భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Comparing Token based Mutual Exclusion Algorithms

Shagun
cs23mtech14013

Index

A log(N) distributed ME Algorithm based on Path Reversal

Local Variables -

- *token_present*: boolean (token–present is true if the process owns the token, false otherwise)
- *requesting_c_s:* boolean (true if the process has invoked the critical section and remains true until i releases the critical section.
- *next, father:* 1..n < {nil} (n is the number of processes; nil means indefinite)

A log(N) distributed ME Algorithm based on Path Reversal

Data Structures -

1. *Queue*

- ➢ Each process knows the next process in the queue only if this "next" exists.
- ➢ The head is the process possessing the token.
- ➢ The tail is the last process which has requested the critical section (excepted when there is only one process in the queue).
- ➢ A path is organized so that, when a process requests entering the critical section, the request message is transmitted to the tail.

2. *Logical rooted tree*

Algorithm Overview

- every node maintains two pointers, **father and next.**
- **Initially**, the graph topology of the system is present in the form of a logically rooted tree, whereby the **root node holds the token** to enter CS and all other processes are pointing to it, either directly or indirectly.
- Father tries to maintain the node to which the request messages are sent. From ‘father’ to ‘father’, a **request is transmitted** to the root which has the token. Furthermore, if the requesting process is not the root, the rooted tree is transformed, the original requesting process is the new root and the processes located between the requesting process and the root will have the new root as the father.
- The next variable is responsible for passing of the token to that request which has requested for the token at the earliest. From ‘next’ to ‘next’, the token for entering CS is passed to all the requested processes in a FIFO order.

Path reversal Algorithm

Algorithm of every process i

```
Initialization
begin
        father :=1       {the initialization of father is the same for every
        process}
        next :=nil
        requesting_c_s := false
        token_present := (father = i)
        if (father = i)   then
                father := nil
        endif
end  {Initialization}

Procedure Request_C_S
begin
    requesting_c_s := true
    if (father ≠ nil) then
    begin
            send Req(i) to father
            father := nil
    endif
end {Request_C_S}
```

```
Procedure Release_C_S
begin
    requesting_c_s:=false
    if next ≠ nil then
    begin
            send Token() to next
            token_present:= false
            next := nil
    endif
end {Release_C_S}

Upon receiving the message Req(k)
            {k is the requesting process}
do
      if (father=nil then
            if requesting_c_s then
                  next:= k
            else token_present := false
                  send Token() to k
            endif
      else
        send Req (k) to father
      endif
      father := k
od

Upon receiving the message Token()
do
      token_present := true
od
```

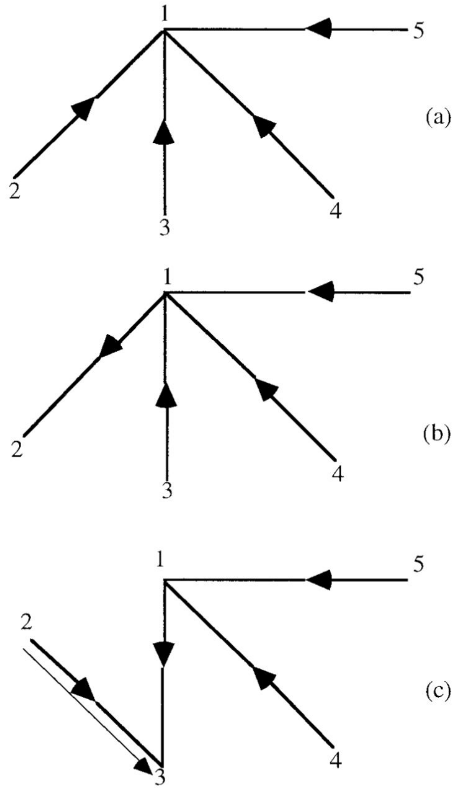
A log(N) distributed ME Algorithm based on Path Reversal

