A Non-blocking Checkpointing Algorithm for Non-Deterministic Mobile Ad hoc Networks

Kusum Saluja*, Praveen Kumar**
**NIMS University, Jaipur (Rajasthan India*
***Meerut Institute of Engineering & Technology, Meerut (INDIA)-250005*

Abstract— **Mobile Ad-hoc Networks are a collection of two or more devices equipped with wireless communication and networking capability. These devices can communication with other nodes that immediately within their radio range or one that is outside their radio range. The transient failure probability of the computing process increases greatly with the enlarging of system scale. If a failure occurs in a process and there is not an appropriate method to protect it, more cost will be wasted for restarting the program. Coordinated checkpointing can be used to introduce fault tolerance in mobile ad-hoc wireless networks environment. In this paper we propose a new minimum process checkpointing scheme for ad-hoc networks. We assume that Cluster Based Routing Protocol (CBRP) is used which belongs to the class of Hierarchical Reactive Routing Protocols. The number of coordinated messages between a cluster head and its ordinary members is small. The recovery scheme has no domino effect and the failure process can rollback from its latest local consistent Checkpoint. We capture the transitive dependencies among processes by piggybacking dependency vector of the sending process along with the computation messages.**

Keywords— **Mobile Ad Hoc Network; Checkpointing; Fault tolerance; Coordinating Checkpointing.**

1. INTRODUCTION

A. Preliminaries

Wireless networks include infrastructure-based networks and ad hoc networks. Most wireless infrastructure-based networks are established by a one hop radio connection to a wired network. On the other hand, mobile ad hoc networks are decentralized networks that develop through self-organization [1]. The original idea of MANET started out in the early 1970s. At this time they were known as packet radio networks. Lately, substantial progress has been made in technologies like microelectronics, wireless signal processing, distributed computing and VLSI (Very Large Scale Integration) circuit design and manufacturing [2]. This has given the possibility to put together node and network devices in order to create wireless communications with ad hoc capability.

MANETs are formed by a group of nodes that can transmit and receive data and also relay data among themselves. Communication between nodes is made over wireless links. A pair of nodes can establish a wireless link among themselves only if they are within transmission range of each other. An important feature of ad hoc networks is that routes between two hosts may consist of hops through other hosts in the network [3]. When a sender node wants to communicate with a receiver node, it may happen that they are not within communication range of each other. However, they might have the chance to communicate if other hosts that lie in-between are willing to forward packets for them. This characteristic of MANET is known as multihopping. An example is shown in figure 1. Node A can communicate directly (single-hop) with node B, node C and node D. If A wants to communicate with node E, node C must serve as an intermediate node for communication between them. Therefore, the communication between nodes A and E is multi-hop.

The infrastructured networks have fixed and wired gateways or the fixed Base-Stations which are connected to other Base-Stations through wires. Each node is within the range of a Base-Station. A 'Hand-off' occurs as mobile host travels out range of on Base-Stations and into range of another and thus, mobile is able to continue communication seamlessly through out the network. Example applications of this type include wireless local area networks and Mobile Phone.

Mobile Ad-hoc Networks are supposed to be used for disaster recovery, battle field Communications, and rescue operations when the wired network is not available. I t can be provided a feasible means for ground communications, and information access.

The topology of the ad hoc network is represented by an undirected graphic $G= (V, E)$, where V is the set of all the mobile nodes, E is the set of all the mobile links. If edge $(u, v) \in E$, then edge $(u, v), \in E$. Node u and v belong to the communication range of each other, and they are 1-hop neighbors. The set of node i's 1-hop neighbor is denoted $N^1_{i.}$ If two nodes share the same 1-hop neighbor, and the shortest path between them is 2 hops, then the two nodes are each other's 2-hop neighbors. The set of node i's 2-hop neighbors is denoted $N^2_{i.}$ If the shortest path between two nodes is 3 hops, then they are each other's 3-hop neighbors. The set of node i's 3-hop neighbors is denoted $N^3_{i.}$ All the nodes use omni directional antennae, and have the same transmission ranges.

