# Availability Modeling and Analysis of Autonomous In-Door WSNs

Safwan Al-Omari and Weisong Shi
Wayne State University
{somari, weisong}@wayne.edu

## Abstract

*Availability analysis and modeling in autonomous and remotely administered systems composed of cheap and failure-prone components is vital to redundancy management including the prediction of the required number of components and the way these components are scheduled* `ON` *and* `OFF`. *Targeting the application of Wireless Sensor Networks (WSN) for the monitoring of elder people living in their apartments, we use techniques from reliability theory to model the WSN as a* $\kappa$*-out-of-m system with independent components. In addition to predicting the required node redundancy to meet desired availability behavior early in the planning phase, we use our modeling to show that scheduling these nodes* `ON` *and* `OFF` *later on in the operational phase does indeed improve the availability behavior over the entire system lifetime. To validate our model, we design and implement a simulator using nesC/TOSSIM, a well-known simulator for WSN.*

## 1. Introduction

Node redundancy has been shown useful in extending the lifetime of WSNs from the power consumption point of view, in this paper we show that redundancy is also a useful technique to extend the lifetime of WSNs from the reliability point of view. By lifetime, we mean the expected lifetime of the WSN given its reliability function.

By reliability we mean the probability that the WSNs lifetime will extend beyond some value (t) given a specific failure model of individual sensor nodes. The remaining question is how to manage this redundancy (how to schedule nodes on and off).

We prefer to use redundancy management rather than topology management to refer to the node scheduling process, since in our problem setup the topology is fixed.

Fixed topology in which a sensor nodes are pre-associated with a cluster head is proven successful as discussed in (cite Tenet paper).

Managing node redundancy is a vital factor of the overall WSNs performance from the reliability point of view. Making all the nodes to stay active from the beginning does not achieve the best reliability possible.

WSN applications are envisioned to operate for long period of times without close maintenance and supervision, this makes modeling and predicting WSN availability and reliability a very important issue, first, in planning the deployment of sensor nodes and, second, controlling the way these nodes are turned on and off later on.

Self-management WSN has been stressed by many people due to the infeasibility of human intervention and replacement of individual nodes.

In order to enable the WSN to perform self-management and recover from node failures, sensor node redundancy is a key factor.

### 1.1. Motivation

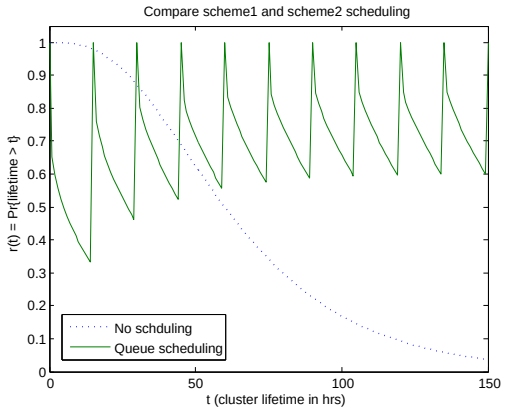



**Figure 1. Comparing cluster reliability under All active and Queue scheduling.**

As shown in Fig. 1, dividing the nodes into groups of $\kappa$ nodes each and making these groups to become active in a queue like schedule exhibits more desirable

availability behavior over the system lifetime than making all the nodes to become active since the beginning. We differentiate four aspects to describe the more desirable behavior. First, queue-like scheduling exhibits higher average availability, which means that at an arbitrary time, queue-like scheduling has higher probability that the system will be available on average. Second, queue-like scheduling exhibits higher minimum availability. Third, queue-like scheduling results in higher expected total uptime, which means that the expected total time in which the system will be available is larger. Fourth, queue-like scheduling exhibits less variation in the system's availability at different stages of the system lifetime, in other words, the system is more stable. In mission-critical systems it is undesirable to have a highly available system in an early stage, while having poor availability in a later stage of the system lifetime as the **no scheduling** scheme does.

## 2. Application Context and Background

It is expected that Adults age 65 years and older will account for more than 20% of the U.S population by the year 2050. Despite several initiatives, the numbers of elders with one or more physical disabilities due to lack of tools to monitor the elders' physical activities, who are livening in their apartments, is still on the rise [].

In this paper, we target the use of Wireless Sensor Networks (WSN) for the monitoring of elder people living in their own apartments. In this application, sensor nodes are deployed in each room to allow for remote and non-obtrusive monitoring and detection of any life-threatening accidents such as falling of elder people.

There are several unique requirements of the application and unprecedented limitations and features of WSN technology that mandate novel approach to network management. First, remote administration and unattended WSN operation, which brings in the need for autonomous, self-managing, and self-healing capabilities. Second, events that need to be detected tend to have short time duration (e.g., falling), which makes availability to take precedence over reliability. By availability, we mean the probability that the WSN will be functioning at any point of time, whereas, reliability means the probability of having longer continues and uninterrupted operation time duration. In this application, high availability of the WSN is very important to detect potential events, however, high reliability is not. Third, fixed topology since nodes that are deployed in the same room can be considered to form one cluster, in which any node is able to do what any other node can do. Fourth, mild operational environment, which makes usage-based sensor node failure model a very reasonable assumption. Fifth, Wireless sensor nodes are envisioned as power-limited, resource-limited, and failure-prone devices [**?**], however, these sensor nodes are expected to be very cheap, which makes it feasible to deploy them in large numbers. These characteristics make the use of node redundancy to overcome these limitations an attractive solution [1, 4, **?**, 5].

