*For any events a,b: if a      b, then C(a) < C(b)*

*a*

*b*

*a* *b*

```
If(I_AM_SERVER && timestamp == start_time)
{
     pingAllMembers( );
     setMyActivity( ACTIVE );
}
```

*Figure A: Server module to request membership verification from all other nodes.*

```
If(I_AM_SERVER && timestamp == (stop_time – 1))
{
     determineNextServer( );
}
```

*Figure B: Server module to determine which node will be the next server for the system.*

```
           }
}
else if(received_msg == system_state)
{
     if(received_msg.key() == request)
     {
           if(resource_available)
           {
                lockForRequestingNode( resource );
                incrementTimestamp( );
                send( RESPONSE );/* to entire group */
           }
     }
     else if(received_msg.key() == release)
     {
                unlockResource ( resource );
     }
     else if(received_msg.key() == ping)
                memberActivity[sender.id] = ACTIVE;
     else {
                /* system state update – i.e.
                   any type of personal information
                   about a node that it needs to
                   notify other members about
                */
           }
}
```

*Figure C: Server module to process incoming messages.*

```
                  removeLeavingMember( );
                  calculateMyTurnAsServer( );
                  if(no_longer_my_turn)
                        I_AM_SERVER = false;
            }
}
else if(received_msg == system_state)
{
      if(received_msg.key() == response)
      {
            processRequestResponse( );
      }
      else if(received_msg.key() == release)
      {
            unlockResource ( resource );
      }
      else if(received_msg.key() == ping)
```

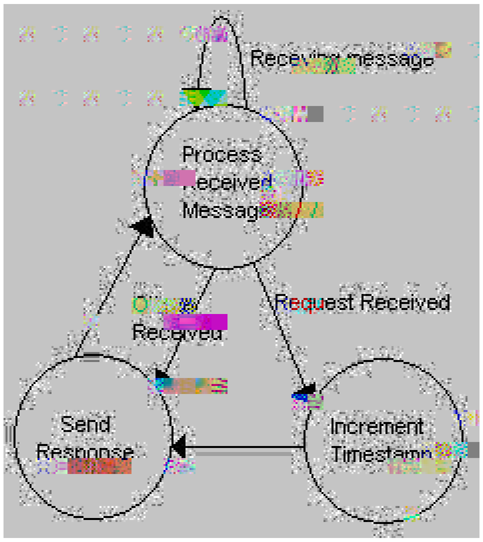

*Figure E: State diagram showing when the server node increments the timestamp value.*

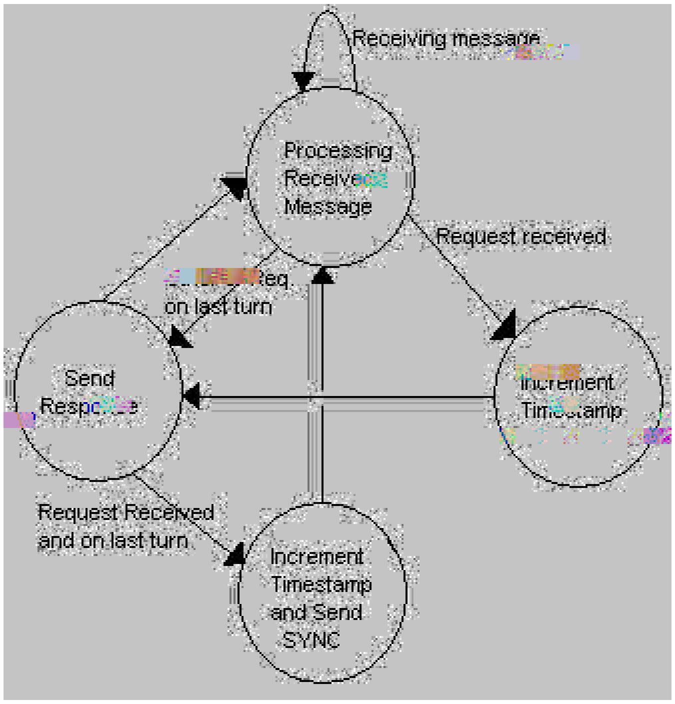

*Figure F: State diagram showing the addition of the SYNC message to server module*

*The proposed algorithm achieves mutual exclusion.*