Cross-Layer Cluster-Based Data Dissemination for Failure Detection in MANETs

David Kidston and Li Li
Canada Communications Research Centre
Ottawa Canada
{david.kidston, li.li}@crc.ca

Walee Al Mamun and Hanan Lutfiyya
Department of Computer Science
The University of Western Ontario
London, Canada
{walee, hanan}@csd.uwo.ca

Abstract— **Node failures may be frequent in MANETs, but there can be many different causes for those failures. Nodes may lose power, crash, or simply move out of range of other nodes in the network. Identifying the root cause is complicated by a lack of fixed monitoring and analysis infrastructure. Past research has focused on monitoring using either ping, heartbeat, or gossip-based approaches, which can incur significant network wide overhead. This paper proposes a novel k-hop cluster based data dissemination scheme that can piggyback on routing messages for more efficient detection of failures including node disconnection. In this scheme, nodes forward their neighbour-hood observations to a per-cluster failure detector based on the observed spanning tree. Simulations show that detecting disconnected nodes using a cross-layer implementation of the data dissemination scheme is more efficient while an application layer implementation is faster. This effect is more pronounced in sparse networks.**

Keywords- cross-layer, data dissemination, network efficiency, failure detection, MANET

I. Introduction

A mobile ad hoc network (MANET) is an autonomous system of mobile nodes connected by wireless links. Wireless links provide lower bandwidth and higher error rates compared with fixed networks. These nodes may be resource constrained, have limited battery power, and because of mobility must continuously monitor and react to changes in their transmission neighbourhood.

The combination of self-organization and resource constraints suggests the need for self-management and adaptation. One aspect of this is recognising and being able to react to system behaviour that deviates from the desired behaviour. This is referred to as a *failure* or *symptom*. The cause of a failure is a *fault* or a *root cause*. A symptom by itself should not necessarily be used to determine a corrective action since a symptom may be explained by more than one fault. *Fault localization* is the process of taking a set of observations of system state and using these observations to determine a fault.

Fault localization in MANETs is complicated by its dynamic topology. At a given point of time t, the topology of the MANET can be described as a directed graph $G_t = (V_t,E_t)$ where V_t is the set of nodes and E_t is the set of links at time t. For any two nodes $u,v \in V_t$, $(u,v) \in E_t$, if the transmitter of u can reach v. The nodes u and v are said to be one hop neighbours. MANET topologies may change for a variety of reasons other than node mobility: a node's battery may be down, a node's radio transmitter may fail, a node may be shutdown, etc.

Previous research in failure detection in MANETs typically used either a unicast heartbeat-based data dissemination approach, which has high network-wide bandwidth overhead, or more recently a gossip-based dissemination approach with increased delay [4]. Previous work in this area has provided only limited analysis of the impact of mobility on data dissemination for the detection of faults.

This paper proposes a novel protocol for data dissemination within a cluster such that all observations are received at the cluster head with minimal overhead. The detector acts as a cluster head for a cluster where no node is more than k-hops away (k-cluster). By building a spanning tree rooted at the detector, changes in connectivity detected by any node can be quickly and efficiently collected for further analysis at the detector using a data gathering protocol. This allows decision-making to be carried out by the cluster head since it has the information to build models that provide a global perspective of system behaviour. We have simulated two related approaches to this scheme. The first is a standalone implementation which produces its own messaging. The second uses extensions to Ad hoc On Demand Distance Vector (AODV) routing protocol. By adding a *detection extension header* to existing routing messages, the protocol can operate with even lower overhead.

The remainder of the paper is organized as follows. We continue in Section II with an overview of the related work on data dissemination schemes for failure detection in MANETs. Section III provides a description of our proposed data dissemination scheme. Section IV introduces a case study based on detecting nodes that lose connection to the rest of the network. In Section V we describe the Qualnet-based simulation of this case study and evaluate the two approaches of our proposed scheme. The paper ends in Section VI with a summary and a discussion of future work.