Designing systems that are highly available on one hand, and autonomous and self-healing on the other hand, puts more demand on developing tools and system models that capture node failure behavior and facilitate the prediction of the required number of nodes that are needed to empower the system to withstand failures and maintain availability and autonomy. In other words, an autonomous system should be designed to sustain failures on its own without external intervention. Therefore, we focus in this paper at developing analytical models to help answer two fundamental questions: how many nodes are required and the way these nodes should be scheduled to obtain best availability behavior in terms of either minimum availability, average availability, expected total uptime, or stability. Rather than using the term topology management as in previous research [], we refer to these two problems as redundancy management as the topology aspect in this application does not have primal effect as the topology is fixed.

We can differentiate two types of node failure models; deployment-based and usage-based. The latter accounts for failures due to normal wear and tear and so the probability of failure is related to the time the node spent in `ON` mode. Whereas, the former accounts for harsh environmental conditions and so once the nodes are deployed they become susceptible to the same failure probability regardless whether the node is `ON` or `OFF`. In our in-door application, it is more reasonable to employ usage-based failure model to mimic real-life sensor node failure behavior.

We assume that all the sensor nodes have similar initial power and perform similar workload, which enable them to function for an identical maximum period of time $T_{max}$. We further assume that all the nodes independently follow the same failure model and once a node fails, it never becomes available again (i.e., fail stop).

## 3. Problem Setup

The sole purpose of our work is to model the availability of the sensor cluster (denoted as $A(t)$) in terms of the availability of the underlying components (denoted as $a_i(t)$) under two scheduling schemes; **no scheduling** and **queue scheduling** schemes, formalize the re-

dundancy management problems in each scheduling scheme, and finally to compare their performance in terms of either minimum availability, average availability, stability, or expected total uptime. We use techniques from reliability theory [] to develop these analytical models and to formalize and solve the redundancy management problems. We consider the sensor nodes of each room to form one cluster, out of which at least $\kappa$ sensor nodes have to be available in order to have an available cluster. $\kappa$ sensor nodes are needed instead of only one node to rule out sensor reading errors due to faulty sensors and noisy environment []. Deciding the value of $\kappa$ is out the scope of this work and is considered as input to our model. We use $m$ to represent the number of nodes in each cluster. Basically, we model the availability of the system as a $\kappa$-out-of-$m$ system. In the rest of the paper, we use the terms system and cluster interchangeably.

We formally define our performance metrics as follows. Minimum availability (denoted as $\min_{A(t)}$) is defined as $\min_{t=0}^{T_{max}} A(t)$. Average availability (denoted as $\text{avg}_{A(t)}$) is defined as $\frac{\sum_{t=0}^{T_{max}} A(t)}{T_{max}+1}$. Stability (denoted as $\sigma_{A(t)}$) is defined as $\sqrt{\frac{1}{T_{max}+1}\sum_{t=0}^{t=T_{max}} (A(t) - avg_{A(t)})^2}$. Total uptime time and expected total uptime are denoted as U and $E[\mathrm{U}]$ respectively and their definitions are given and explained in their corresponding scheduling schemes.

The redundancy management problems have slightly different settings in each scheduling scheme. In the **no scheduling** scheme, all the nodes are made `ON` since the beginning and so, we are only left of finding the required number of nodes (i.e., $m$) to meet some availability requirements (i.e., either $\min_{A(t)}$, $avg_{A(t)}$, $\sigma_{A(t)}$, or $E[U]$). Whereas in the **queue scheduling** scheme, the $m$ nodes are divided into groups of $\kappa'$ nodes each, and made `ON` in a queue-like manner, therefore, we have two problems to solve. First, find $m$ and corresponding $\kappa'$ to meet some availability requirements. Second, given $m$, find $\kappa'$ that optimizes availability in terms of either $\min_{A(t)}$, $avg_{A(t)}$, $\sigma_{A(t)}$, or $E[U]$. Note that **no scheduling** scheme is indeed a special case of the **queue scheduling** scheme (i.e., make $\kappa' = 1$), however, we prefer to model the **no scheduling** scheme separately for two reasons. First, modeling the **no scheduling** scheme is easy as a classical $\kappa$-out-of-m system. Second, this scheme needs no scheduling management at all, which makes the implementation different than the **queue scheduling** at the cluster head.

The first step toward building our models is to assume a sensor node failure model. Typically, the lifetime of usage-based components is divided into three periods, each with a different failure rate. First, an early period that exhibits decreasing failure rate, these failures are due to design and manufacturing faults. Second, a stable period known as the 'useful life' that exhibits a very low and stable failure rate. Third, a wear-out period at the end of the component lifetime that exhibits increasing failure rate, these failures are due to normal tear and wear out.

