Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 462–476
computer communications
www.elsevier.com/locate/comcom

Self-monitoring of wireless sensor networks

Chihfan Hsin[a], Mingyan Liu[b,*]

[a]Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109-2122, USA
[b]Electrical Engineering and Computer Science Department, University of Michigan, 1301 Beal Ave. 4238 EECS, Ann Arbor, MI 48109-2122, USA

Available online 15 February 2005

Abstract

This paper presents an efficient distributed self-monitoring mechanism for a class of wireless sensor networks used for monitoring and surveillance. In these applications, it is important to monitor the health of the network of sensors itself for security reasons. This mechanism employs a novel two-phase timer scheme that exploits local coordination and active probing. Simulation results show that this method can achieve low false alarm probability without increasing the response delay. Under a stable environment analytical estimates are provided as a guideline in designing optimal parameter values. Under a changing, noisy environment a self-parameter tuning functionality is provided and examined.

Keywords: Wireless sensor networks; Monitoring; Surveillance; False alarm; Response delay

1. Introduction

Wireless sensor networks have been the subject of extensive study in recent years due to rapid advances in integrated sensing, actuation, processing and wireless technologies. In this study, we consider a class of surveillance and monitoring systems employing wireless sensors, e.g., indoor smoke detection, surveillance of public facilities for tampering or attack. In these applications, the network itself—meaning the constituent sensors of the network—needs to be constantly monitored for security purposes. In other words, we need to ensure the proper functioning of each sensor (e.g., has energy, is where it is supposed to be, etc.) in order to rely on these sensors to detect and report anomalies. Note that self-monitoring can reveal both malfunctioning of sensors and attacks/intrusions that result in the destruction of sensors. It is possible that in some scenarios, especially where redundancy exists in the sensor deployment, we only need to ensure that a certain *percentage* of the sensors are functioning. In this study we will limit our attention to the case where each individual sensor in the network needs to be monitored. We assume that a (possibly) remote control center will react to any problems revealed by such monitoring. Due to the potentially large area over which sensors are deployed, sensors may be connected to the control center via multiple hops. We will assume that these multi-hop routes exist as a result of certain initial self-configuration and route establishment function.

In general, detection of anomalies may be categorized into the following two types. One is *explicit detection*, where the detection of anomaly is performed directly by the sensing devices, which send out alarms upon the detection of an event of interest. Explicit detection usually involves very clear decision rules. For example, if temperature exceeds some predefined threshold, a sensor detecting it may fire an alarm. Following an explicit detection, an alarm is sent out and the propagation of this alarm is to a large extent a routing problem, which has been studied extensively in the literature. For example, [3] proposed a braided multi-path routing scheme for energy-efficient recovery from isolated and patterned failures; [5] considered a cluster-based data dissemination method; [7] proposed an approach for constructing a greedy aggregation tree to improve path sharing and routing. Within this context the accuracy of an alarm depends on the pre-set threshold, the sensitivity of the sensory system, etc. The responsiveness of the system depends on the effectiveness of the underlying routing mechanism used to propagate the alarm.

* Corresponding author. Tel.: +1 734 7649546; fax: +1 734 7638041.
E-mail addresses: chsin@umich.edu (C. Hsin), mingyan@umich.edu (M. Liu).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.12.031

The other type of detection is *implicit detection*, where anomalies disable a sensor from being able to communicate. The occurrence of such an event thus has to be inferred from the *lack* of information. An example is the death of a sensor due to energy depletion. To accomplish implicit detection, a simple solution is for the control center to perform *active monitoring*, which consists of having sensors continuously send existence/update (or keep-alive) messages to inform the control center of their existence. If the control center has not received the update information from a sensor for a pre-specified period of time (timeout period), it may infer that the sensor is dead. The problem with this approach is the amount of traffic it generates and the resulting energy consumption. This problem may be alleviated by increasing the timeout value but this will also increase the response time of the system in the presence of an intrusion. Active monitoring can be realized more efficiently in various ways, including the use of data aggregation, inference, clustering, and adaptive updating rate. For a more detailed discussion see [6].

A distinctive feature of active monitoring is that decisions are made in a centralized manner at the control center, which becomes a single point of data traffic concentration (same applies to a cluster head). Subsequently, the amount of bandwidth and energy consumed affects its scalability. In addition, due to the multi-hop nature and high variance in packet delay, it will be difficult to determine a desired timeout value, which is critical in determining the accuracy and responsiveness of the system. An active monitoring based solution may function well under certain conditions. However, we will pursue a different, distributed approach in this paper.

Our approach is related to the concept of *passive monitoring*, where the control center expects nothing from the sensors unless something is wrong. Obviously, this concept alone does not work if a sensor is disabled from communicating due to intrusion, tampering or simply battery outage. However, it does have the appealing feature of low overhead. Our approach to a distributed monitoring mechanism is thus to combine the low energy consumption of passive monitoring and the high responsiveness and reliability of active monitoring.

Due to the energy constraint of sensors and the nature of random topology, it has been argued in [11] that it is inappropriate to use CSMA/CD, TDMA or reliable point-to-point transmissions like IEEE 802.11 in wireless sensor networks. We assume in this paper that the wireless channel is shared via MAC of the random access type, which is subject to collision. Under this assumption, we define two performance measures to evaluate a self-monitoring mechanism. The first is the *false alarm probability* defined as the probability that a particular sensor has been determined to be dead while the truth is the opposite. This can happen if consecutive update packets from a sensor are lost due to collision or noise in the environment. The second is the *response delay*, which is defined as the time between when a sensor dies (either due to energy depletion or attacks) and when such an event is detected. These are inherently conflicting objectives. Time-out-based detection necessarily implies that longer timeout value results in more accurate detection result (smaller false alarm probability) but slower response (longer response delay). Our approach aims at reducing the false alarm probability for a given response delay requirement, or equivalently reducing the response delay for a given false alarm probability requirement. In addition to these two metrics, *energy consumption* associated with a self-monitoring mechanism is also an important metric. We will use these three metrics in evaluating different approaches.

The rest of the paper is organized as follows. In Section 2, we describe our approach to the self-monitoring problem, as well as a number of variations. Section 3 provides simulation results for performance evaluation. In Section 4, the performance of some instances of this mechanism are studied analytically. Section 5 provides a self-parameter tuning scheme to adjust the control parameters of our approach under a changing, noisy environment. Section 6 gives a review of related work and Section 7 concludes the paper.

2. The two-phase self-monitoring system

The previous discussions and observations lead us to the following principles. First, some level of active monitoring is necessary simply because it is the only way of detecting communication-disabling events/attacks. However, because of the high volume of traffic it incurs, active monitoring should be done in a localized, distributed fashion. Secondly, the more decision a sensor can make, the less decision the control center has to make, and therefore less information needs to be delivered to the control center. Arguably, there are scenarios where the control center is at a better position to make a decision with global knowledge, but whenever possible local decisions should be utilized to reduce traffic. Similar concepts have been used for example in [8], where a sensor advertises to its neighbors the type of data it has so a neighbor can decide if a data transmission is needed or redundant. Thirdly, it is possible for a sensor to reach a decision with local information and minimum embedded intelligence, and it should be exploited.