Example :-

Initial state of the distributed algorithm.
Process 1 has the token (fig. 2. a)

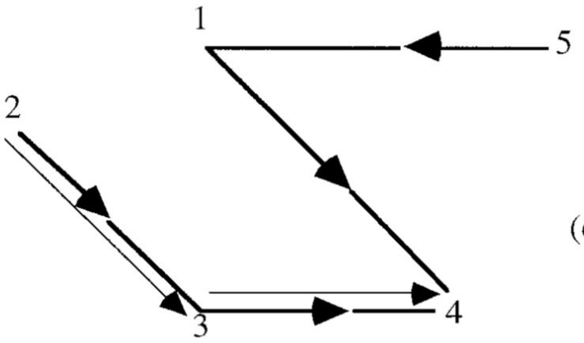
Process 2 invokes the mutual exclusion. It sends a request to process 1, and becomes a new root.
Process 1 receives the request from process 2 and sends the token to it. The father of process 1 becomes 2.
Process 2 receives the token and enters the critical section (fig. 2. b)

Process 3 invokes the mutual exclusion. It sends a request to process 1 which transmits it to process 2.
Process 2 receives the request, and considers 3 as its next (thin line). The father of processes 1 and 2 becomes 3 (fig. 2. c)



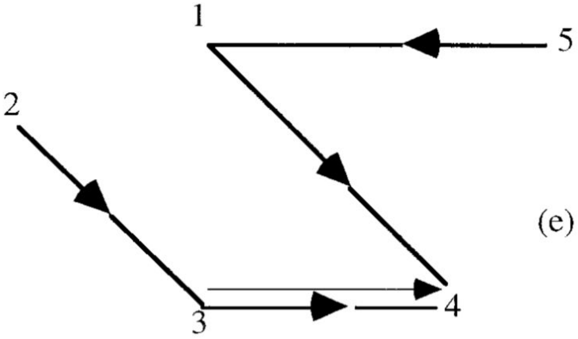
Example (contd..)

Process 4 invokes the mutual exclusion.
It sends a request to process 1 which transmits it to process 3.
Process 3 receives the request sent by process 1, its next becomes the process 4.
The father of process 1 becomes 4 (fig. 2. d)



(d)

Process 2 releases the critical section, and sends the token to its next, the process 3 (fig. 2. e).



(e)

A log(N) distributed ME Algorithm based on Path Reversal

Message Complexity : -

Since the token and requested messages are always transferred in a logical tree topology, the average number of messages for n processes is of **O(log(n))** complexity where n refers to the number of nodes in the system.

A distributed algo for ME in an Arbitrary Network

Assumptions :

- Interprocess communication lines facilitate **bidirectional** transmission.
- The network has no loss of messages; no message alteration; finite transmission Delay.
- FIFO ordering is not a requirement for channels.
- Network topology assumptions are limited to **connectivity**.
- Processes possess only local knowledge: their own ID and the identities of neighboring processes.

A distributed algo for ME in an Arbitrary Network

Message Format

```
enum messageType
{
    TOKEN,
    REQUEST,
    TERMINATE
};
```

```
typedef struct RequestMessage
{
    enum messageType type;
    int senderID;
    int reqOriginId;

    LLONG reqTime;

    char alreadySeen[MAX_LENGTH];

} RequestMessage;
```

```
typedef struct Token
{
    enum messageType type;
    int senderID;

    int elecID;

    LLONG lud[MAX_NODES];
} Token;
```

A distributed algo for ME in an Arbitrary Network

- When a process wants to enter the critical section it **sends a request message** to its neighbours and **waits for the token** message.
- **Upon receiving a request** message, a process P **broadcasts** that request to its neighbours not belonging to the control part of the message; this part is a subset of the set of process names carried by every request message (. The request is added to P's set of known pending request.
- If the process P owns the token and is outside the critical section , it extracts the oldest request R from its known requests set . It then sends the token to the creator of R through N, the neighbour of P which sent him the request R . If process P is inside the critical section, it will behave as stated above upon exiting it.
- Upon receiving the token message, process P keeps it if the token's addressee is itself. Otherwise P hands it over to N, the neighbour which sent him the request being serviced .
- A process can enter the critical section only if it owns the token.