II. Related Work

There is considerable previous work on failure detection in MANETs. The following is a discussion of work focusing on information collection and dissemination techniques.

A) *Ping- and Heartbeat-Based Approaches*

Ping-based architectures for network fault detection use detector nodes to send "are you alive?" messages to which receiving nodes must reply within a certain amount of time or be considered as failed. The selection of detector and receiver nodes can be fixed in advance, dynamically assigned [9], or randomised [1]. These solutions are easy to implement but have the disadvantage of having both a ping and reply per monitored node that must traverse limited bandwidth and error-prone links.

Fault detection can be achieved with half the transmission overhead using a heartbeat architecture where receiver nodes send unsolicited "I'm alive" messages to detector nodes. A detector that does not receive a heartbeat after a certain timeout considers the sender to have failed. In MANETs heartbeat based architectures often have multiple detectors that collaborate in an overlay [8], where detectors with a defined set of nodes to monitor periodically sense each other's heartbeat messages as well, or in a fully distributed manner [13], where each receiver sends heartbeat messages towards the sender, recruiting other nodes on the path to localise where a failure occurs.

If we take the example network on the left in Figure 2, a ping-based architecture would require that the cluster head (node A) send a ping message to nodes B, C, D, E, F, and G and then a reply is sent back from each node to A. A heartbeat-based approach would require all other nodes to periodically send updates directly to A. Our approach reduces the number of heartbeat messages needed by using a spanning tree to aggregate information from each node as probes are forwarded towards the detector. This increased message efficiency comes at the cost of potentially increased detection delay.

B) *Cluster-Based Approaches*

A method for detector nodes to collaborate and reduce messaging overhead is to divide the network into dynamic clusters such that each detector is close to the monitored nodes. For example, in the scheme described in [16], all nodes first broadcast a heartbeat message and then a digest message summarising each nodes view of who was heard in the first phase. This is followed by a health status update from the cluster head identifying the failed nodes based on the analysis of the digest messages. Since messages are confined to the local cluster, this provides more efficient messaging. This solution assumes static single-hop cluster membership (no mobility). Our approach explicitly considers the case of both multi-hop clusters and node mobility. This allows us to determine if a node has failed as opposed to becoming disconnected from the MANET (see case study).

The data collection module in [5] uses a cluster head election approach to identify a single detector node. It builds a tree for collecting information from normal nodes and aggregating before sending it towards the detector in a similar fashion to our work. The difference in this case is that this work focuses on intrusion detection while we are focusing on node failures.

C) *Gossip-Based Approaches*

Yet another method for multiple detectors to collaborate is to use the gossip (epidemic) protocol to globally distribute and evaluate heartbeat messages. Gossip protocols typically broadcast information from the local node and from any gossip from other nodes it has heard about within its one-hop neighbourhood. Failure detection work that uses some form of gossip protocols MANETS includes [2], [3], [5], and [15]. Most related work in this area assumes a static topology (no mobility) allowing the global network information to converge in a reasonable amount of time. However, the work in [2] takes a broadcast-based approach that can deal with limited mobility. Using a 1-hop reliable broadcast, this method is able to detect both hard and soft faults in MANETs. By comparing the outcomes to proscribed test tasks returned by different units (neighbouring nodes), faults in those units can be detected. While this method works well for detecting errors in individual nodes, it does not scale up to network wide failures and is not appropriate for detecting node disconnection

III. Data Dissemination Scheme

This section describes our approach for data dissemination by the cluster head. The approach is based on the construction of a spanning tree for the nodes in a cluster. Topological and other failure related information is gathered using this tree. The approach is briefly described in this section.

The decision on how to respond to a failure is based on the type of fault. This analysis is assigned to specific nodes, which collect regional or global information for informed analysis. We will refer to this node as a *detector* and we will assume that each detector is associated with a set of nodes. Since the detector makes the decisions on how to respond to failures, all observations should be sent at least to the detector node and not necessarily all other nodes.