The first principle points to the concept of *neighbor monitoring*, where each sensor sends update messages only to its neighbors, and every sensor actively monitors its neighbors. Such monitoring is controlled by a timer associated with a neighbor. If a sensor has not heard from a neighbor within a pre-specified period of time, it will assume that the neighbor is dead. Note that this neighbor monitoring works as long as there is no partition in the network. Since neighbors monitor each other, the monitoring effect is propagated throughout the network, and the control center only needs to monitor a potentially very small subset of sensors. The second and the third principles lead us to the concept of *local decision making*. By adopting a simple neighbor coordinating scheme with which a sensor

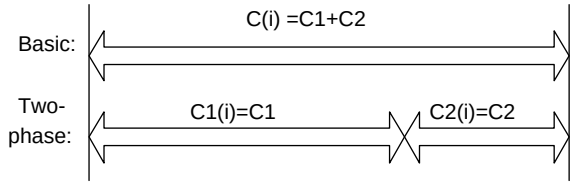

Fig. 1. The basic system vs. the two-phase system.

consults with its neighbors before sending out an alarm, we may significantly increase the accuracy of such a decision. This in turn reduces the total amount of traffic destined for the control center. The above discussion points to an approach where active monitoring is used only between neighbors, and network-wide passive monitoring is used in that the control center is not made aware unless something is believed to be wrong with high confidence in some localized neighborhood. Within that neighborhood a decision is made via coordination among neighbors.

Our approach consists of a two-phase timer where a sensor uses the first phase to wait for updates from a neighbor and uses the second phase to consult and coordinate with other neighbors in order to reach a more accurate decision. Fig. 1 shows the difference between our approach (with neighborhood coordination) and a typical system based on a single timer (without neighborhood coordination), subsequently referred to as the two-phase (TP) system and the basic system, respectively. In the basic system a single timer of length C_1+C_2 is maintained for monitoring sensor i by a neighboring sensor s. Each sensor periodically sends an update packet (UPD) to its neighbors with an average update interval of T. If an update packet from i is received by s within this timer, it will be reset. If no packets are received within this timer, sensor s times out and decides that sensor i does not exist or function anymore. It will then trigger an alarm to be sent to the control center.

In the two-phase system, two timers are maintained for monitoring sensor i, with values C_1 and C_2, respectively. If no packets from sensor i is received before the first timer $C_1(i)$ expires, sensor s activates the second timer $C_2(i)$. During the second timer period, sensor s will query other neighbors regarding the status of i with an alarm query packet (AQR), which contains IDs of s and i. A common neighbor k of both i and s may corroborate s's observation if k's own $C_1(i)$ has expired with an alarm confirmation packet (ACF), or negate s's observation if it has an active $C_1(i)$ with an alarm reject packet (ARJ). This ARJ packet contains IDs of k and i and the k's remaining timer $C_1(i)$ as a reset value. If sensor i is still alive and receives s's query, it may directly respond to s with an update. If sensor s does not receive any response to its query before $C_2(i)$ expires, it will send out an alarm. If any packet from i is received during any one of the two phases, the timer will be reset. In subsequent discussion, we will use UPD(i) to indicate an update packet *from* sensor i, and use AQR(i), ACF(i) and ARJ(i) to indicate an alarm query, confirmation or rejection packet *regarding* sensor i, respectively. We will also refer to the sensor suspected of having a problem as the *target*.

The intuition behind using two timers instead of one is as follows. Let $P_{\mathrm{F\ A_{basic}}}$ and $P_{\mathrm{F\ A_{TP}}}$ be the probability of false alarm with respect to a monitoring sensor in the basic mechanism and the two-phase mechanism, respectively. Let $f(t)$ be the probability that there is no packet received from target i over time t. Let p be the probability that the coordination/alarm checking of TP fails. Then we have the following relationship. $P_{\mathrm{F\ A_{basic}}} \approx f(C_1+C_2)$ and $P_{\mathrm{F\ A_{TP}}} \approx f(C_1+C_2)p$, thus we have $P_{\mathrm{F\ A_{TP}}} < P_{\mathrm{F\ A_{basic}}}$ approximately.

Note that all control packets (UPD, AQR, ACF and ARJ) are broadcast to the immediate neighbors and are subject to collision, in which case packets involved in the collision are assumed to be lost. There are no acknowledgments or retransmissions. Also note that in the presence of data traffic, any packet received from a sensor should be taken as an update packet. Any data packet sent from a sensor can also cancel out the next scheduled UPD packet.

2.1. State transition diagram and detailed description

We will assume that the network is pre-configured, i.e. each sensor has an ID and that the control center knows the existence and ID of each sensor. However, we do not require time synchronization. Note that timers are updated/reset by the reception of packets. Differences in reception times due to propagation delays can result in slightly different expiration times in neighbors. A sensor keeps a timer for each of its neighbors, and keeps an instance of the state transition diagram for each of its neighbors. Fig. 2(a) shows the state transitions sensor s keeps regarding neighbor i. Fig. 2(b) shows the state transition of s regarding itself. They are described in more detail in the following.

Neighbor Monitoring: Each sensor broadcasts its update packet UPD with TTL = 1 with inter-arrival time chosen from some probability distribution with mean T. Each sensor has a neighbor monitoring timer $C_1(i)$ for each of its neighbor i with an initial value C_1. After sensor s receives UPD or any packet from its neighbor i, it resets timer $C_1(i)$ to the initial value. When $C_1(i)$ goes down to 0, a sensor enters the random delay state for its neighbor i. When sensor s receives an alarm query packet AQR(i) in neighbor monitoring, it broadcasts an alarm reject packet ARJ(i) with TTL = 1. When sensor s receives an alarm reject packet ARJ(i) in this state, it resets $C_1(i)$ to the reset (residual timer) value in carried in the ARJ if its own $C_1(i)$ is of a smaller value.

Random delay. Upon entering the random delay state for its neighbor i, s schedules the broadcast of an alarm query packet AQR with TTL = 1 and activates an alarm query timer $C_2(i)$ for neighbor i with initial value C_2. After the random delay, sensor s enters the alarm checking state by sending AQR. Note that if a sensor is dead, timers in a subset of neighbors expire at approximately the same time

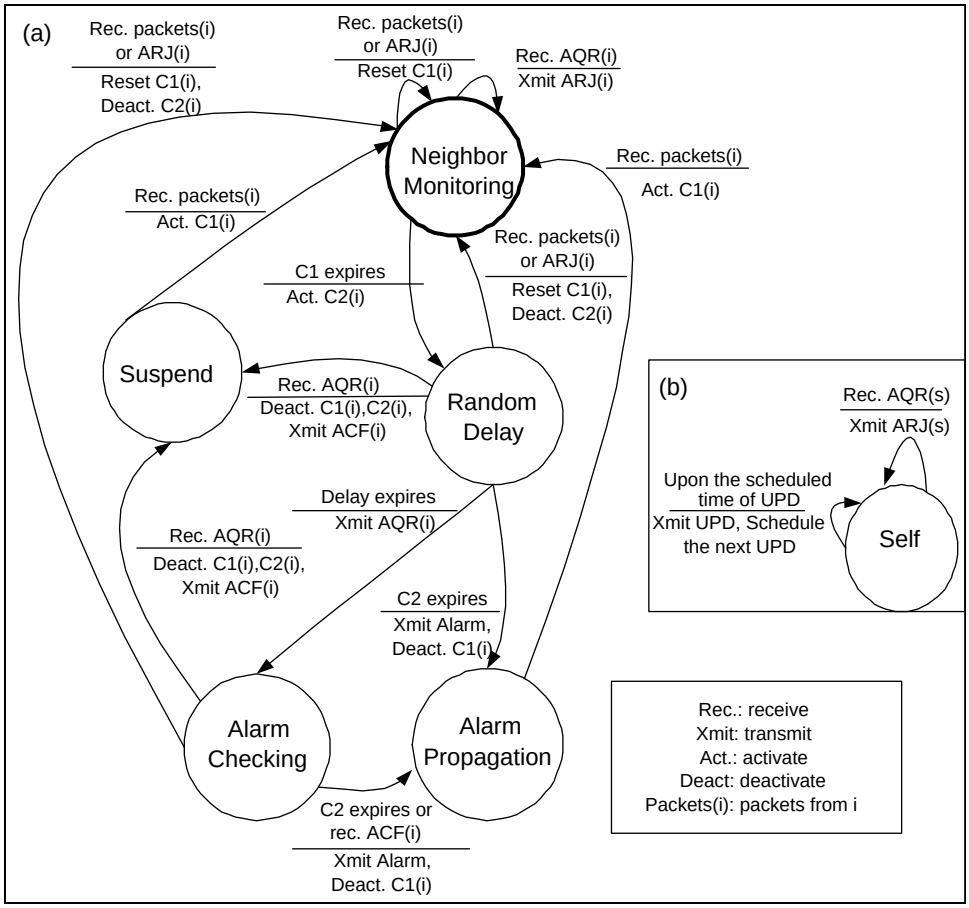

Fig. 2. State diagram for (a) neighbor i and (b) sensor s itself with transition conditions (condition/action).