A distributed algo for ME in an Arbitrary Network

```
procedure enter_CS;
  begin
    if token_here_i
      then in_CS_i := true
      else
      --broadcast a request
      ∀k ∈ neighbours_i: send req((i,C_i),(i,neighbours_i ∪
            {i}))to P_k;
    endif;
    wait in_CS_i;
    --May be interrupted upon receiving a
      message.
  end enter_CS;

procedure exit_CS;
  begin
    in_CS_i := false;
    transmit_token;
  end exit_CS;
```

A distributed algo for ME in an Arbitrary Network

```
procedure receive_request(req((req_origin,req_time),
        (sender,already_seen)));
  begin
    if ∃(req_origin,t) such that(req_origin,t) ∈ req_array_i
          ∧ (t < req_time)
      then
      --Delete this old request
      req_array_i := req_array_i-(req_origin,t);
    endif;
    if (req_origin,req_time) ∉ req_array_i ∧ ¬
        (∃(req_origin,x) such that x > req_time)
      then
      --The request just received is a new
        one and is the youngest that P_1 ever
        received from process P_req_origin
      C_i := max(C_i,req_time) + 1;
      req_array_i[sender] := req_array_i[sender] ⊕
          (req_origin,req_time);
      --Broadcast the request
      ∀k ∈ neighbours_i − already_seen:
        send    req((req_origin,req_time),(i,already_
            seen ∪ neighbours_i)) to P_k;
      if token_here_i ∧ ¬ in_CS_i then transmit_token;
      endif;
    endif;
  end receive_request;
```

```
procedure receive_token(token(lud,elec));
  begin
    token_here_i := true;
      if elec = i
      then in_CS_i := true
      else
      --The token is following the path
        that the corresponding request
        established.
      let via be such that (elec,x) ∈ req_array_i[via];
      token_here_i := false;
      send (token(lud,elec)) to P_via;
    endif;
  end receive_token;
```

