The Mechanisms to Decide on Caching a Packet on Its Way of Transmission to a Faulty Node in Wireless Sensor Networks Based on the Analytical Models and Mathematical Evaluations

Mahdi Nasrullah Al-Ameen
MD. Rakib Hasan
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
E-mail: mahdi.cse.buet03@gmail.com, rakib.cse.buet03@gmail.com

Abstract—**The behavior of sensor networks is highly unpredictable because of randomness in individual node state and network structure. When a node fails, the packets transmitted to this faulty node is dropped if it is not cached by an operational node. In this paper some analytical models based on fault tolerance mechanisms have been proposed to be used in different aspects of caching a packet regarding this situation. The ratio of the time to be allocated to cache the packet by an operational node and the expected repair time of the faulty node represents a threshold that decides whether this operational node should cache the packet or it will pass the packet to a selected neighbor node. Analytical models developed in this paper have made detail and clear inspections in these respects. Distributed caching mechanism has been described in this paper, which makes a way to use the energy of the nodes in a distributed fashion to cache a packet. The models and mechanisms, described in this paper are distributed and highly scalable, which ensure energy efficient caching of a packet and thus highly applicable to the sensor networks.**

Keywords-Sensor networks, Fault tolerance mechanisms, Caching packets of faulty nodes, Allocation factor for caching, Expected reinstallation time of a faulty node, neighbor selection for caching, Distributed caching mechanism.

I. INTRODUCTION

A sensor network consists of hundreds of tiny battery-powered sensing nodes communicating via radio. The sensors are deployed throughout the sensor field to collect information or data for a base station [4].

When a node fails in sensor networks, maximum connectivity among nodes and efficient data transmission can be ensured by using fault tolerance mechanisms. In this case, a packet transmitted to the faulty node by a source, should be cached by an operational node (let us say this node: c) which is within the communication path of this packet and directly connected to the faulty node.

Each node acknowledges the failure of its neighbor node (let us say this node: f) by using a watchdog timer, which does not receive 'alive signal' from node: f at a predefined interval for a specific number of times, when node: f becomes faulty. Afterwards when node: f is repaired and reinstalled to operation, a 'reinstalled message' is sent by node: f to acknowledge all its neighbors.

During the period of time when node: f is in 'repair state' after the failure, a number of packets may have been transmitted to this node by source nodes and the mechanisms described in this paper have guided to an energy efficient approach to cache these packets by operational nodes so that node: f can receive the packets when it is reinstalled to operation.

Hence this paper has made a clear inspection on the mechanisms of minimizing the effect of the failure of a node on the network system. And the whole approach to be undertaken in this case has been represented by analytical models with necessary mathematical evaluations.

II. CONSIDERED FACTORS FOR CACHING A PACKET

When a packet (let us say this packet: p) transmitted to node: f by a source node (let us say this node: s) arrives at node: c, the decision on caching packet: p is made by node: c based on the value of a threshold T_{H}, which is calculated from the following equation.

$$T_H = \frac{t_c}{t_o} \quad (1)$$

t_c = Available time for caching packet: p by node: c.

t_o= Expected time for node: f to get repaired and reinstalled to operation after the failure.

A. *Computation of t_c*

t_c is computed from the value of E_{ac}, which stands for the energy to be allocated by node: c for caching packet: p.

The value of E_{ac} is calculated from the following equation.

$$E_{ac} = \alpha\left[E_a - \left(E_{th} - E_e\right)\right] \tag{2}$$

Here,

E_a = Current available energy of node: c.

E_{th} = Amount of energy of node: c, which is not available for caching as it is preserved by this node for other services and operations.

E_e=Amount of energy; already used from E_{th}.

α = Allocation factor.

At this point on getting the value of E_{ac}, t_c can be easily calculated because the time for caching a packet is fixed when E_{ac} has been determined. Thus in this paper we have concentrated on determining the factors to calculate the value of E_{ac}.

The value of α is dependent on the following factors (considering the current case).

- Number of neighbors of node: c.
- Number of packets; currently cached by node: c.
- Priority of node: f.
- Priority of packet: p.
- Availability of node: c at the expected reinstallation time of node: f.