(subject to differences in propagation delays which is likely very small in this case) with a high probability. The random delay therefore aims to de-synchronize the transmissions of AQR. Typically this random delay is smaller than C_2, but it can reach C_2 in which case the sensor enters the alarm propagation state directly from random delay. In order to reduce network traffic and the number of alarms generated, when sensor s receives AQR(i) in the random delay state, it cancels the scheduled transmission AQR(i) and enters the suspend state. This means that sensor s assumes that the sensor which transmitted AQR(i) will take the responsibility of checking and firing an alarm. Sensor s will simply do nothing. If sensor s receives any packet from i or ARJ(i) in the random delay state, it knows that i is still alive and goes back to neighbor monitoring. Sensor s also resets its $C_1(i)$ to C_1 if it receives packets from i.

Alarm checking. When sensor s enters the alarm checking state for neighbor i, it waits for the response ARJ from all its neighbors. If it receives any packet from i or ARJ(i) before $C_2(i)$ expires, it goes back to neighbor monitoring. Sensor s also resets its $C_1(i)$ to C_1 if it receives packets from i or to the $C_1(i)$ reset value in ARJ if it receives ARJ(i). When timer C_2 expires, sensor s enters the alarm propagation state.

Suspend. The purpose of the suspend state is to reduce the traffic induced by AQR and ARJ. If sensor s enters suspend for its neighbor i, it believes that i is dead. However, different from the alarm propagation state, sensor s does not fire an alarm for i. If sensor s receives any packet from i, it goes back to neighbor monitoring and resets $C_1(i)$ to C_1.

Alarm propagation. After sensor s enters the alarm propagation state, it deletes the target sensor i from its neighbor list and transmits an alarm to the control center via some route. If sensor s receives any packet from i, it goes back to the neighbor monitoring state and resets $C_1(i)$ to C_1. If sensor s receives packets from i after the alarm is fired within reasonable time, extra mechanisms are needed to correct the false alarm for i. On the other hand, a well-designed system should have very low false alarm probability. Thus this situation should only happen rarely.

Self. In the self state, if sensor s receives AQR with itself as the target, it broadcasts an alarm reject packet ARJ with TTL = 1. In this state, sensor s also schedules the transmissions of the update packets. In order to reduce redundant traffic, each sensor checks its transmission queue before scheduling the next update packet. After a packet transmission completes, a sensor checks its transmission queue. If there is no packet waiting in the queue, it schedules the next

transmission of the update packet based on the exponential distribution. If there are packets in the transmission queue, it will defer scheduling until these packets are transmitted. This is because each packet transmitted by a sensor can be regarded as an update packet from that sensor.

2.2. Variations within the two-phase mechanism

For performance comparison purposes, we will examine the following variations of the mechanism described above.

(1) *Alarm rejection only* (*ARJO*). Under this approach, a neighbor responds to an AQR(i) packet only when it has an active timer for *i*. It replies with an ARJ(i) along with the remaining value of its timer. If its first timer $C_1(i)$ has also expired it will not respond but will enter the suspend state. A sensor in the alarm checking state will thus wait till its second timer $C_2(i)$ expires to trigger an alarm.
(2) *Alarm confirmation allowed* (*ACFA*). This approach is the same as ARJO except that neighbors receiving AQR(i) are also allowed to respond with an ACF(i) if their timers also expired. We specify that upon receiving one ACF(i) packet, a sensor in the alarm checking state can terminate the state by triggering an alarm before the second timer C_2 expires. By doing so we can potentially reduce the response delay, but we will see that this comes with a price of increased false alarms.
(3) *Alarm target only* (*ATGO*). Under this approach, only sensor *i* is allowed to respond to an AQR(i) packet from sensor *s*. In other words, *s proactively probes i* to see if *i* is still alive. In doing so if *i* is dead *s* will have to wait for timeout before triggering an alarm. Intuitively by doing so the amount of responding traffic is reduced. However, the correlation between neighbors' observations is not utilized.

Note that a key to these schemes is the determination of parameters C_1 and C_2. This may be based on the expected time to the next UPD packet arrival (T) and the likelihood of a collision and consecutive collisions. In all the above schemes these parameters are assumed to be pre-determined. An alternative approach is to have the sensors announce the time to their next scheduled UPD transmission in the current UPD packet. This scheme (subsequently denoted by ANNOUNCE) does not require time synchronization as the time of next arrival may be computed by the local time of arrival of the current UPD and the time difference contained in the current UPD. This scheme may be used with any of the previous variations to determine the timeout value C_1.

3. Simulation studies

In this section, we compare the performance of the various schemes introduced in Section 2, denoted by 'basic', 'ARJO', 'ACFA', 'ATGO', and 'ANNOUNCE', respectively. Section 3.1 presents the results under the basic and the first three variation schemes, when the UPD inter-arrival time is exponentially distributed and the environment is noiseless (meaning that the packet losses are only due to collisions). As mentioned before, the proposed readily admits any arbitrary distribution. The choice of exponential in this part is largely due to the fact that the analysis in Section 4 is made tractable under this assumption and may be compared with simulation results. A second reason is that it provides a reasonably large variance of the interarrival times to randomize transmission times of UPD packets. In Section 3.2, we consider the effect of noise, i.e. packet losses are not only due to collisions, but also due to noise. Here noise is modeled via independent loss probability (in addition to collision loss) for every packet rather than via explicit computation of signal to noise ratio. The performance of ANNOUNCE is examined in Section 3.3.

Table 1
System parameters

	Notation	Value
UPD size	τ_{up}	60 bytes
AQR size	τ_{aq}	64 bytes
ARJ size	τ_{ar}	80 bytes
ACF size	τ_{ac}	64 bytes
Channel BW	W	20 kbps
Xmission range	R	200 m

Our simulation is implemented in Matlab. A total of 20 sensors are randomly deployed in a square area. Sensors are assumed to be static during the simulation. The transmission radius is fixed for all sensors, thus the size of the square area may be altered to obtain different sensor node degree, denoted by d (degree is defined as the average number of sensors within direct communication area of a sensor). For AQR(i), ARJ(i), and ACF(i) packets, a sensor waits for a random period of time exponentially distributed with rates $d\tau_{aq}/W$, $d\tau_{ar}/W$, and $d\tau_{ac}/W$ before transmission, where W is the channel bandwidth and τ_{aq}, τ_{ar}, and τ_{ac} are the sizes of the AQR, ARJ, and ACF packets, respectively.[1] During a simulation, sensor death events are scheduled periodically. The sensor death periods (time between two successive death events) are 100 time units and 500 time units when update periods T are 10 units and 60 units, respectively. Under the same sets of parameters (e.g., T), when the sensor death becomes more frequent, the amount of control packets incurred by sensor failure increases. Thus the false alarm probability increases due to the increasing packet collision. However, the response delay decreases because the control timers are less likely to be reset when packet collision increases. Table 1 shows the parameters and notations used in simulation results and subsequent analysis. Different

[1] The purpose is again to randomize the packet transmission times. The random delay also needs to scale with the network degree d and the packet transmission time.

devices have different packet overhead. The packet size ranges from tens of bytes [7] to hundreds of bytes [8]. Since we only consider control packets, we used 60 bytes as the basic size similar to [7]. Without specifying packet format, these choices are arbitrary but reasonable.

