, with these above configuration and assumptions, a naming self-stabilization can exist. As in the line 01 and 02 stated that, the leftmost node, which does not have any right neighbor (also stated in the initial configuration), will adopt the value $left = 0$. This will make it in to a distinguished node. The leftmost adopts this attribute as its ID, hence floods for all other nodes to get an ID as well. Moreover, in the pseudo code, the leftmost acts as an anchor for the flooding. After the convergence, each nodes will have a unique value in left field of its register, which can be treated as a unique property.
Also, this system's symmestric was broken by the topology. Nodes are not identical, because we have leftmost and rightmost are not the same with other nodes (have neighbors from both side).

□

Question 2.

Consider the following definition of a tree graph, which we later extend:

- **Connectedness:** There is a path between any pair of vertices in the tree.

- **Acyclicity:** The tree contains no cycles.
- **Finite vertices:** The tree has a finite number of vertices and edges.
- **Edges:** Suppose there are n vertices in the tree. The number of edges is $n - 1$.

This question assumes that the network topology is a *non-rooted tree*, which is a tree network with only local identifiers. This is unlike rooted tree networks, where one of the nodes is distinguished.

Proof. Consider this pseudo code block:
Connect the graph when leader is elected

```
P_i
00 do forever:
01     for each P_j in N(P_i):
02         sum_from_other_neighbors = 0
03         for each P_k in N(P_i) that k ≠ j:
04                 sum_from_other_neighbors += P_k.size_for[P_i]
05         P_i.size_for[P_j] = 1 + sum_from_other_neighbors
06     total = 1
07     for each P_j in N(P_i):
08         total += P_j.size_for[P_i]
```

The above pseudo code is the solution for couting problem in a tree. The main idea is each node read from its neighbors the information about the total nodes rooted at that children. For example, the node A has B, C and D as children, it will read the value about the total node from subtree rooted in B, C and D, and counted 1 for itself, it will know the value of total node in this system.

- **Legal configuration and legal execution set** the safe configuration for this system is every leaves keep writing the $total = 1$ and all the nodes write the same information in the P_i.size_for[P_j] (keeps broadcasting the same value for all its neighbors).
- **Correctness** in this tree, because it has the same setting with question 1 that leftmost node (now we call it leaf) that does not have any neighbor except itself. This ensure that after the $doforever$ loop, it will only has $sum\_from\_other\_neighbors = 1$. Regardless a garbage value occurs, this nodes always propagate 1 to its neighbors. When this value is passed to P_j, neighbor of the leaf, it will further broadcast this value to all other nodes, creates a way of correction. Because this tree contains no loop and there is always a way from every two nodes, the propagation will reach every nodes. When all the garbage values are eliminated, each node P_i will calculate the total by the value from every neighbors, hence knows the total nodes of this system.

- Self-stabilizing if the system enters an arbitrary state, the leaf node (node that has only 1 neighbor) will always adopt the value $sum\_from\_other\_neighbors = 1$, this ensures that all the leaves propagate correct value, thus begins the correction way to other nodes.

□

For the question: *Can a naming self-stabilization algorithm exist on such a system*
Unlike question 1, every leaf nodes in this tree are the same. They are identical and run the same code (try to write 1). The easiest imagination is the star-model. With one central node and several leaves, these nodes are exactly the same and only have one neighbor. To have a naming algorithm the symmestry must be break.