A detail analysis on the mentioned factors has guided to the development of analytical models to compute the value of α and so on the value of E_{ac}.

1) Number of Neighbors

Due to energy constraints of sensor networks, a node can allocate energy to a certain extent for caching packets. So, a threshold (n_{max}) is considered by any node in this case. When the number of neighbors is less than n_{max}, less energy is likely to be allocated for caching with the increase in the number of neighbors. But if the number of neighbors is greater than n_{max}, this factor (number of neighbors) has a constant effect on the allocation of energy for caching, because a minimum amount of energy must be allocated for caching a packet and if a large number of neighbors is considered to be served at their faulty state, in fact no packet can be cached because of insufficient energy, allocated for caching a packet. The following membership function defines the effect of number of neighbors on E_{ac}.

$$\mu_1 = \begin{cases} \dfrac{1}{n}, & \text{for } n_{\min} \le n < n_{\max} \\ \dfrac{1}{n_{\max}}, & \text{for } n \ge n_{\max} \end{cases} \tag{3}$$

n = number of neighbors.

n_{min} = minimum possible number of neighbors of any node in the network.

n_{max} = maximum possible number of neighbors that can be served by a node considering its energy constraint.

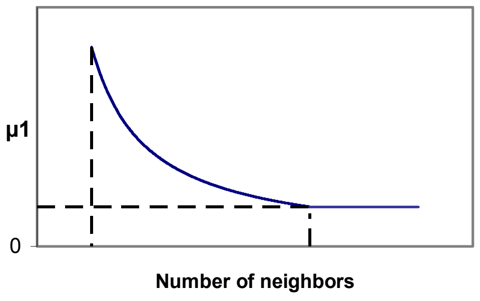

Figure 1. Effect of number of neighbors

2) Number of Currently Cached Packets

To decrease network jam and communication delay due to the overload of packets on a node, heuristics can be applied depending on the portion of buffer; used for caching. Considering,

n_{th} = A threshold, regarding number of cached packets.

n_b = Maximum number of packets allowed to be cached at a time (considering buffer size).

n_c = Current number of cached packets.

At this point, the following scenarios have to be considered.

• $n_c < n_{th}$: In this case, this attribute (number of cached packets) has no negative effect on caching a newly arriving packet.

• $n_{th} \le n_c < n_b$: In this scenario E_{ac} is likely to be decreased as n_c increases. Again the change of negative effect on E_{ac} gradually increases. Thus quadratic function, rather than simple linear function, can be applied to define this criterion.

•$n_c \ge n_b$: No more arriving packet can be cached in this case.

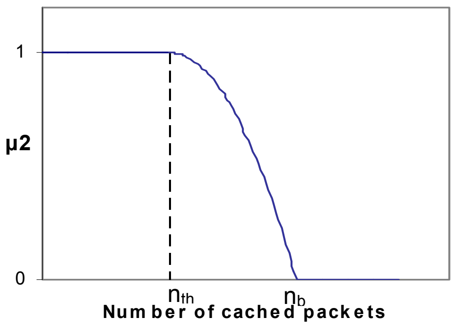

Figure 2. Effect of number of cached packets

The corresponding membership function can be defined as follows.

$$\mu_2 = \begin{cases} 1, & \text{for } n_c < n_{th} \\ 1-\left(\dfrac{n_c - n_{th}}{n_b - n_{th}}\right)^2, & \text{for } n_{th} \le n_c < n_b \\ 0, & \text{for } n_c \ge n_b \end{cases} \quad (4)$$

3) Priority of the Faulty Node (P_f)

The monitoring node (the root node of the network tree) [1] sets a priority for each node in the network after the topology discovery phase [1]. The priority of a node may depend on its number of neighbors, importance of its coverage region and other pre-specified considerations. The priority of a node may be updated by the monitoring node at any time and each time a 'priority set' message with the revised priority of this node (let us say this node: p) is broadcasted. In this case, only the neighbor nodes of node: p store the priority of node: p and other nodes ignore this 'priority set' message in this case.