The following metrics are considered. The probability of false alarm, denoted by P_{FA}, the response delay, and the total power consumption. Denote the number of false alarms generated by sensor s for its neighbor i by α_{si} and denote the total number of packets received by s from i by β_{si}. $P_{\mathrm{F\,A}}$ is then estimated by

$$P_{\mathrm{F\,A}} = \frac{\sum_s \sum_i \alpha_{si}}{\sum_s \sum_i \beta_{si}}.$$

This is because s resets its timer upon every packet received from i, so each packet arrival marks a possible false alarm event. The response delay is measured by the delay between the time of a sensor's death and the time when the first alarm is triggered by one of its neighbors. The total power consumption is the sum of communication and idle power consumption. The former is calculated by counting the total transmission/receiving time and using the communication core parameters provided in [1]. The power dissipated in node n_1 transmitting to node n_2 is $(\alpha_{11}+\alpha_2 d(n_1,n_2)^2)r$ where $\alpha_{11}=45nJ$/bit, $\alpha_{12}=135nJ$/bit, $\alpha_2=10pJ/\mathrm{bit}/m^2$, r is the transmission rate in bits/s, and $d(n_1,n_2)$ is the distance between n_1 and n_2 in meters. The power dissipated in n_1 receiving from n_2 is $\alpha_{12}r$. The idle power consumption (sensing power and data processing power) per sensor is 1.92 mW based on the energy consumption ratio in [13]. Each data point is the average of multiple runs with a fixed set of parameters but different random topologies.

3.1. Comparison of different schemes

First, we will investigate a relatively high-alert system with $T=10$ time units. The amount of time per unit may be decided based on the application. Different time units will not affect the relative results shown here. In our simulation, we choose one time unit to be 1 s. The upper 2 graphs of Fig. 3 show the two performance measures with average update interval $T=10$, where C_2 is set to 1. With this set of parameters, the three two-phase schemes, ARJO, ATGO, and ACFA, result in very similar and much lower false alarm probability than the basic scheme. Compared to the basic scheme, the largest false alarm probability decrease is up to 82%. As expected, false alarm decreases as timer C_1 increases for all cases. The response delay under all schemes increases with C_1, with very little difference between different schemes (maximum 3 s). There is also no consistent tendency as to which scheme results in the highest or lowest response delay. The alarm confirmation scheme (ACFA) does not help reduce the overall response delay in this case because C_2 is very small, in which case a sensor either does not receive a confirmation, or the time saving due to confirmation is very limited.

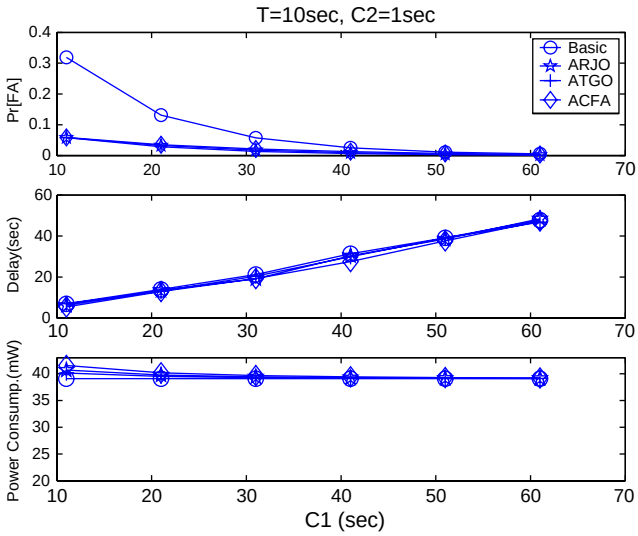

Fig. 3. Simulation results with $T=10$, $C_2=1$, and $d=6$.

The upper 2 graphs of Fig. 4 show results with $T=10$ and C_1 fixed at 21. False alarm decreases with increasing C_2, while the response delay increases. We see a dramatic reduction in the response delay when the alarm confirmation scheme (ACFA) is used. However, this does not come for free. Table 2 shows comparison between ARJO and ACFA at $C_2=31$. The increase in false alarm probability is mainly due to incorrect confirmation given by a neighbor. Due to the collision nature of the wireless channel and correlation in observations, multiple neighbors may have expired timers. This increase in false alarm can be potentially alleviated by requiring more than one confirmation packet (ACF) to be received before a sensor can trigger an alarm. However, the same trade-off between accuracy and latency remains. Furthermore, as can be seen in Fig. 4, the basic

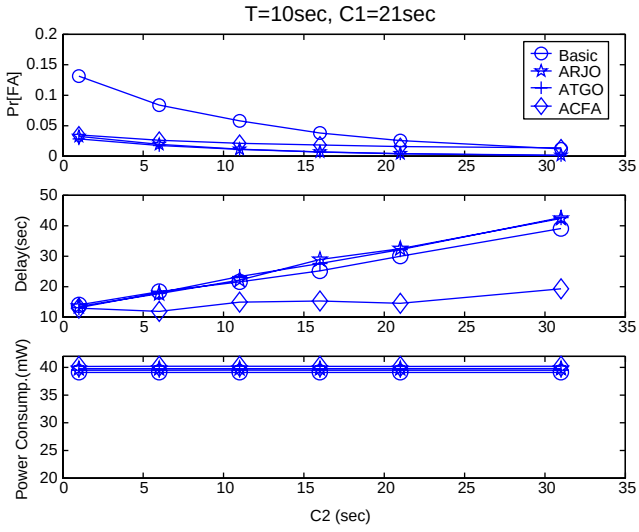

Fig. 4. Simulation results with $T=10$, $C_1=21$, and $d=6$.

Table 2
Effect of alarm confirmation

	ARJO	ACFA	Change (%)
P_{FA}	0.0014	0.0136	+871
Resp. delay	42.4	19.28	−54.5

scheme has slightly smaller response delay than ARJO and ATGO. This is because the last update packet (UPD) sent by neighbor i before i's death may be lost due to collision and therefore the timer for i is not reset in the basic scheme. When the timer expires, an alarm is sent. However, in ARJO and ATGO as long as the alarm coordination in $C_2(i)$ phase succeeds, the timers for i may still be reset even if the last UPD from i is lost. Therefore, the sensors in ARJO and ATGO need to wait for a longer time till the timers expire to send an alarm than the sensor in the basic scheme. When C_2 is larger, the alarm coordination is more likely to succeed and thus the response delay is larger.

The bottom graphs of Figs. 3 and 4 show the total power consumption. In the bottom graph of Fig. 3, we see that in general the two-phase schemes result in slightly higher energy consumption due to extra traffic incurred by the local coordination mechanism. When C_1 is not too large, ARJO and ACFA have the highest power consumptions. ATGO consumes less because only the target participates in alarm checking. When C_1 becomes larger, the difference becomes small. Overall the largest increase does not exceed 6%. In the bottom graph of Fig. 4, ACFA still has the highest power consumption since it generates most control traffic. The basic scheme still has the lowest power consumption. However, the power consumption does not vary with C_2. The reason is that the amount of controlled traffic is control by the length of C_1 rather than C_2. Overall the largest increase does not exceed 2.7%.

