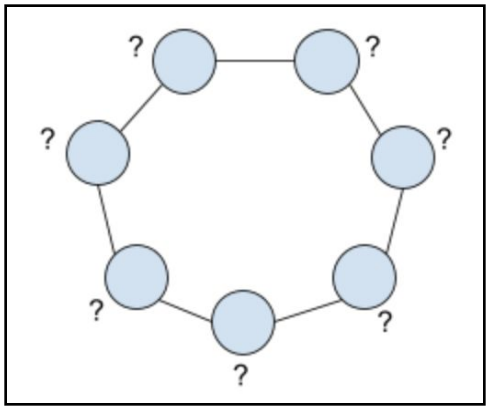
Monte Carlo Algorithm for MIS in uniform anonymous ring assuming CONGEST

Assumptions made

1. **CONGEST model, aka a fail-free, synchronous** model where each round i) a node sends message ii) receive messages and iii) does processing.
2. Uniform ring (nodes have **no knowledge of size 'n'**)
3. **Anonymous** ring (nodes do not have unique ID)



Impossibility result

- **No Las Vegas algorithm exists** that can solve the Leader Election problem in an Anonymous, uniform ring.
- Only a monte carlo algorithm is possible, which has a non zero probability of breaking correctness (> 1 leader).

Conclusion

- Therefore the **best we can target is a Monte Carlo algorithm** with the given assumptions.
- We can think of a **trivial Monte Carlo algorithm**:

a) All nodes pick a random n-digit number
b) If all the numbers are distinct -> they have unique ids -> we can compute size of ring -> Node with larger ID is leader -> Success
c) If all numbers are NOT distinct -> More than 1 leader possible

- We can make this **P(Failure) arbitrarily low** by increasing n-digits.

Motivation

- We want something stronger than the previous trivial solution.
- We can combat this “Monte Carlo-ness” using Public-Private Key Authentication. (Explained later)
- As long as authentication is not solvable in polynomial time, this algorithm will will not a feasible probabilistic chance of failure and be relied upon to applications.
- However even though correctness is protected by our authentication based solution, it is still technically a Monte Carlo Solution. (As even RSA can be broken by brute forcing a password)

Literature Review

- We searched for papers with our same settings/assumptions which could do it in **O(nlogn) message complexity.**
- No algorithm is available for these assumptions. Many are available for other assumptions in the ring (such as when the size of the ring is known).
- The idea of combating Monte Carlo failures with authentication seems to be a novel direction of thinking.

Literature Review

Itai-Rodeh Election Algorithm

Consider an anonymous, directed ring.
Let all nodes know the ring size N.

Each node selects a random identity from $\{1, \ldots, N\}$. Now run the Chang-Roberts algorithm.

Average-case message complexity: $O(N \log N)$

Literature Review

Ring Size

There is no Las Vegas algorithm to compute the size of an anonymous ring!

Cidon and Yuval [1] show that it is impossible to produce a Las Vegas algorithm that computes non-symmetric functions such as Leader Election and the size of the ring.

Itai and Rodeh gave a Monte Carlo algorithm to compute the size of an anonymous ring. Such an algorithm may terminate *incorrectly*.

Literature Review

Symmetry breaking in distributed networks [Itai and Rodeh]

In the above cases the size of the ring was assumed to be known to all the processors. If the size is not known then finding it may be done only with high probability: any algorithm may yield incorrect results (with nonzero probability) for some values of n. Another difficulty is that, if we insist on correctness, the processors may not explicitly terminate. Rather, the entire ring reaches an inactive state, in which no processor initiates communication.

For the synchronous case several algorithms are presented: The simplest requires, on the average, the transmission of no more than $2.442n$ bits and $O(n)$ time. More sophisticated algorithms trade time for communication complexity. If the processors work asynchronously then on the average $O(n\log n)$ bits are transmitted.