A widely known and accepted approach to model this behavior is to use a bathtub-shaped **failure rate** function, denoted as ($\lambda_i(t)$). $\lambda_i(t)$ represents the conditional probability intensity that node $i$ will fail in the next moment, given that it has survived until time $t$ (i.e., $\lambda_i(t) = Pr\{X_i \in [t+\mathrm{d}t] | X_i > t\} = \frac{-S_i(t)'}{S_i(t)}$), where $X \in [0, T_{max}]$ represents node $i$ lifetime and $S_i(t)$ is known as the survival function and represents the unconditional probability that node $i$ has no failures by time $t$ and so survives beyond time $t$. It is known that $S_i(t) = exp\{-\int_0^t \lambda(\tau)\mathrm{d}\tau\}$ []

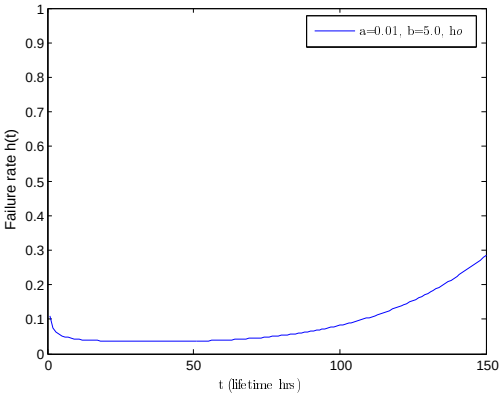



**Figure 2. Comparing cluster reliability under All active and Queue scheduling.**

In this paper, we use a failure rate function proposed lately in []. $\lambda_i(t)$ and the corresponding $S_i(t)$ are defined as follows:

$$\lambda_i(t) = a\,b(at)^{b-1} + (\frac{a}{b})(at)^{\frac{1}{b}-1} + h_\circ \qquad (3.1)$$

$$S_i(t) = \exp\{-(at)^b - (at)^{\frac{1}{b}} - h_\circ t\} \qquad (3.2)$$

Fig. **??** and Fig. **??** depict $\lambda(t)$ and $S_i(t)$ respectively with assumed values for $a$, $b$, and $h_\circ$.

Based on our assumption that once a node dies it never become available again (i.e., fail-stop), we may think of $S_i(t)$ as the availability of node $i$ at time $t$, which equals to $Pr\{node\ i\ is\ available\ at\ time\ t\}$.

## 4. Model

We use reliability theory to model availability of the sensor cluster as a parallel system with independent

components (i.e., sensor nodes). The model answers two fundamental questions. First, how many redundant nodes (i.e., $m$) are needed to meet desired availability behavior (i.e., $\min_{A(t)}$, $avg_{A(t)}$, $\sigma_{A(t)}$, or $E[U]$). Second, the model guides the process of scheduling these nodes to achieve optimal average reliability or maximize the minimum reliability and maximizing the expected lifetime.

We formally define these problems in the following two sections.

### 4.1. No scheduling scheme

In this scheduling scheme, all the nodes become ON since the beginning. Therefore, we simply model the availability of the cluster as a classical $\kappa$-out-of-m system. We say that the sensor cluster is available at time $t$ (i.e., $A_{no}(t)$) if and only if there exits at least $\kappa$ nodes available at time $t$, put formally as follows:

$$A_{no}(t) = \sum_{i=\kappa}^{m} \binom{m}{i} a_i(t)^i \cdot (1 - a_i(t))^{(m-i)} \qquad (4.1)$$

Fig. **??** shows an $A_{no}(t)$ with $m = 12$ and $\kappa = 1, 3,$ and 6.

To find $textrmE[U]$, Note that the system as a whole exhibits a fail-stop behavior following its fail-stop components (i.e., sensor nodes). In other words, once there are less than $\kappa$ sensor nodes available, the system fails and never become available again. Therefore, $U$ is equivalent to a random variable representing the time until first failure (denoted as $\tau$) with $Pr\{\tau > t\} = A_{no}(t)$. Note that the random variable $\tau \geq 0$, and so the expected time until first failure and hence $\mathrm{E}[U]$ can be defined as follows:

$$\mathrm{E}[U] = \sum_{t=0}^{T_{max}} A_{no}(t) \qquad (4.2)$$

As there is no scheduling in this scheme (i.e., all nodes are ON all the time, we are left with one question that hope the model to answer:

PROBLEM 1: GIVEN A VALUE OF $\kappa$, FIND LOWEST $m$ NEEDED TO MEET EITHER $\min_{A(t)}$, $\mathrm{avg}_{A(t)}$, $\sigma_{A(t)}$, or $\mathrm{E}[U]$.

PROBLEM 1 is a simple optimization problem that can be solved iteratively over $m$ starting with $m = \kappa$, incrementing $m$ by one each time, and checking whether the current value of $m$ meets the requirements (i.e., $\min_{A(t)}$, $\mathrm{avg}_{A(t)}$, $\sigma_{A(t)}$, or $\mathrm{E}[U]$).