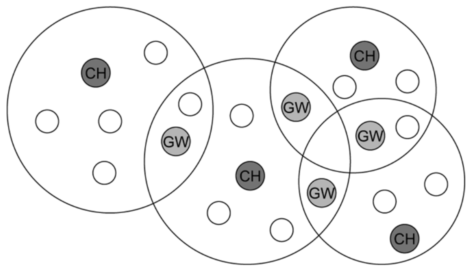

Figure 1:-A topology example after clustering

Each node may work as one of the four following roles: cluster- head, gateway, compound gateway and cluster-member. As shown in figure 1, there are four clusters; each cluster is having one cluster head (denoted by CH). Gateway Nodes (denoted by GW) interconnect the cluster head nodes.
First of all, since the nodes in Wireless Ad-hoc Network are free to move arbitrarily at any time .So the networks topology of MANET may change randomly and rapidly at unpredictable times. This makes routing difficult because the topology is constantly changing and nodes cannot be assumed to have persistent data storage. In the worst case, we don not even know whether the node will still remain next minute, because the node will leave the network at any minute.

Bandwidth constrained is also a big challenge. Wireless links have significantly lower capacity than their hardwired counterparts. Also, due to multiple access, fading, noise, and interference conditions etc. the wireless links have low throughput.

Some or all of the nodes in MANET may rely on batteries. In this scenario, the most important system design criteria for optimization may be energy conservation.

Mobile networks are generally more prone to physical security threats than are fixed cable networks. There are increased possibility odeavesdropping, spoofing and denial-of-service attacks in these networks.

In each cluster, it has a unique leader, called a cluster head, to enforce channel allocation. A cluster head is a local manager of all mobile hosts within a cluster. In the same cluster, the mobile host called clusters members that controlled by the cluster-head. One of the basic functions for a cluster head is broadcasting beacon packets to all mobile hosts in the cluster.

Local checkpoint is the saved state of a process at a processor at a given instance. Global checkpoint is a collection of local checkpoints, one from each process. A global state is said to be “consistent” if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost. Initial global state is always consistent, because, it cannot contain any orphan message. A transit message is a message whose send event has been recorded by the sending process but whose receive event has not been recorded by the receiving process. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last check pointed state and only the computation done thereafter needs to be redone. Processes in a distributed system communicate by sending and receiving messages.

B. Contribution of the Paper

In this chapter, we devise a minimum process non-blocking checkpointing algorithm for mobile Ad hoc Networks. There is no common clock, shared memory or central coordinator. Message passing is the only mode of communication between any pair of processes. Messages are exchanged with finite but arbitrary delays. In our algorithm, we consider that the processes which are running in the distributed mobile ad hoc network systems are non-deterministic. The algorithm is distributed in nature. There is no centralized controlling node. To avoid any waste of bandwidth or CPU consumption, the algorithm is loop free. We assume that Cluster Based Routing Protocol (CBRP) is used which belongs to the class of Hierarchical Reactive Routing Protocols. Clustering Routing Strategy is highly employed in Ad hoc Networks to surpass scalability problem. By limiting the network view of each node, clustering reduces the routing complexity and the size of the routing table. The local movement of nodes is handled only within the cluster without affecting other parts of the network and so the overhead is highly reduced. Our system model consists of a number of MHs which communicate through Cluster Heads (CHs). Each CH provides wireless communication support for a fixed geographical area, called a cluster. CHs are linked together over the Wireless data networks through Gateway Nodes. We assume that wireless channels and logical channels are all FIFO order. If a MH moves to the cell of another CH, a wireless channel to the old CH is disconnected and a wireless channel in the new cluster is allocated. There is no common clock, shared memory or central coordinator. Message passing is the only mode of communication between any pair of processes. Any process can initiate checkpointing. It is assumed that processes may be failed during processing but there is no communication link failure. Messages are exchanged with finite but arbitrary delays. In our algorithm, we consider that the processes which are running in the mobile Ad hoc Network are non-deterministic.

2. THE PROPOSED CHECKPOINTING ALGORITHM

A. *Informal Discussion of the Proposed Algorithm with an Example*

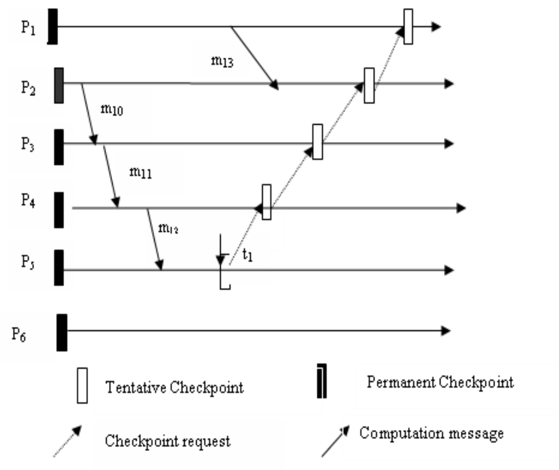

Figure 2

In figure 2, at time t_1, suppose process P_5 initiates checkpointing process. It should be noted that our proposed algorithm is distributed in nature and any process can initiate checkpointing. If two processes concurrently initiate checkpointing, the checkpoint initiation of the process with lower process_ID will prevail. In this way, concurrent initiations will not lead to concurrent executions of the proposed protocol. If we use the technique to capture the transitive dependency by direct dependencies proposed by Cao Singhal [2] and other similar algorithms; the following scenario will take place. P_5 sends the checkpointing request to P_4 due to m_{12}. On receiving checkpointing request P_4 takes its tentative checkpoint and sends the checkpointing request to P_3 due to m_{11}. Similarly, after taking its tentative checkpoint, P_3 sends the checkpointing request to P_2 due to m_{10}. In this way, checkpointing tree of height three is generated and the checkpointing time may be exceedingly high in ad hoc networks.

In the proposed scheme, every process maintains a dependency vector (say DV[]) of length n where n is the number of processes in the ad hoc network. $DV_i[j]=1$ implies P_i is causally dependent upon P_j. $DV_i[j]$ is set to '1' only if P_i processes m received from P_j such that P_j has not taken any permanent checkpoint
after sending m.

In our algorithm, dependency vectors are maintained as follows. Let the initial dependency vectors of P_1, P_2, P_3, P_4, P_5, P_6 are DV_1 [000001], DV_2 [000010], DV_3 [000100], DV_4 [001000], DV_5[010000], DV_6[100000], respectively. In figure 2, P_2 sends m_{10} to P_3 along with its dependency vector DV_2[000010]. When P_3 receives m_{10}, it appends its dependency vector DV_3 by taking the bitwise logical OR of DV_2[000010] and DV_3 [000100], which comes out to be [000110]. Similarly, P_3 sends m_{11} to P_4 along with its own dependency vector DV_3[000110].

After receiving m_{11} by P_4, DV_4 becomes [001110]. At time t_1, P_5 initiates checkpointing process with the DV_5 [011110], and sends the checkpointing request to P_2, P_3, P_4. In this way no checkpointing tree is formed as found in Cao-Singhal algorithm as detailed above. In this way, the time to collect the global state will be significantly low as compared to Cao-Singhal algorithm. Therefore, the time to collect the global checkpoint will be less and the number of useless checkpoints will also be reduced considerably. The original idea of capturing the transitive dependencies during normal processing was proposed by Prakash-singhal [5].

In figure 2, when P_2 takes its tentative checkpoint C_{21} and finds that P_1 is in the dependency set of P_2, but is not available in the minimum set {P_2, P_3, P_4, P_5} received from P_5. In this case, if P_1 does not take its checkpoint in the current initiation, m_{13} will become orphan. Therefore, P_2 sends checkpoint request to P_1 and P_1 takes its tentative checkpoint C_{11}. In this way, we get [C_{11}, C_{21}, C_{31}, C_{41}, C_{51}, C_{60}] as the consistent global state.

In figure 3, P_0 takes its tentative checkpoint and sends m_{11} to P_1. P_1 has neither taken its tentative checkpoint nor received any checkpointing request from any other process. By the piggybacked information along with m_{11} and certain other data structures, P_1 concludes that P_0 has taken its tentative checkpoint for some new initiation. In this case if P_1 takes its checkpoint after processing m_{11}, m_{11} will become orphan. Therefore, we propose that P_1 will take a forced checkpoint (say induced Checkpoint) before processing m. If P_1 does not receive any checkpointing request during the current initiation, P_1 will discard it on commit. In this case, if we find that P_1 has not sent any message to any process since its last committed checkpoint, then P_1 will process m without taking its induced checkpoint. Because, we can say that P_1 will not be included in the minimum set in this case.

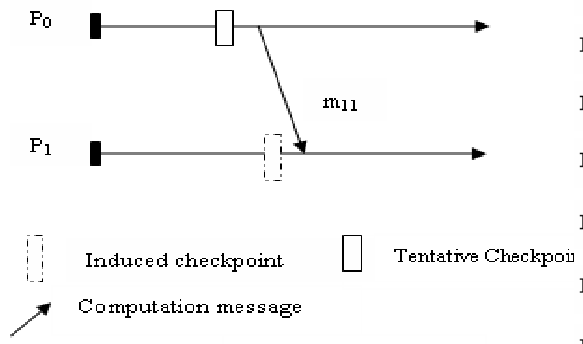

Figure 3

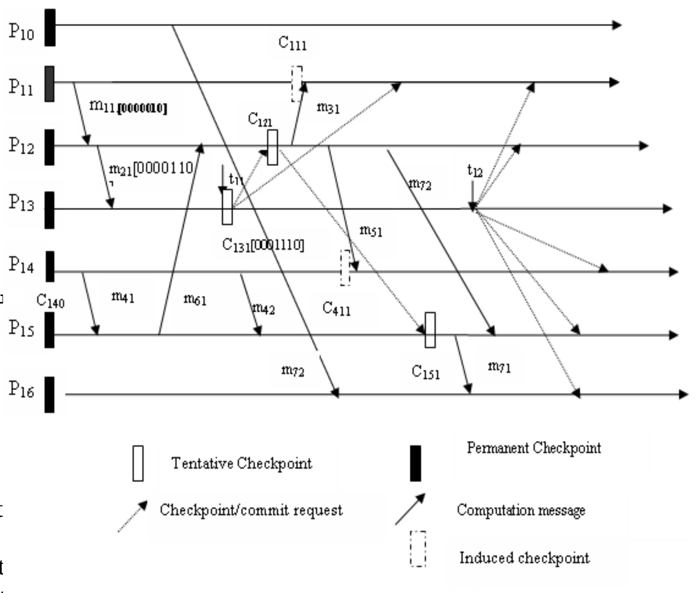

Figure 4

As shown in figure 4, at time t_{11}, P_{13} initiates checkpoint processfor the mobile ad hoc network under consideration. captures minimum set $\{P_{11}, P_{12}, P_{13}\}$. It takes its own tentat checkpoint C_{131} and sends the checkpointing request to P_{11} P_{12} [it should be noted that R_3 at P_{13} at t_{11} is [0001110]. On receiving the checkpointing request from P_{13}, P_{12} takes its tentative checkpoint C_{121}. At the time of taking its tentative checkpoint, P_{12} finds R_2=[0100110]. It means P_{12} is dependent on P_{15} due to m_{61} and P_{15} is not included in the minimum set $\{P_{11}, P_{12}, P_{13}\}$ computed so far. Therefore, P_{12} sends checkpointing request to P_{15} which in turn takes its tentative checkpoint C_{151}. After taking its tentative checkpoint, P_{12} sends message m_{31} to P_{11}. P_{11} has not received the checkpointing request so far. As csn of P_{12} is greater than expected one at the time of sending m_{31}, P_{11} takes its induced checkpoint before processing m_{31}. When P_{11} actually gets the tentative checkpoint request, it converts its induced checkpoint C_{111} into tentative one. After taking its tentative checkpoint, P_{12} sends message m_{51} to P_{14}. Obviously, P_{14} takes its induced checkpoint before processing m_{51}. All the concerned processes after taking their tentative checkpoints, inform the initiator [not shown in figure], and finally initiator P_{13}, sends the commit request at time t_{12} to all the processes. On commit, P_{14} finds that it has not received the formal tentative checkpointing request from any process. Therefore, P_{14} discards its induced checkpoint. In this way the resultant consistent state is [C_{100}, C_{111}, C_{121}, C_{131}, C_{140}, C_{151}, C_{160}]. After taking its tentative checkpoint, P_{15} sends m_{71} to P_{16}; P_{16} does not take induced checkpoint before processing m_{71} because P_{16} has not sent any message to any process since its last committed checkpoint.

B. *Data Structures*

We have used the following data structures in our checkpointing protocol.

The following section describes the notations and data structures used in our algorithm. In our algorithm, any process can initiate the checkpointing operation. Data structures are initialized/ updated on the completion of a checkpointing process. We assume that there are n processes running in the system.

- **new_csn$_i$** checkpoint sequence number of process P_i and is incremented when P_i takes a tentative checkpoint; otherwise, it shown the csn of the last committed checkpoint.
- **DV $_i$ []** an array of n bits for a process. $DV_i[j]$ becomes ‘1’ when P_i receives a message from P_j in the current checkpointing interval. In the beginning of every checkpointing interval, this vector is to zero for all processes except for itself which is intialized as ‘1’. Maintenance of $DV_i[]$ is shown in basic idea.
- **ch_state$_i$** A boolean which is set to ‘1’ when P_i takes a tentative checkpoint; otherwise is zero on receiving abort or commit request from the initiator process.
- **Mess_send_flag[i]** A bit vector of size n for n processes. **Mess_send_flag** $_i$[j]=1 if P_i sends m to P_j
- **set_dp** An array of size n used to save minimum set of processes on which initiator process is transitively depends on. Initially, when

checkpointing operation is started, set_dp is $DV_i[]$ of the initiator process..

- **Chk_set[]** An array of size n to save information about the processes which have taken their tentative checkpoints. When process P_j takes its tentative checkpoint then j^{th} bit of this vector is set to 1.
- **Timeout_flag** a flag used to provide timing in checkpointing operation. It is initialized to zero when timer is set and becomes '1' when maximum allowable time for collecting coordinating checkpoints is expired.
- **p_CH** An array of size n used to save the information on every CH regarding the processes which are running in its cell. p_ch[k] = 1 indicate that process P_k is running in the cell of this CH. Information about disconnected MH, if any, which are supported by this CH, is also stored in this array.
- **tent_chk_set** An array of n bits maintained by the CH. Tent_chk_set [j]=1 whenever process P_j which is in the cell of CH has taken tentative checkpoint.
- **chk_request[]** An array of n bits maintained also on every CH. The j^{th} bit of this array is set to 1 whenever initiator sends the checkpoint request to P_j and P_j is in the cell of this CH.
- **error_flag** A flag maintained on every CH, initialized to '0' and set to '1' when any process in the cell of CH fails to take tentative checkpoint
- **P_{in}** The process which has initiated the checkpointing operation
- **CH_{in}** The CH which has P_{in} in its cell
- **new_csn_{in}** checkpoint sequence number of initiator process
- **g_chkpt** A flag which indicates that some global checkpoint is being saved
- **comm_csn_array []** An array of size n, maintained on every CH, for n processes. comm_csn_array[i] represens the most recently committed checkpoint sequence number of P_i. After the commit operation, if set_dp[i]=1 then comm_csn_array[i] is incremented. It should be noted that entries in this array are updated only after converting tentative checkpoints in to permanent checkpoints and not after taking tentative checkpoints.
- **set_dp1[]** An array of size n maintained on every CH. It contains those new processes which are found on getting checkpoint request from initiator.
- **set_dp2[]** An array of size n. for all j such that set_dp1[j] $\neq$ o, set_dp2= set_dp2 $\cup$ set_dp1.
- **set_dp3[]** An array of length n; on receiving set_dp3, set_dp, set_dp1 along with checkpoint request [c_req] or on the computation of set_dp1 locally: set_dp3=set_dp3$\cup$ c_req.set_dp3; set_dp3=set_dp3$\cup$set_dp; set_dp3=set_dp3 $\cup$c_req.set_dp1; set_dp3=set_dp3 $\cup$ set_dp1; set_dp3 maintains the best local knowledge of the minimum set at an CH;

C. The Checkpointing Protocol

In a distributed mobile adhoc network system, due to less bandwidth of wireless channels and vulnerability of storage of MH, all the information regarding the checkpointing are stored in the stable storage of the MH itself. In the proposed protocol, when an MH sends an application message, it is first sent to its local CH over the wireless channel. The CH then attaches the dependency vector of the process with the message and sends it to the CH for which it was issued. The destination CH strips this dependency vector from the application message and transmit it to the destination MH over the wireless channels. The destination CH updates the dependency vector of destination MH (maintenance of dependency vector is explained in basic idea). In this way, no data structures are allowed to travel over the wireless channels. It should be noted that a dependency vector of mobile hosts are maintained at CHs.

We propose that any process in the system can initiate the checkpointing operation. When a process (say P_i) want to initiate checkpointing, it takes its tentative checkpoint and send the request to its local CH (initiator CH). This local CH coordinates the checkpointing operation on behalf of the initiator MH. If two processes initiate checkpointing at the same time then the checkpointing initiation of the lower id will prevail. CH_{in} [initiator CH] sends the checkpointing request to all CHs alongwith set_dp { set_dp[ ] = $DV_i[]$ }. It should be noted that the dependency vector of P_i i.e. $DV_i[]$ contains all the processes on which it is directly or transitively dependent. set_dp is a tentative minimum set computed from DVi[] of the initiator process. When an CH receives the checkpointing request, it sends checkpointing request to P_j if $P_j \in$ set_dp[] and P_j is in its cell and stores such processes in $chk\_request_j[\ ]$. We take the following action with every process, say P_j, which is required to take its tentative checkpoint. If there exists any process P_k such that P_k does not belong to set_dp [] and P_k belongs to $DV_j[]$, then P_j sends checkpoint request to P_k. During checkpointing process, if a process P_j receives the message m from P_i, it takes the following actions:

If P_i has taken its tentative checkpoint before sending m and P_j has not taken its tentative checkpoint at the time of receiving m, in this case, P_j will take its induced checkpoint before receiving m. It should be noted that if P_j takes its tentative checkpoint after receiving m, m will become orphan and resulting consistent global state will be inconsistent.

For a disconnected MH that is a member of minimum set, the CH that has its disconnected checkpoint, converts its disconnected checkpoint into tentative one. When a CH learns that its concerned processes in its cell have taken their tentative checkpoints, it sends the response to CH_{in}. On receiving positive response from all concerned CHs, the CH_{in} issues the commit request to all CHs. On commit when a process learns that it has taken an induced checkpoint and has not received the formal tentative checkpointing request from any process, it discards its induced checkpoint.

D. Formal Outline of the checkpointing Algorithm:

1) Actions taken when P_i sends m to P_j:

send (m, new_csn_i, ch_state_i);

2) Algorithm executed at initiator CH (say CH_{in})

Suppose P_{in} initiates checkpointing. P_{in} sends the request to CH_{in}. CH_{in} computes set_dp..

```
i.   On the basis of computed set_dp, CH_in computes set_dp1, set_dp2, set_dp3.
ii.  Set_dp = set_dp3.
iii. CH_in sends c_req to all CHs alongwith set_dp.
iv.  Set timeout_flag.
v.   Wait for response.
vi.  On receiving response (P_in, CH_in, CH_s,tent_chk_set,set_dp2,error_flag) or at timer out
       (i)   If (timeout_flag) ∪ (error_flag) { send message abort (P_in,CH_in,new_csn_in} to all CH_s, Exit;
             [“∪” is a set union operator]
       (ii)  Set_dp = set_dp ∪ set_dp2.
       (iii) Chk_set[] = Chk_set[] ∪ tent_chk_set[]
vii. For (k=0;k<n; k++)
       If ( ∃ k such that Chk_set [k] ≠ set_dp[k]) then go to step 5;
viii. Send message commit (P_in, CH_in,new_csn_in, set_dp) to all CH_s;
       // set_dp is the exact minimum set//
```

3) Algorithm Executed at a process P_j on receiving of m from P_i:

```
If (m.new_csn_i = =comm_csn_array[i])
{   rec(m);
    DV_j[i]=1};
If (m.new_csn_i<comm_csn_array [i]; rec (m));
If m.new_csn_i>comm_csn_array [i])
    {if (new_csn_j>comm_csn_array [j];
    {rec (m); DV_j[i]=1}
        Else
         if (new_csn_j=comm_csn_array [j] ∧ ∀i sendj [i] ≠ 0 )
           {P_j takes induced checkpoint; ++new_csn_j; c-state=1;
           rec(m); DV_j[i]=1}
           Else // if (own-csn_j =comm_csn_array ∧ ∀ i (mess_send_flagi]= 0//
         {rec(m); DV_j[i]=1;}
```

i. Algorithm executed at any CH (say CHs)

```
1. Wait for Response
2. Upon receiving message c_req (P_in, CH_in, new_csn_i, set_dp) from CH_in
   (i)   For any P_i such that p_CH[i]=1 ∧ set_dp[i]=1; send c_req to P_i
   (ii)  ++new_csn_i; chk_request[i]=1, ch-state_i=1
   (iii) Compute set_dp1, set_dp2, set_dp3
   (iv)  If ∃ i such that set_dp1[i]=1;
           send c_req to P_i.  //set_dp1 contains the new processes found for the minimum set//
3. On receiving c_req from some other CH say CH_p
   ∀i such that set_dp3[i] = = 1 ∧ p_CH[i]= =1 ∧ E[i]==0
   { send c_req to P_i; compute set_dp1, set_dp2, set_dp3}
   If ∃ i such that set_dp1[i]=1;
   send c_req to P_i;
   ∀i set_dp1[i]=0;
4. On receiving response to checkpointing from P_j
```

```
        (i)   If ($P_j$ has taken the tentative checkpoint
                  successfully the tent_chk_set[j]=1
                  else set error_flag.)
        (ii)  If (error_flag) $\vee$ ( $\forall$ j
                  tent_chk_set[j]=chk_request[j];
                  Send response ($P_{in}$, $CH_{in}$,
                  chk_request, error_flag, set_dp2)
                  to $CH_{in}$;
5.      On receiving commit().
        (i)   Convert the tentative checkpoints in to
                  permanent ones and discard old
                  permanent checkpoints.
        (ii)  Discard induced checkpoints if any.
        (iii) $\forall$ j such that
                  set_dp[j]=1,comm_csn_array[j]+
                  +;
        (iv)  Initialize relevant data structures.
6.      On receiving abort().
        Discard the tentative checkpoints and
        induced checkpoints, if any.
        Update relevant variables.
4.      Algorithm executed at any process $P_i$;

On receiving tentative checkpoint request.
Take tentative checkpoint and inform local CH.
```