Now we investigate the case with $T=60$ time units. For brevity we only show the results for $T=60$ and $C_2=1$ in Fig. 5. For other parameter settings, the observations and the interpretations of the results remain largely the same as given here. As can be seen, the comparison and the change over C_1 are similar to the case of $T=10$. The two-phase schemes result in much lower probability of false alarm than the basic system. The response delay of different schemes are approximately the same, with differences within 10 s. Due to the light traffic, the power consumption of different schemes become more and more similar, and is thus not shown here. It may seem surprising that although on average a sensor updates its neighbors once every 60 s, there is still significant false alarms when C_1 is below 200 s. This is because as T increases, false alarms are more likely to be caused by the increased variance in update interval than caused by collisions as when T is small. Since the update intervals are exponentially distributed, in order to achieve low false alarm probability comparable to results shown in Fig. 3, C_1 needs to be set appropriately. In this case, either a constant update interval or the ANNOUNCE scheme may

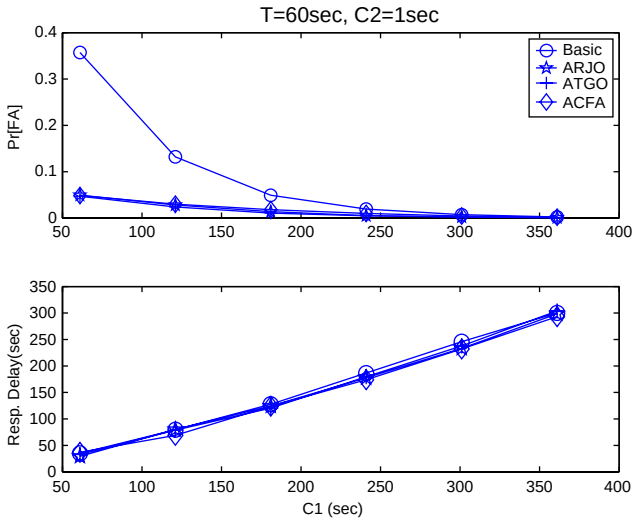

Fig. 5. Simulation results with $T=60$, $C_2=1$, and $d=6$.

be used to reduce the amount of uncertainty in estimating the time till the next UPD packet arrival.

We ran the same set of simulations for topology scenarios with average sensor degrees of 3 and 9[2]. Overall all results on performance comparisons remain the same as shown above. Power consumption increases with sensor degree. However, the increase in energy consumption by using the two-phase schemes remains very small. In Fig. 8 of Section 4, we will show that increasing degree does not necessarily mean decreasing false alarm probability.

3.2. Results in a noisy environment

Recent measurements on real systems [4] show that links with heavy packet loss are quite common due to the corrupted received packets under a noisy environment. Below we evaluate our system under a noisy environment. Let P_f be the probability that a received packet is corrupted due to the noisy environment (not due to packet collision). Fig. 6 shows the simulation results under the same scenario as Fig. 3 but with $P_f=0.1$ and $P_f=0.5$, respectively. (We only show the results for the basic system, ARJO, and ATGO for clearer presentation. The results for the confirmation scheme ACFA follow similar observations.) Comparing Figs. 3 and 6, we can see that the false alarm probability increases and the response delay decreases as P_f increases. This is because the last packet sent by neighbor i before i's death is more likely to be lost when P_f is larger. The subsequent alarm coordination in $C_2(i)$ phase is also more likely to fail when P_f is larger. Therefore, timers $C_1(i)$ and $C_2(i)$ are less likely to be reset when P_f is larger, which

[2] From Ref. [12], in order to achieve asymptotic connectivity each sensor should have number of neighbors $c\log(n)$ where c is a critical parameter (assigned to 1 here) and n is the total sensor number. When the degree is 9, the total number of sensors in which the asymptotic connectivity can still be achieved is $e^9 \simeq 8103$, which is a fairly large network.

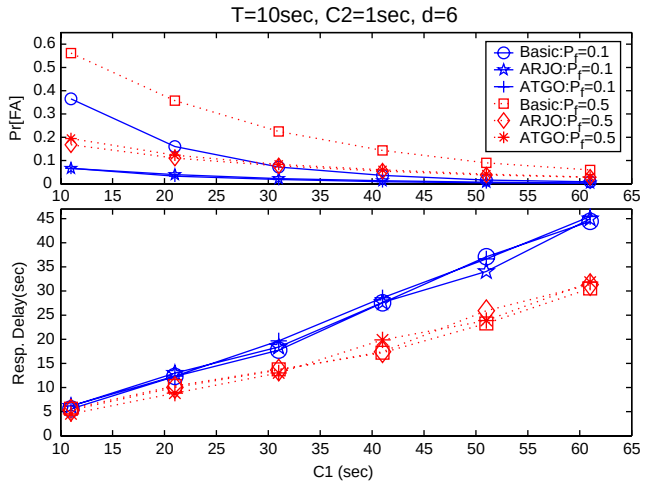

Fig. 6. Simulation results with $T=10$, $C_2=1$, $d=6$.

results in less waiting time (response delay) till the timer expiration to send an alarm for i. If we want to satisfy a predefined false alarm probability limit, we need to use different initial values for $C_1(i)$ or $C_2(i)$ under different environment, i.e. different P_f. In Section 5, we present a method for sensors to adjust the initial timer values according to changes in the environment.

3.3. Announcement of the time till next UPD

ANNOUNCE was described in Section 2.2. In the simulation the initial value of $C_1(i)$ is set to $2T+1$. Upon receiving a UPD packet from neighbor i containing a time difference t_i (time to next UPD arrival), $C_1(i)$ is updated to be equal to t_i. When using ANNOUNCE with the basic scheme, only one timer of value $C_1(i)$ is used. All other operations remain the same as before.

Fig. 7 shows the simulation results with $T=10$ and $d=6$ under this scheme. Since the basic scheme has only one timer updated by the received t_i, which is unaffected by the value of C_2, it has approximately constant false alarm probability and response delay. In contrast, the basic scheme in Fig. 4 produces lower false alarm probabilities at large values of C_2. This is primarily because in Fig. 4 the basic scheme has a timer value of C_1+C_2 with fixed C_1 and increasing C_2, whereas under ANNOUNCE the basic scheme has timer values of $C_1(i)$ updated by t_i. The latter essentially does not leave any room for the possibility of packet losses, resulting in higher false alarm probability. ARJO, ATGO, and ACFA in Fig. 7 have higher false alarm probabilities than the ones in Fig. 4 for all C_2 values simulated (especially for ARJO and ACFA with a lot of control traffic and thus large packet collision probabilities). Correspondingly the response delays are in general lower than those shown in Fig. 4. The power consumption in Fig. 7 is similar to the one in Fig. 4 and is not shown here.

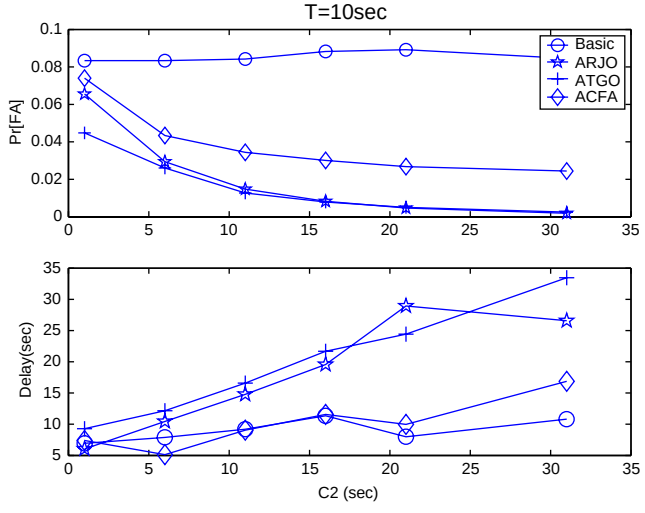

Fig. 7. Simulation results with $T=10$ and $d=6$ using ANNOUNCE.

The results shown here indicate that even though better estimates on when to expect the next UPD packet may be obtained via explicit announcement, the uncertainty in packet collision can still result in high false alarm probabilities. In particular for exponential distribution with mean $T=10$, setting $C_1=21$ in Fig. 4 means that C_1 is roughly the mean plus one standard deviation, which results in a value greater than t_i (which is drawn from the same distribution) most of the time. Consequently Fig. 4 is a result of on average larger timeout values C_1+C_2, hence lower false alarm and high response delay (as we will show in the next section, the performance of these schemes primarily depends on the combined C_1+C_2). We could easily increase C_1 to be the *sum* of t_i contained in the received UPD packet and some extra time, which should lead to reduced false alarm probability, but the tradeoff between the two performance measures would remain the same. On the other hand, as mentioned before as T increases packet collision decreases and thus the performance of ANNOUNCE is expected to improve.