To find $\min_{A_{no}(t)}$ for a given $m$, simply substitute ($t = T_{max}$) in equation 4.1. Note that $A_{no}(t)$ is consistently decreasing with time as we can observe from Fig. **??**, hence, $min_{A_{no}(t)} = A_{no}(T_{max})$. For $\mathrm{avg}_{A(t)}$ and $\sigma_{A(t)}$, we simply follow the definition to calculate them for a given $m$ value. From the definition of $\mathrm{avg}_{A(t)}$ in Section **??**, we get $(\mathrm{T}_{max}+1) \cdot \mathrm{avg}_{A_{no}(t)} = \sum_{t=0}^{\mathrm{T}_{max}} A_{no}(t)$, by substituting in equation 4.2, we get:

$$\mathrm{E}[U] = \mathrm{avg}_{A_{no}(t)} \cdot (\mathrm{T}_{max}+1) \qquad (4.3)$$

Thus, the value of $m$ needed to meet $\mathrm{avg}_{A(t)}$ requirement, is the same $m$ that is needed to meet $\mathrm{E}[U]$.

### 4.2. Queue scheduling scheme

Unlike the **no scheduling** scheme, **queue scheduling** divides the $m$ nodes into $\eta = \frac{m}{\kappa'}$ groups (denoted as $g_i$, where $i = 1, \ldots, \eta$). Each group, $g_i$, consists of $\kappa'$ nodes, where ($\kappa \leq \kappa' \leq m$). Given these $\eta$ groups, the time is divided into $\eta$ epoches with equal periods denoted as $\Delta$. Each group $g_i$ is switched ON, in a queue-like scheduling, at the beginning of its corresponding epoch (denoted as $\varepsilon_i$).

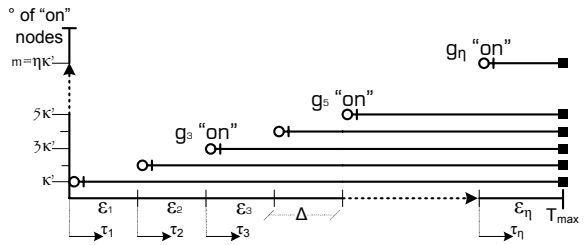



**Figure 3.**

Fig. 3 depicts a queue scheduling time line. The x-axis represents time, while the y-axis represents the total number of ON nodes shown as discrete values multiple of $\kappa'$.

Perhaps the first thing that comes to mind when trying to model the availability of queue scheduling is the renewal process model []. Unfortunately, the fact that renewals (i.e., bringing node groups ($g_i$) ON), are asynchronous to failures causes two major incompliances with the classical renewal process model assumptions. First, lack of instantaneous repair, in other words, should the system fail during epoch $i$ ( i.e., $\eta_i$), it will not be available until the beginning of the next time epoch (i.e., $\eta_{i+1}$). Recall that in our usage-based failure model assumption **??**, nodes has to be in SLEEP mode to avoid failures, which makes them unresponsive to external events and therefore can not be asked to become active in case of failures. On the other hand, in un-attended and remotely administered WSNs, human intervention is infeasible and violates the key non-obtrusive application requirement. Second, non-homogeneity of the availability probability distributions

during different time epoches, in other words, $\tau_i$ random variables in Fig. 3 are not identically distributed. non-homogeneity results from having different number of ON nodes and even different groups being in different times of their lifetime during different epoches.

In light of the above queue scheduling algorithm complications, we use recursive numerical function to model the system availability at an arbitrary time instance (i.e., $A_Q(t)$). Let $Q(t,e,i,p)$ be the probability that there are exactly $i$ nodes ON at time $t$, then:

$$A_Q(t) = \sum_{i=\kappa}^{\kappa' \cdot e} Q(t,e,i,1.0) \text{ , where}$$
$$e = \lfloor \frac{t}{\Delta} \rfloor + 1$$

```
1. function result=Q(t, e, i, p)
2. t' = t – (e - 1) * Δ
3. if (e == 1){ // base case
4.   if (i > k'){
5.     return 0 // no enough nodes to choose from
6.   }
7.   return (k' i) * S(t')^i * (1 - S(t'))^(k' - i) * p
8. }else{
9.     minJ = max(0, i - (e - 1) * k')
10.    if (min > k'){
11.      return 0 // no enough nodes to choose from
12.    }
13.    maxJ = min(i, k')
14.    tmpP = 0
15.    for (j = minJ; j <= maxJ; j++){
16.      current_p = p * (k' j) * S(t')^j * (1 - S(t'))^(k' - j)
17.      tmpP = tmpP + Q(t, e - 1, i - j, current_p)
18.    }
19.    return tmpP
20. }}
```

**Figure 4.**

The function **Q** finds the probability of having $\kappa$ nodes available out of $e$ groups each one consists of $\kappa'$ nodes by considering all possible $\kappa$ node combination out of the $e$ node groups and summing their corresponding probabilities. In line 2 of Fig. 4, $t'$ represents the current group ON time. $t'$ is used in lines 5 and 16 to find the availability probability of the current group nodes using the survival function (i.e., $S(t')$). At each recursive call,

Line 7 of Fig. 4, which is the base case with a single node group, marks the end of one possible node combination in the recursive who's probability is calculated as the

Each recursive call in **Q** is equivalent to asking

PROBLEM 2: GIVEN $\kappa$, FIND LOWEST $m$ AND CORRESPONDING $\kappa'$ NEEDED TO MEET EITHER $\min_{A_Q(t)}$, $\text{avg}_{A_Q(t)}$, OR $\text{E}[U]$.