A) *Construction of the Spanning Tree*

The spanning tree construction algorithm is presented in Figure 1. Each node executes this algorithm. Line 1 indicates that the node is waiting for an event, which in this case is message arrival. The spanning tree construction is initiated when the cluster head sends a HELLO message to its one hop neighbours. Each node has variables that represent the node's parent and the node's children. When a node, *u*, receives a HELLO message from its neighbour (line 3:), it checks to see if it has a parent. If a node receiving the HELLO message has no parent it designates the sender of the HELLO message as its parent and sends its parent an OK message (line 6:). A node's parent is the node that it receives its first HELLO message from. When a node receives an OK message (line

11:) it adds that node to its child set. A node knows it is a leaf node if it does not receive any OK messages within a pre-defined interval time in response to the HELLO messages it sent. We refer to this pre-defined interval as the OK timeout.

Algorithm 1 Spanning Tree Construction Algorithm

```
SpanningTreeConstruction(event)
Input: event
 1:   Switch (event)
 2:   //node v sends a HELLO message
 3:   CASE: HELLO message
 4:       if (parent($u$) == $nil$) {
 5:           parent($u$) = $v$;
 6:           Send OK message to node $v$;
 7:           Send HELLO message one-hop
               to neighbours (except parent)
 8:       }
 9:       $N(u)_t = N(u)_t \cup v$
10:   //node u received an OK message from $v$
11:   CASE: OK message
12:       $\mathrm{Child}(u) = \mathrm{Child}(u) \cup v$
```

Figure 1: Spanning Tree Construction

Figure 2 shows a network with seven nodes: A, B, C, D, E, F, G. The left hand side of the figure shows the links between nodes in the network. All links represent one hop neighbours. Assume that node A is the cluster head. The cluster head initiates the construction of the spanning tree by sending a HELLO message to its one hop neighbours which are nodes B and C. Nodes B and C send back an OK message and set their parent variable to node A. All subsequent HELLO messages received by node B and node C are discarded. Thus both node B and node C have as its parent the node A and the children of node A are nodes B and C. This is seen in the right hand side of Figure 2. Nodes B and C both send a HELLO message since the HELLO message from node A is the first HELLO message received. Node C sends a HELLO message to its one hop neighbours nodes: D, E and F. Node B sends a HELLO message to its one hop neighbour nodes: D and G. Assume that node C's HELLO message arrives before B's HELLO message at node D. Node D sets its parent variable to node C and sends an OK message to node C. Node D discards node B's HELLO message. Assume that node C's HELLO message arrives first at nodes E and F. Both nodes set their parent variables to node C and send OK messages back to node C. All other HELLO messages received by nodes E and F will be discarded. If node D's HELLO message arrives at node G before node B's HELLO message then node B's HELLO message is discarded and node G sets its parent attribute to node D and node G sends an OK message to node D.

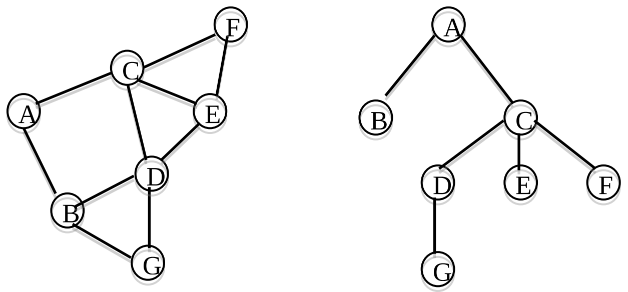

Figure 2: Network and Spanning Tree

It is possible for an alternative spanning tree to be determined. For example if node B's HELLO message is received by node G before node D's HELLO message then node G will set its parent to node B and send an OK message to node B. Graphically the right hand side of Figure 2 would have a link from node B to node G instead of there being a link between node D and node G. Thus the spanning tree depends on the order of arrival of HELLO messages.