3.4. Discussion

These simulation results show that the two-phase approach performed very well with the parameters we simulated. We can achieve much lower false alarm probability compared to a single timer scheme for a given response delay requirement (up to 82% decrease). Equivalently we can achieve much lower response delay for a given false alarm probability requirement. In particular, the alarm confirmation scheme (ACFA) poses additional flexibility in achieving a desired trade-off. Such benefit comes with a very slight increase in energy consumption. In general, false alarm probability decreases with increased timer values C_1 and C_2, while the response delay increases. The choice of update rate $1/T$ also greatly affects these measures. In the next section, we will show how to choose

these system parameters via analysis for certain instances of this scheme.

In the simulation results shown, we only considered control traffic incurred by our algorithms. In reality, other background traffic may also be present in the network. As mentioned earlier, all traffic may be regarded as update packets from neighbors. Therefore, if the background traffic is light (e.g. less frequent than the scheduled UPD packets), then it simply replaces some of the regularly scheduled UPD packets, and the performance of the system should not change much. On the other hand, if the background traffic is heavy, it mounts to updating neighbors very frequently which may completely replace the regularly scheduled UPD packets. The performance, however, in this scenario is not easy to predict since the background traffic is likely to require reliable transmission between nodes (e.g. using the collision avoidance method RTS-CTS sequence provided by IEEE 802.11, and packet retransmission).

There are many other choices in addition to an exponential distribution that was used for scheduling UPD packets in these simulations. In general, sufficient variance in this candidate distribution is desired as it properly de-synchronize the transmission of UPD packets (with the exception of perhaps the scenario of very light traffic, i.e. long update cycles, where initial randomization followed by fixed update intervals or the ANNOUNCE scheme may suffice). On the other hand, this variance should not be too large, as it may make the determination of values C_1 and C_2 difficult. In general, there is a delicate balance between (1) sufficiently large timeout values to ensure high confidence that the timer expires because a sensor is indeed dead rather than because there happens to be a very large interarrival time of the UPD packets or because there has been a packet collision, and (2) not overly large timeout values to ensure reasonable response time.

4. System analysis and optimization

In this section, we attempt to analyze the performance of the scheme introduced in this paper for a few simplified instances. The goal is to obtain further insight into the relationship between the performance of the system and the timer values C_1 and C_2. If the analytical models are accurate enough, then they can also be potentially used to properly select these system parameters for certain performance requirement on false alarm probability and response delay.

We assume that the UPD inter-arrival time is exponential distributed with rate $1/T$. We assume that the transmission queue in each sensor is stable and the arrival rate equals the departure rate. This is because compared to channel bandwidth, the packet generation rate is typically very small. We also assume that the packet propagation delay, data processing delay in a sensor, and transmission queuing delay are all negligible. Furthermore, the amount of sensor update packets, UPD, is much larger than AQR and ARJ packets. Thus we may only consider the collision induced by UPD packets. Packet loss probability due to a noisy environment is denoted by P_f.

Below we first define reference values for C_1 and C_2 that serve as benchmarks. We then provide models of $P_{\mathrm{F\,A}}$ and response delay, denoted by D, as functions of a number of system parameters for the basic system, ARJO, and ATGO schemes, respectively. We will not consider ACFA here for brevity as ACFA is significantly more complicated than other variations and require additional assumptions. The details of the derivations of the results presented below can be found in Appendix A.

Result 1. The reference values of the two timers C_1 and C_2 are:

$$C_1^* = \frac{T}{1-P_f}\exp\left(\frac{2d\tau_{up}}{TW}\right),$$

and

$$C_2^* = \frac{\tau_{aq}+\tau_{ar}+4d\tau_{up}}{W}.$$

Result 2. Basic scheme has

$$P_{\mathrm{F\,A}} = \exp\left(-\frac{(1-P_f)\mathrm{e}^{-2d\tau_{up}/TW}(C_1+C_2)}{T}\right)$$

and

$$\begin{aligned} D \approx {} & P_{\mathrm{succ}}\int_0^{C_1+C_2}(C_1+C_2-x_1)\frac{\mathrm{e}^{-x_1/T}}{T}\mathrm{d}x_1 + P_{\mathrm{succ}}(1-P_{\mathrm{succ}}) \\ & \times\int_0^{C_1+C_2}\int_0^{C_1+C_2-x_2}(C_1+C_2-x_1-x_2)\frac{\mathrm{e}^{-x_1/T}}{T} \\ & \times\frac{\mathrm{e}^{-x_2/T}}{T}\mathrm{d}x_1\mathrm{d}x_2, \end{aligned}$$

where

$$P_{\mathrm{succ}} = (1-P_f)\mathrm{e}^{-2d\tau_{up}/TW}.$$

Result 3. ARJO has

$$\begin{aligned} P_{\mathrm{FA}} = {} & \exp\left(-\frac{(1-P_f)^2\mathrm{e}^{-(2d\tau_{up}/TW)}(C_1+C_2)}{T}\right) \\ & \times\left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right] \\ & \times\left\{\exp\left(-\frac{(1-P_f)\mathrm{e}^{-(2d\tau_{up}/TW)}C_1}{T}\right)\right. \\ & +\left(1-\exp\left(-\frac{(1-P_f)\mathrm{e}^{-(2d\tau_{up}/TW)}C_1}{T}\right)\right) \\ & \left.\times\left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right]\right\}^{0.6885d} \end{aligned}$$

and

$$D \approx P_{\text{succ}} \int_0^{C_1+C_2} (C_1+C_2-x_1)\frac{e^{-x_1/T}}{T}dx_1 + P_{\text{succ}}(1-P_{\text{succ}}) \times \int_0^{C_1+C_2}\left[P\int_0^{C_1+C_2-x_2}(C_1+C_2-x_1-x_2)\frac{e^{-x_1/T}}{T}dx_1 + (1-P)\int_0^{C_1+C_2}(C_1+C_2-x_1)\frac{e^{-x_1/T}}{T}dx_1\right]\frac{e^{-x_2/T}}{T}dx_2,$$

where

$$P = \left\{\exp\left(-\frac{(1-P_f)e^{-(2d\tau_{up}/TW)}C_1}{T}\right) + \left(1-\exp\left(-\frac{(1-P_f)e^{-2d\tau_{up}/TW}C_1}{T}\right)\right) \times\left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right]\right\}^{0.6885d}.$$

Result 4. ATGO has

$$P_{\text{FA}} = \exp\left(-\frac{(1-P_f)e^{-(2d\tau_{up}/TW)}(C_1+C_2)}{T}\right) \times\left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right]$$

and the response delay is the same as Result 2.

First consider P_{FA}. We will use Result 4 instead of Result 3 since the former is much easier to interpret. We see that P_{FA} decreases exponentially with the increase in the ratio (C_1+C_2/T). Though not a surprising result, it is interesting to observe that it is the sum of C_1+C_2 that determines the false alarm rather than individual values (same with response delay). This is because the performance is decided by how packets are received over the sum of C_1 and C_2 period; furthermore, the individual C_2 value ($C_2>0$) does not affect the probability that the alarm coordination succeeds since AQR and ARJ packets are only transmitted once (no retransmission) and we have discarded the queuing delay in the derivation. However, individual C_1 and C_2 values do have an effect on energy consumption. Furthermore, increasing the channel bandwidth W will decrease P_{FA}.