PROBLEM 3: GIVEN $m$ AND $\kappa$, FIND $\kappa'$ THAT MAXIMIZES $\min_{A_Q(t)}$, $\text{avg}_{A_Q(t)}$, OR $\text{E}[U]$.

## 5. Simulation

We use TOSSIM simulator

### 5.1. simulation setup

To verify our modeling results, we build a simulation in nesC/TOSSIM. nesC is a well-known programming language for WSN applications and TOSSIM is a simulator that can simulate nesC applications.

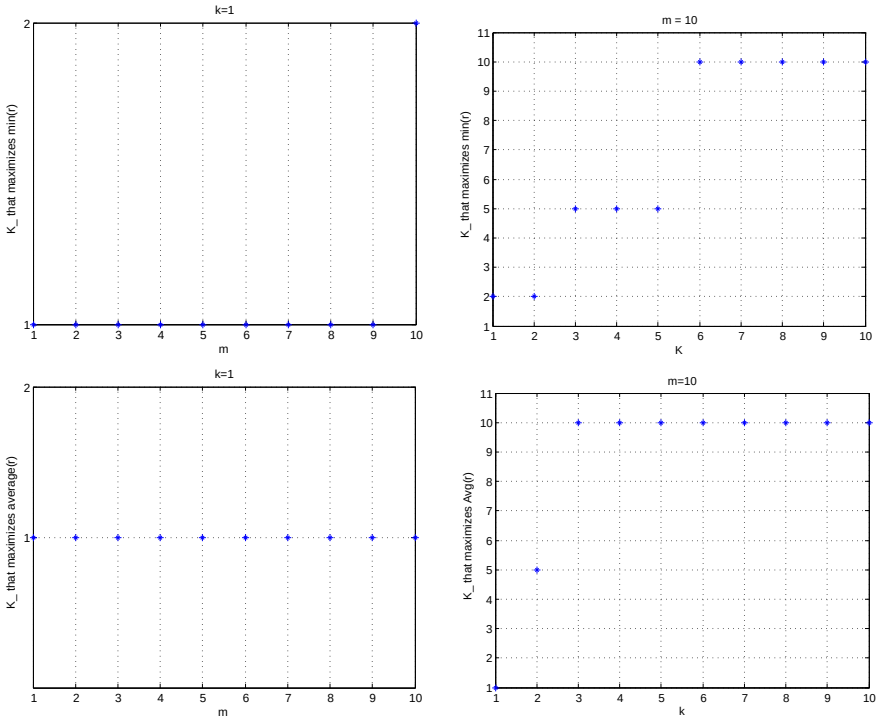
Fig. **??** depicts an overview of the simulation components and the way they interact. Fig. **??**(a) depicts a cluster with a **cluster head**, **sensor nodes**, **simulation clock** on the left, and a **global trace** file on the right.

The cluster head is aware of the total number of nodes in the cluster ($m$) and the required number for a functioning system ($k$). As we discussed earlier, $m$ is determined in the planning phase using the our model, whereas $k$ is an input to the simulator and made available at the cluster head as a booting parameter. Upon starting the simulation, the cluster head find the optimal $k\prime$ (i.e., either for $min[a(t)]$ or $avg[a(t)]$), divide the $m$ sensor nodes into groups and assign each a starting time (i.e. $\alpha_i$ relative the global simulation clock).

The global simulation clock in its turn is responsible to advance the simulation time (i.e., **T** and to fire an event (i.e., new hour) at the sensor nodes every time unit. The global clock uses a TinyOS **Timer** to implement this. Simulation time starts at zero and ends at $T_{max}$.

Each sensor node is responsible to write two evens on the global simulation trace, **start time** and **fail time**. At the end of the simulation, we parse and analyze the **global trace** to find our results.

As shown in Fig. **??**(b), each sensor node consists of three major modules: **local time**, **failure rate**, and **top-level control** modules. The **local time** module acts as a switch and keeps track of the node local time, which advances only when the switch is ON (i.e., node is turned on). **failure rate** module simply implements the failure rate function and decides whether the node should fail in the next hour or not. Since the failure time of the node is not decided at the beginning of the simulation, we use the failure rate function to calculate the probability of failure, which represents the conditional failure probability, rather than survival function. The **rand()** function generates a value distributed uniformly between zero and one. Once the a node fails, it writes




its failure time on the **global trace** and turns the **local time** switch back OFF. Since the **global trace** maintains global time, **start time** (i.e., $\alpha_i$) is added to the local failure time before being written to the **global trace**. The **top control** module simply writes down the start time and turns the **local time** switch ON when global simulation time equals its start time.

we simulate a single cluster with several $m$ and $k$ values.