A distributed algo for ME in an Arbitrary Network

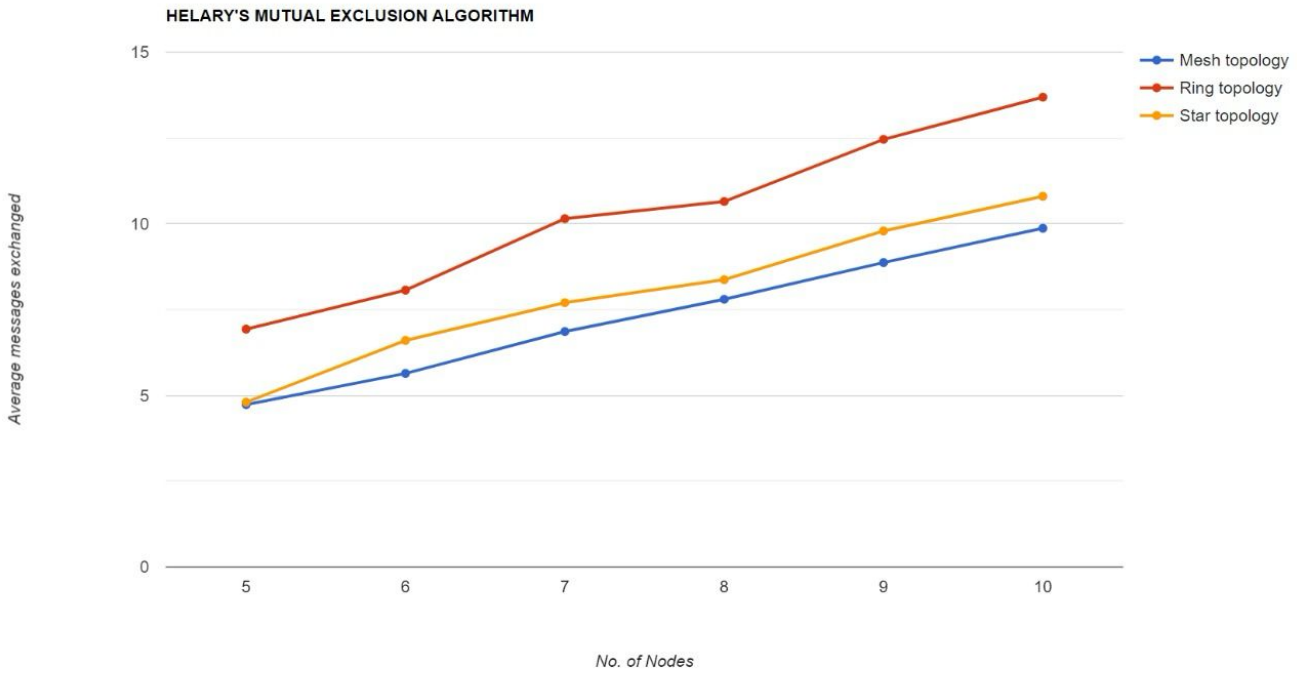
```
procedure transmit_token;
  begin
    --Compute the set X of the processes
      owning a pending request then find
      the oldest and send it the token.
    let X be {(orig,t) such that((orig,t) ∈ req_array_i
      ∧ (lud[orig] < t))};
    if X ≠ { }then
      (elec,x):= min (X);
      let via be such that (elec,x) ∈ req_array_i[via];
      req_array_i[via]:= req_array_i[via] − (elec,x);
      lud[i]:= C_i;C_i:= C_i+1;--Line ref-
        erenced (A) in the proof.
      token_here_i:= false;
      send token(lud,elec) to P_via;
    endif;
  end transmit_token;
```

A distributed algo for ME in an Arbitrary Network

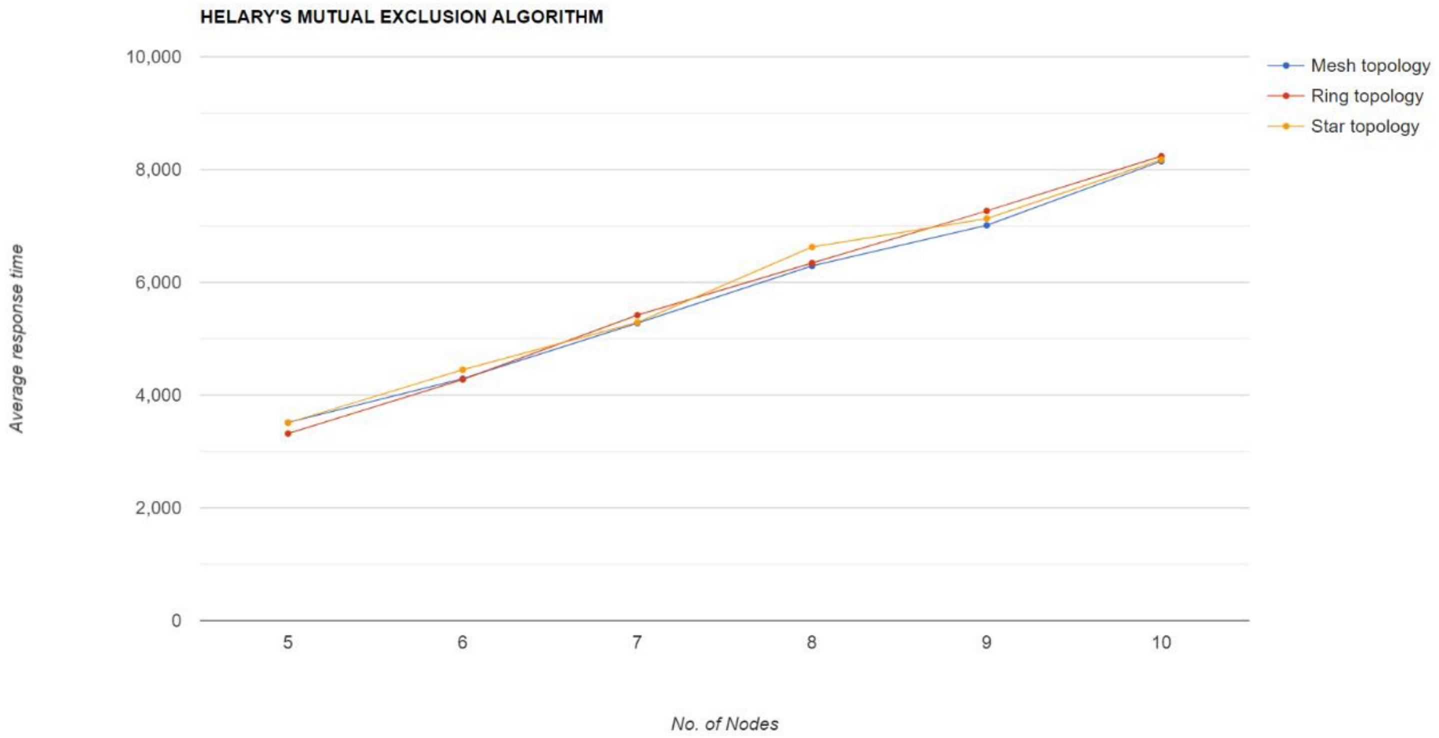
Message Complexity :