B) Data Gathering

In this section we describe the algorithm for data gathering. This is presented in Figure 3. At the end of construction of the spanning tree each node determines if it is a leaf node or not. After a pre-defined period of time a leaf node sends its neighbourhood information to its parent using a REPORT message. The REPORT message consists of the one-hop neighbours of the sending node. For a leaf node, u, the neighbourhood information is in the form of $(u, N(u)_t)$.

Algorithm 2 Data Gathering Algorithm

```
DataGathering(event)
Input: event
 1:   $T = \emptyset$;
 2:   Switch (event)
 3:   //node $u$ receives a REPORT message ($m$) from $v$
 4:   CASE: REPORT MESSAGE
 5:       if (parent($u$) != $nil$) {
 6:           $T = T \cup$ extractNeighbourhoodSets($m$);
 7:           $\mathrm{Child}(u) = \mathrm{Child}(u) \backslash v$;
 8:           if (Child($u$) == $nil$)
 9:               Send REPORT message with
                  $T \cup (u, N(u)_t)$ to parent($u$);
10:       }
```

Figure 3: Data Gathering Algorithm

When a node receives a REPORT message from one of its children it extracts the neighbourhood information using the *extractNeigbhourhoodSets* function (line 6:). When a node receives REPORT messages from all its children it sends a REPORT message with its neighbourhood set as well as all the neighbourhood sets received from its children preprocessed to reduce the information sent (lines 7: to 10:).

This process ends with the cluster head that determines the topology from the neighbourhood sets it has received. We note that for each $v \in N(u)_t$ information that characterizes the link

(u,v) can also be sent e.g., the number of incoming and outgoing packets.

As an example, consider the network and spanning tree presented in Figure 2. The leaf nodes are B, G, E, F whose neighbourhood sets are represented as (B, {A, D, G}), (G, {B, D}), (E, {C, D, F}) and (F, {C, E}), respectively. When node D receives the neighbourhood set from node G it sends {(D, {B, C, E, G}), (G, {B, D}). When C receives all the neighbourhood sets from each of its children (nodes D, E, F) it sends {(C, {A, D, E, F}), (D, {B, C, E, G}), (G, {B, D}), (E, {C, D, F}), (F, {C, E})} to node A.

C) Topology Construction

The cluster head can construct the cluster's topology from the neighbourhood sets. We note that it is possible that for two nodes u and v, that $u \in N(t)_t$ but $v \notin N(u)_t$. The reason for this is that messages may be lost. We assume that in constructing the topology that a link is said to exist between u and v if and only if $u \in N_v(t)_t$ and $v \in N_u(t)_t$.

IV. CASE STUDY

In this section we describe how the data dissemination protocol was used to detect that a node has either moved out of range or has gone down. Further cross-layer work will be required to distinguish between these two cases. We start with a discussion of how nodes no longer visible within a cluster can be detected. Assume that the cluster head node maintains $G_t = (V_t,E_t)$ for different values of t. $D_{t+i} = V_t \backslash V_{t+i}$ (where $i > 0$) is the set of nodes in G_t but not in G_{t+i}. A simple definition of loss of visibility of a node u is if it is in D_{t+1}. However, it is a possible that a message is lost since UDP is the typical transport protocol used in MANETS. If an OK message is lost and there is only one node within transmission range then the node would be considered to no longer visible. In the next construction of the spanning tree the node may be visible. A node, u, is considered not visible if $u \in D_{t+j}$ for each value of j between 1 and n. We refer to each construction of the spanning tree as an iteration. The value of n represents the number of iterations before a node is considered not visible.

For example, assume that in the network presented in Figure 2 that node G moves. Node G may temporarily not be visible due to lost OK messages e.g., node G may be visible at D_t but not at D_{t+1} due to a loss of OK messages. Node G may become visible again at t+2 and hence will not be in $D_{t+2.}$

Note that sometimes the reported neighbourhood set of nodes is not completely accurate since nodes move. It is possible that a node will report an incorrect neighbourhood set. The topology may need multiple iterations to determine the correct topology.

V. SIMULATION AND MEASUREMENTS

In this section, we analyse the performance of the data dispersion protocol based on a Qualnet simulation of the two implementations. The parameters used in these simulations are presented in Table 1.