Our Monte Carlo Algorithm

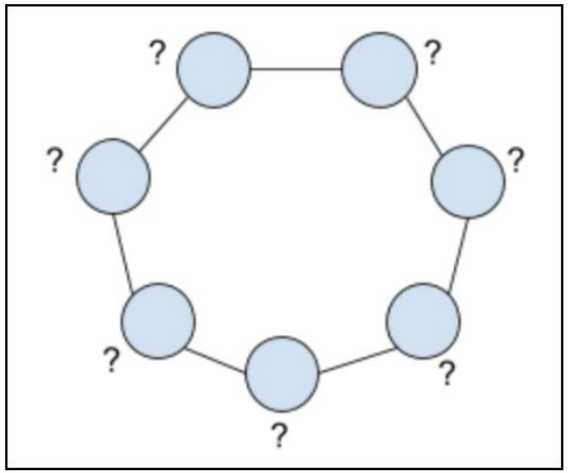
High-level overview

1. The algorithm uses the idea of recursively adjacent merging nodes to form "supernodes"/a group of nodes thus reducing the size of the problem from n → n/2 (on average).
2. In each round of the algorithm, a supernode can either vote for its left or right neighbour. This process of merging between 2 adjacent supernodes happens when they both vote for each other.
3. After the formation of a supernode of say size k, information of the supernode metadata is disseminated to all k nodes in the SNode.
4. Thus on each iteration, the size of the problem recursively decreases and the expected number of rounds is O(logn). Each round disseminates O(n) messages in the graph. Therefore it is O(nlogn) message complexity.

An Example

Let us consider an example with n = 7 (this size is not known any node in the graph). Ignore the '?' marks in the diagrams.

Initially, there are 7 nodes (or 7 supernodes of size = 1).



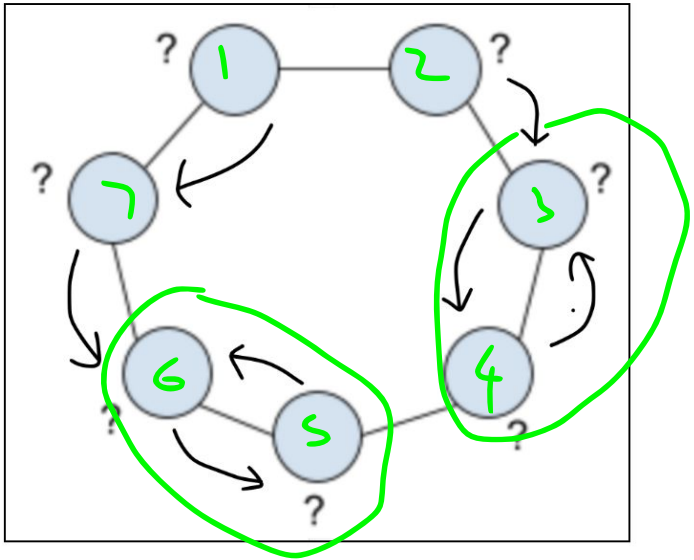
An Example

Each round consists of the following:

a) Sending a vote by "head"/negotiator of the supernode
b) Receiving vote
c) Checking for Termination condition (if needed)
d) Merging 2 supernodes and disseminating information insider supernode

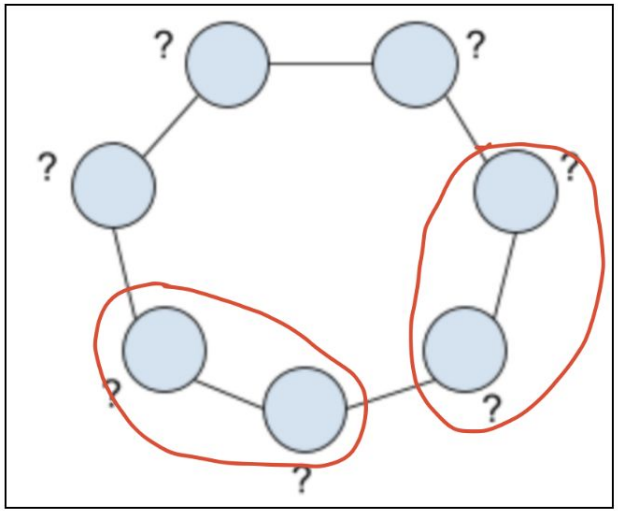
An Example

Assume the nodes vote like this in round = 1.



An Example

Each node will check and see if it received a mutual vote, if true, will merge to a new supernode, or will remain as is. The resultant ring is as shown below. We can see that 2 new supernodes have formed of size = 2 and the size of the problem has reduced from 7→ 5.

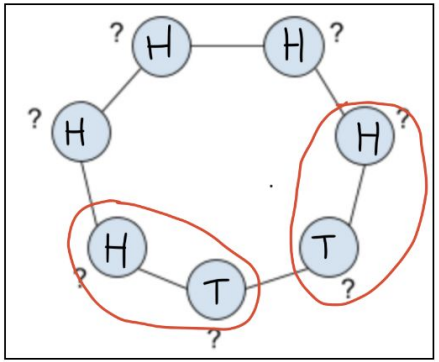


An Example

Furthermore, by exchanging of information between the last 2 heads of the Snodes, each node in the merged supernode have knowledge of:

a) Size of the supernode
b) Its position inside the supernode
c) The "head" of the Snode, or the voter
d) The "tail" of the supernode

