# Homework #03

M^ed Amine
Chorfi
CA-CS-803

## Problem 3.1:

- In case of msg duplication for Wave Algorithms, issues might arise, notably, Increased Network traffic, Msg duplication yields an exponential increase in messages exchanged, each duplicate msg would cause a Node to send more duplicates, even flooding the Network.

⟹ Network Congestion: Excessive Msg can saturate the network slowing down communication & causing delays.

⟹ Resources Exhaustion: Nodes might run out of bandwidth, memory or processing power, to deal with increased number of messages.

- According to the Wave Algorithm definition, Chap 3.2, Nodes transition their states based on messages received.

⟹ Redundant Processing: Nodes may process same information twice, wasting computational ressources.

⟹ State Corruption: Receiving Same message multiple times might cause nodes to enter incorrect state, if Algorithm is not designed Accordingly.

- Wave Algorithms relies on detecting of procs after ending their tasks, and No msg circulating in order to terminate properly.

⟹ False Continuation: duplicates messages might prevent Algorithm from recognizing Termination time, causing it to run infinitly.

⟹ Premature Termination: Algorithm mistakenly assumes that all necessary messages, have been exchanged/processed it might terminate too early, before task is Completed.

- The Correctness of Wave Algorithms often relies on assumption each msg is sent exactly Once.

⟹ Incorrect Results: The Algorithm might produce incorrect Output is message is processed multiple times.

## Degrade Performance:

- The efficiency of algorithm is compromised, as processes spend additional time & resources handling duplicates messages.

## Problem 3.2:

- Modified Echo Algorithm mitigating messages duplication issues.
- We would implement a type set parameter into our Node .received_messages = set() to keep track of processed message IDs and discard duplicates

Each node uses a self.lock() to ensure its state is accessed and modified safely when handling concurrent messages reception

Check ("A3_P3.py") Attached

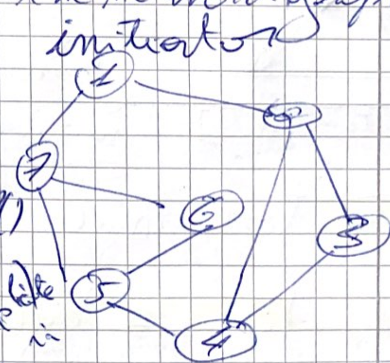
- Each Node work method ("initiator_work"/"non_init_work") is executed in separate thread, enabling processes to operate concurrently
- The "send" method simulates network delay using "threading.timer".
- Both "initiator_work" and "non_init_work" methods include a "time.sleep(0.1)" to avoid busy waiting while checking for condition to proceed.

## Problem 3.3:

("check A3P3.py")

## Problem 3.4:

Executing Echo Algorithm in the network graph below:




* Threads are used to simulate concurrent processing and message handling
* Concurrency managed using locks

$rec = 0$ $p(v_p \in G)$; $rec_{3,7} = 1$; $rec_{3,4,5,6} = 1$; $rec_{4,1} = 3$; $rec_{4,1} = 3$

; $rec = 0$ $p(v_p \in G)$; $rec_{2,7} = 1$; $rec_{3,6} = 2$; $rec_{3,6} = 2$

$rec_1 = 0$; $rec_{2,7} = 2$; $rec_{2,7} = 3$

$rec_1 = 0$ $rec_7 = 2$

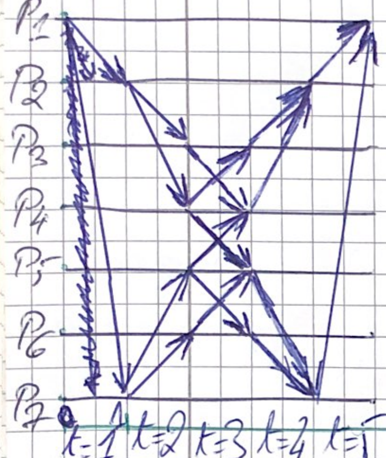
$t = 1$, $2$, $3$, $4$, $5$

Lamport Logical Clock

Initial State $t=1$ ; $rec = 0$ $(\forall p \in G)$

at $t=5$ ; $rec_1 = \#Neigh_1$

and since ~~Node~~ ① is the init in our network topology it would decide at that moment after all nodes of the network have been reached, and eventually termination of Algorithm



$rec_{1,3,6} = 2$

$rec_{4,5} = 3$

$rec_{2,7} = 3$

Lamport Logical Clock

$t=5$ terminal State

$(\forall p \in K)$ $rec_p = \#Neigh_p$

$\forall k = [1;7]$ All nodes have received ⟨tok⟩ the token transfered it to their neighbours and received and echo back through a child-parent relationship

refer to "A3P3.py" for an actual implementation in a multi-thread environment monitoring message delay and measuring it.

## Problem 3.5:

a) Hamiltonian Paths or cycles:

N-dimensional HyperCube has $2^N$ vertices each vertex is connected to N others leading to a total of $\frac{N.2^N}{2} = N.2^{N-1}$ edges

- We are looking for the number of Hamiltonian paths or cycles, which visit each vertex exactly once and traverse $2^N - 1$ edges

$\Rightarrow$ each vertex can be visited in a sequence $2^N!$

- Given the symmetries of HyperCube, not all sequences are distinct. Specifically, HyperCube can be rotated in $N!$ ways (the N! of permutations of N dimensions) and reflected in $2^N$ ways. (each dimension can be independently flipped)

Thus, Num of Hamiltonian paths $\simeq \frac{(2^N)!}{2^N . N!}$

b) Deterministic Hamiltonian Path.

Gray Code Ordering

a binary sequence where
two successive values
differ in exactly one bit
Method: traverse the HyperCube
vertices in order given by
an N-bit Grey code.
This ensures that each vertex
is visited exactly once
& only Adjacent vertices in
the Gray Code differ by one edge
in the HyperCube.
providing a deterministic
Hamiltonian Path.