Suppose the priority of the faulty node is P_f. In this case, E_{ac} is linearly dependent on P_f.

$$\mu_3 = \frac{P_f - a_{\min}}{a_{\max} - a_{\min}} \quad (5)$$

This equation defines the effect of P_f where,

a_{min}= minimum possible priority of any node.

a_{max}= maximum possible priority of any node.

4) Priority of the Arriving Packet (Pp)

Each transmitted packet contains a priority, which may depend on the type of information contained in the packet. Even it may be set by the source node depending on the importance of reaching the packet to the destination node. In this case, E_{ac} is linearly dependent on P_p.

The effect of P_p can be defined by the following equation.

$$\mu_4 = \frac{P_p - b_{\min}}{b_{\max} - b_{\min}} \quad (6)$$

b_{min}= minimum possible priority of any packet.

b_{max}= maximum possible priority of any packet.

5) Availability of Node: c at the Expected Reinstallation Time (T_d) of the Node: f

To define the effect of availability, Let T_{fi} and T_{ri} be the expected instants of time for the node c to be faulty and then reinstalled, respectively. T_c is the current time.

If T_{r0} is the last reinstallation time, then node c is likely to be faulty at T_{f1}. In this case, we will use two well recognized term for any device or system: Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

According to their definition, MTBF = $T_{f(i+1)}$ - T_{ri} and MTTR = T_{ri} - T_{fi} . Probability of being operational (P_t) at any time depends on the elapsed time from T_{r0}. Using probabilistic measure we can assume that P_t is constant (P_r) until the midpoint of MTBF of this node. The value of P_r is maximum and equal to the Reliability of c.

$$P_r = \frac{MTBF}{MTBF + MTTR} \quad (7)$$

After midpoint it is expected to decrease exponentially to 0. But at T_{f1}, c is likely to become faulty. As a result P can decrease from P_r to P_{th} (probability of being operational just before T_{f1}) and then from P_{th} it sharply decreases towards 0 since caching memory is volatile i.e. node c will lose all its buffer data after being faulty.

Let t = T_d-T_{r0}. Then the membership function can be defined as follows.

$$\mu_5 = P_t = \begin{cases} P_r, & for \quad t \le \dfrac{MTBF}{2} \\ P_r e^{\frac{-(2t-MTBF)}{MTBF}\ln(\frac{P_r}{P_{th}})}, & for \quad \dfrac{MTBF}{2} < t \le MTBF \\ P_{th} e^{-(t-MTBF)}, & for \quad t > MTBF \end{cases} \quad (8)$$

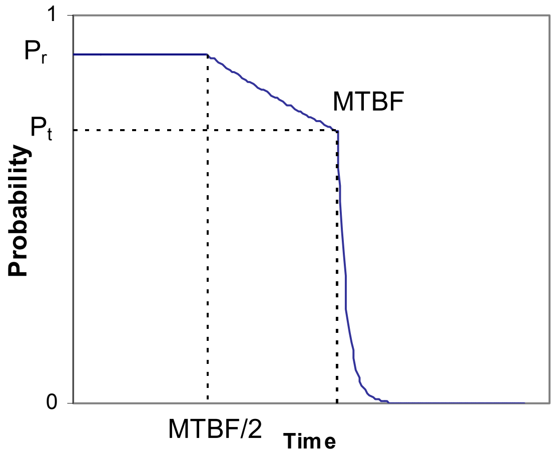

Figure 3. Availability of node c (for volatile buffer)

Now, if we use non-volatile buffer the situation becomes a bit more complex.

The scenario can be visualized as follows.