Parameter	*Value(s)*
Antenna Type	Omni directional
Channel Frequency	2.4 GHz
Pathloss Model	Two ray
Maximum Propagation Range	400m
Radio Type	802.11b
Data Rate	2mbps
Routing Protocol	AODV
AODV Hello Message Interval	1 second
Scenario Area	1500 x 1500m
Node Count	10, 30, 50
Number of runs	15
Percentage of Nodes Moving	10%

Table 1: Simulation Parameters

Simulations ran with 10, 30 and 50 node networks. 10% of the nodes in each case are mobile (random waypoint model) which is typically found in tactical MANETs. The results should apply to higher rates of mobility. Nodes were placed within close proximity for the 30 node scenario and more dispersed for the 10 and 50 node scenarios. Each experiment was repeated 15 times and the average outcome is given in the results. The spanning tree constructed five times in each scenario.

A) Implementation

For simulation, we have developed two versions of the data dissemination protocol. First, the cross-layer implementation makes use of the HELLO messages found in the AODV routing protocol to construct the spanning tree. We chose to start with AODV since it is best suited for resource constrained environments. AODV periodically sends HELLO messages to determine link connectivity as part of its route maintenance process. There are two issues that had to be addressed in this implementation. First, it may be desirable to construct the spanning tree only for every N AODV HELLO messages. This is a configurable parameter in our implementation. The second issue is that not all AODV HELLO messages get an OK message in response. To deal with this we added a reply flag bit to the AODV HELLO message. If the bit is one then this indicates that an OK message should be returned. The frequency of setting this bit to one is also a configurable parameter.

Second, an application layer based implementation independent of any lower level network protocols was used for comparison. All HELLO, OK and REPORT messages are produced independently.

B) Performance Results

The metric that we used to evaluate efficiency is the number of messages generated during the simulation. These are protocol-specific. The results for the application-layer implementation do not include any lower-layer packets. The results for the cross-layer protocol does not include the packets generated for the HELLO messages of the AODV routing protocol since these messages are already being generated for routing purposes. It does include the OK messages and the REPORT messages since these are not part of the AODV protocol.

Figure 4 shows that the total number of packets generated for 10, 30 and 50 node scenarios. For 10 and 30 nodes, the timeout for the OK message was 10 seconds. When this timeout was used in the 50 node case the spanning tree could not be fully constructed and the information the cluster head received was incomplete. Increasing the timeout to 20 seconds eliminated this problem. Doubling the simulation time allowed five spanning trees to be constructed as in the previous cases.

