Journal of King Saud University – Computer and Information Sciences (2015) **27**, 402–415

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Rollback recovery with low overhead for fault tolerance in mobile ad hoc networks

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

[a] *Department of Computer Science, Jaypee Institute of Information Technology, Noida, India*
[b] *Department of Computer Engineering, National Institute of Technology, Kurukshetra, India*

Received 14 November 2013; revised 4 February 2014; accepted 13 March 2014
Available online 18 June 2015

KEYWORDS

Ad hoc network;
Mobile backbone;
Rollback recovery;
Checkpointing;
Message logging;
Routing protocols

Abstract Mobile ad hoc networks (MANETs) have significantly enhanced the wireless networks by eliminating the need for any fixed infrastructure. Hence, these are increasingly being used for expanding the computing capacity of existing networks or for implementation of autonomous mobile computing Grids. However, the fragile nature of MANETs makes the constituent nodes susceptible to failures and the computing potential of these networks can be utilized only if they are fault tolerant. The technique of checkpointing based rollback recovery has been used effectively for fault tolerance in static and cellular mobile systems; yet, the implementation of existing protocols for MANETs is not straightforward. The paper presents a novel rollback recovery protocol for handling the failures of mobile nodes in a MANET using checkpointing and sender based message logging. The proposed protocol utilizes the routing protocol existing in the network for implementing a low overhead recovery mechanism. The presented recovery procedure at a node is completely domino-free and asynchronous. The protocol is resilient to the dynamic characteristics of the MANET; allowing a distributed application to be executed independently without access to any wired Grid or cellular network access points. We also present an algorithm to record a consistent global snapshot of the MANET.

1. Introduction

Mobile ad hoc networks (MANETs) have extensively enhanced the wireless networks as they eliminate the need for any fixed infrastructure, in the form of base stations, routers etc. These networks are formed by nodes that communicate over wireless links without the control of any central or fixed administration. Each node performs the dual roles of a node as well as a router. As MANETs are self organizing and rapidly deployable, these have been frequently used for communication in places where it is either expensive or difficult to install network infrastructure, such as in battlefields, search-and-rescue or space exploration. In addition, the computational power of mobile computing platforms of the present day exceeds that of the workstations from a few years ago. The explosive and continuing growth of wireless devices and networks along with their widespread availability provides

* Corresponding author.

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2014.03.022

a thrust for understanding and utilizing the computing potential of mobile ad hoc networks. These networks are increasingly being used in collaboration with LAN/WAN scenarios, for parallel processing systems, as a means of expanding the computing capacity of existing networks such as cellular mobile systems and even for implementing mobile Grid computing systems (Darby, 2010; Wang et al., 2006; Rao et al., 2006; Jipping, 2001).

A variety of lightweight, distributed applications can be executed successfully on mobile ad hoc platforms without the support of fixed infrastructure. These applications include mobile agents providing location-aware services; local and collaborative processing of sensor data collected from a number of MHs, update of maps in real-time battle scenarios etc. Other applications include collaborative mobile gaming, context-aware applications for internetworked vehicles, bio-informatics and other scientific applications; especially in remote areas where access to the wired network is infeasible (Darby and Tzeng, 2010). Smart phones having high computational capabilities along with laptops and Personal Digital Assistants (PDAs) may be used for creating computing clouds in trains, colleges etc. Such clouds could be used to calculate weather forecasts for passengers at their destination using environmental data from public sensing systems, cracking of encryption codes, development of mobile health care and education applications besides participating in scientific projects (Buschin et al., 2012). Some mobile Grid projects, such as the Akogrimo (2010), have explored the use and practical applications of mobile Grid concepts, so that idle resources from a great number of mobile devices could be used for the development of a mobile Grid computing platform.

Due to the vast number of feasible practical applications, the current mobile computing platforms are increasingly being utilized as viable compute resources. However, the nodes in such systems vary greatly in their capabilities such as computation power and battery power and may be susceptible to different types of transient and permanent failures. Therefore, the applications designed to execute on these systems should be fault tolerant so that they can complete successfully without access to any wired Grid or cellular network access points. Checkpointing and rollback recovery have been used widely and effectively to provide fault tolerance for distributed systems in static as well as dynamic environments (Elnozahi et al., 2002). Checkpointing results in a significant performance enhancement as it allows a failed node to resume execution from its latest saved error-free state at the time of recovery and thus avoids the need to restart job execution from the very beginning. In contrast, in the absence of execution checkpointing the failure at one node may cause some other nodes to suspend execution as well, if they are waiting for intermediate results from the failed node. Thus, process failures can lead to severe performance degradation or even total job abortion in the absence of checkpointing.

Though a number of checkpointing and rollback recovery protocols exist for static distributed systems or cellular mobile computing systems, these are not trivially applicable to MANETs as they pose some categorically different set of challenges. Ad hoc wireless networks are characterized by limitation of resources as wireless bandwidth, stable storage, battery power etc. Moreover, the absence of fixed infrastructure generates new problems for ad hoc networks, such as self-routing and a highly unpredictable and dynamic topology. The traditional systems rely on stable storage available at nodes or Base Transceiver Stations, for saving recovery related information (Prakash and Singhal, 1996; Li and Shu, 2005; Tantikul and Manivannan, 2005). On the other hand, the ad hoc environment lacks such capable stations and large data carrying reliable links. The mobile ad hoc networks also have an intrinsic scalability limitation. As the size of the network increases, the performance of the ad hoc network rapidly degrades because a large network with flat structure results in long hop paths which are susceptible to link breaks.

The paper presents a checkpointing and rollback recovery protocol to provide fault tolerance in MANETs. We consider a backbone based mobile ad hoc network which is a type of hierarchical network used for scalability and implementation of efficient protocols (Rubin et al., 2004). Such a network comprises of some particular backbone capable nodes (BCNs) which have powerful radios and are functionally more capable than other ordinary nodes. A virtual backbone is formed by dynamically electing some BCNs to act as backbone nodes (BNs) and forming links between interconnecting neighboring BNs. Each of the other BCNs and ordinary nodes affiliate with one BN such that clusters of nodes are formed with the BN acting as the cluster-head. The communication between the nodes uses the backbone and thus, avoids long hop paths and improves the network performance. Nodes communicate with each other through the BNs to which they are affiliated. If the communicating nodes are affiliated to the same BN, routing is straightforward. However, if they are at remote locations, the routing protocol existing in the network is used for routing through the backbone network. A location based routing protocol, GOAFR+ (Kuhn et al., 2008), has been assumed for the current work. It employs a combination of greedy and face routing to reach destinations using their geographic information. However, our recovery protocol is independent of and can be integrated with any routing protocol for MANETs.

The presented recovery protocol has been designed to handle the specific challenges posed by the dynamic topology and resource constraints of a MANET. The protocol does not add a high overhead to the normal process execution as it takes advantage of the routing protocol already existing in the network. The proposed scheme is an application of cross-layer optimization where the routing protocol existing in the network has been utilized for implementing a message efficient checkpoint and recovery mechanism. The use of the backbone clustered structure provides for added scalability of the protocol. The contributions of the paper can be summarized as follows: (1) The paper presents a checkpointing and rollback recovery protocol which does not assume access to any fixed host or wired network and is therefore appropriate for MANETs. (2) The checkpointing process does not require control messages as the control information required by the protocol is piggybacked on the application messages. (3) The recovery procedure may involve a few control messages; imposing only a low overhead on the network. (4) Rapid and efficient recovery of a mobile node is possible despite the dynamic topology of the network. Even if a mobile node recovers at a location different from the location of its crash, its checkpoint and related information can be located in the network without much delay using the network backbone.

(5) An algorithm to construct a consistent global snapshot of the system is also provided. Simulation experiments have been performed to evaluate the proposed scheme.

The rest of the paper is structured as follows: Section 2 discusses the background of the checkpointing and message logging techniques. Section 3 outlines the related work done in the areas of checkpointing as well as application message routing in MANETs. The underlying system model used in the proposed algorithm is described in the Section 4; explaining the construction of the network backbone and the assumed routing methodology. Subsequently, we present the message logging based checkpointing algorithm and the recovery protocol in Section 5. An algorithm to compute the global snapshot of the system is described in Section 6. The presented scheme is compared with related schemes and simulated under varying application message traffic and failure rates. The comparative performance analysis is presented in Section 7. Section 8 concludes the presentation.