- **Broadcasting** a request requires exactly **n — 1** message.
- **Token transmission** requires between **1** (if the sender and the addressee are neighbors) and **d** messages, where the diameter d is here the length of the longest path (1 < d < n — l).
- Thus the total number of messages per CS request: **n — 1 +d**
- n<< n-1+d

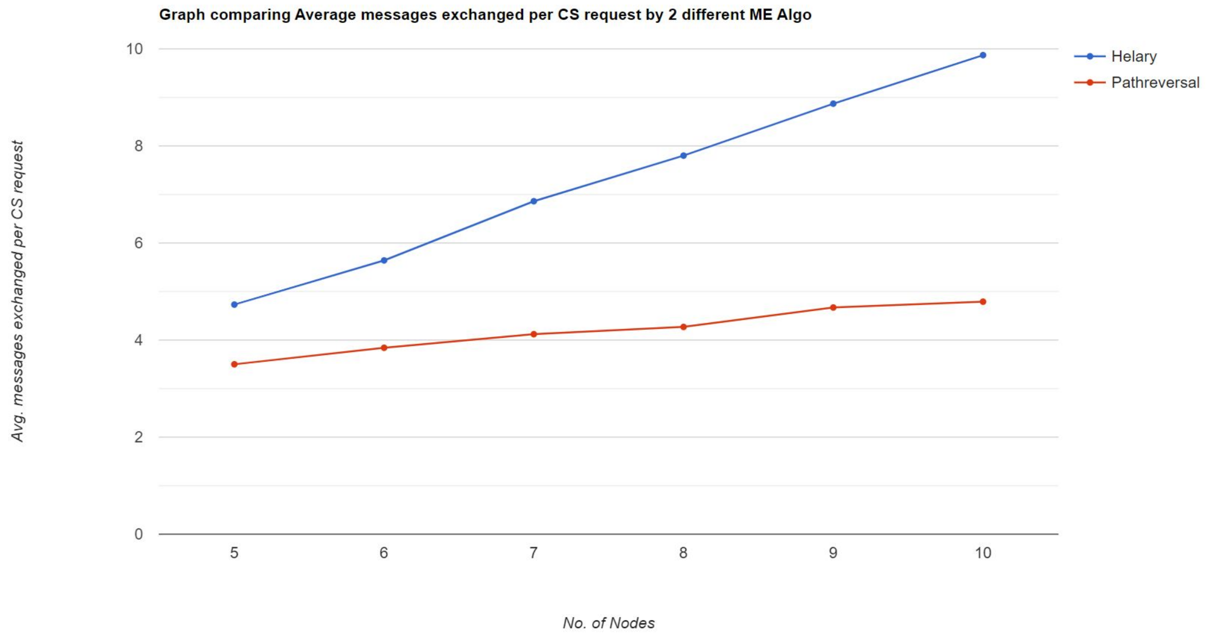
Average messages exchanged for different topologies for Helary's Algorithm