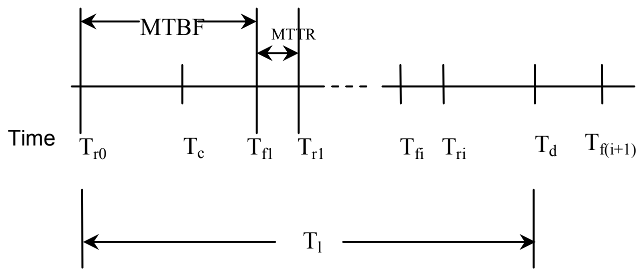

Figure 4. Node failure and recovery

Considering Fig. 4, we have to find P_t at T_d considering the fact that node: c can become faulty and again operational, several times, when node: f is in 'faulty state'. As a result, here

$$t = (T_d - T_{r0}) \% (MTBF + MTTR) \quad (9)$$

Since MTBF and MTTR are statistically measured in this case, T_{fi} and T_{ri} are not likely to be defined in practice. As a result we cannot use the previous relation to find P_t. In this case, probability will decrease in both directions from the midpoint (MTBF/2).

Other situations are similar to the previous. Since the curve in Fig. 5 is symmetric,

$$t = |((T_d - T_{r0}) \% (MTBF + MTTR)) - (MTBF/2)| \quad (10)$$

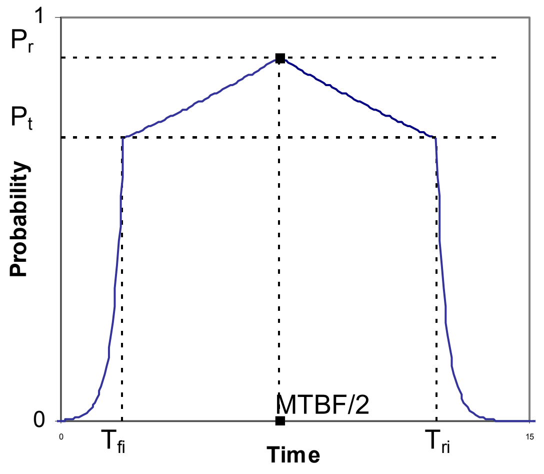

Figure 5. Availability of node c (for non-volatile buffer)

In this case, membership function is defined as,

$$\mu_5 = P_t = \begin{cases} P_r e^{\frac{-2t}{MTBF}\ln(\frac{P_r}{P_{th}})}, & for \quad t \le \frac{MTBF}{2} \\ P_{th} e^{-(t-\frac{MTBF}{2})}, & for \quad t > \frac{MTBF}{2} \end{cases} \quad (11)$$

6) Calculating α Using Fuzzy Evaluation

Five criteria have been described for the evaluation of α and each criterion is assigned a numerical evaluation by fuzzy membership function. Thus α can be calculated by aggregation of these five criteria. So,

$$\alpha = \sum_{i=1}^{5} w_i \mu_i \quad (12)$$

Where w_i ($w_i \ge 0$) denotes the corresponding weights for criterion: i (i = 1… 5) such that

$$\sum_{i=1}^{5} w_i = 1 \quad (13)$$

To note, a dominant criterion should have a higher value as its weight.

B. Computation of t_o

Initially the value of t_o for each node is set by the monitoring node and a message is broadcasted with a node-id (let us say this node: e) along with its t_o. The neighbors of node: e stores this value of t_o and other nodes ignore the message in this case. Later on when node: e fails, its neighbors keep track of the instant of time of its failure (t_f) and the instant of time of its getting reinstalled to operation (t_r). From the difference $|t_r$-$t_f|$, the time required for node: e to be in operational state after the failure is calculated and stored by each of its neighbor nodes. If node: e fails multiple times the mean of $|t_r$-$t_f|$ is calculated, which actually represents t_o for node: e. So,

$$t_o = \frac{nt_{op} + t_{oc}}{n+1} \quad (14)$$

n=number of times a node has failed before the current ((n+1)th) failure.

t_{op}=Expected time for a node of being reinstalled to operation after the failure, considering first n-1 failures.

t_{oc}= The time for a node of being reinstalled to operation after the n-th failure.

C. Significance of T_H

T_H drives the decision of caching a packet and different scenarios may come to consideration according to the value of T_H.

1) Scenario 1

When $T_H \geq 1$, maximum possible allocation time (t_c) for caching is greater than t_o. In this case, packet is cached for maximum t_o period of time. If node: f is reinstalled to operational state before t_o elapses, the cached packet is transmitted to node: f. Thus it is an event driven approach where the faulty node receives its cached packet as soon as it is reinstalled to operation.