2. Background

A global checkpoint of a distributed execution is formed as a set of local checkpoints, one from each process in the system. However, the message passing between processes creates dependencies among checkpoints of different processes. (It may be noted that a message implies application message in the subsequent discussion, unless explicitly specified as a control message). As a consequence, any given set of local checkpoints may not be consistent if there exists a message whose receive event is recorded in some process's local checkpoint, but its send event is not recorded in the local checkpoint of any process. Such a message is an orphan message and a system state formed as a set of local checkpoints, one from each process, is consistent if and only if it does not include any orphan message.

A straightforward approach to construct a consistent global checkpoint of a distributed computation is provided by coordinated checkpointing (Elnozahi et al., 2002). This technique requires that the processes in the system synchronize with each other at the time of checkpointing, i.e., for saving their local state on stable storage periodically. To recover from a crash failure, the system rollbacks to its latest saved consistent global state formed as a set of the checkpoints of all processes. Even if one process fails, multiple processes may have to roll back to their latest checkpoints in order to restore a consistent system state. Further, the recording of a consistent global checkpoint on stable storage requires a large number of messages between processes to synchronize their checkpointing activities. Therefore, coordinated checkpointing suffers from high communication and synchronization overhead associated with the checkpointing process (Li and Shu, 2005).

On the contrary, the independent or uncoordinated checkpointing schemes allow a process to take its checkpoints at periodic intervals independently, without any synchronization or message passing among processes (Elnozahi et al., 2002). A major drawback is that it may lead to *domino effect* which is a condition where the rollback of one process may trigger a cascaded rollback by multiple processes (Randell, 1975). In the worst case, the domino effect can take the computation to the initial state. Further, uncoordinated checkpointing may result in useless checkpoints at processes, i.e., checkpoints which cannot be a part of any consistent global state. The necessary and sufficient condition for a set of local checkpoints to form a consistent global state is derived from the results on zigzag paths (z-paths), a generalization of Lamport's happened before relation (Netzer and Xu, 1995). It requires that there be no z-path from any checkpoint from the set of local checkpoints to the other. Moreover, a checkpoint can never be a part of any consistent global checkpoint if it is involved in a zigzag cycle (z-cycle), i.e., the checkpoint has a z-path to itself.

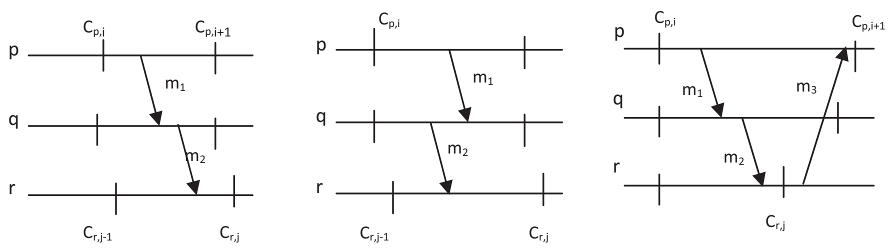
The ith checkpoint at a process, p is referred to as $C_{p,i}$ and the period between the ith and $(i + 1)$st checkpoints at a process as the ith checkpoint interval. A zigzag path (Netzer and Xu, 1995) exists from a checkpoint $C_{p,i}$ to a checkpoint $C_{r,j}$ if there exists a message sequence $m_1, m_2, \ldots m_n$ $(n \geqslant 1)$ such that

1. m_1 is sent by process p after $C_{p,i}$.
2. If m_k $(1 \leqslant k \leqslant n)$ is received by process q, then m_{k+1} is sent by q in the same or a later checkpoint interval either before or after the receipt of m_k.
3. m_n is received by process r before $C_{r,j}$.

Fig 1a depicts a causal path from $C_{p,i}$ to $C_{r,j}$ due to the message chain, $m_1,$ m_2 while Fig 1b depicts a non-causal z-path from $C_{p,i}$ to $C_{r,j}$ due to the message chain m_1, m_2; where the message m_2 is sent by q before the receipt of m_1 in the same checkpoint interval. Zigzag paths do not always represent causality; hence, a checkpoint C may be involved in a zigzag cycle if there is a zigzag path from C to itself. The receipt of m_3 by process p in Fig. 1c completes a z-cycle involving $C_{r,j}$ due to the message chain m_1, m_2, m_3; where m_1 is sent before m_3 is received and in the same checkpoint interval of process p. It has been proved that a global checkpoint, formed of a set S of local checkpoints, is consistent iff there is no zigzag path between any two checkpoints in S (including a zigzag cycle from any checkpoint to itself) (Netzer and Xu, 1995). This also leads to the result that a checkpoint C can belong to a consistent snapshot if and only if C is not involved in any zigzag cycle (Xu and Netzer, 1993).

In order to detect z-cycles online, the dependency information of the sender of a message needs to be available to the receiver of the message when it receives the message. However, only the information about the sender's dependencies in the causal past of the message can be appended with an outgoing message. The dependencies created at the sender by the receipt of messages after the sending of the message cannot be known at the receiver when the message is received. Since it is not possible to inform the non-causal dependencies and thus impossible to track all z-paths online (Allulli et al., 2007), we propose to use the backbone network to propagate the dependency information in the system by appending it with application messages. This approach reduces the number of useless checkpoints taken without any extra control messages.

Another approach to rollback recovery combines checkpointing with message logging to achieve asynchronous recovery of a failed process. 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, a process uses its checkpoint and logged determinants to rerun the corresponding non-deterministic events. Thus, the recovery procedure can reconstruct the process's pre-failure state exactly, i.e. beyond the latest checkpoint, by combining checkpointing with message logging. Variations of



a. Causal Path b. z-path c. z-cycle

Figure 1 Paths between checkpoints.

receiver based synchronous message logging have been described in the literature. Of these, sender based logging (Johnson et al., 1987) replaces the pessimistic logging of a message at the receiver with volatile logging at the sender's memory and thus, lowers highly the failure-free overhead of synchronous logging. Messages are kept in the volatile memory of the sender and transferred to stable storage only when the sender takes a new checkpoint.

3. Related work

The technique of checkpointing and rollback recovery has been extensively used to provide fault tolerance in wired as well as mobile distributed systems. Various coordinated as well as independent checkpointing protocols have been proposed in the literature (Elnozahi et al., 2002). The aim of any checkpointing protocol is to achieve a consistent global checkpoint in the system with a minimum checkpointing and communication overhead. Upon failure, a process should be able to recover to an error-free state which is consistent with the global system state.

Since coordinated checkpointing involves a high synchronization and message overhead, authors have worked upon approaches to achieve independent checkpointing techniques that avoid uncontrolled rollback propagation. The issue of forming consistent global checkpoints containing a given set of independent, local checkpoints by processes is addressed in Wang (1997). The authors define maximum and minimum consistent global checkpoints containing a set S of checkpoints and give algorithms based on reachability analysis on a rollback-dependency graph. Maximum and minimum consistent global checkpoints including a set of checkpoints can be constructed with the protocols in Wang (1995). The work in Manivannan et al. (1997) defines exactly which local checkpoints can be included in a consistent global checkpoint. An algorithm is presented to list all such consistent global checkpoints. The work in Xu and Netzer (1993) puts forth an adaptive independent checkpointing algorithm to detect zigzag cycles with the objective of reducing rollback propagation. If a process receives a message such that its current checkpoint has a causal path to the sender's current checkpoint, then the received message completes a zigzag cycle involving the sender's checkpoint. The process takes a checkpoint before processing the message. However, using the algorithm, each local checkpoint may still not belong to some consistent checkpoint and the domino effect could occur in the worst case. The quasi-synchronous checkpointing algorithm (Tantikul et al., 2005) combines coordinated and uncoordinated checkpointing approaches to allow processes to take checkpoints asynchronously and also to eliminate the useless checkpoints. However, their algorithm, like the work in Xu and Netzer (1993), tracks only the causal paths on-line and non causal paths are not detected. Recent works, such as Allulli et al., (2007) have built on the earlier work of Netzer and Xu (1995), Wang (1997, 1995), Manivannan et al. (1997), Xu and Netzer (1993); yet these perform by placing restrictions on the checkpoint and message pattern. It is shown in Allulli et al. (2007) that it is impossible to detect non causal z-cycles online without using control messages.

Our protocol outperforms the earlier protocols as it allows a fraction of non-causal z cycles to be detected without the use of control messages. Additionally, the size of inter-process dependency information maintained at each process is $O(N)$, where N is the backbone size or the number of clusters in the network as compared to $O(n)$, where n is the number of hosts in the system, as used by Xu and Netzer (1993). Further, all the above discussed algorithms are designed to detect useless checkpoints taken by processes in either the static wired or cellular mobile networks. The wired systems have no limitation of stable storage, have high bandwidth wired links and fixed topology. The decisions about where and how to store checkpoint information and retrieve the same at the time of recovery are not considered. Similarly, the rollback recovery schemes designed for cellular mobile systems assume unlimited and static support in the form of fixed Mobile Support Stations (Prakash and Singhal, 1996; Li and Shu, 2005; Tantikul and Manivannan, 2005). Almost every solution for cellular mobile systems delegates the task of storing checkpoints and message logs of the processes to the MSSs. Such assumptions cannot be extended to the mobile ad hoc environment and thus, the design of checkpointing and rollback recovery protocols for the ad hoc networks is challenging. The problem has received attention in the literature lately.

A quasi-synchronous checkpointing and pessimistic logging scheme for ad-hoc wireless networks is presented in Men et al. (2008). Upon failure, a process can rollback to its latest consistent checkpoint and the rollback procedure does not cause the domino effect. The checkpoint protocol proposed in Ono and Higaki (2007) 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 clustered model of Juang and Liu (2002), each cluster manager maintains a dependency matrix of size dependent on the total number of mobile hosts and clusters in the system. A mobility-aware checkpointing and failure recovery algorithm for cluster based mobile ad hoc network is described in Biswas and Neogy

(2012). A mobile node utilizes its neighboring nodes to save its checkpoint if the mobility of a node among the clusters crosses a pre-defined threshold value. The scheme prevents orphan and lost messages. However, none of these protocols attempt to reduce the size of recovery related information to be stored at hosts. Moreover, the recovery related messages are broadcast resulting in a high message overhead.

An emerging computing paradigm for the future is that of mobile Grids (MoGs), i.e., computational Grids involving mobile hosts to allow users to access the Grid and as well as to offer computing resources (Darby and Tzeng, 2010; Wang et al., 2006; Rao et al., 2006). Mobile devices can participate in Grid as a resource provider and as well as a resource recipient. The MoGs are in particular beneficial in situations where access to the wired Grid is not possible, and autonomous, collaborative computing is required. A decentralized, QoS-aware middleware for checkpointing arrangement in mobile Grid computing systems is presented in Darby and Tzeng (2010). Each mobile host (MH) sends its checkpointed data to one chosen neighboring MH, and also serves as a checkpoint storage node for another neighboring MH. The authors prove that finding a globally optimal checkpoint arrangement is NP-complete and therefore, present QoS-aware heuristics, to construct efficient checkpointing arrangements. The Reliability Driven (*ReD*) methodology of Darby and Tzeng (2010) utilizes the values of reliability for the links of each MH to converge to a checkpointing arrangement. However, the ReD does not have any provision of message logging or maintaining inter-process dependencies. Thus, the checkpoint of a failed process can be retrieved from its 'provider' at the time of recovery, but the global snapshot of a system cannot be computed as there is no record of inter-process dependencies. In comparison, our algorithm tracks inter-process dependencies and logs messages as well so that a consistent global snapshot may be computed. The *ReD* methodology assumes that a process will recover at a node which is a neighbor of its 'provider'. On the other hand, our protocol does not place such a restriction and allows a process to recover at any location in the network regardless of where its last checkpoint is stored.

A proxy-based coordinated checkpointing scheme with pessimistic message logging for fault recovery in mobile Grid systems is presented in Rao et al. (2006). The mobile hosts store checkpoints on their respective proxies running on the middleware. The system can roll back to the latest consistent global snapshot, without the direct participation of the mobile hosts, thus resulting in less processing and storage overhead on mobile device as compared to existing schemes. However, unlike our protocol, the solution in Rao et al. (2006) relies on proxies, i.e., static hosts residing on Mobile Access to Grid Infrastructure middleware which is assumed to be resource-rich.

It has been observed that none of the existing approaches to checkpointing and message logging address all the problems faced by the checkpointing nodes in ad hoc networks comprehensively. Moreover, these approaches suffer from a high message overhead, rendering the problem open to the development of efficient solutions. The presented protocol aims to provide a rollback recovery protocol which avoids useless checkpoints at processes with a low message overhead. The protocol is scalable due to the clustered backbone based system architecture and resilient to the node mobility.

4. System model

A dynamic Mobile Backbone Network is used to achieve message efficient communication among the nodes of the network. The hierarchical architecture of such a network combines backbone capable nodes (BCNs), which have superior processing and communication capability, with ordinary nodes (ONs), which may have relatively constrained capability. A virtual backbone is formed by dynamically electing BCNs to act as backbone nodes (BNs) and forming links among the neighboring BNs to achieve a connected backbone network. The remaining nodes (unelected BCNs and ONs) affiliate with a BN by executing a clustering algorithm and join its group or cluster. Mobile backbone networks were described by Rubin et al. (2004) as a solution to the scalability issues characteristic of mobile ad hoc networks and have been extensively studied and used (Srinivas et al., 2009; Craparo et al., 2011; Ju et al., 2004; Pandey et al., 2006; Ju and Rubin, 2005). Such a structure is illustrated in Fig. 2

4.1. Network backbone synthesis

For the network backbone election, we compute a weight, W, for each BCN on the basis of the parameters of residual energy, mobility rate and node degree of BCNs. The BN node selection method prefers BCNs with greater weight, i.e., higher energy, lower mobility rate and higher node degree. Let, θ be the mobility rate, γ be the residual energy and η be the node degree of a BCN. Assuming size of network as A, where A represents the operational area size

$W \propto \gamma\eta$ and $W \propto 1/\theta$

Thus, $W = k * (\gamma * \eta/\theta)$; where k is a constant equal to the inverse of size of the network.

A number of backbone election algorithms have been proposed in the literature (Ju et al., 2004; Pandey et al., 2006; Ju and Rubin, 2005; Wu and Li, 1999). We adapt the election algorithm known as the *MBN Topology Synthesis Algorithm* (Ju and Rubin, 2005) for our system as it converges in $O(1)$ time and its message complexity is of the order of $O(1)$ per node. A BCN will convert to BN if it needs to provide client coverage for its neighboring BCNs or to increase the

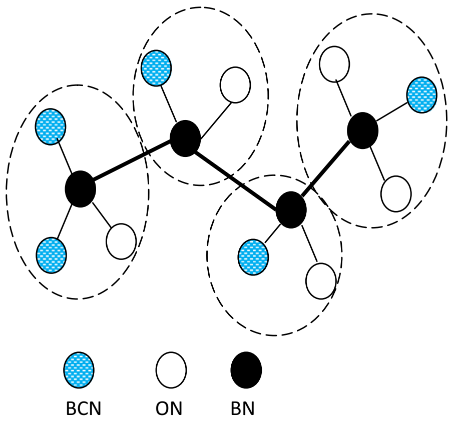


Figure 2 System model.

connectivity among its neighboring BNs. A BN will convert back to BCN if it finds that it is not required for client coverage or local connectivity.

Firstly, a BCN tries to identify the BN with the highest weight in its 1-hop neighborhood to affiliate with and sends an affiliation request to it. In case there is no neighboring BN, the node attempts to affiliate with the neighboring BCN, including itself, with the highest weight. Every node sends periodic beacon messages to its neighbors. A BCN appends its weight and the id of the BN to which it has affiliated, with the periodic message. Every node (ON, BCN or BN) also includes its list of adjacent BNs in the message and thus, shares its full 1-hop neighborhood and 2-hop BN neighborhood knowledge with its neighbors.

A BCN x identifies itself as a BN if any of the following conditions are satisfied at a BCN x:

(1) BCN x has the highest weight among its unaffiliated BCN neighbors or BCN x has received one or more affiliation requests.
(2) Two or more of its BN neighbors are not directly connected (say, BN y and BN z) and BCN x has the highest weight among its BCN neighbors (say, BCN u) that can connect those BNs as in Fig. 3 a.
(3) At least one of its BN neighbors (say, BN y) and one of its BCN neighbors (say, BCN z) do not connect to each other directly or through BNs and (i) BCN x has the highest weight among all of its BCN neighbors that can connect y and z and (ii) none of the BCN neighbors of node x (say, BCN u) can directly connect to BN y and to at least one of BCN z's BN neighbors as in Fig. 3b.

4.2. Routing in the network

There has been an extensive research on routing in mobile ad hoc networks. A survey of routing protocols for ad hoc networks is available in Royer and Toh (1999) and Boukerche et al. (2011). The routing protocols fall in the categories of on-demand and proactive protocols. The route selection is initiated by the sender only when it has a packet to transmit in the on-demand protocols. Conversely, with proactive protocols, mobiles periodically exchange routing control packets and update their routing tables. The on-demand or the reactive approach results in lesser control packets and adapts to changes in topology, but leads to longer delay in route setup before a packet may be sent. In contrast, proactive protocols need to maintain routing tables, independent of traffic load, and thus may have a high overhead when data traffic is lower than mobility rate. It is also possible that in a dynamic network, the pre-computed route is incorrect, leading to potential lost packets. Hence, the performance of proactive protocols degrades in large networks. On the other hand, though reactive protocols provide better scalability these protocols suffer from the broadcast storm problem due to the flooding approach used in the route discovery process; causing redundancy and collision problems. Some reactive routing protocols (Khamayseh et al., 2011) have made an effort to reduce the effects of the broadcast problem by restricting the rebroadcast messages on the slow moving and low loaded nodes.

Geographic routing or location-based routing has received considerable attention in the ad hoc environment. Geographic routing is particularly of interest, as it does not require any routing tables and once the position of the destination is known, all operations are strictly local and independent of remotely occurring topology changes. In this approach, it is assumed that every node knows its own and its network neighbors' positions (with the aid of positioning systems). Moreover, the source of a message is assumed to be informed about the position of the destination. Geographic routing algorithms that provide guarantee of reaching the destination are based on faces, continuous regions separated by the edges of planar network sub graphs. The first geographic routing algorithm that guarantees delivery was Face Routing (Kranakis and et al., 1999; Bose and Morin, 1999). This protocol walks along faces of planar graphs and proceeds along the line connecting the source and the destination. Face routing has been combined with *greedy forwarding* where each node forwards the message to be routed to its neighbor located "best" with regard to the destination. GOAFR+ (Kuhn et al., 2008) is a combination of greedy routing and face routing. The algorithm tries to route in a greedy manner, if it encounters local minima with respect to the distance from the destination, it switches to face routing.

Locating mobile nodes contributes to the checkpointing and recovery costs and therefore, the presented protocol utilizes the routing protocol of the network. Our approach uses GOAFR+ for routing messages destined for nodes not

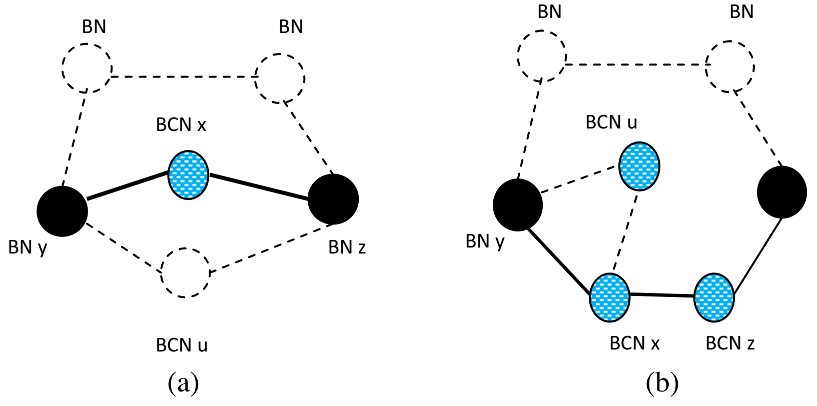


Figure 3 Network backbone synthesis. (a) BCN to BN conversion: Condition 2. (b) BCN to BN conversion: Condition 3

present in the same cluster as the sender. None of the approaches to checkpointing have utilized the routing protocol in the network for an efficient implementation. Once the network backbone is synthesized, the interconnections among the BNs can be modeled as a graph G (V, E). For the location based routing algorithms, the network graph is required to be planar, i.e., without intersecting edges. A planar graph features faces, which are contiguous regions separated by the edges of the graph. A Gabriel Graph (Gabriel and Sokal, 1969) can be computed in order to achieve planarity on the unit disk graph. The Gabriel Graph can be computed locally on the unit disk graph as a network node can determine all its incident nodes in by just an inspection of its neighbors' locations.

When a node has to send an application message to another node, it sends it to the BN to which it is affiliated. If the destination node is also in the same cluster, the BN forwards the message to it. Otherwise, the BN uses GOAFR+ routing protocol to route the message in the backbone network. When a node in the backbone network receives this message, it checks if the destination node is within its cluster. If not, then it again routes the message using GOAFR+, else it forwards the message to the destination node. It may be noted that though GOAFR+ has been assumed, the presented checkpointing and recovery protocol is independent of the underlying routing protocol. The presented protocol can exploit any MANET routing protocol with suitable adaptation.

4.3. Failure model

The mobile environment is constrained due to the characteristics of MHs as well as the wireless links. MHs possess limited computational resources, such as processor capability and storage capacity. The effective wireless bandwidth available for MHs is also limited and dynamic; being dependent on the wireless technology, the number of MHs sharing the wireless link etc. These characteristics affect the availability and connectivity of the MHs. Transient failures are the most likely failures of MHs in the mobile environment. A frequent cause of transient failures is the limitation of battery power. We assume a crash-recovery model for MHs, both ONs and BCNs, i.e., if a MH crashes, it stops receiving or sending messages until its recovery is complete. We assume that the failure frequency of BNs is lower than of ONs. When a BN fails, another BCN converts from BCN to BN (due to the rules of Section 4.1) to keep the backbone connected. The proposed algorithm presented in the next section handles the BNs' failures effectively.

5. Proposed algorithm

The presented scheme combines checkpointing with controlled sender based message logging to deliver a low overhead rollback recovery procedure.

5.1. Sender based message logging at a BN

The sender based logging requires processes to log their sent messages in the limited volatile memory as the recovery process at a recipient node may need messages to be replayed from the log. The proposed protocol requires a BN to log in its volatile memory any message sent by a MH in its cluster before routing it to the destination. Since the messages sent by a node are routed through the BN, no extra communication overhead is placed for logging them at the BN. However, the size of the message log may outgrow the size of volatile memory at the BN. It is not straightforward to determine the duration for which a message should be present in the sender's log and after which it may be removed. Therefore, controlled message logging is applied; where a message is removed from the sender BN's log on receiving the information that this message will not be required by the receiver again.

We apply a simple strategy employing the routing protocol; where a node sends an acknowledgment (*ack*) to the senders of those messages which will never be required to be resent. The *ack* will be the highest sequence number of the message received by the node from the sender before the latest cluster checkpoint. Such messages were received by the node prior to its latest checkpoint and therefore, will not be required to be resent by the sender in future.

Instead of sending any extra *ack* message, the *ack* is piggybacked on any subsequent application messages being routed from its BN to nodes located along the same face as the sender. Any BN, source or intermediate, along the route of an application message can append the acknowledgment for some message received earlier by it. On receiving this *ack*, a BN removes the message from its log. At the time of checkpointing, the BN transfers a MH's volatile message log to the stable storage along with its checkpoint in a single write; thus avoiding the overhead of synchronous logging.

Theorem. *Controlled sender based message logging removes only the log information that will not be used for recoveries in the future.*

Proof. We prove this by contradiction. Assume that a message, m sent from process p to q is removed from p's log after q sends an *ack*, though, m may be useful for q's recovery in future.

When a process q takes a checkpoint C, as in Fig. 4, it sends an *ack*, in the form of the sequence number, say SN_{qp}, of the latest message, m' it received from some process p in the previous checkpoint interval. On receiving the *ack*, p removes m' from its log. If q fails, it restarts its execution from its latest checkpointed state. To reinstate its state as just before failure, it needs messages which it had received after its latest checkpoint. The sequence numbers and the ids of the senders of such messages are saved at q (in *RCVD_LST* as described in the next section). The sequence number of such a message from p (here m'') $> SN_{qp}$ and hence, p need not resend m' to q. Thus, m' is not useful for q's recovery again. This contradicts the hypothesis. □

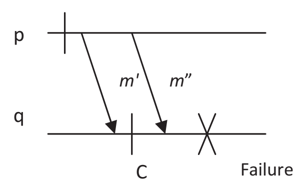


Figure 4 Sender based message logging.

5.2. Checkpointing

Considering the hierarchical architecture of the clustered network, different checkpointing techniques can be used within and between the clusters. The nodes within a cluster are affiliated to the same BN and hence, can synchronize their checkpointing procedure. In this case, a set of checkpoints, one from each node in the cluster forms a consistent local checkpoint for the cluster. The failure of a cluster implies the failure of one or multiple nodes in its cluster. Further, a set of local checkpoints, one from each cluster, forms a global checkpoint. However, it is not feasible to coordinate each global checkpoint due to the highly dynamic nature of a MANET and hence, the checkpointing at each cluster is independent of the others. A global checkpoint will be consistent if and only if there do not exist any inter-cluster orphan messages between any pair of local checkpoints.

Each mobile host (MH), either ON or BCN, affiliates itself with a BN for routing the application messages in the network. This BN, in addition, serves as the host's *Checkpoint and log Storage Node* (*CSN*) to save the checkpoint and message log of the MH. In order to take a local checkpoint in a cluster, a BN broadcasts a *take_chkpt* message to the nodes in its cluster. In response, each MH transfers its checkpoint to the BN (which is also its *CSN*) which then stores the checkpoint in its own stable storage. A local checkpoint at a cluster is taken periodically or in case the receipt of a message by some MH in the cluster completes a z-cycle involving the sender node. When a new checkpoint is taken in a cluster, the previous checkpoints for the MHs are deleted from the stable storage at the *CSN*.

Message passing among clusters creates dependencies between the checkpoints of various clusters. The receipt of a message by a process may render useless the checkpoint at the sender process if the sender's checkpoint is involved in a z-cycle. Therefore, in order to detect the formation of z-cycle involving the checkpoint of a message's sender, the dependency information of the sender should be available to the receiver. The proposed protocol aims to eliminate any control messages during the checkpointing process and consequently the dependency information of a MH node is appended with the application messages.

The inter-process dependencies of a cluster are stored at the BN in a list, namely the *Dep_List*, of maximum size *N*, where *N* is the number of BNs in the network. The records in this list for a BN correspond to the BNs on which it depends. Each record stores a BN's id and the index of its checkpoint on which this BN depends. The sender BN saves the destination's id in a *Sent_list* for the current checkpoint interval (CI). It generates a unique sequence number for the message and appends this sequence number, its own id, its *Sent_list* and its *Dep_List* with the message. Along the route from the source to destination of a message on the network backbone, each intermediate BN checks if the destination node is affiliated to it. If it is not, it checks if the *Sent_list* includes any node affiliated to it, in which case, the BN updates its own *Dep_List* using the appended *Dep_List* (The BN takes a component-wise maximum for common records and adds the records for the BNs not existing in its *Dep_list*). It removes its member from the *Sent_List* in the message before forwarding the message. When the message reaches the BN of the destination node, this BN updates its own *Dep_list* by using the *Dep_list* appended to the message before sending the message to the destination MH. A MH on receiving the message saves the sender node's id and the message sequence number in a *RCVD_LST* maintained in its own stable storage.

The presented scheme may not be able to prevent all useless checkpoints as the information about all non-causal dependencies may not reach their intended destinations by this method. It has been proved that it is impossible to track all z-cycles online without the use of control messages (Allulli et al., 2007). Our algorithm detects the z-cycles involving causal as well as non-causal dependencies between checkpoints at different processes. The simulation results show that up to 34% of z-cycles are detected by this algorithm without using any control messages. Moreover, the performance of the algorithm improves as the network traffic increases. The complete checkpointing and message logging algorithm is presented next.

Data Structures

- Used at a BN

Aff_List$_i$: list of nodes affiliated with a BN *i*

Dep_List$_i$ (of maximum size *N*, where *N* is the number of BNs in the network): Each record of this list stores a BN's id and the index of its checkpoint on which the BN *i* depends.

Sent_list$_i$: list of MHs to which messages have been sent in a CI by BN *i*

Sent_flag$_i$: Set to 1 when a message is sent in a CI by BN *i*

- Used at a MH

RCVD_LSTk $\langle BN\ id,\ seq\ no\rangle$: stores the id of a BN from which a message has been received in the current Checkpoint Interval by the MHk and the sequence number of the latest message received.

CHK$_k$: to save the id of the BN holding the checkpoint of the MH$_k$

Checkpointing & Message Logging Protocol

When it is time to checkpoint in cluster p, actions performed by the BN$_p$

- Broadcast a *take_chkpt* message in the cluster p.
- On receipt of the checkpoint and *RCVD_LST* of each affiliated node, for each message, *m* received in the previous checkpoint interval from some cluster *q*,
  if an application message, *m'* is destined for *q*
    then append acknowledgment of *m* with *m'*
  else send a control message to *q* for acknowledging the receipt of *m*

On receiving take_chkpt message from its BN, actions performed by each node, MH$_k$, in the cluster

- Save own checkpoint at the BN$_p$
- Send *RCVD_LST* to BN$_p$
- Set *CHK$_k$* = BN$_p$

When MH$_i$ affiliated to BN$_p$ sends an application message, m to MH$_j$ affiliated to BN$_s$, actions performed by the BN$_p$

- Append *p* and *Dep_list$_p$* to *m*
- If *Sent_flag$_p$* = = 1, append *Sent_list$_p$* to *m*
- Route *m* using GOAFR+

When BN_s receives m destined for MH_j, actions performed by the BN_s

```
• If $MH_j \notin Aff\_list_s$ {
    If ($\exists$k: $m.Sent\_list.MH_k \in Aff\_list_s$)
        Update $Dep\_List_s$ using $m.Dep\_list$
        Remove $MH_k$ from the $m.Sent\_List$
        Forward $m$}
    } else if ($MH_j \in Aff\_list_s$) {
    Update $Dep\_List_s$ by using $m.Dep\_list$
    If ($Dep\_List_p[s] == CI_s$)
        {/* message completes a zigzag cycle */
        Take a new checkpoint
        Set $Sent\_flag_s = 0$ and clear $Sent\_list_s$}
    Save $m$ in message log for $MH_j$
    Forward $m$ to $MH_j$}
```

When MH_j receives m from BN_p, actions performed by the BN_p

```
• If p $\in RCVD\_LST_j$
    Set $RCVD\_LST_j \langle p, seqno \rangle$ = m. seqno
    else
    Save $\langle$p, m.seqno$\rangle$ in $RCVD\_LST_j$
```

If a BN converts its state to BCN

- It affiliates to a new BN and transfers the message logs of the nodes to the new BN.
- The new BN broadcasts a *take_chkpt* message for its affiliated nodes to take a local checkpoint in the cluster.

5.3. Asynchronous recovery of a mobile host

The recovery process of a mobile host is completely asynchronous as it does not require any other node to rollback and hence, is completely domino-free. We consider the various scenarios of recovery of a mobile host.

Case 1: The crashed node recovers and affiliates to the same BN as before failure

The recovery related data of the MH are available at the current BN and hence, no control messages are required.

Case 2: A failed node affiliates to a different BN upon recovery

The current BN needs to retrieve the checkpoint from the *CSN* of the node before failure. The id of the BN holding the latest checkpoint of a node, MH_k is available as CHK_k in the node's own stable storage.

Step 1: The current BN firstly locates the *CSN* in the network in the following manner. If an application message is destined for some node along the same direction as the required *CSN* of the recovering node, the current BN appends the recovery related information with the application message. The current BN appends a *recover* field, the required *CSN*'s id and the recovering node's id with it. If the application message carrying the recovery information is destined for some BN which is reached before the *CSN*, it retrieves and appends the recovery related information with some other application message being sent along the same face as of the required *CSN*.

A separate control message will have to be sent only when there is no such application message at some BN. Our simulation results show that under normal traffic conditions, approximately 33% of the recovery procedures do not require control messages.

Step 2: Once the *CSN* for the node is located, the current BN retrieves the checkpoint from the *CSN*.

Step 3: Each node in the *RCVD_LST* of the MH is sent the highest sequence number of messages received from it prior to the latest checkpoint. Any message, with a higher sequence number, sent earlier to the recovering node will be still available in the log of the sender node. This message will be resent to the recovering node again.

Step 4: The recovering node then rolls back to the retrieved checkpoint, replays the messages received and thus reconstructs its state just before failure.

Apart from eliminating the need for control messages each time a node is recovering, the routing assisted recovery process presented has multiple additional advantages. Firstly, it enables the simultaneous recovery of multiple nodes. This is possible as any BN on the path from source to destination BN may append the information about another failed node along with the application message. Further, the scheme is also resilient to changes in network topology. The recovery process by BNs uses face routing to locate the checkpoint and members of *RCVD_LST* of the node. The required BNs may currently not be a part of the backbone; but they will still be affiliated to some BN and hence can be located.

Theorem. *Recovery process does not create orphans in the system and leads the system to a consistent state.*

Proof. The recovery of a process results in the creation of an orphan message if the send event of a message is undone due to the rollback of the sender but the receive event is not undone as the receiver is executing normally. Considering the receipt of a message at a process *p* as a nondeterministic event, *e*, the following are defined:

(1) Depend(e) is the process, *p* and those set of processes whose state depends on the event *e* according to Lamport's happened before relation.
(2) Log(e) is the set of processes that have logged a copy of e's determinant in their volatile memory.
(3) Stable(e), is a predicate that is true if e's determinant is available in stable storage (Elnozahi et al., 2002).

A process *p* becomes an orphan if *p* itself does not fail and *p*'s state is dependent on the execution of some nondeterministic event *e* whose determinant is neither available in stable storage nor in the volatile memory of a surviving process. Formally

$$\forall(e) : \neg Stable(e) => Depend(e) \subseteq Log(e)$$

This property is called the *always-no-orphans* consistency condition (Elnozahi et al., 2002).

In the proposed recovery scheme, if a BN j has received a message for a MH affiliated with it in a checkpoint interval, then some BN i (whose id is present in the MH's *RCVD_LST*) must have logged the message content and the corresponding send event in its stable storage. This is because controlled sender based logging is utilized which ensures that the message is not removed from the sender's log till the receiver has taken its next checkpoint. Hence, the determinant of a nondeterministic event is always available upon recovery and the nature of the backbone network ensures that it can be retrieved by the recovering process despite the dynamic network topology. Therefore, there cannot be any orphan messages and the protocol satisfies the *always-no-orphans* condition.

The recovery of a MH is completely asynchronous as it does not require the rollback of any other host in the system. The MH's state before failure can be reconstructed independently by restoring the latest checkpoint from its *CSN* and replaying the messages from the log at the nodes in its *RCVD_LST*. Since there are no orphan messages, it is guaranteed that the pre-failure execution of the MH is repeated and any message, whose send event was undone due to the rollback of the MH, will also, be resent during the MH's recovery. Therefore, the recovering process is able to reconstruct its state consistent with the system state.

6. Global snapshot

Global snapshot or global checkpoint computation is an essential problem of distributed computing. It finds application in fault tolerance of long-executing programs by providing an intermediate consistent global checkpoint of the system. In case a failure occurs, the system can restart from the saved checkpoint in place of restarting the execution of the program from the initial state. Global snapshots are also employed for monitoring stable properties of the system, such as termination detection, deadlock detection, loss-of-a-token, etc. (Chandy and Lamport, 1985; Garg et al., 2010).

A global checkpoint can be constructed by a set of local checkpoints, one per cluster. Since the local checkpoints in each cluster are independent of each other, any arbitrary collection of local checkpoints may not be consistent. The presented procedure will form a consistent global checkpoint by requiring only those clusters to take an additional checkpoint which have sent some message in the current checkpoint interval. Other clusters include their latest local checkpoint in the global checkpoint. This eliminates the formation of orphans. The procedure for the construction of global checkpoint in the system is as follows:

```
Global_chkpt_initiation()
//Executed by BN i to initiate global checkpointing
1. If $send\_flag_i == 1$, BN i takes a checkpoint with sequence number, say $n$, in its cluster.
2. BN i executes chkpt_req_propagation(i, n).
3. Wait for a reply (checkpoint or Deny) from each neighboring BN.
4. On getting replies from all its neighbors, complete the global checkpointing process.
```

```
Chkpt_req_propagation(i, n)
//Executed by a BN to propagate a global checkpoint request
1. If an application message is being sent to any neighboring BN, except the sender, append a flag take_global $(i, n)$ with it.
2. Send a control message with the flag take_global $(i, n)$ to a neighboring BN to which no application message was sent in step 1.
```

```
Global_chkpt(j)
// Executed by each BN j
On receipt of a message with the take_global flag by BN j
1. if $global\_chkpt\_taken_j == 0$ && $send\_flag_j == 1$, $BN_j$ takes a checkpoint.
       else if $global\_chkpt\_taken_j == 1$, send Deny to sender, exit.
2. Set $global\_chkpt\_taken_j = 1$, include latest checkpoint in the global checkpoint.
3. Execute chkpt_req_propagation(i, n).
4. Wait for a reply (checkpoint or Deny) from each neighboring BN.
5. On getting replies from all its neighbors, send own checkpoint to the sender.
```

The number of additional checkpoints required to be taken by the clusters may be further reduced by delaying the taking of the checkpoint by a BN till the time it receives replies from its neighbors. If the periodic checkpointing time period gets over by that time, the periodic checkpoint will be a part of the global checkpoint.

7. Performance study

The proposed protocol combines checkpointing with controlled sender based message logging in order to prevent orphans, limit the rollback propagation at the time of recovery and eliminate the domino effect. Another salient characteristic of the protocol is that, considering stable storage is limited in a MANET, each cluster is required to save only its latest checkpoint. Moreover, controlled sender based logging ensures that any message is logged at a sender BN for a finite duration only. The recovery can handle multiple, concurrent failures in the system. Comparing the proposed scheme with the techniques of Tantikul and Manivannan (2005), Xu and Netzer (1993), while the latter can track only the causal z-cycles, the proposed algorithm can handle causal as well as non-causal z-dependencies. In the best scenario that can be envisaged, the algorithm can detect all z-cycles online. The backbone based architecture of the system makes it feasible to scale the protocol to large systems efficiently.

7.1. Message complexity

A local checkpoint creation for a single cluster requires a single broadcast message by the BN of the cluster. Thus, for a system with N clusters, local checkpointing requires N broadcast messages. The propagation of dependency information in the system does not involve any control message as this information is appended with the application messages. The construction of a global checkpoint requires the collection of local checkpoints, one from each cluster. In the best case, the global checkpoint collection request would be tagged with the application messages and no control message is required. However, the worst case would require one *take_global* message along each face of the initiator cluster. If the initiator has m neighbors, this results in m control messages. Further, each recipient of the *take_global* message propagates the message to all its neighbors except the sender. If it is assumed that the BNs are fully connected, i.e., each BN is a neighbor of all the other BNs in the network or $m = N - 1$, there would be m *take_global* messages by the initiator and $m*(m - 1)$ *take_global* messages by the neighbors of the initiator in case no application message is being sent. Thus, the global checkpoint creation requires m^2 control messages in the worst case. However, in a normal scenario and considering average application message traffic, the message complexity of the algorithm will be much lower than $O(m^2)$, where m is the average number of faces or neighbors of any given BN. It may be noted that the average number of neighbors of a given BN, $m < N \ll n$, where N is the number of clusters or total BNs in the system and n is the number of nodes in the system. Thus, the protocol does not add a significant message overhead even in the worst case.

The recovery procedure for a node requires locating the CSN for the node. In the best case, this can be done by appending the required data along with application messages and hence, requires no control messages. In the worst case, one control message will be sent to retrieve the recovery related data of a node from its CSN. Each node in the *RCVD_LST* of the MH is also sent a message to retrieve the message log for the recovering node. Thus, the recovery process results in $k + 1$ control messages, where k is the number of processes in the *RCVD_LST* of a node. Since, $k \ll n$, the communication overhead of the recovery process of a single node is insignificant.

7.2. Comparison with related work

The proposed protocol utilizes the routing protocol existing in the network for the design of an efficient checkpointing and controlled sender based message logging mechanism. To evaluate the performance of the proposed recovery protocol, we have compared our protocol with the related checkpointing schemes for purely mobile computing Grid (Darby and Tzeng, 2010), cellular mobile system (Tantikul and Manivannan, 2005) and a wired distributed system (Xu and Netzer, 1993). The work of Darby and Tzeng (2010) is related to our algorithm as both store checkpoints at neighboring MHs and do not rely on any static hosts or access points for the checkpointing protocol. The communication induced checkpointing and selective message logging protocol (Tantikul and Manivannan, 2005) is comparable to our protocol as it has a low checkpointing and message overhead; although it utilizes MSSs for the implementation. We also compare our protocol with the adaptive algorithm of Xu and Netzer (1993) which is an early representative work on z-cycle detection methods. Table 1 summarizes the salient features of checkpointing and rollback based recovery protocols across the various schemes.

7.3. Simulation experiments

Simulation experiments have been performed to analyze the performance of the presented protocol using the Network Simulator, ns2. The *MBN Topology Synthesis Algorithm* (Ju and Rubin, 2005) has been used for the backbone synthesis. We consider representative values from Ju and Rubin (2005) for the parameters to assess our approach. All nodes in the network use random waypoint mobility model. These nodes are randomly distributed in a rectangular 1500×1500 operational area and the communication range of each node is 300 m. Each node in the simulation has different pause time randomly distributed between 0 and 600 s. The movement of the nodes may lead to a change from one cluster to another cluster. The message sending rate of a process follows an exponential distribution with a rate $\lambda_c = 5$ and for each message sending event, the recipient of the message is selected randomly. The process also takes a checkpoint with a fixed time interval of $C_c = 200$. The failure rate of a MH follows an exponential distribution with a rate $\lambda_f = 1/300$ and upon failure; the MH instantly performs the proper action for the recovery. For the simulation, the network size has been kept as 100 nodes and 25% of the mobile nodes are backbone capable. The topology synthesis algorithm with BCN-to-BN restricting rules results on an average in a backbone size of 10.

We firstly measure the percentage of z-cycles detected by our checkpointing protocol under average data traffic conditions. The dependency information appended with the application messages helps to detect z-cycle formation. The number of z-cycles prevented by the algorithm corresponds directly to the number of useless checkpoints prevented at processes. The algorithm can detect about 34% of the z-cycles, without the addition of overhead to the application execution (since no control messages are used), as shown by Fig 5. In comparison, the schemes of Tantikul and Manivannan, (2005), Xu and Netzer (1993) can detect only the z-cycles which result due to causal dependencies between processes. The simulation results show that the schemes of Tantikul and Manivannan, (2005), Xu and Netzer (1993) detect 20% of the z-cycles on the average in the simulated network conditions.

For complete z-cycle detection, additional control messages are required. The proposed algorithm does not employ any control messages and uses the application messages to propagate inter-process dependencies. Therefore, as expected, the performance of the protocol improves significantly if the application message sending rate at the processes increases. When there are a high number of application messages, there is a high probability of the availability of outgoing application messages to which the dependency information can be appended. This ensures that nodes will receive the latest dependency information without any message overhead and z-cycles will be detected. Fig. 6 shows the increase in the number of detected z-cycles as the application message traffic grows in the network. However, there is no corresponding performance improvement in the schemes of Tantikul and Manivannan, (2005), Xu and Netzer (1993) as can be observed from Fig 7.

Fig. 8 demonstrates the performance of the simulated recovery procedure. The recovery related control information is tagged with the application messages, where possible. It can be seen that approximately 33% of recovery procedures do not require control messages as the recovery requests were appended with outgoing application messages. The other instances of recovery procedure generated control messages for propagating the recovery requests in the network. As compared, if the application messages are not utilized by the recovery procedure, as in Tantikul and Manivannan (2005), a higher message overhead is added to the system, as can be observed from Fig 8.

Further, for a constant failure rate, an increase in the application message sending rate decreases the number of recovery

Table 1 Comparison of rollback recovery schemes.

Features	Proposed scheme	Decentralized QoS aware scheme Darby and Tzeng, 2010	Communication induced checkpointing and selective message logging (Tantikul et al., 2005)	Adaptive checkpointing (Xu and Netzer, 1993)
Type of checkpointing	Uncoordinated inter-cluster checkpointing and coordinated intra-cluster checkpointing	Independent	Communication induced checkpointing	Periodic checkpointing combined with forced checkpoints to avoid z-cycles
Type of message logging	Controlled sender based logging	No message logging	Selective message logging at receiver	Pessimistic logging of all messages
Asynchronous recovery	Possible	Inter-process dependencies not considered	Possible, but causes processes dependent on the recovering process also to rollback	No, domino effect possible in worst case
Assumption about availability of fixed host	No	No	Yes, MSSs used	Yes, algorithm for a static system only
Location of recovery of a MH	MH can recover at any location in the network	MH can recover at a location which is a neighbor of checkpoint storage node or the 'provider'	MH can recover at any location in the network	Same location as where node crashed as fixed hosts assumed
Useless checkpoints avoided	Yes, both due to causal and non-causal z-cycles	Not considered as global snapshot not being created	Yes, but only causal dependencies tracked	Yes, but only causal dependencies tracked
Control messages required by the protocol	1 broadcast message in each cluster at the time of checkpointing; $k + 1$ control messages for recovery of a node, where k is the number of processes in the *RCVD_LST* of a node in the worst case, 0 messages for recovery in the best case	Checkpoint relation request and break messages of $O(n)$ at each node, where n is the number of neighbors of the node	None at the time of checkpointing, $O(l)$ messages for recovery, where l is the number of processes to which some message was sent after the latest saved checkpoint	None at the time of checkpointing, $O(n)$ for recovery, where n is the number of processes to which the MH had sent application messages to, after the latest checkpoint it has rolled back to
Size of dependency information required to be saved	$O(N)$ where N is the backbone size or the number of clusters in the network	Not considered	An integer, i.e. the sequence number of current checkpoint	$O(n)$ where n is the number of nodes in the network
Global snapshot collection	Possible	Not considered	Possible, but algorithm not discussed	Possible, but algorithm not discussed

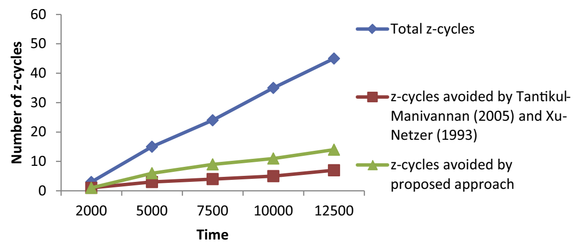


Figure 5 Performance of the z-cycle detection algorithm.

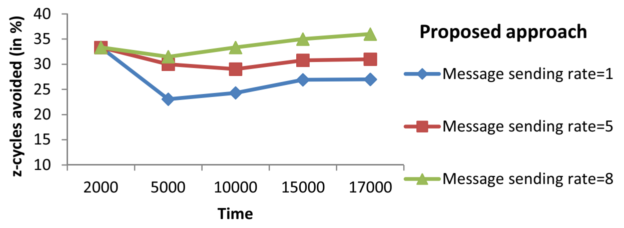


Figure 6 Effect of increase in data traffic on the performance of proposed protocol.

messages required by the proposed algorithm, as depicted by Fig 9. As the number of outgoing messages at the processes increases, fewer recovery messages will be required to be sent. However, the increase in the application message sending rate does not affect the performance of Tantikul and Manivannan (2005)

Fig. 10 depicts the number of control messages required by the proposed protocol for the recovery of hosts in a given time under varying rates of failure and a constant network traffic rate, $\lambda_c = 5$. It is observed that though an increase in the failure rate increases the number of control messages required; yet the increase is not significant.

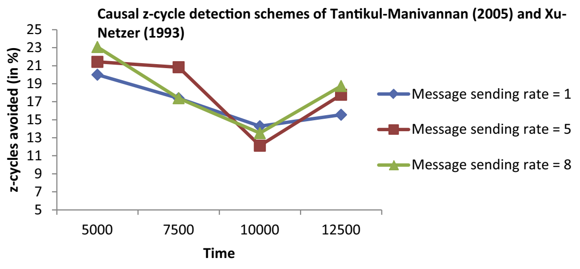


Figure 7 Effect of increase in data traffic on the performance of the scheme of Tantikul et al. (2005) and Xu and Netzer (1993).

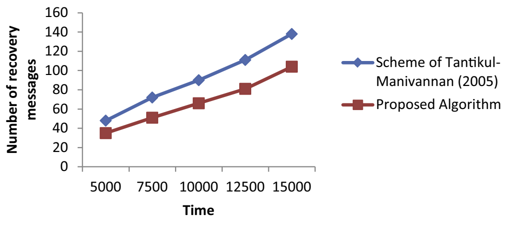


Figure 8 Performance of the recovery protocol.

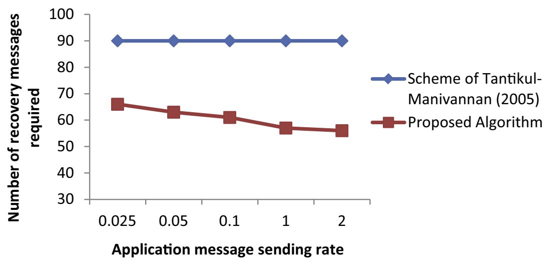


Figure 9 Effect of increase in application message sending rate on the recovery message overhead.

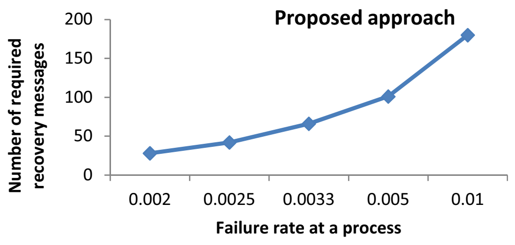


Figure 10 Effect of increasing failure rate at a process on the recovery message overhead.

8. Conclusion

The paper presented a checkpointing and asynchronous rollback recovery protocol for nodes to provide fault tolerance in a backbone based mobile ad hoc network. The checkpointing and recovery method is integrated with the underlying routing protocol of the network and hence, the proposed protocol does not need any control messages at the time of checkpointing. Moreover, the performance of the checkpointing protocol improves as the network traffic increases since a higher number of z-cycles are detected online. The amount of prevented z-cycles enhances the efficiency of our protocol in terms of storage requirement as it reduces the number of useless checkpoints taken by the nodes. The protocol allows a completely asynchronous and domino-free recovery of the mobile nodes and also avoids sending control messages during the recovery process. We also used the presented checkpointing scheme to develop an efficient algorithm for the computation of a consistent global snapshot of the system. The presented protocols are scalable to large systems due to the backbone based architecture used for the system. The implementation of the presented protocols will make it possible to utilize the computing capabilities of mobile nodes connected in an ad hoc fashion without access to any wired or cellular network.

References

Allulli, L., Baldoni, R., Laura, L., Piergiovanni, S.T., 2007. On the complexity of removing Z-cycles from a checkpoints and communication pattern. IEEE Trans. Comput. 56 (6), 853–858.

Biswas, S., Neogy, S., 2012. Checkpointing and recovery using node mobility among clusters in mobile ad hoc network. Adv. Intell. Syst. Comput. 176, 447–456.

Bose, P., Morin, P., Stojmenovic, I., Urrutia, J., 1999. Routing with guaranteed delivery in ad hoc wireless networks. In: Proc. 3rd Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications. 48–55.

Boukerche, A., Turgut, B., Aydin, N., Ahmad, M., Bölöni, L., Turgut, D., 2011. Routing protocols in ad hoc networks: a survey. Comput. Netw. 55 (13, 15), 3032–3080.

Busching, F., Schildt, S., Wolf, L., 2012. DroidCluster: towards smartphone cluster computing – the streets are paved with potential computer clusters. In: 32nd International Conference on Distributed Computing Systems Workshops. 114–117.

Chandy, K.M., Lamport, L., 1985. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. 3 (1), 63–75.

Craparo, E.M., How, J.P., Modiano, E., 2011. Throughput optimization in mobile backbone networks. IEEE Trans. Mob. Comput. 10 (5), 560–572.

Darby, P.J., Tzeng, Nian-Feng, 2010. Decentralized QoS-aware checkpointing arrangement in mobile grid computing. IEEE Trans. Mob. Comput. 9 (8), 1173–1186.

Elnozahi, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B., 2002. A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. 34 (3), 375–408.

Gabriel, K.R., Sokal, R.R., 1969. A new statistical approach to geographic variation analysis. Syst. Zool. 18 (3), 259–270.

Garg, R., Garg, V.K., Sabharwal, Y., 2010. Efficient algorithms for global snapshots in large distributed systems. IEEE Trans. Parallel Distrib. Syst. 21 (5), 620–630.

Jipping, M., Lewandowski, G., 2001. Parallel processing over mobile ad hoc networks of handheld machines. In Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing. 267–270.

Johnson, D.B., Zwaenepoel, W., 1987. Sender-based message logging. Proc. of Fault Tolerant Computing Systems, pp. 14–19.

Ju, L.H., Rubin, I., 2005. Performance analysis and enhancement for backbone based wireless mobile ad hoc networks. BROADNETS. 789–798.

Ju, H., Rubin, I., Ni, K., Wu, C., 2004. A distributed mobile backbone formation algorithm for wireless ad hoc networks". Proc. IEEE Int. Conf. Broadband Netw., 661–670

Juang, T.T., Liu, M.C., 2002. An efficient asynchronous recovery algorithm in wireless mobile ad hoc networks. J. Internet Technol. 4, 143–152.

Khamayseh, Y., Obiedat, G., Yassin, M.B., 2011. Mobility and load aware routing protocol for ad hoc networks. J. King Saud Univ. – Comput. Inf. Sci. 23 (2), 105–111.

Kranakis, E., Singh, H., Urrutia, J., 1999. Compass routing on geometric networks. In: Proc. 11th Canadian Conf. Computational Geometry. 51–54.

Kuhn, F., Wattenhofer, R., Zollinger, A., 2008. An algorithmic approach to geographic routing in ad hoc and sensor networks. IEEE/ACM Trans. Netw. 16 (1), 51–62.

Li, G., Shu, L., 2005. A low-latency checkpointing scheme for mobile computing systems. In 9th Annual International Computer Software and Applications Conference COMPSAC. 2 (225). 491–496.

Manivannan, D., Netzer, R.H.B., Singhal, M., 1997. Finding consistent global checkpoints in a distributed computation. IEEE Trans. Parallel Distrib. Syst., 623–627

Men, C., Xu, Z., Li, X., 2008. An efficient checkpointing and rollback recovery scheme for cluster-based multi-channel ad hoc wireless networks. In: Proc. of the ISPA'08. IEEE Computer Society, pp. 371–378.

Netzer, R.H.B., Xu, J., 1995. Necessary and sufficient conditions for consistent global snapshots. IEEE Trans. Parallel Distrib. Syst. 6, 165–169.

Ono, Higaki, H., 2007. Consistent checkpoint protocol for wireless ad-hoc networks. The 2007 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA. 1041–1046.

Pandey, A., Ahmed, M.N., Kumar, N., Gupta, P., 2006. A hybrid routing scheme for mobile ad hoc networks with mobile backbones, In: 13th International Conference on High Performance Computing, 411–423.

Prakash, R., Singhal, M., 1996. Low-cost checkpointing and failure recovery in mobile computing systems. IEEE Trans. Parallel Distrib. Syst. 7 (10), 1035–1048.

Randell, B., 1975. System structure for software fault tolerance. SIGPLAN Not. 10 (6), 437–449.

Rao, I., Imran, N., Woo Lee, P., Huh, E., Chung, T., 2006. A proxy based efficient checkpointing scheme for fault recovery in mobile grid system. In: Proceedings of the 13th International Conference on High Performance Computing (HiPC'06), 448–459.

Royer, E., Toh, C.K., 1999. A review of current routing protocols for ad-hoc mobile wireless networks. IEEE Pers. Commun. 6, 46–55.

Rubin, A., Behzad, H., Ju, R., Zhang, X., Huang, Y.-C., Liu, R., Khalaf, 2004. Ad hoc wireless networks with mobile backbones. In: Proceedings of IEEE International Symposium on Personal, Indoor and Radio Communications, vol. 1, 566–573.

Srinivas, A., Zussman, G., Modiano, E., 2009. Construction and maintenance of wireless mobile backbone networks. IEEE/ACM Trans. Netw. 17 (1), 239–252.

Tantikul, T., Manivannan, D. A communication-induced checkpointing and asynchronous recovery protocol for mobile computing systems. In: Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2005, vol. 2005, 70–74.

The Akogrimo Project, http://www.akogrimo.org, 2010.

Wang, Y.M., 1995. Maximum and minimum consistent global checkpoints and their applications. In: Proc. 14th IEEE Symp. Reliable Distributed Systems. 86–95.

Wang, Y.M., 1997. Consistent global checkpoints that contain a given set of local checkpoints. IEEE Trans. Comput. 46 (4), 456–468.

Wang, Z., Chen, Q., Gao, C., 2006. Implementing grid computing over mobile ad-hoc networks based on mobile agent. In: Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops, USA, 321–326.

Wu, J., Li, H., 1999. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In: Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications. 7–14.

Xu, J., Netzer, R.H.B., 1993. Adaptive independent checkpointing for reducing rollback propagation. Proc. Fifth IEEE Symp. Par. and Distributed Processing. 754–761.