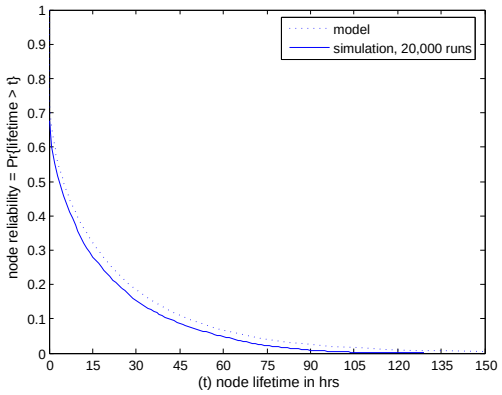



**Figure 6. Comparing cluster reliability under All active and Queue scheduling.**

In Fig. 6, we show that our simulator can regenerate experimentally the same single node survival function (i.e., $S(t)$) **??**, which is very important to show the validity of our system model later on. The x-axis represents time ($t$), y-axis represents single node availability at time ($t$). We perform a thousand single-node simulation runs and record the failure times. Each point represents the ratio of the number of runs in which the node was available to the total number of runs at time $t$. We can notice an excellent match.

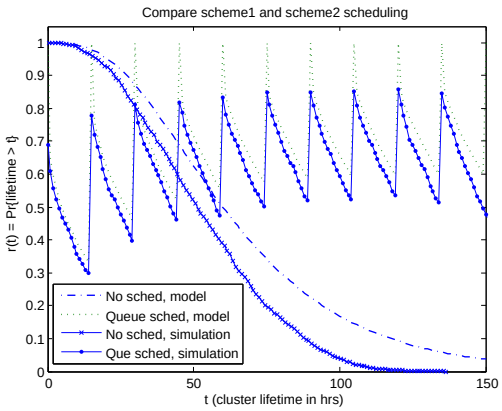



**Figure 7. Comparing cluster reliability under All active and Queue scheduling.**

In Fig. 7 we move on to validate our simulator against the two scheduling scheme models, the **no scheduling** scheme model **??** and **Queue scheduling** scheme mode **??**. As in Fig. 6, x-axis represents time,

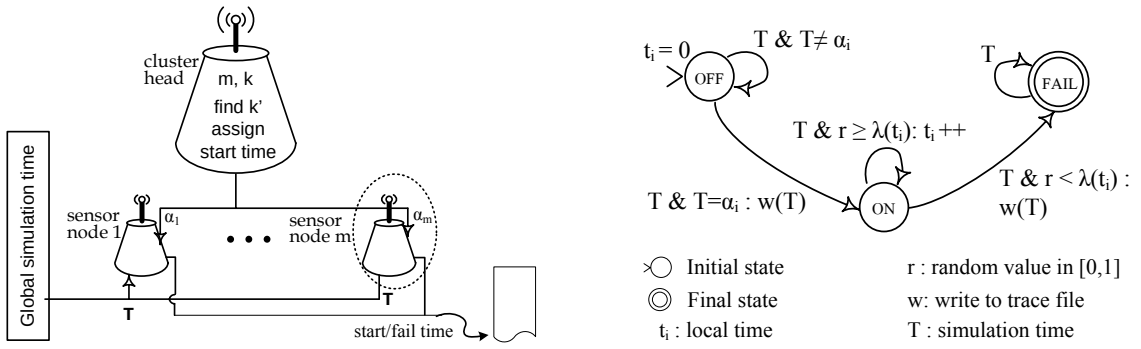



**Figure 5.**

y-axis represents the availability of the system. For each scheduling scheme we perform a thousand runs, record single-node failure times, and calculate system availability at time $t$ as the ratio of runs in which more than $\kappa$ nodes are available, at time $t$, to the total number of runs (i.e., 1000 runs). There are two important observations that need further explanation. First, the mismatch between the model and the simulator is larger than that of the single node scenario, Fig. 6. Second, the mismatch in the **No scheduling** scheme is larger than that of **Queue scheduling** scheme. Larger mismatch in both cases is due to the same reason, which is mismatch accumulation resulted from the several nodes in the system. This is easy to understand in the first case. In the second scenario, note that the number of nodes that are turned ON in the **no scheduling** scheme is larger than that of the **queue scheduling** scheme, which results in larger mismatch accumulation. This explanation is also supported by the observation that mismatch becomes greater as more and more nodes are turned ON as time passes in the **queue scheduling**.

## 6. Evaluation

We validate our model results through simulation. Furthermore, We show the advantage of **Queue scheduling** scheme over the **No scheduling** scheme in terms of four performance metrics.

We compare the performance of **no scheduling** and **queue scheduling** schemes in terms of minimum availability, average availability, stability, and expected total uptime as node redundancy increases. We show that **queue scheduling** indeed exhibits better node redundancy leverage. We control node redundancy by changing both $m$ and $\kappa$. For the **queue scheduling** scheme, $\kappa'$ is chosen to optimize the corresponding performance metric under study.

Fig. 8 shows analytical as well as experimental minimum availability comparison results as $m$ increases, Fig. 8(a), and as $\kappa$ increases, Fig. 8(b). Analytically, we discussed how to obtain minimum availability in Section **??**. Experimentally, we perform a thousand runs for each $m$ and $\kappa$ values, calculate the system availability as we did in Fig. 7 for all $t \in [0, T_{max}]$, and finally find corresponding minimum availability. We can observe that **queue scheduling** exhibits consistent and larger increase in the minimum availability compared to **no scheduling**. Also, note that as $m$ and $\kappa$ get closer (i.e., less redundancy), the performance of **queue scheduling** and **no scheduling** becomes closer, which is simply because **queue scheduling** converges to **no scheduling**. In other words, if $m$ and $\kappa$ are the same, the only way to schedule the nodes is to make all of them ON since the beginning, which is the same as **no scheduling**.

Like in Fig. 8, in Fig. 9 we perform a thousand runs for each $m$ and $\kappa$ values, calculate the system availability as we did in Fig. 7, and finally find corresponding average availability over $t \in [0, T_{max}]$. Again, we observe that **queue scheduling** outperforms **no scheduling**, in particular for high node redundancy.

In Fig. 10, we move on to compare **no scheduling** and **queue scheduling** in terms of stability. As we mentioned, we use standard deviation as a measure of stability (i.e., y-axis), note that unlike Fig 9 and Fig. 8, lower values on the y-axis means better stability. Again, we observe better stability in case of **queue scheduling**, furthermore, we observe that **queue scheduling** achieves better and better stability as redundancy increases, whereas, **no scheduling** keeps on getting worse stability as redundancy increases.

## 7. Related Work

Sensor node redundancy and node scheduling have been extensively studied in the wireless sensor network research community [1, 2, 3, 6]. The basic idea in their work is to use node redundancy and scheduling primarily to work around the battery lifetime limitation of the sensor nodes and extend the network lifetime while maintaining coverage and connectivity. Our work complements their work by adding a new dimension (i.e., availability) in the node redundancy and scheduling protocols design space and exploring more sensor node failure models, namely usage-based, in addition to the trivial running-out-of-battery sensor node failure. Furthermore, coverage and connectivity are minor objectives when performing the node scheduling process in our application as all the nodes in the same cluster can communicate directly to the cluster head and can provide the same coverage. This setting has been also supported in the latest wireless sensor network architecture (i.e., Tenet) proposed in []. The authors in [] argue, based on their experience with real-life deployments, that the WSN should be arranged into clusters with predetermined cluster head, which makes our modeling analysis even applicable in more application contexts.

DADA [], the authors proposes two adaptive sleeping schedules of redundant nodes based on both application demand and network conditions. The primary goal of their work is the association of coverage and routing backup sets with each active node, these sets are made active regularly to take over in case an active node failure is detected. They considered two node failure modes; aging failure model and catastrophic event failure model which correspond to our usage-based and deployment-based failure models. Our work differs from theirs in two aspects. First, they adopt a re-active failure detection and recovery approach in contrast to our pro-active approach, in which we optimize for the best possible scheduling scheme in advance given a specific failure model. Second, we tried to solve a more fundamental problem, which is predicting the required node redundancy to reach a desired fault tolerance behavior, using our analytical model. In a sense, their adaptive scheduling protocol can be integrated in our solution framework, which may result in more efficient scheduling schemes in different application contexts. PEAS, proposed in [**?**], leverages node redundancy and scheduling to overcome harsh deployment environments, which causes frequent node failures, in other words, they target deployment-based failure model. Like DADA [], their approach is re-active in contrast to our pro-active approach.

Autonomous WSN and self-*

Reliability and Availability

## 8. Conclusion

Address assumptions

Our model can be easily adapted to incorporate any sensor node failure model. (lack of real deployment failure trace).

Usage-based failure model and how it can be adapted to deployment-based failure model.

## References


[1] Alberto Cerpa and Deborah Estrin. Ascent: Adaptive self-configuring sensor networks topologies. *IEEE Transactions on Mobile Computing Special Issue on Mission-Oriented Sensor Networks*, 3(3), July-September 2004.

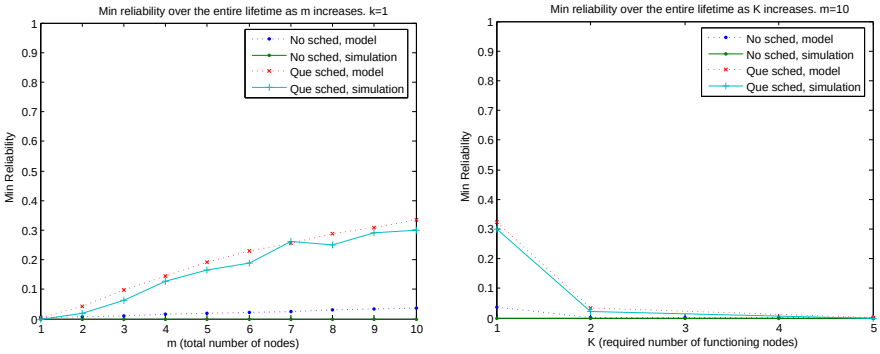
[2] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. SPAN: An energy-efficient coordination algorithm for topology maintenance in ad-hoc wireless networks. In *Proc. of MobiCom'01*, July 2001.

[3] W. Steven Conner, Jasmeet Chhabra, Mark Yarvis, and Lakshman Krishnamurthy. Experimental evaluation of synchronization and topology control for in-building sensor network applications. In *Proc. of WSNA'03*, 2003.

[4] R. H. Katz, J. M. Kahn, and K. J. Pister. Mobile networking for smart dust. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, Seattle, WA, August 1999.

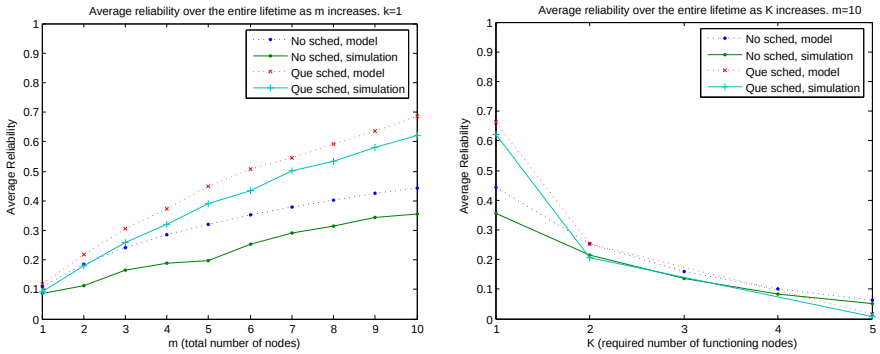
[5] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, October 2000.

[6] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proc. of MobiCom'01*, July 2001.



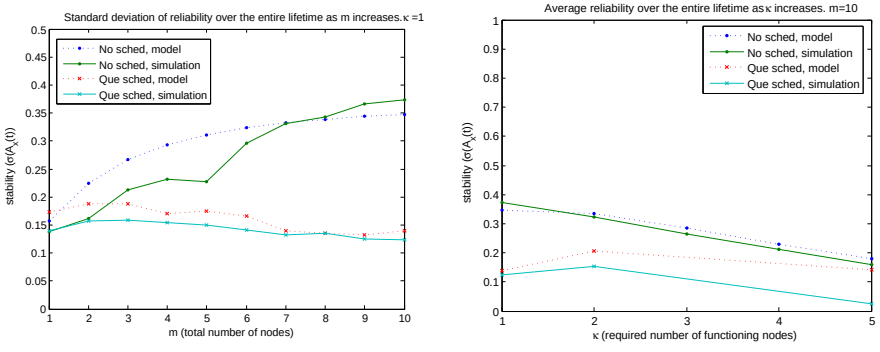

(a) (b)

**Figure 8. Comparing min cluster availability under no scheduling and queue scheduling, (a) as m increases, (b) as $\kappa$ increases.**



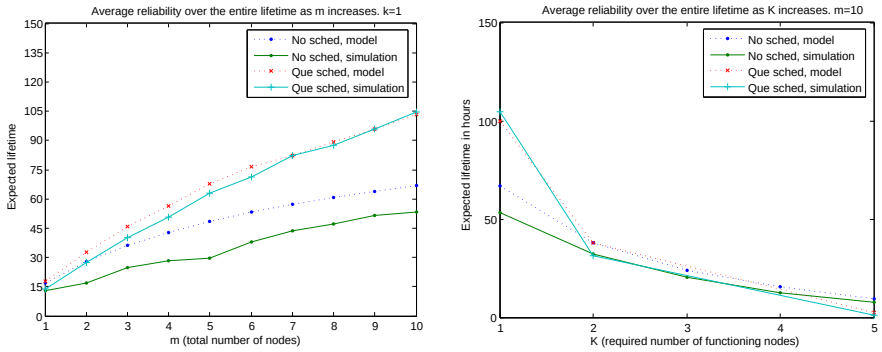

(a) (b)

**Figure 9. Comparing average cluster availability under no scheduling and queue scheduling, (a) as m increases, (b) as $\kappa$ increases.**




(a) (b)

**Figure 10. Comparing stability of cluster availability under no scheduling and queue scheduling, (a) as m increases, (b) as $\kappa$ increases.**




(a) (b)

**Figure 11. Comparing expected total uptime of the cluster under no scheduling and queue scheduling, (a) as m increases, (b) as $\kappa$ increases.**