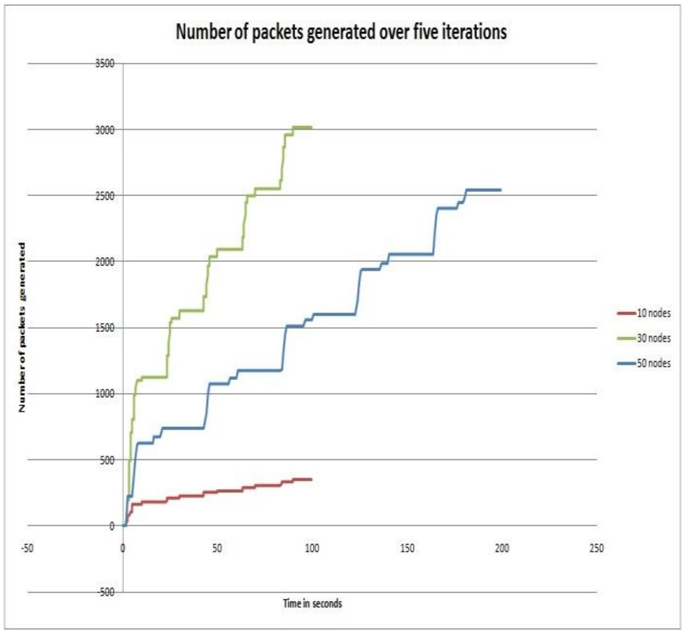

Figure 4: Total number of Generated Packets - Application-Layer

At any point in time the number of packets received is cumulative since the start of the simulation. The x-axis represents time while the y-axis represents packets generated. As can be seen in the graph the number of generated packets increases in a step-wise fashion. Note that although the graph trends are similar for each case the number of packets generated and received is smaller for the 50 node case compared to the 30 node case. The reason is that the cluster was less dense and thus the average number of edges per node was higher for the 30 nodes case. The average number of one hop neighbours for the 10 node scenario was 2, for the 30 node scenario 15, and for the 50 node scenario 8.

It should be noted that each scenario shows a sharp increase in packets generated at the start of the simulation. This represents packets generated in the initialization of the protocol simulation, and not in the discovery protocol.

While the results in Figure 4 are for the application-level implementation, Figure 5 shows results for the cross-layer implementation. The cross-layer implementation exhibits a similar trend, but with a lower overhead. Again, note the closeness of the lines for the 30 node and 50 node cases. This is again due to the low node density of the 50 node case requiring fewer messages.

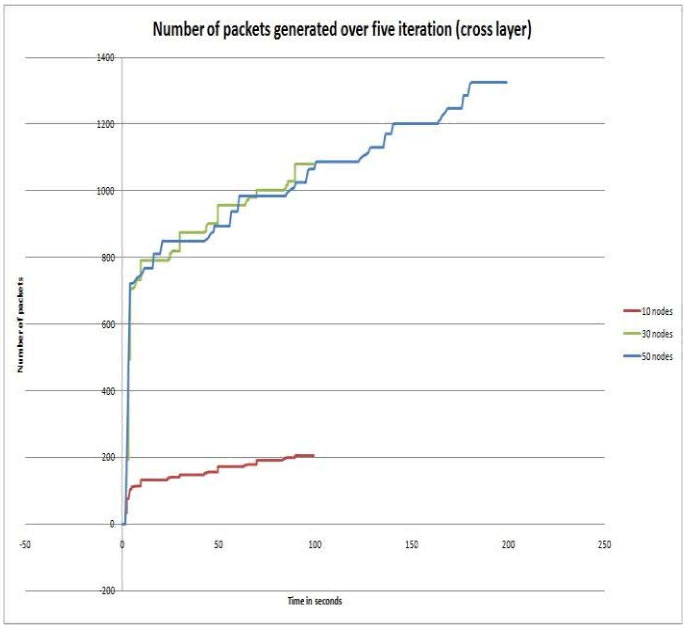

Figure 5: Total packets generated – Cross-Layer

Another view of overhead is seen with a comparison of the average number of packets received by each node. This is presented in Table 2.

Scenario	10 nodes	30 nodes	50 nodes
Cross-Layer Implementation	24	36	32
Application Implementation	36	101	52

Table 2: Packets Received per Node

As can be seen, the cross-layer implementation produces fewer packets than the application-layer implementation both globally and per-node. The reason is that the application-layer and the cross-layer implementations both generate packets for the routing protocol HELLO messages. However, the application-layer protocol generates separate HELLO messages for the spanning tree construction and thus increases the total number of packets generated. The only additional messages needed for the cross-layer implementation are the OK messages. A comparison of the average time taken to construct the spanning tree is presented in Table 3.

Scenario	10 nodes	30 nodes	50 nodes
Cross-Layer Implementation	2.59	4.12	8.03
Application Implementation	0.83	3.69	4.98

Table 3: Spanning Tree Construction Times (seconds)

As we can see in the table, for the 30 node scenario the spanning tree construction time is similar for both implementations. The reason is that for a very dense graph both implementations of the spanning tree construction take about same amount of time. However, if the graph is less dense, as in the 50 node scenario, the cross-layer approach is two times slower. For a sparser graph, e.g. 10 nodes, the cross-layer approach is three times slower. This suggests that if the graph is very dense, the cross-layer may become more responsive. A dense graph implies that a node has many edges. Since the 50 node scenario had fewer edges it generated fewer messages.