Average Response time for different topologies for Helay's Algorithm

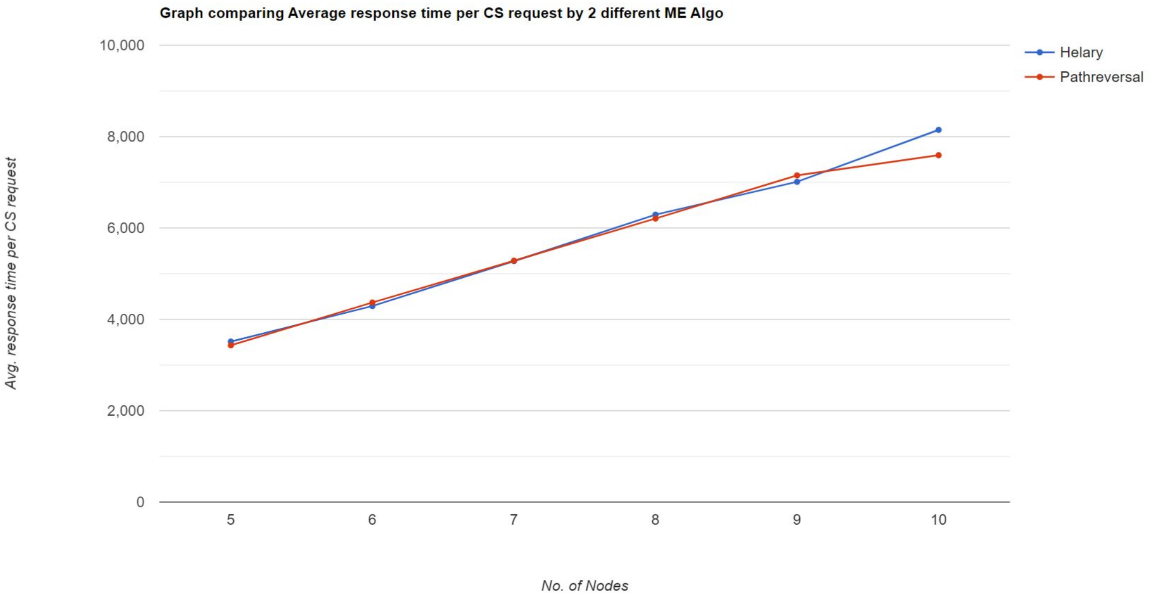


Average messages exchanged per CS request vs Number of nodes



Average response time per CS request VS Number of nodes

Time is in millisecs



Conclusion

1. Pathreversal Mutual exclusion algorithm **exchanges a lesser number of messages** as compared to Helary's Mutual Exclusion algorithm.

   Helary is **O(num_of_nodes + diameter -1) and Pathreversal is O(log (num_of_nodes))** because in Path reversal algorithm, messages are sent along a **logically connected tree** rather than broadcasting the message to every other node in the distributed setup.

2. The **response time** of both the algorithms is approximately the **same**.

References

- A Log (N) Distributed Mutual Exclusion Algorithm Based on Path Reversal by Mohamed Naimi, Michel Trehel & André Arnold
- A Distributed Algorithm for Mutual Exclusion in an Arbitrary Network by J. M. Helary, N. Plouzeau & M. Raynal

Thank You

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad