International Journal of Enhanced Research in Management & Computer Applications, ISSN: 2319-7471
Vol. 2 Issue 6, June-2013, pp: (14-23), Available online at: www.erpublications.com

Asynchronous Rollback Recovery in Cluster Based Multi Hop Mobile Ad Hoc Networks

Awadhesh Kumar Singh[a], Parmeet Kaur Jaggi[b]

[a]Dept. of Computer Engineering, National Institute of Technology, Kurukshetra, Haryana, India
[b]Dept. of Computer Science, Jaypee Institute of Information Technology, NOIDA, UP, India

Abstract: The dynamic topology and constraints of limited resources in mobile ad hoc networks (MANETs) pose unique challenges to execution of processes as compared to traditional distributed systems. The failure probability of the computing process increases when the system scales up. The paper presents an algorithm for the asynchronous recovery of a mobile host in cluster based and multi hop MANETs using movement based checkpointing. Though checkpointing is well explored for crash recovery of applications in the distributed systems, yet it is not trivially applicable to MANETs. The paper addresses problems specific to MANETs such as limited availability of stable storage, movement of nodes and network partitioning due to lost links. The algorithm imposes a self stabilizing spanning tree over the network topology to reduce the number of recovery messages. The findings of the algorithm have been substantiated by simulation for hosts with varied mobility rates and different number of clusters.

Keywords: ad hoc network; checkpointing; asynchronous recovery; network partition; spanning tree.

1. Introduction

A MANET is formed by a set of independent mobile nodes that communicate with each other by forming a multihop radio network. The nodes are self organizing and each node in the ad hoc network may perform the dual roles of a host and a router. The nodes may also move freely across the network and may join or leave the network at any time leading to a dynamic network topology. A node may communicate with another node either over a shared direct wireless link (i.e. single hop network), or over a sequence of wireless links including one or more intermediate nodes (i.e. multi-hop network).

When mobile nodes move within transmission range of each other, wireless links are created. On the other hand, a wireless link between nodes fails when nodes move so that they are no longer within transmission range of each other. The limited bandwidth is a major constraint faced by the MANETS. Also the MANETs face frequent link creation and disruption due to the mobility of the nodes. The recurrent link creation and disruption may result in disconnected components in the network i.e. such networks are susceptible to network partitioning. Though processes in the same component can communicate with each other, they cannot communicate with processes in other components.

The failure probability of the computing processes increases greatly due to the nature of the system itself. The ad hoc and dynamic topology of MANETs creates challenges for computing processes that were either not present or have been handled well for cellular mobile systems. The need for reliability in such a scenario leads to the requirement of some fault tolerance methods specific to the system.

Techniques such as group communication, transactions and rollback recovery have been used to improve the availability and reliability of processes in the distributed systems. Of these, rollback recovery protocols restore a process to a consistent state after failure. These protocols require that the state of a process be stored onto stable storage i.e. checkpointed frequently during error free operation. At the time of recovery, the failed process rollbacks to the latest checkpoint and resumes computation from that state. The asynchronous recovery of a process can be accomplished by combining message logging with checkpointing. In the log-based rollback recovery the determinants of non-deterministic events are logged into the stable storage during failure-free operation. At the time of recovery, the

processes use the checkpoints and logged determinants to rerun the corresponding non-deterministic events to achieve a consistent state. Message logging also reduces the extent of rollback during recovery.

The paper presents an algorithm that allows the asynchronous rollback recovery of a process in the ad hoc environment after a crash failure using movement based checkpointing and message logging. The objectives of our work are mainly to address the challenges faced by the recovery process due to the following constraints of MANETs:

- inadequate stable storage
- limited wireless bandwidth
- dynamic topology and hence network partitions

For accomplishing the above, the MANET is organized into a hierarchical structure by segregating the nodes into disjoint and virtual groups called clusters. The clustering approach is utilized to handle the limitation of stable storage A self stabilizing spanning tree is maintained over the resultant topology to reduce the number of recovery related messages. This conserves the wireless bandwidth and also deals with the changing topology or partitions in the network. Our scheme combines message logging with movement based checkpointing to limit the overhead due to checkpointing.

The paper is organized as follows. The next section discusses the related works in the area of checkpointing and rollback recovery in mobile computing systems. Then we present the underlying system model of the proposed algorithm. Subsequently, we describe the checkpointing algorithm and the recovery protocol. The proof of correctness of the algorithm is presented. The results of simulation of the scheme for hosts with different mobility rates and for networks with different number of clusters are described. Lastly we conclude our presentation

Related work

Considerable work has been done in the area of checkpointing and rollback recovery of cellular mobile computing systems. Most of the schemes proposed rely on stable storage at static base stations. An uncoordinated checkpointing scheme for mobile hosts is described in [1] where application messages are employed by MHs to achieve a global consistent checkpoint. There is no overhead due to extra coordination messages. Time is used to create a global consistent state of the system in the protocol described by [2]. There is no use of message exchanges for the same. [3] provides a detailed insight of pessimistic, optimistic and causal message logging protocols .

An asynchronous recovery scheme based on optimistic message logging is presented in [4] the mobile support stations are responsible for logging and dependency tracking while mobile hosts need to store only a small amount of information for mobility tracking. An independent failure recovery scheme for mobile applications based on movement-based checkpointing and logging is put forth in [5]. A mobile host takes a checkpoint only after it has encountered a threshold of handoffs. The optimal threshold is determined dynamically. An asynchronous recovery protocol based on communication-induced checkpointing for mobile computing systems is presented in [6]. Stable storage is assumed at Mobile support stations which store and trace the checkpoints, message logs and handle rollback of the mobile hosts. The recovery algorithm does not cause domino effect. To address the stable storage problem, the work in [7] presented an asynchronous consistent global checkpoint collection algorithm which prevents contention for network storage at the file server .In the algorithm, a process initiates consistent global checkpoint collection by saving its state tentatively and asynchronously (called tentative checkpoint) in local memory or remote stable storage if there is no contention for stable storage while saving the state.

The above schemes can not be applied directly in ad hoc wireless network due to the absence of any extra static centralized administration or fixed Mobile Support Stations. Recently the area of checkpointing and rollback recovery in the ad hoc networks has received attention in literature. A cluster based scheme designed for ad-hoc wireless networks is presented in [8]. It utilizes the cluster-based multi-channel management protocol (CMMP). The mobile nodes in the network act as cluster heads, gateways or ordinary members. Quasi synchronous checkpointing algorithm is employed along with pessimistic logging. The recovery scheme is free from any domino effect and a process can rollback from the latest local consistent checkpoint.

The checkpoint protocol proposed in [9] employs message exchanges for checkpointing. A request to checkpoint is broadcast by flooding and the same message carries the state information of mobile nodes. In the model of [10] the MANET is geographically partitioned into several disjoint and equal sized clusters. Each cluster is assigned one manager which can directly communicate with the adjacent managers For the recovery algorithm each manager must keep an

($n_{total\ h}$ * $n_{cluster\ h}$) dependency matrix where $n_{total\ h}$ is the total number of mobile hosts in the system and $n_{cluster\ h}$ is the total number of mobile hosts in its cluster.

None of the above approaches addresses the problem of network partitions or the problem of access to limited stable storage by the checkpointing nodes in ad hoc networks. Moreover the above approaches have not tried to reduce the overhead due to recovery related messages as most of them have utilized flooding for broadcasting the same.

2. Background

2.1 System Model

Organizing a network hierarchically by clustering allows the spatial reuse of system resources such as the storage and bandwidth. A clustered network can deal with the node movement, node failures, node insertions and removals from the network. We consider a multi hop clustered MANET model as in Fig 1. The network is divided into disjoint clusters with one node chosen as the cluster head (CH) in each cluster. A node, for instance, may identify itself as the CH if it meets some predefined qualifying criteria, such as the availability of stable storage or having the least mobility.

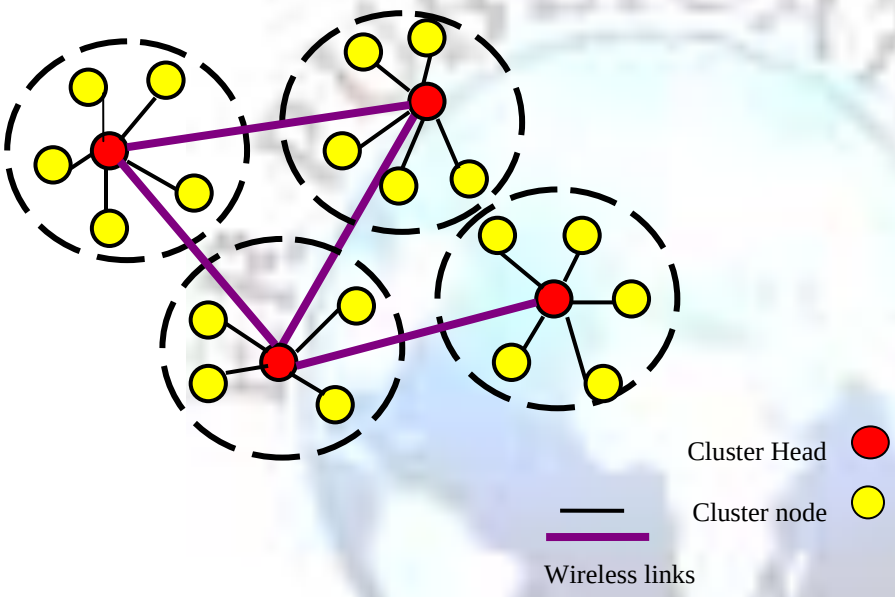


Fig 1: System Model

Ordinary nodes within the cluster are assumed to be within one-hop distance of the CH. The CHs can communicate directly with other cluster heads in their transmission range i.e. by a one hop path. Within a cluster, the ordinary nodes may communicate via the CH while they can communicate with nodes in other clusters through their respective CHs. A communication between CHs not directly in the transmission range of each other will require a sequence of wireless links including one or more intermediate CH nodes i.e. by a multi-hop path.

The CHs are assumed to have enough stable storage for saving the checkpoints of nodes in their cluster initially but we also provide a storage management module for dealing with the shortage of stable storage. The processes in the system are fail-stop and communicate with each other only by exchanging messages. No message will be lost in the channel. The CHs do not fail and have limited or none mobility. However links between nodes, CHs or ordinary, may be created or disrupted at any instant. The dynamic topology of the system may lead to disconnected clusters in the system, thus partitioning the network.

2.2 Self Stabilizing Spanning Tree Construction

The CHs form a connected graph G(V,E) in which V is the set of nodes representing the CHs and E is the set of edges representing the connections between the neighboring CHs. We adapt the algorithm of [11] to impose a spanning tree

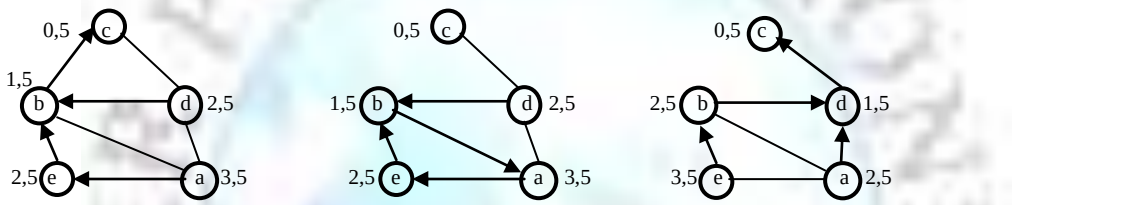
over the nodes of the graph. The algorithm is self- stabilizing because regardless of the initial state, the algorithm always builds a spanning tree in the system in a finite number of moves .Each node i in the tree knows its level L(i) and its parent P(i) in the tree. If r is the root of the spanning tree, the following predicate is always true:

$\forall i, p : i \neq r$ AND $p = P(i) : L(i) = L(p) + 1$

The nodes have to execute the following actions to maintain the spanning tree in the system [11]:

1. if ((L(i) <> n) AND (L(P(i)) <> n) AND (L(i)<> L(P(i)) + 1)) then set L(i) = L(P(i))+1
2. if ((L(i) <> n) AND (L(P(i)) = n)) then set L(i) = n
3. if ((L(i) =n) AND ($\exists$ k $\in$ N(i) : L(k) < n-1)) then set L(i) = L(k) + 1; set P(i)=k;
   where N(i) represents the neighbors of node i in the graph.

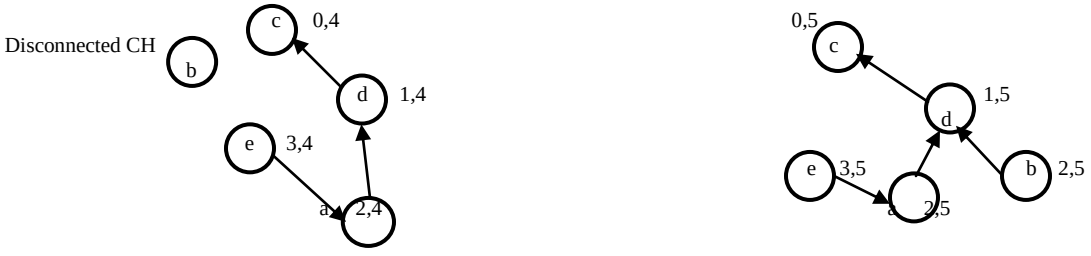
For the system used in our work, any one node is randomly chosen as the root of the tree. Each node in the spanning tree additionally knows the total number of CHs and its own child nodes in the tree. The self-stabilizing property of the algorithm used ensures that the system is able to maintain a spanning tree of the connected CHs even after any change in the topology of the system due to the movement, if any, of the CHs. Fig 2 (i) shows an initial spanning tree in a MANET where a, b, c, d and e represent CHs. The root at node c is at level 0 while a, b, d, e are at levels 3, 1, 2, 2. Each node has a pointer to its parent in the tree and stores the number of nodes in the tree i.e. 5 along with its own level in the tree. Fig 2 (ii) illustrates the situation where the topology of the system changes due to the change in the links between CHs Fig 2 (iii) shows how the self stabilizing algorithm succeeds in building a spanning tree in the system.



(i) Spanning Tree in the initial system (ii) Illegitimate configuration (iii) Final spanning tree due to change in topology

Fig 2: Self Stabilizing spanning tree in MANET

Though the algorithm of [11] does not consider partitions in the network, a CH may lose connections with the network and create in a partition in our system. If initially there are n clusters in the network, the maximum level of a node in the network is n-1. We assume that only one cluster may lie in a partition at a given time. Thus the maximum legitimate level of a node can be n-2 after a partition. The nodes in the tree make moves to reach a legitimate state in a finite number of moves after a partition. Fig 3(i) shows the spanning tree in the system with four CHs after the CH e is disconnected from the system. The last neighbor, such as e, to lose connection with a CH, such as b, gets the recovery related data from b (as will be described later). It thus knows that a loss in connection with b implies that a network partition has been created. It decreases the number of total nodes to four. Its child and parent nodes see the change and make the moves necessary to maintain the spanning tree in the system. When a CH comes in the range of the network, it sends a join message to a CH in the network in its transmission range. This causes such a neighbor to add the CH to its Neighborlist and increment the number of CHs in the system by one. This further leads the system to make the moves required to generate a spanning tree in the network as in Fig 3(ii).



(i) Spanning Tree in the system after a partition (ii) Spanning Tree in the system after a CH joins the network

Fig 3: Spanning Tree after network partition

3. The Log Based Checkpointing Algorithm

3.1 Concept

We present a movement based checkpointing and message logging algorithm that allows the asynchronous recovery of a mobile host subsequent to a crash failure. Any mobile host in the network is always a member of some cluster in the network. It may be a special node i.e. the Cluster Head or an ordinary member affiliated to the Cluster Head. As the mobile host moves in the network, it may change its cluster and hence the affiliation to a CH. Since stable storage is limited at a CH, the checkpoint and recovery related information of a MH are distributed among the CHs with which the mobile host affiliates as it moves across the network. This approach provides an additional benefit of avoiding the simultaneous access to stable storage present at one place by all the MHs. Each MH uses the stable storage at the CHs with which it affiliates as it moves.

A weight, W, is assigned to each cluster head based on its connectivity with other CHs at a given time. This weight of a given CH is equal to the number of its neighbor CHs. Since the topology of the network is dynamic, the weight of the CH may change over time.

The CH of the first cluster that a MH joins on entering the network becomes its Checkpoint & Movement Coordinator (CMC). The MH saves its initial state at the CMC. The further movement of the MH defines a virtual region comprising the cluster of the CMC and its neighboring clusters. The messages of a MH are logged at the current CH but till the time the MH is in a virtual region, its checkpoint remains at the CMC. However the message log unifying scheme ensures that the message log of the MH at all CHs except its current and previous to current CHs keeps getting unified at the CMC as the MH moves in a virtual region. Once the MH moves to a CH which is not a neighbor of the CMC, a new checkpoint is taken at that CH and it becomes the new CMC of the MH. Therefore at any given instant, the recovery related data of a MH is distributed amongst the CMC and a maximum of two CHs.

3.2 Working of the algorithm

For accomplishing the above concept, the following data structures have been used in the algorithm:

CURRENT_LOC <MH id, Current CH of the MH >: a CH maintains the current location of all the MHs for which it is the CMC
Cluster_list$_x$: a list of MH ids in the cluster of CH_x
Neighbor_list$_x$: a list of the neighboring CHs of CH_x ;
weight of CH_x = number of neighbors at a given time
The MH identifies itself by : **<own MH id, id of its CMC, id of its previous CH >**

The algorithm uses the following messages between the CHs

delete_log (CH_j,MH) **:**sent by some CH to CH_j asking CH_j to delete the message log of MH present with it
unify_log (CH_i, CH_j, MH): sent by CH_i to CH_j asking CH_j to unify the message log of MH present with it at CH_i and delete its own copy.
inform_loc(CH_i ,MH) :sent by CH_i to another CH informing the presence of MH in the cluster i

3.3 Pseudo Code

As a MH moves in the network, its latest checkpoint remains with the CMC while the message log of the MH keeps getting unified at the CMC. At a given instant only the message logs at the current & previous to current CHs is not present at the CMC. We denote the set of CHs< CMC, current CH, previous CH> as the RECOVERY_SET of the MH. For instance, when the MH is in the cluster of CMC, its RECOVERY_SET is of the form <CMC, CMC, null>. When the MH moves to a neighbor CH_i from the CMC, its RECOVERY_SET is of the form <CMC, CH_i, CMC>. At other times, the RECOVERY_SET may be of the form <CMC, CH_i, CH_j> which implies that the MH is at the CH_i while the previous CH was CH_j. This RECOVERY_SET is not stored explicitly as the MH stores the CMC and previous CH information and the CMC knows the current CH of the MH. The message logging and unifying scheme is detailed next and illustrated by Fig 4.

PROCEDURE enter_network(MH,CH_m){

// MH initially becomes a part of cluster m in the network
i. Set CMC = CH_m
ii. Save initial state of the MH at the CH_m
iii. Add a record to CURRENT_ LOC at the CH_m<MH id, CH_m >
//Further messages of the MH are logged at the CMC. **}**

PROCEDURE move_from_CMC(MH,CMC,CH_i){

// When the MH moves from the CMC to a neighboring cluster i, i.e. the $Neighbor\_list_i$ contains CMC
i. Send inform_loc(CH_i,MH) from CH_i to the CH_m
ii. Update CURRENT_LOC at CH_m as <MH id, CH_i >
iii. Set previous CH of MH = CH_m
// Further messages of MH are logged at CH_i **}**

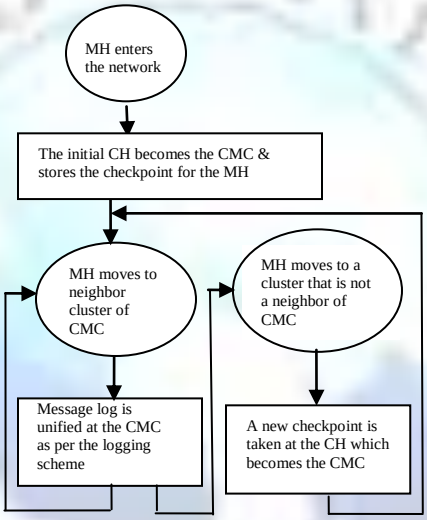


Fig 4: Message Logging & Unification

PROCEDURE move_from_CH(MH, CH_j , CH_k){

// When the MH moves from an ordinary CH, say CH_j such that CH_j was a neighbor of CMC to its neighbor CH_k
If the CH_k is same as the previous CH of MH in the movement pattern then {
i. Unify the log of MH with CH_j at CH_k
ii. Set the previous CH = CMC
iii. exit
} elseif the $Neighbor\_list_k$ contains CMC of the MH then {
i. Send inform_loc(CH_k,,MH) from CH_k to the CMC
// CMC updates the CURRENT_LOC of MH
ii. If previous to CH_j in the movement pattern of the MH was the CMC of MH, then{
Perform no action at CH_j}
else {
Send unify_log from CH_j to its previous CH // Previous CH appends its log for the MH at the CMC and deletes its own copy}
iii. exit
} elseif the $Neighbor\_list_k$ does not contain CMC of the MH then {

i. Take a new checkpoint of the MH at CH_k
ii. Delete the log at the CH_j for the MH
iii. Send delete_log to the CMC and the previous CH of the MH, if different from the CMC.
} **}**

3.4 Proactive approach for handling network partitions

A MH may crash or voluntarily disconnect in a cluster and recover in any other cluster later. Upon subsequent recovery of a MH, the state of the MH before failure can be reconstructed in the correct order. This is achieved by unifying the checkpoint and message log of the MH at the CMC with the message log before failure at the other CHs in the RECOVERY_SET. However this recovery related information of a mobile host may be present with a disconnected CH since CHs can dynamically lose connection with other CHs. Hence a proactive approach is employed by the CHs for dealing with the network partitions.

If only one neighbor is left in the neighbor list of a CH, it proactively takes actions to ensure the availability of the recovery related data of a mobile host later .Here we assume that the CH does not lose connection while transferring the data.

If the CH is the CMC for some MHs currently in the cluster of the only neighbor of this CH; it asks those MHs to take a checkpoint at the current CH. For all other connected or disconnected MHs, the CH transfers the recovery related data present with it to its only neighbor. Later if the connection is actually lost, the neighbor can reflect the decreased number of CHs over the existing spanning tree and any CH at an illegitimate level in the spanning tree makes moves to take the system to a legitimate state. However if a CH's weight becomes greater than one, it may ask the neighbor to discard the backup data. The neighbor based on its stable storage availability may discard or not for later availability.

3.5 Storage Management at a Cluster Head

If a MH remains in the neighborhood of a CMC for long, its message log at the CMC may increase, so storage management is required Since a CMC has a record of the current CH of each active MH, it can ask the MH to take a checkpoint at the current CH. The current CH passes a delete_log message to the previous CH, if any, for the MH. The previous CH and the CMC then remove the previous checkpoint & message logs for the MH present with them.

4. The Asynchronous Rollback Recovery Algorithm

A MH may crash in a cluster, C_m and recover in the same or a different cluster, C_n. The recovery related data is distributed amongst the CHs in the RECOVERY_SET which are neighbors of each other and within 1 hop distance of each other. Thus for successful recovery, it is necessary to locate either of the CHs in the RECOVERY_SET. After the required CHs have been found, it is required to unify the message logs with the latest checkpoint at the CMC. This recovery related data is then sent to the current CH of the MH. The recovery protocol in different scenarios is described next.

A. The MH may recover in a cluster such that the recovery information is available amongst the neighbors

PROCEDURE Recovery(MH, CH_i) {

// The MH recovers in the cluster i
If the CH_i is the same any CH of the RECOVERY_SET of MH then
{
i. If $CH_{i=}$ the CMC for the MH, then {
restore the latest checkpoint of the MH ;
send unify_log to the last and previous CHs
} else {
send unify_log to the CH of the RECOVERY_SET other than the CMC }
ii. Unify the message log at the CMC
iii. Present the unified log to the MH along with the checkpoint.

iv. Roll back to this checkpoint and replay the logs in order at the MH.
v. Set previous CH of the MH as NULL and set CH_i as the CMC
} elseif the MH recovers in a cluster whose neighborlist contains either the CMC or the previous CH, say CH_x {
i. Send the CMC & previous CH information from CH_i to CH_x .
ii. Pass unify_log from CH_x to other CHs in the RECOVERY_SET
iii. Send the unified recovery data from CH_x to the current location of the MH
iv. Set previous CH = NULL and set CMC= CH_i
}

B. The MH may recover in a cluster such that the recovery information is not available amongst the neighbors.

The MH has a record of its CMC and the previous to last CH. The last CH of the MH can then be retrieved from the CMC. The following message is used to locate the CMC of the MH:
locate_CH(seq no, MH, CH_i , CH_j): sent by CH_i to its parent and child nodes in the spanning tree for finding the location of CH_j and retrieving the message log of MH
The following data structures are required for traversing the spanning tree:
PARENT(CH_i): the parent CH of the CH_i in the spanning tree maintained
CHILD(CH_i): the list of child nodes of CH_i in the spanning tree maintained

PROCEDURE Recovery(MH, CH_i) {

// The MH recovers in the cluster i but the recovery related data is not present in the neighboring clusters of CH_i
i. Send locate_CH(seq no, MH, CH_i , CMC) from CH_i to its parent and child nodes in the spanning tree. //The sequence number is used to detect duplicates.
ii. At each intermediate node which is not the CMC, check the neighborlist, append this node's id and forward the message to its parent and child nodes in the spanning tree other than the sender of the message if the message was not a duplicate. //This approach gives a significant improvement as compared to the flooding of the network with the locate_CH message.
iii. As soon as the CMC is located, stop the propagation of the locate_CH message.
iv. Restore the latest checkpoint of the MH and send unify_log message to the last and previous CHs
v. Present the unified log to the MH along with the checkpoint.
vi. Roll back to this checkpoint and replay the logs in order at the MH.
vii. Set previous CH of the MH as NULL and set CH_i as the CMC **}**

Even if a CH in the RECOVERY_SET is currently in a network partition, the recovery related data is available at a connected component due to the proactive actions of dealing with partitions.

5. Proof of Correctness

Theorem 1: Recovery involves additional O(n) messages where n is the number of clusters in the network.

Proof: Let there are n clusters in the network. At the time of recovery, if the previous CMC of the MH is not available within the neighborhood of the cluster in which the MH recovers, then a maximum of n-1 messages are passed across the spanning tree of the CHs in the network to locate a CH in the RECOVERY_SET. The response of the message log to the sender Cluster Head involves messages equal to the length of path between the node representing the previous CMC in the spanning tree and the current CH. As this distance is bound by a maximum of n-1, the recovery process involves a maximum of 2 *(n-1) additional messages i.e. of the O(n).

Theorem 2: The spanning tree created in the MANET is self-stabilizing.

Proof: To prove the validation of the spanning tree constructed by the algorithm, we need to prove the convergence of the system i.e. the system reaches a legal configuration from any state in a finite number of steps and the closure i.e. any legal transition will keep the system in a legal state.
The spanning tree created for connecting the CHs in the MANET is self-stabilizing by virtue of the rules used from the algorithm in [11]. However these rules do not account for the change in topology or disconnected nodes which is possible in the system considered by us. If there is a change in the links between a CH and its neighbors, it chooses any

one neighbor randomly as its Parent taking the system to an illegitimate state. This causes the CHs to again make the necessary moves as per the rules from the algorithm in [11] to take the system to a legitimate state. Also each CH knows the total number of CHs in the system. Hence, when a CH loses connection with the network, the last CH to lose connection with it decreases the counter denoting the total number of CHs in the system. This change is detected by the Parent and Child nodes of this CH, thus triggering the system to move to a legitimate state in a finite number of moves, thus proving the convergence of the algorithm. Moreover each type of change in topology of the system has been handled by the algorithm and thus is legal, therefore closure of the algorithm is guaranteed.

Theorem 3: Recovery process is consistent and can handle multiple failures concurrently.

Proof: The message logging scheme ensures that the message log keeps getting unified in the correct sequence at the CMC as per the movement pattern of the MH. Upon recovery, the latest checkpoint of the MH is available at the last CMC of the MH and the message log for the MH is distributed among the CHs of the RECOVERY_SET. These CHs may be among the neighbors of the current cluster head or not. However, any one of the CHs in the RECOVERY_SET can be determined as described in the prior sections and after that it is possible to unify the message log at the last CMC to reconstruct the state as before failure. Also the recovery procedures of multiple MHs can proceed concurrently as the CHs involved in the process can send and receive recovery related messages independently of each other. All such messages can use the spanning tree paths by the CHILD and PARENT data structures for locating the recovery related data

6. Performance Analysis

6.1 Simulation Model

We have simulated a mobile ad hoc system with a clustered organization and have varied the number of clusters as 6, 12 and 18. Each node in the system is part of some cluster at any given time. Every cluster has one node acting as the Cluster Head and all other nodes in the cluster use the stable storage at the CH for checkpointing. Each cluster has a random number of neighboring clusters and the CHs of the neighboring clusters are within one hop distance of each other. A MH may move from one cluster to another neighboring cluster. We select the next cluster for each movement out of the neighboring clusters randomly. The time interval between two inter cluster movements follows an exponential distribution with an average of $1/\lambda$. We take $\lambda = 0.05$ initially .

6.2 Simulation Results

Since our scheme is a movement based scheme, our simulation experiments have compared the number of checkpoints taken by a MH w.r.t. the number of clusters changed by it across the network. Till the time a MH is in a particular virtual region, no checkpoints are taken. Only when there is a movement to a CH which is not the neighbor of a CMC, a checkpoint is taken. Fig 5 demonstrates the number of checkpoints taken by a MH with $\lambda = 0.05$ in networks with clusters varying between 6, 12 and 18. As the number of clusters increases, the MH has more area to move and hence the number of checkpoints increases but the increase is not substantial.

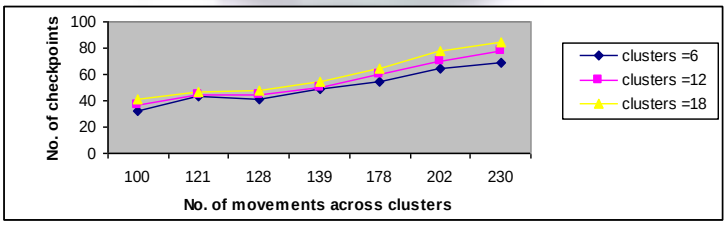


Fig 5: Comparison of checkpoints in different sized networks

However, the number of recovery messages decreases with the growing number of clusters as shown by fig 6. This is due to the fact that the probability of a CH of the RECOVERY_SET of a MH being in the neighborhood of the cluster in which the MH recovers increases. Furthermore, the number of recovery messages due to our scheme is significantly lower than the number of messages which would be broadcast if flooding was to be employed for locating a CH of the RECOVERY_SET. Fig 7 demonstrates the same.

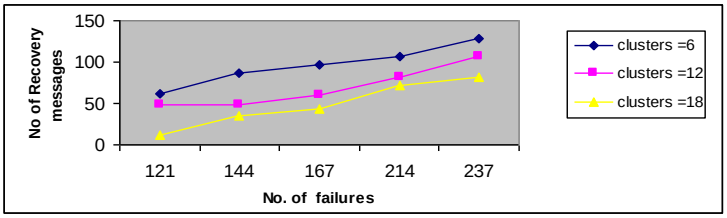


Fig 6: Comparison of recovery messages in different sized networks

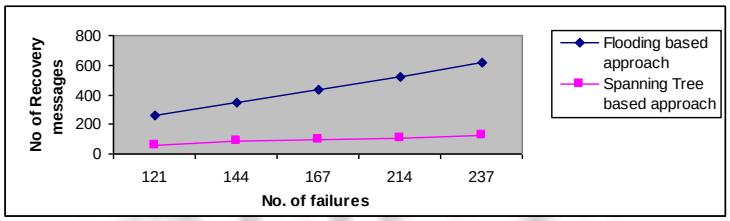


Fig 7: Comparison of recovery messages with flooding based approach

Conclusion

The paper has proposed an algorithm for the asynchronous recovery of a mobile host after a crash failure using log based checkpointing. The checkpoints are taken by a mobile host depending on its movement across the network. The stable storage at a Cluster Head is utilized for storing the checkpoints and message logs of a MH in its cluster. While a MH remains in the neighboring clusters of the CH acting as its Checkpoint and Movement Coordinator, no new checkpoints are taken. Therefore, the number of total checkpoints is comparatively less. Further, the CHs are organized into a spanning tree to reduce the number of recovery related messages as compared to flooding. The simulation results show that the algorithm takes less number of checkpoints; performs well for hosts with low as well as high mobility rates and scales up as we increase the number of clusters in the system.

References

[1]. Acharya, A., Badrinath,B.R, “Checkpointing distributed applications on mobile computers”, In: 3rd Int’l Con. on Parallel and Distributed Information Systems, pp. 73–80. October 1994.
[2]. Neves, N. Fuchs, W. K., “Adaptive recovery for mobile environments”, ACM Press, Vol.40, Issue 1, pp. 68 – 74, 1997.
[3]. Elnozahy, E. N., Alvisi,L. Wang, Y.M., Johnson, D.B., “A survey of rollback-recovery protocols in message-passing systems”, ACM Computing Surveys (CSUR), v.34 n.3, p.375-408, September 2002.
[4]. Park, T., Yeom, H.Y, “An asynchronous recovery scheme based on optimistic message logging for mobile computing systems”, 20th International Conference on Distributed Computing Systems, pp.436-443 , 2000.
[5]. George S. E., Chen, I. and Jin, Y., “Movement based checkpointing and logging for recovery in mobile computing systems”, In Proc of MobiDE’06, pp. 51-58, June 2006.
[6]. Tantikul T., Manivannan D., “A Communication-Induced Checkpointing and Asynchronous Recovery Protocol for Mobile Computing Systems”, In Proc. of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT '05). IEEE Computer Society, Washington, DC, USA, 70-74., 2005.
[7]. Jiang, Q.,Manivannan,D., “An optimistic checkpointing and selective message logging approach for consistent global checkpoint collection in distributed systems”, in Proc. IEEE International Parallel and Distributed Processing Symposium, pp. 1-10. (2007).
[8]. Men, C., Xu,Z., Li,X.., “An Efficient Checkpointing and Rollback Recovery Scheme for Cluster-Based Multi-channel Ad Hoc Wireless Networks”, In :Proc. of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA '08). IEEE Computer Society, Washington, DC, USA, 371-378 (2008).
[9]. Ono,.Higaki., H, “Consistent Checkpoint Protocol for Wireless Ad-hoc Networks”, The 2007 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada,USA, pp. 1041-1046.(2007).
[10]. Juang ,T. T., Liu, M.C., “An Efficient Asynchronous Recovery Algorithm In Wireless Mobile Ad Hoc Networks”, J. of internet technology P.143-152 vol 4 (2002).
[11]. Nian-Shing, C., Yu, H., Huang,S.,“A self-stabilizing algorithm for constructing spanning trees. Information Processing Letters”, 39 pp. 147-151. (1991).