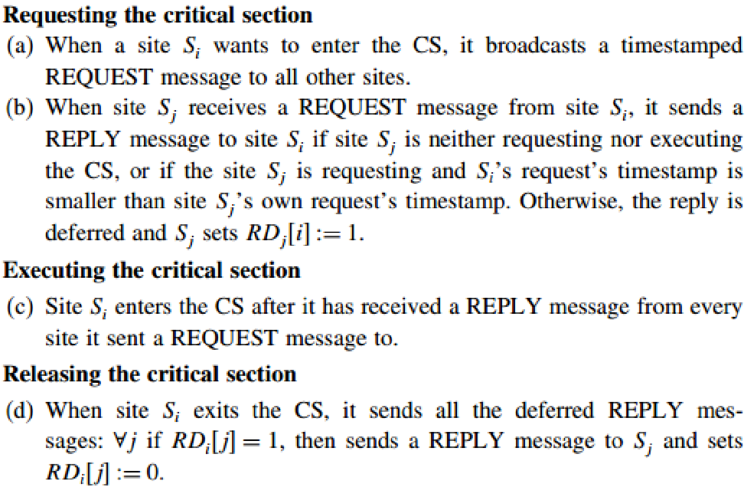
3.3 RICART–AGRAWALA ALGORITHM

- Ricart–Agrawala algorithm is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala.
- This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm.
- It follows permission based approach to ensure mutual exclusion.
- Two type of messages (REQUEST and REPLY) are used and communication channels are assumed to follow FIFO order.
- A site send a REQUEST message to all other site to get their permission to enter critical section.
- A site send a REPLY message to other site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests.
- Smaller timestamp gets high priority over larger timestamp.
- The execution of critical section request is always in the order of their timestamp.



Requesting the critical section

(a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.

(b) When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. Otherwise, the reply is deferred and S_j sets $RD_j[i] := 1$.

Executing the critical section

(c) Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

Releasing the critical section

(d) When site S_i exits the CS, it sends all the deferred REPLY messages: $\forall j$ if $RD_i[j] = 1$, then sends a REPLY message to S_j and sets $RD_i[j] := 0$.

Fig 3.2: Ricart–Agrawala algorithm

To enter Critical section:

- When a site S_i wants to enter the critical section, it send a timestamped REQUEST message to all other sites.
- When a site S_j receives a REQUEST message from site S_i, It sends a REPLY message to site S_i if and only if Site S_j is neither requesting nor currently executing the critical section.
- In case Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request.
- Otherwise the request is deferred by site S_j.

To execute the critical section:

Site S_i enters the critical section if it has received the REPLY message from all other sites.

To release the critical section:

Upon exiting site S_i sends REPLY message to all the deferred requests.

Theorem: Ricart-Agrawala algorithm achieves mutual exclusion.
Proof: Proof is by contradiction.

- Suppose two sites S_i and S_j ‘ are executing the CS concurrently and S_i ’s request has higher priority than the request of S_j . Clearly, Si received S_j ’s request after it has made its own request.
- Thus, S_j can concurrently execute the CS with S_i only if Si returns a REPLY to S_j (in response to S_j ’s request) before Si exits the CS.
- However, this is impossible because S_j ’s request has lower priority. Therefore,Ricart-Agrawala algorithm achieves mutual exclusion.

Message Complexity:

Ricart–Agrawala algorithm requires invocation of 2(N – 1) messages per critical section execution. These 2(N – 1) messages involve:

- (N – 1) request messages
- (N – 1) reply messages

Drawbacks of Ricart–Agrawala algorithm:

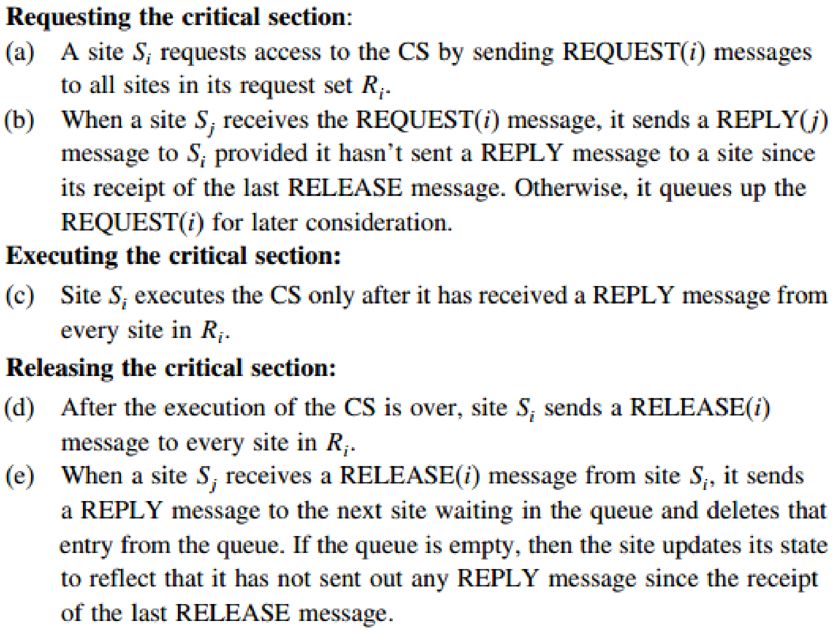
- **Unreliable approach:** failure of any one of node in the system can halt the progress of the system. In this situation, the process will starve forever. The problem of failure of node can be solved by detecting failure after some timeout.