We next consider the effect of the degree d on P_{FA} of ARJO. On the one hand, P_{FA} tends to decrease with increasing degree since there are more sensors to correctly report the state of a sensor; on the other hand, the probability of collision also increases with node degree, which leads to higher number of false alarm events. The combined effect of these two factors is shown in Fig. 8, computed from Result 3. As can be seen, when T is small (e.g. $T=1$ in Fig. 8), the packet collision resulted from the increasing degree d outweighs the benefit of alarm coordination. When T is large (e.g. $T=7$ in Fig. 8), the benefit of alarm coordination becomes more prominent as d increases.

Below we compare the numerical results from the above models with those of simulation. For brevity we only show the results for $T=10$, degree $d=6$, $P_f=0$, and $C_2=1$ s.

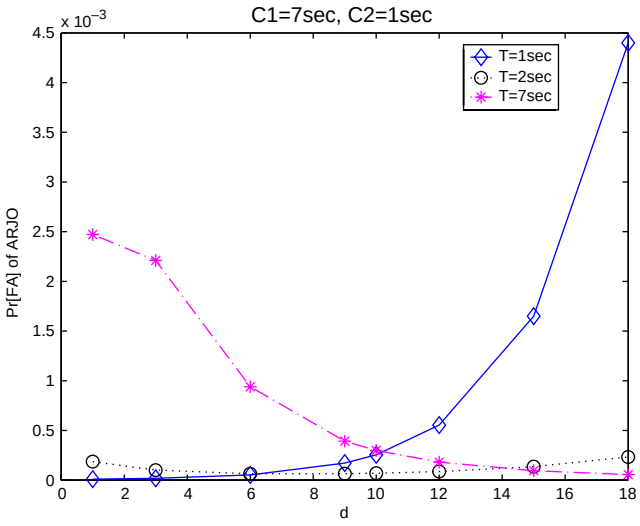

Fig. 8. P_{FA} of ARJO with $C_1=7$ s and $C_2=1$ s.

The observations are similar under other parameter settings. The initial values of C_1 and C_2 are chosen based on the analytical reference values. Fig. 9 shows numerical results with $T=10$ and $C_2=1$. The analysis and the simulation of P_{FA} of the basic system match quite well. However, analysis on ARJO and ATGO are less satisfactory, producing gross underestimates. The main reason with ARJO is that we assumed that two unsuccessful packet receiving events at two neighbors are independent. When a sensor did not successfully receive a broadcast packet, it is likely that its neighbors also did not successfully receive the same broadcast due to the shared wireless channel. Thus those two events may be highly correlated. The discrepancy with ATGO is likely related to the fact that as probability of false alarm falls very low, the variance in simulation results increases. For example, a small rise in the number of false alarm events (numerator of P_{FA} measure) caused by a random perturbation in the simulation can result in a large difference when probability is low. For the response delay,

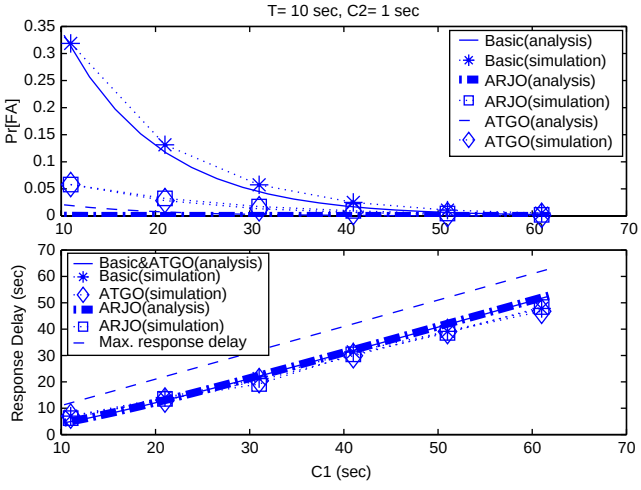

Fig. 9. Numerical results with fixed C_2.

the analysis gives very good estimates under all schemes examined, with differences not exceeding 3 s.

5. Self-parameter tuning function

In previous sections, we have shown via both simulation and analysis that the system performance greatly depends upon system parameters, in particular C_1 and C_2. On the other hand, given different deployment environment, e.g. node degree or noise level, the best values of C_1 and C_2 may vary. It is thus desirable to have a mechanism that adapts to an unknown and potentially changing environment and produces a stable false alarm probability under different (or changing) conditions. This means that the sensors need to dynamically adjust their C_1 and C_2. Below we show one simple way of achieving this, by using the average of packet reception rate as an indicator of how good the environment is and adjusting these parameter accordingly.

When a sensor s receives the a-th packet from sensor i, it updates its initial value of $C_1(i)$ to a moving average

$$C_1 = M\frac{\sum_{b=a-N+1}^{a} t_b}{N},$$

where t_b is the reception period between the $(b-1)$-th and the b-th packets received from sensor i, N is the size/window of the moving average, and M is a control parameter. A larger M results in smaller false alarm probabilities but higher response delays. When sensor s receives the a-th ARJ from sensor i, it updates its initial value of $C_2(i)$ to

$$C_2 = M\frac{\sum_{b=a-N+1}^{a} t'_b}{N},$$

where t'_b is the period between the b-th AQR sent and the b-th ARJ received from sensor i.

This self-parameter tuning function is essentially a low-pass filter of the received packet interarrival times. Here for brevity we only show the results of ARJO with $T=10$, $d=6$, $N=20$, and note that the results are similar in other cases. Fig. 10 gives the simulation results for $M=2$ and $M=4$ under different P_f. Comparing these results to that shown in Fig. 6, we see that the false alarm probabilities for both $M=2$ and $M=4$ are much more stable under different P_f compared to the difference of the false alarm probabilities with $P_f=0.1$ and $P_f=0.5$ in Fig. 6. In Fig. 10, the response delay increases with P_f because sensors use larger C_1 and C_2 when the packet losses are more often, i.e. P_f is larger. A large M (e.g. $M=4$ in Fig. 10) gives us large C_1 and C_2, which results in small false alarm probabilities but large response delays.

6. Related work

Problems related to the monitoring of a sensor network have been studied in the literature for various applications and purposes. [2] proposed a single timeout scheme to monitor the system-level fault diagnosis. [14] proposed an approach to construct abstracted scans of sensor network health by applying in-network aggregation of network state. Specifically, in [14] a residual energy scan was designed to approximately depict the remaining energy distribution within a sensor network. Such in-network aggregation was also studied in [7]. Ref. [9] proposed an approach to trace the failed nodes in sensor networks, whereby the corrupted routes due to node failure can be recovered. The approach used in [9] was centralized, where the control center (base station) monitors the health of sensors and recovers the corrupted routes. Coordination among neighbors similar to the one used in this paper was also proposed in [8] for information dissemination among sensors, where sensors use meta-data negotiations to eliminate the transmission of redundant data. [10] presented a hierarchical architecture for the self-organization of sensor networks, in which a monitoring mechanism was proposed. Under this scheme, sensors report periodic updates to their cluster heads. This falls under the active monitoring method mentioned before, whose efficiency depends on the size of the cluster.

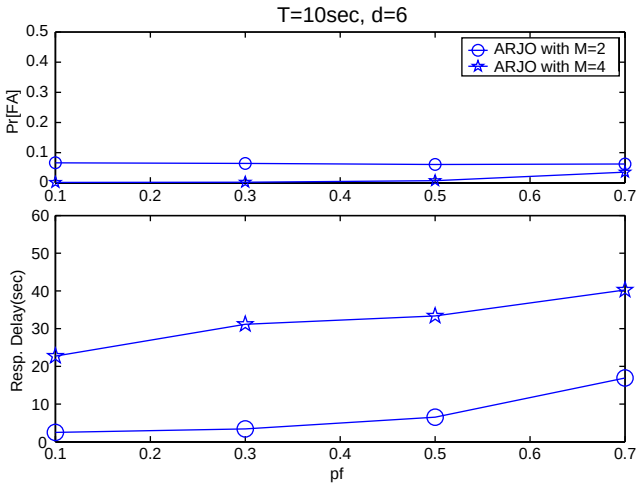

Fig. 10. Performance of the self-parameter-tuning function.