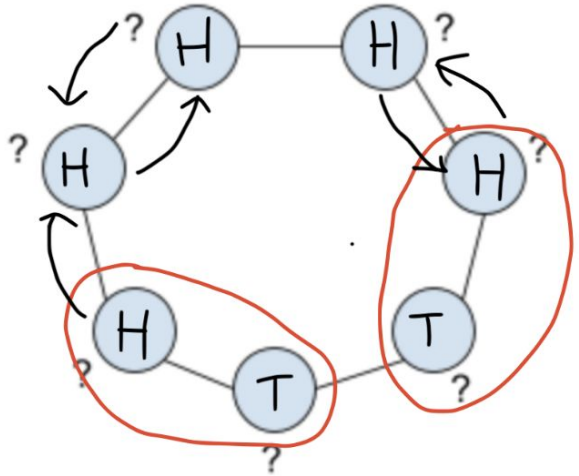
After dissemination of information, the f graph is obtained.



An Example

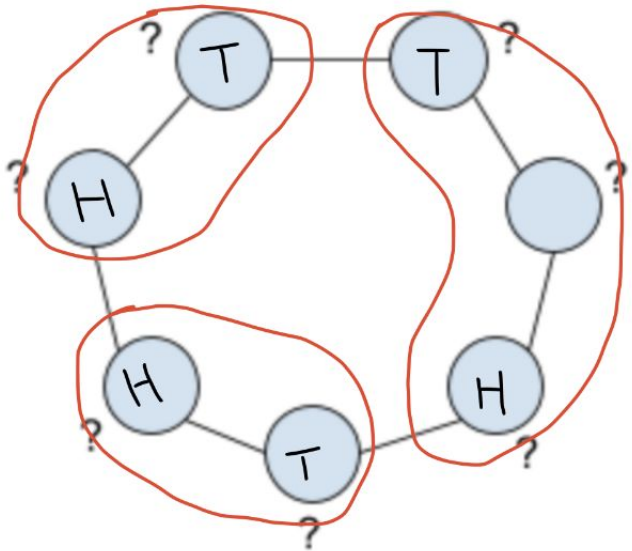
The 'H' on a node indicates that it is the head of its supernode i.e. It will vote next round for its adjacent neighbor. The 'T' on a node indicates that it is the tail of the supernode.

The next round of voting proceeds as follows,



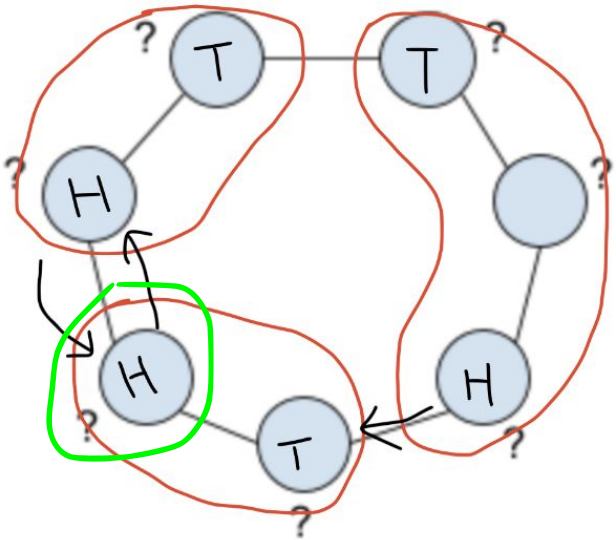
An Example

Again we can see there have been 2 matches and therefore these will be merged like before. After merging, new H and T nodes will be allocated as shown below.



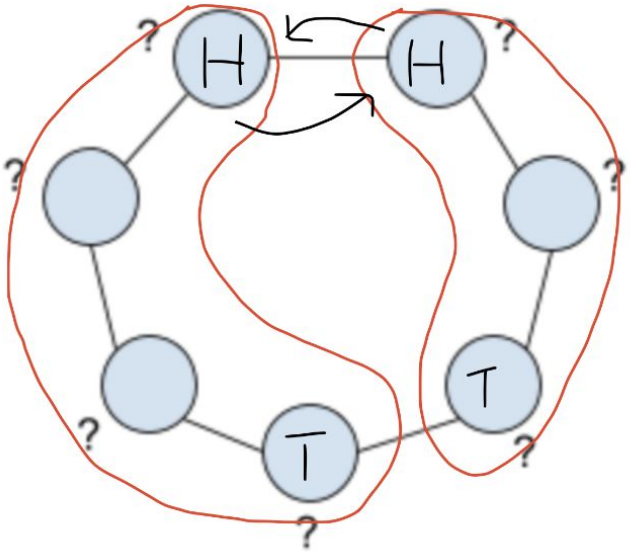
An Example

The next round of voting will happen as follows (the H of each supernode will vote for its adjacent Snode).