Performance:

Synchronization delay is equal to maximum message transmission time It requires 2(N – 1) messages per Critical section execution.

3.4 MAEKAWA‘s ALGORITHM

- Maekawa’s Algorithm is quorum based approach to ensure mutual exclusion in distributed systems.



Requesting the critical section:

(a) A site S_i requests access to the CS by sending REQUEST(i) messages to all sites in its request set R_i.

(b) When a site S_j receives the REQUEST(i) message, it sends a REPLY(j) message to S_i provided it hasn’t sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration.

Executing the critical section:

(c) Site S_i executes the CS only after it has received a REPLY message from every site in R_i.

Releasing the critical section:

(d) After the execution of the CS is over, site S_i sends a RELEASE(i) message to every site in R_i.

(e) When a site S_j receives a RELEASE(i) message from site S_i, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

Fig 3.3: Maekawa‘s Algorithm

- In permission based algorithms like Lamport's Algorithm, Ricart-Agrawala Algorithm etc. a site request permission from every other site but in quorum based approach, a site does not request permission from every other site but from a subset ofsites which is called quorum.
- Three type of messages (REQUEST, REPLY and RELEASE) are used.
- A site send a REQUEST message to all other site in its request set or quorum to get their permission to enter critical section.
- A site send a REPLY message to requesting site to give its permission to enter the critical section.
- A site send a RELEASE message to all other site in its request set or quorum upon exiting the critical section

The following are the conditions for Maekawa's algorithm:

M1 $(\forall i\ \forall j : i \neq j,\ 1 \leq i, j \leq N :: R_i \cap R_j \neq \phi)$.
M2 $(\forall i : 1 \leq i \leq N :: S_i \in R_i)$.
M3 $(\forall i : 1 \leq i \leq N :: |R_i| = K)$.
M4 Any site S_j is contained in K number of R_is, $1 \leq i, j \leq N$.

Maekawa used the theory of projective planes and showed that N = K(K – 1)+ 1. This relation gives |Ri|= √N.

To enter Critical section:

- When a site S_i wants to enter the critical section, it sends a request message REQUEST(i) to all other sites in the request set R_i.
- When a site S_j receives the request message REQUEST(i) from site S_i, it returns a REPLY message to site S_i if it has not sent a REPLY message to the site from the time it received the last RELEASE message. Otherwise, it queues up the request.

To execute the critical section:

- A site S_i can enter the critical section if it has received the REPLY message from all the site in request set R_i

To release the critical section:

- When a site S_i exits the critical section, it sends RELEASE(i) message to all other sites in request set R_i
- When a site S_j receives the RELEASE(i) message from site S_i, it send REPLY message to the next site waiting in the queue and deletes that entry from the queue
- In case queue is empty, site S_j update its status to show that it has not sent any REPLY message since the receipt of the last RELEASE message.

Correctness

Theorem: Maekawa's algorithm achieves mutual exclusion.

Proof: Proof is by contradiction.

- Suppose two sites Si and Sj are concurrently executing the CS.

- This means site Si received a REPLY message from all sites in Ri and concurrently site Sj was able to receive a REPLY message from all sites in Rj .
- If Ri ∩ Rj = {Sk }, then site Sk must have sent REPLY messages to both Si and Sj concurrently, which is a contradiction

Message Complexity:

Maekawa’s Algorithm requires invocation of 3√N messages per critical section execution as the size of a request set is √N. These 3√N messages involves.