Node density also explains the results seen in Figure 5. The reason is that the cross-layer implementation uses the HELLO messages of the AODV protocol. When a node receives an AODV HELLO message it must wait until the routing protocol is ready to send out its AODV HELLO message. In the application-layer protocol the application-layer sends out its HELLO messages immediately after receiving HELLO message.

The reason why density helps is that the network diameter is typically smaller which means that the length of paths is smaller. If the network is dense then the diameter of the graph is small so spanning tree construction takes less time for the AODV implementation than if the nodes were more sparse.

C) *Detection of Single Node Movement*

The next part of our analysis focuses on the number of iterations used to detect a node becoming not visible. One of the reasons that a node may not be briefly visible is because of a lost message. Thus, it may not be feasible to consider a node down unless it is not visible for multiple iterations. The experiments show that on average the number of iterations needed to detect that a node is not visible is one iteration for 10 nodes, two iterations for 30 nodes and three iterations for 50 nodes. This is to be expected since a larger number of nodes imply a larger number of packet losses. The average number of iterations needed to determine node movement (assuming that 10% of nodes have moved) was one iteration for 10 nodes and 30 nodes and two iterations for 50 nodes. This may suggest that node movement within a cluster requires fewer iterations to determine (and hence less time). A node *u* that moves within a cluster may move out of range of one node but within the range of another node. Thus *u* is still visible although the actual topology calculated by the cluster head may be incorrect. Note that the results for the application layer and cross-layer implementation were the same.

D) *Detection Correctness - Topology*

In the case of ten nodes one node was allowed to move. This node moved within the first two iterations before moving out of transmission range of all nodes of the network. The node moved for the first two iterations. For the ten node scenario all node movement was correctly detected in all fifteen runs of the simulation in the third iteration without knowledge of a correct initial topology. Each node's movement was detected in the next iteration of the algorithm.

In the case of thirty nodes three nodes were allowed to move. Among these three nodes, one node moved out of range and the other two nodes moved to other parts of the network. Nodes moved for the first two iterations. Among the fifteen runs we were able to detect the movements in 11 cases without correct initial topology. With correct initial topology we detected the movement correctly 14 out of 15 cases. We failed to detect the movement correctly in one case that is due to the limited five iterations used. We present a summary in Table 4.

	Detected	Failed to Detect
Correct Initial Topology	14	1
Incorrect Initial Topology	11	4

Table 4: Detection Rate

We determined the movement in the next iteration in 73% of the runs. We detected the movement in the second next iteration in 18% of the runs and in the third next iteration in 9% of the simulation runs.

VI. Conclusions and Future Work

In this paper we have proposed a novel cross-layer protocol for data dissemination within a cluster such that observations are received at the cluster head with minimal overhead. This protocol is used to collect cluster-wide topology information at a detector node that acts as a virtual cluster head where the k-cluster is a sub-network of nodes of interest for failure monitoring.

Simulations show that by building a spanning tree rooted at the detector, changes in topology detected by any node can be more efficiently collected for further analysis using a cross-layered implementation. An application layer implementation, which produces its own messaging, was found to have higher overhead but was faster and more accurate at detecting nodes that have moved out of range of the k-cluster. These effects were magnified in networks with lower node density.

The work presented in this paper considers only a single k-cluster. However, the results suggest that having multiple clusters may be feasible for larger networks. For future work we will be investigating inter-cluster head communication of topological information. While this will increase the network side detection overhead, we expect it to remain an improvement over centralised detection schemes. Another alternative we are planning to investigate is the use of a localised broadcast to track down potentially disconnected nodes such as the scheme presented in [6]. We will also incorporate the splitting of large clusters as needed based on the work presented in [14].

We are also investigating the use of other protocols such as OLSR and NHDP as a substitute for AODV. Finally, we are investigating how this scheme can be used to detect and respond to and distinguish between different types of failure ([7], [11])) using cross-layer information (e.g., SNR values [12]) and policy-based response.