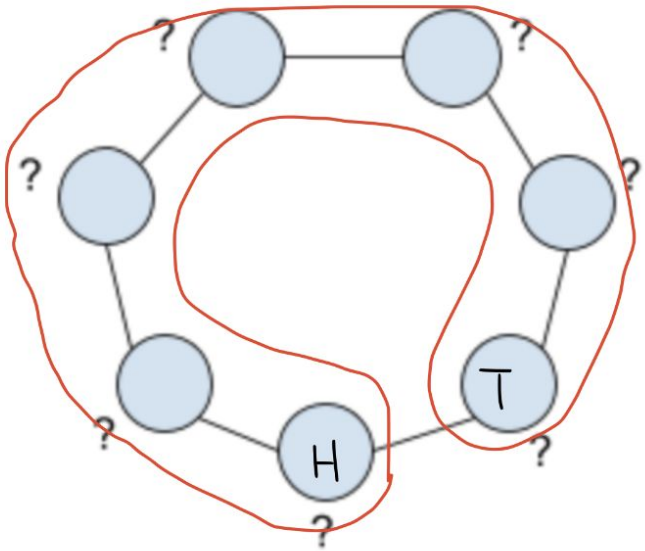


An Example

The merging is done and new H and T are randomly selected for each supernode. The next round of voting is as shown below.

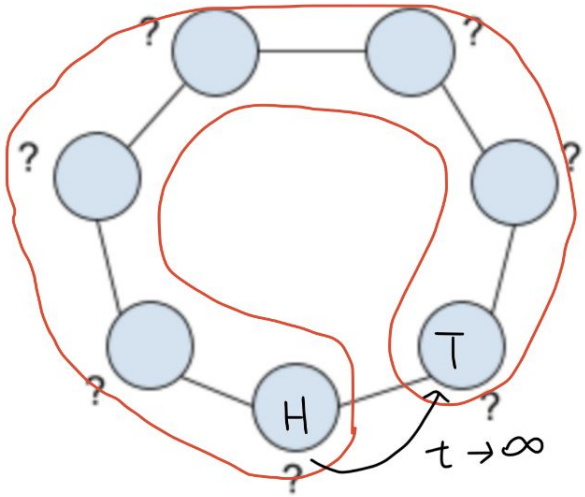


An Example



The trickiest part of this algorithm is the question of termination detection or "how does the final supernode know that it is the final one?". If this is not addressed, then the H of the supernode will go on voting forever and will never get a reply. Therefore the algorithm will not terminate.

An Example



As we can see, the Head of the Snode will have no choice but to vote for its own Tail. The head will not know that the tail belongs to itself. SImilarly the tail will not know that the vote it is receiving is from the Head of its own supernode. In this case, it is the tails responsibility to check for Termination detection.

Termination Detection

Aim: The tail of the last supernode needs to inform the head that it is voting for the tail of its own supernode. After the head knows this, the algo is over.

Key question: “How can the head TRUST that the tail actually belongs to its own Snode? Can it just look at the Snode ID of the tail?”

Non-Trivial: This last question is tricky and is the reason why there is no deterministic solution. There is no way the head can confirm that the tail belongs to its own Snode with 100% chance.

Our Solution: Authentication

Steps:

1. Each member of an snode has access to the “Snode Password”. This information is disseminated when the merging of 2 Snodes occurs.
2. When the tail thinks its own head is working for it, it will send a TD_check message to the head.
3. This TD check message contains Snode password.
4. If head sees that the password matches its own Snode password, Termination detection, SUCCESS.
5. Else, False termination detection, continue algorithm to completion.

Our Solution: Password leakage problem

- The tail is revealing its RAW SNode password to a potential foreign node.
- This foreign node can use this password to pretend its from Snode and break correctness.
- Therefore an adversarial node can make a False termination happen.

Our Solution: Public private authentication

- To prevent sharing of raw password in TD_check message, we first encrypt the password using a one way hash function F(public key, private key).
- This new encrypted password is shared during TD check.
- After head decrypts the password and sees there is match, termination detection.

Progress so far

1. Algorithm implemented and it is working for n-nodes. We can observe that it is doing what is expected using logs.
2. (TODO) “Is this topic of academic value? Will it be a useful finding/ accepted? Maybe need to ask specialist before pursuing”
3. (TODO) Analysis and plots (n vs #rounds)
4. (TODO) Verify O(nlogn) T.C.
5. (TODO) Add proper authentication (RSA)

Thank you! Questions?