- √N request messages
- √N reply messages
- √N release messages

Drawbacks of Maekawa’s Algorithm:

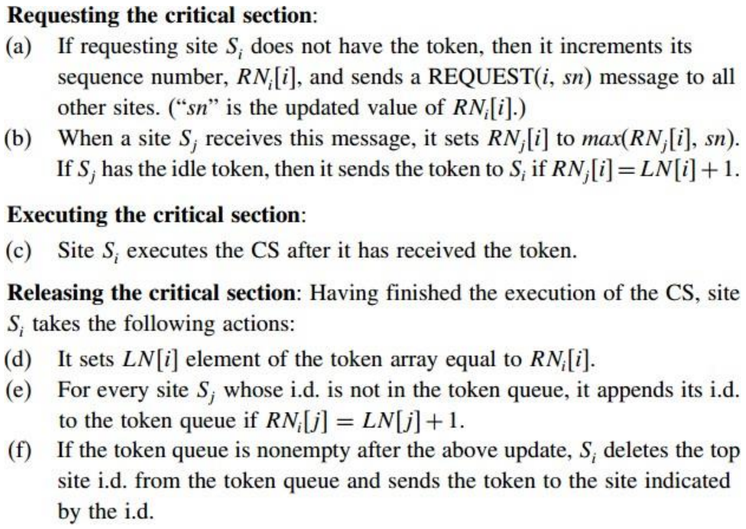
This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

Performance:

Synchronization delay is equal to twice the message propagation delay time. It requires 3√n messages per critical section execution.

3.5 SUZUKI–KASAMI‘s BROADCAST ALGORITHM

- Suzuki–Kasami algorithm is a token-based algorithm for achieving mutual exclusion in distributed systems.
- This is modification of Ricart–Agrawala algorithm, a permission based (Non-token based) algorithm which uses REQUEST and REPLY messages to ensure mutual exclusion.
- In token-based algorithms, A site is allowed to enter its critical section if it possesses the unique token.
- Non-token based algorithms uses timestamp to order requests for the critical section where as sequence number is used in token based algorithms.
- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.



Requesting the critical section:

(a) If requesting site S_i does not have the token, then it increments its sequence number, $RN_i[i]$, and sends a REQUEST(i, sn) message to all other sites. (“sn” is the updated value of $RN_i[i]$.)

(b) When a site S_j receives this message, it sets $RN_j[i]$ to $max(RN_j[i], sn)$. If S_j has the idle token, then it sends the token to S_i if $RN_j[i] = LN[i] + 1$.

Executing the critical section:

(c) Site S_i executes the CS after it has received the token.

Releasing the critical section: Having finished the execution of the CS, site S_i takes the following actions:

(d) It sets $LN[i]$ element of the token array equal to $RN_i[i]$.

(e) For every site S_j whose i.d. is not in the token queue, it appends its i.d. to the token queue if $RN_i[j] = LN[j] + 1$.

(f) If the token queue is nonempty after the above update, S_i deletes the top site i.d. from the token queue and sends the token to the site indicated by the i.d.

Fig 3.4: Suzuki–Kasami‘s broadcast algorithm

To enter Critical section:

- When a site S_i wants to enter the critical section and it does not have the token then it increments its sequence number $RN_i[i]$ and sends a request message REQUEST(i, s_n) to all other sites in order to request the token.
- Here s_n is update value of $RN_i[i]$
- When a site S_j receives the request message REQUEST(i, s_n) from site S_i, it sets $RN_j[i]$ to maximum of $RN_j[i]$ and s_ni.e$RN_j[i] = \max(RN_j[i], s_n)$.
  After updating $RN_j[i]$, Site S_j sends the token to site S_i if it has token and $RN_j[i] = LN[i] + 1$

To execute the critical section:

- Site S_i executes the critical section if it has acquired the token.

To release the critical section:

After finishing the execution Site Si exits the critical section and does following:

- sets $LN[i] = RN_i[i]$ to indicate that its critical section request $RN_i[i]$ has been executed
- For every site S_j, whose ID is not prsent in the token queue Q, it appends its ID to Q if $RN_j[j] = LN[j] + 1$ to indicate that site S_j has an outstanding request.
- After above updation, if the Queue Q is non-empty, it pops a site ID from the Q and sends the token to site indicated by popped ID.
- If the queue Q is empty, it keeps the token

Correctness

Mutual exclusion is guaranteed because there is only one token in the system and a site holds the token during the CS execution.

Theorem: A requesting site enters the CS in finite time.

Proof: Token request messages of a site Si reach other sites in finite time.

Since one of these sites will have token in finite time, site S_i 's request will be placed in the token queue in finite time.

Since there can be at most $N - 1$ requests in front of this request in the token queue, site S_i will get the token and execute the CS in finite time.

Message Complexity:

The algorithm requires 0 message invocation if the site already holds the idle token at the time of critical section request or maximum of N message per critical section execution. This N messages involves

- (N – 1) request messages
- 1 reply message

Drawbacks of Suzuki–Kasami Algorithm:

- Non-symmetric Algorithm: A site retains the token even if it does not have requested for critical section.

Performance:

Synchronization delay is 0 and no message is needed if the site holds the idle token at the time of its request. In case site does not holds the idle token, the maximum synchronization delay is equal to maximum message transmission time and a maximum of N message is required per critical section invocation.