Acknowledgement.

This work was supported by Defence R&D Canada (DRDC).

REFERENCES.

[1] T. Chandra, G. Goldszmidt and I. Gupta “On Scalable and Efficient Distributed Failure Detectors”, Proceedings of ACM symposium on Principles of Distributed Computing, 2001.

[2] S. Chessa and P. Santi, “Comparison-based system-level fault diagnosis in ad hoc networks”, Proceedings of IEEE Symposium on Reliable Distributed Systems, 2001, pp. 257-266.

[3] M. Elhadef and A. Boukerche, “A Failure Detection Service for Large-Scale Dependable Wireless Ad-Hoc and Sensor Networks”, The Second international Conference on Availability Reliability, and Security, 2007.

[4] M. Elhadef and A. Boukerche, “A Gossip-Style Crash Faults Detection Protocol for Wireless Ad-Hoc and Mesh Networks,” Proceedings of the IEEE Performance Computing and Communications Conference (IPCCC), 2007, pp. 600-605.

[5] M. Elhadef, A. Boukerche and H. Elkadiki, “Performance Analysis of a Distributed Comparison-Based Self-Diagnosis Protocol for Wireless Ad-Hoc Networks” MSWiM’06, 2006.

[6] O. F. Gonzalez, M. Horwath and G. Pavlou, “Detection And Accusation of Packet Forwarding Misbehavior In Mobile Ad-Hoc Networks”, Journal of Internet Engineering, Vol. 2, No. 1, June 2008, pp. 181-192.

[7] A. Hadjiantonis, A. Malatras, G. Pavlou, “A Context-Aware, Policy-Based Framework for the Management of MANETS”, Proceedings of the Seventh IEEE International Conference on Policies for Distributed Systems and Networks, 2006.

[8] D. Khedher, R. Glitho and R. Dssouli, “A Novel Overlay Based Failure Detection Architecture for MANET Applications”, Proceedings of IEEE International Conference on Networks, 2007, pp. 130-135.

[9] D. Liu and J. Payton, “Adaptive Fault Detection Approaches for Dynamic Mobile Networks,” IEEE Consumer Communications and Network Conference, 2011, pp. 735-739.

[10] M. Natu and A. Sethi, “Intrusion Detection System to Detect Wormhole using Fault Localization Techniques”, Proc. WORLDCOMP SAM'07, International Conference on Security and Management, June 2007.

[11] M. Nodine, G. Levin, K. Kurachik, D. Woelk, Y. Lin, R. Chadha, J. Chaing, K.Moeltner, Y.Kumar, S. Ali, R. Bauer, “Distributed Diagnosis of Network Faults and Performance Problems for Tactical Miltary Networks”, IEEE Military Communications Conference (MILCOM 2010), pp. 1647-1652.

[12] M. Nodine, G. Levin K. Kurachik, D. Woelk, Y. Lin, R.Chadha, J. Chaing, K. Moeltner, T. D’Silva, and Y. Kumar, “Issues with and Approaches to Network Monitoring and Problem Remediation in Military Tactical Networks”, IEEE Military Communications Conference (MILCOM 2009), pp. 1-7.

[13] R. Sivakami and G.M.K. Nawaz, “Reliable Communication for MANETs in Military through Identification and Removal of Byzantine Faults,” International Conference on Electronics Computer Technology, 2011, pp. 377-381.

[14] W. Song, S. Rehman, H. Lutfiyya, “A Scalable PBNM Framework for MANET Management”, IEEE/IFIP International Conference on Integrated Network Management, 2009

[15] N. Sridhar, “Decentralized Local Failure Detection in Dynamic Distributed Systems”, Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems, pp. 143-154, 2006.

[16] A. Tai, K. Tso, W. Sanders, “Cluster Based Failure Detection Service for Large-Scale Ad Hoc Wireless Network Applications”, Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks, 2004, pp. 805-814.