But if t_o elapses before node: f comes to operational state two different actions can be taken depending on the value of T_H:

- According to the first action, cached packet is passed to node: f after t_o elapses. If it is the case that node: f is reinstalled to operation but its 'reinstalled message' has not reached node: c, then passing the cached packet to node: f after t_o elapses increases the probability for node: f to receive this packet.
- According to the second action, packet is transmitted for caching to a selected neighbor node, which is within the overlapping region of the coverage areas of node: c and node: f. In fact, this action is taken when the value of T_H reflects the high importance of the packet to reach node: f.

A neighbor node is also required to be selected for caching in a case when node: c cannot cache a packet any more before t_o elapses due to the energy constraint, arising from sudden energy requirement for other services or operations.

2) Scenario 2

When $T_H<1$ the maximum possible allocation time for caching (t_c) is less than t_o. So node: c cannot cache the packet with a good probability of having it cached until node: f is reinstalled to operation. Thus one of the described actions can be taken in this case depending on the value of T_H.

- Packet is passed for caching to a selected neighbor node, which is within the overlapping region of the coverage areas of node: c and node: f with enough available energy to cache this packet until t_o elapses.
- Packet is just dropped if no such neighbor with enough available energy is found to cache the packet. The packet also may be dropped due to its low and negligible importance to reach node: f.
- Distributed caching mechanism is followed to cache the packet if no such neighbor is found with enough available energy to cache the packet. Details on this mechanism have been described later.

D. Selection of Neighbor Node for Caching

In the neighbor selection mechanism considering scenario: 1, currently computed t_c of node: c is also needed for comparison with the t_c of its neighbors. Hence node: c may cache the packet for more period of time if it is found more energy efficient in comparison to its neighbors.

When a neighbor node is to be selected for caching a packet, node: c broadcasts a 'neighbor selection' message to its neighbor nodes with the node-id of node: f and the period of time (t_{ac}) that has been already spent for caching this packet. A neighbor of node: c which is not the neighbor of node: f, ignores the message. But a neighbor of node: c which is also a neighbor of node: f computes the value of t_c for the packet to be cached after getting the 'neighbor selection' message. In this case, if $t_c<t_o$ is found no reply is sent to node: c. But if $t_c \geq t_o$ is found, this neighbor node replies with the value of T_{rh} (Revised version of T_H).

$$T_{rh} = t_c / (\sigma^* t_o). \tag{15}$$

Due to the consideration of already cached time while calculating T_{rh}, T_{rh} can differ from T_H, when $t_{ac} \geq t_o$. In this case, $\sigma = 1$ if $t_{ac} < t_o$, but for $t_{ac} \geq t_o$, $\sigma = \sigma_h$ (σ_h<<1 based on the priority of that node as well as packet).

Node: c starts a timer after broadcasting 'neighbor selection' message and as the timer elapses, on the basis of the following factors a neighbor node is selected and the packet for caching is passed to this node.

- T_{rh} of the neighbors
- Distance between node: n (the selected neighbor node) and node: f, D_{nf}.
- Distance between node: c and node: n, D_{cn}.

As the distance decreases, the probability of the packet drop during transmission through this distance also decreases. The transmission period is also reduced with the decrease in distance. Hence a node: with minimum D_{nf} and D_{cn} is likely to be selected as the neighbor to cache the packet.

Here we again use fuzzy evaluation (μ_n). Node c chooses the node with maximum μ_n. μ_n can be defined as follows.

$$\mu_n = w_1 T_{rh} + w_2 \frac{1}{1+D_{nc}} + w_3 \frac{1}{D_{nf}} \tag{16}$$

As has been already described, node: c itself can also cache the packet in this case and then D_{nc} will be zero. Thus to avoid 'divide by zero' error D_{nc} + 1 has been used instead of D_{nc}, in this equation. Here w_1, w_2, w_3 are corresponding weights based on the significance of these three factors.

E. Distributed Caching Mechanism

In distributed caching mechanism, energy of the neighbor nodes (of the faulty node) can be used in a distributed fashion to cache a packet on its way of transmission to the faulty node. If $T_H<1$, distributed caching mechanism can be followed.

- Node: c caches the packet for t_c period of time and selects a neighbor to cache the packet for t_{cr}= (t_o- t_c) period of time.

- As the t_c elapses, node: c sends a 'neighbor selection for distributed caching' message to all its neighbors. Neighbor selection mechanism is slightly different in this case. Node: c sends the value of t_{cr} and the node-id of node: f with this message. A neighbor of node: c which is not the neighbor of node: f ignores the message. But a neighbor of node: c which is also a neighbor of node: f computes the value of t_c for the packet to be cached after getting the 'neighbor selection for distributed caching' message and sends the value of its t_c to node: c.

- Node: c starts a timer after broadcasting 'neighbor selection for distributed caching' message and as the timer elapses, it computes the following fuzzy evaluation for each of the neighbors, which has replied to this message.

$$\mu_{n1} = w_1 t_c + w_2 \frac{1}{D_{nc}} + w_3 \frac{1}{D_{nf}} \tag{17}$$

- The node with maximum μ_{n1} is selected (node: c itself will not cache the packet any more) and the packet is transmitted to this node (let us say this node: s) for caching.

- If the t_c (of node: s) $< t_{cr}$, as the t_c elapses, it selects another node (except node: c) to cache this packet for $(t_{cr} - t_c)$ period of time.

- If the t_c(of node: s) $> t_{cr}$, t_{cr} period of time is allocated by node: s to cache the packet. The mechanism to decide on caching the packet has been already described, when the faulty node is not reinstalled to operation within $t_o = ((t_c \text{ of node: c}) + t_{cr})$ period of time.

- Node: s forwards the packet to node: f if it is reinstalled to operation within t_o period of time.

Distributed caching mechanism has a great impact on caching a packet, because even if the t_c of all the neighbor nodes of node: f are less than t_o, the packet can be cached by following this mechanism. Instead of passing the packet to a neighbor node for caching, distributed caching mechanism can be followed so that the energy of node: c and node: s can be utilized in a distributed manner to cache the packet.

III. Conclusion and Future Work

The analytical models that have been developed in this paper for a clear and detail inspection on the mechanisms of caching packets on its way of transmission to a faulty node can be considered as an important aspect of fault tolerance mechanisms for sensor networks. Thus the mechanisms contain great opportunities in the field of ensuring better performance of sensor networks with efficient use of energy.

Our future work will involve an elaborate discussion on fault tolerance mechanisms for sensor networks with the development of corresponding analytical models. Working on data dissemination and aggregation with duty cycle assignment mechanism is another field of interest to work on.

References

[1] "A Topology Discovery for Sensor Networks with Application to Network Management", http://www.cs.rutgers.edu/dataman/papers/TopDisc.pdf.

[2] Iavor Georgeff, "A Distributed Topology Discovery Algorithm for Wireless Sensor Networks", http://www.csse.uwa.edu.au/~rachel/AdHocProjects/iavor.georgeff.hons2004.dissertation.pdf.

[3] Hui Tian, "Network Topology Discovery and Its Application", http://www.jaist.ac.jp/library/thesis/is-doctor-2006/paper/hui-t/paper.pdf.

[4] Ranveer Chandra, Christof Fetzer, Karin Hogstedt, "Adaptive Topology Discovery in Hybrid Wireless Network", http://research.microsoft.com/~ranveer/docs/topology.pdf

[5] Krishnamachari, L.; Estrin, D.; Wicker, S, "The impact of data aggregation in wireless sensor networks", Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on Volume, Issue, 2002 Page(s): 575 – 578.

[6] JingPeng Li, R.S. Kwan, "A Self-Adjusting Algorithm for Driver Scheduling" Journal of Heuristics, 11:351–367, 2005, Springer Science + Business Media, Inc. Manufactured in The Netherlands.

[7] S. K. Prasad, Aung Aung, "Distributed Algorithms for Improving Wireless Sensor Network Lifetime with Adjustable Sensing Range", etd.gsu.edu/theses/available/etd-04202007-162055/unrestricted/aung_aung_200705_ms.pdf