3. A PERFORMANCE EVALUATION

A. General Comparison with existing non-blocking minimum process algorithms:

In [7],[4], initiator process/CH collects dependency vectors for all the processes and computes the minimum set and sends the checkpointing request to all the processes with minimum set. The algorithm is non-blocking; the message received during checkpointing may add processes to the minimum set. It suffers from additional message overhead of sending request to all processes to send their dependency vectors and all processes send dependency vectors to the initiator process. But in our algorithm, no such overhead is imposed. The Cao-Singhal [2] suffers from the formation of checkpointing tree as shown in basic idea. In our algorithm, theoretically, we can say that the length of the checkpointing tree will be considerably low as compared to algorithm [2], as most of the transitive dependencies are captured during the normal processing. We do not compare our algorithm with Prakash-Singhal [2], as Cao-Singhal proved that there no such algorithm exists [1].
Average number of useless checkpoints in the proposed algorithm will be significantly less as compared to [2] algorithm in many situations. As in [2] algorithm, a checkpointing tree is formed, therefore, the time to collect the global state in [2] will be higher than the proposed one. Excessive checkpointing time may trigger many mutable checkpoints which may lead to higher number of useless checkpoints as compared to our algorithm. Furthermore, in [2] algorithm, transitive dependencies are captured by direct dependencies. Hence the average number of useless checkpoints requests will be significantly higher than the proposed algorithm. In [2], huge data structure are piggybacked along with checkpointing request, because they are unable to maintain exact dependencies among processes. Incorrect dependencies are solved by these huge data structures. In our case, no such data structures are piggybacked on checkpointing request and no such useless checkpoint requests are sent., because we are able to maintain exact dependencies among processes and furthermore, are able to capture transitive dependencies during normal computation at the cost of piggybacking bit vector of length n for n processes.

B. Performance of the proposed algorithm

The average blocking time in our algorithm is nil. The average number of checkpoints comes out to be $N_{min} + N_{useless.}$

Average message overhead:

A process taking a tentative checkpoint needs two system messages; request and reply. A process may receive more than one request for the same checkpoint initiation from different processes. However, we have used techniques to reduce the occurrence of this kind of situation. Therefore, the system message overhead is approximately 2 N_{min}* C_{pp}. in the second phase, CH_{in} broadcasts commit request in C_{bst} time. Hence the total message overhead will be 2* N_{min}* C_{pp}+ C_{bst}.

4. CONCLUSION

We have proposed a minimum process coordinated checkpointing algorithm for mobile ad hoc networks, where no blocking of processes takes place. We try to reduce the number of useless checkpoints by avoiding checkpointing tree which may be formed in Cao-Singhal [2] algorithm. We captured the transitive dependencies during the normal execution. The Z-dependencies are well taken care of in this protocol. We also avoided collecting dependency vectors of all processes to find the minimum set as in [4], [7]. In this way, we reduced the message complexity to a significant extent, as compared to these algorithms. Thus the proposed protocol is simultaneously able to attain the zero blocking time and to reduce the useless checkpoints to bare minimum, by maintaining exact dependencies among processes and piggybacking checkpointing sequence number and dependency vector on to the normal messages.

REFERENCES

[1] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.

[2] Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.

[3] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.

[4] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" Proceedings of IEEE ICPWC-2005, pp 491-95, January 2005.

[5] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.

[6] J.L. Kim, T. Park, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[7] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[8] Murthy & Manoj, "Ad hoc Wireless Networks Architectures and Protocols", Pearson Education, 2004.

[9] D.J. Baker and A. Ephremides, "The Architectural Organisation of a Mobile Radio Network via a Distributed algorithm", IEEE Trans. Commun., vol. 29, no. 11, pp 1694-1701, Nov., 1981

[10] D.J. Baker, A. Ephremides and J.A. Flynn "The design and Simulation of a Mobile Radio Network with Distributed Control", IEEE J. sel. Areas Commun.., pp 226-237, 1984

[11] B.Das, R. Sivakumar and V. Bharghavan, "Routing in Ad-hoc networks using a Spine",Proc. Sixth International Conference, 1997.

[12] B.Das, R. Sivakumar and V. Bharghavan, "Routing in Ad-hoc networks using Minimum connected Dominating Sets",Proc. IEEE International Conference, 1997.

[13] M.Gerla, G. Pei, and S.J. Lee, "Wireless Mobile Ad-hoc Network Routing", Proc. IEEE/ACM FOCUS'99, 1999.

[14] M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.