7. Conclusion

In this paper, we presented a comprehensive two-phase mechanism developed for the self-monitoring of a large sensor network using neighbor monitoring and local coordination. Via both simulation and analysis, we showed that this two-phase, self-monitoring mechanism achieves low false alarm probability without increasing the response delay and with very limited extra power consumption. We presented variations of this mechanism which help achieve better tradeoff between the two performance objectives.

Acknowledgements

This work was supported in part by the Engineering Research Centers Program of the National Science Foundation under NSF Award Number EEC-9986866. An earlier version of this work reporting part of the results here appeared in 2002 ACM Workshop on Wireless Security (WiSe'02).

Appendix A. System analysis

A.1. Timer references

It is reasonable to choose a reference value of C_1 to be the expected time for a sensor (say node A) to receive successfully a UPD packet from a given neighbor. For this to happen, no other neighbors of A (including A itself) may transmit simultaneously. Adopting a random access model (unslotted), the transmissions of UPD from different sensors are independent. Thus we may model the UPD transmissions by all A's neighbors excluding A as a single Poisson process with rate d/T. Then the probability of A successfully receiving a UPD from any given neighbor is the probability that no other transmission takes place during the period $2\tau_{up}/WT$, i.e.

$$P_{\text{succ}} = (1 - P_f)\exp\left(-\frac{2d\tau_{up}}{TW}\right)$$

Hence the successful reception of UPDs at A from a given neighbor is a Poisson process with rate

$$\lambda_{ra} = \frac{Psucc}{T}.$$

The average time between two consecutive successful UPDs from this neighbor is $1/\lambda_{ra}$, which is taken to be the reference value for C_1, i.e.

$$C_1^* = 1/\lambda_{ra} = \frac{T}{(1 - P_f)}\exp\left(\frac{2d\tau_{up}}{TW}\right).$$

Next consider C_2. For a neighbor to respond with an ARJ, the delay is approximately the summation of the transmission delays of AQR and ARJ and twice the average random delay. Note that under the assumptions given earlier, ARJ transmissions are always successful. The random delay is exponentially distributed with rate $\mu_{ra} = (W/d\tau_{up})$ with mean $1/\mu_{ra}$. We take twice the mean for the purpose of deriving a reference value for C_2, which is taken to be

$$C_2^* = \frac{\tau_{aq} + \tau_{ar} + 4d\tau_{up}}{W}.$$

This reference is a little more than the expected time it takes to successfully transmit an AQR and successfully receive an ARJ.

A.2. Performance of the basic scheme

The basic scheme only has one timer of initial value $C_1 + C_2$. Given all sensors are alive, the probability that a sensor transmits a false alarm packet targeting a neighbor is the probability that this sensor has not received UPD from that neighbor for $C_1 + C_2$. We know that the successful reception of UPDs is a Poisson process with rate λ_{ra}, which immediately leads to the expression of P_{FA} given in Result 2.

In calculating the response delay, we use Fig. A1 to illustrate possible scenarios. In Fig. A1 (1), a sensor successfully receives the last UPD from i, but sensor i is dead x_1 time units after this event. Denoting the response delay in this case by D_0, we have $D_0 = C_1 + C_2 - x_1$. The delay between the last update and the death event is a random variable denoted by X_1. In Fig. A1 (2), a sensor fails to receive the last UPD from i but did successfully receive the one before the last one. Thus the response delay in this case, denoted by D_1, is $D_1 = C_1 + C_2 - x_1 - x_2$. The delay between two consecutive UPD from i is also a random variable denoted by X_2. In general, we can define A_i to be the event that a sensor has lost the last i (consecutive) UPDs sent by another sensor that subsequently died. D_i is then defined as the response delay as a result of event A_i. Note that events A_i form a partition of the space of all possible events that can happen to a sensor in detecting another sensor's death. We only consider consecutive losses since any reception will reset the timer. We can then derive the average response delay:

$$\begin{aligned} D &= \sum_{i=0}^{\infty} \Pr[A_i]D_i = P_{\text{succ}}D_0 + P_{\text{succ}}(1 - P_{\text{succ}})D_1 \\ &\quad + P_{\text{succ}}(1 - P_{\text{succ}})^2 D_2 + \cdots = P_{\text{succ}}D_0 \\ &\quad + P_{\text{succ}}(1 - P_{\text{succ}})D_1 + o(P_{\text{succ}}) \end{aligned}$$

The following can be obtained:

$$D_0 = \int_0^{C_1+C_2} (C_1 + C_2 - x_1) f_{X1}(x_1)\mathrm{d}x_1$$

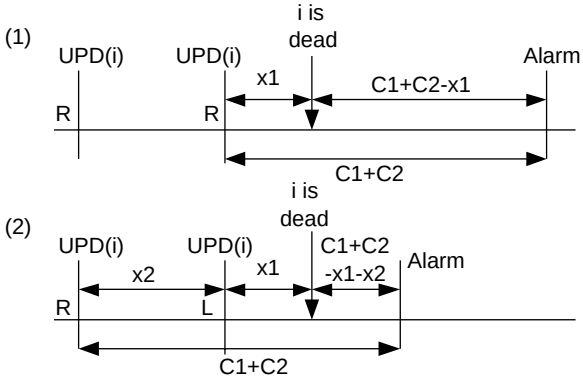

Fig. A1. Possible scenarios. 'R' means that UPD is received from i. 'L' means that UPD is lost.

and

$$D_1 = \int_0^{C_1+C_2}\int_0^{C_1+C_2-x_2}(C_1+C_2-x_1-x_2) \times f_{X1}(x_1)f_{X2}(x_2)\mathrm{d}x_1\mathrm{d}x_2$$

where $f_{x1}(x_1)$ and $f_{x2}(x_2)$are pdf's of X_1 and X_2, respectively. Both are exponentially distributed with rate $1/T$, assuming sensor death is a random event. For D_0, the range of x_1 is from 0 to C_1+C_2. If x_1 is larger than C_1+C_2, a false alarm event will happen which results in a negative response delay. This, however, is considered as a false alarm event and not taken into account in deriving response delay. Result 2 given in Section 4 follows.

A.3. Performance of alarm reject only scheme (ARJO)

Consider the simple scenario shown in Fig. A2 and assume all sensors are alive. A sensor would send out a false alarm if it failed to receive a UPD from a neighbor within C_1+C_2, and failed to receive an ARJ from its neighbors within C_2. We denote these two events by E_1 and E_2, respectively. We thus have $P_{\mathrm{F\,A}}=\Pr[E_1]\ \Pr[E_2|E_1]$, where $\Pr[E_1]$ was obtained earlier.

Further define E_3 to be the event that after C_1 expires and an AQR is sent, a sensor (e.g., B in Fig. A2) does not receive an ARJ from the target (e.g., A in Fig. A2) within C_2. Define E_4 to be the event that under the same scenario a sensor does not receive an ARJ from a common neighbor of itself and the target (e.g., C in Fig. A2) within C_2. Define E_5 to be the event that a common neighbor's timer C_1 has also expired when the ARJ was transmitted (by B). Define E_6 to be the event that the AQR-ARJ communication failed between the sensor (e.g., B) who initiated the AQR and a common neighbor (e.g., C). Assuming that the events of not successfully receiving ARJ from neighbors are mutually independent, and assuming the expected number of common neighbors are d', we have the following approximation: $\Pr[E_2|E_1]\approx \Pr[E_3](\Pr[E_4])d' = \Pr[E_3](\Pr[E_5]+\ \Pr[\bar{E}_5]\Pr[E_6|\bar{E}_5])^{d'}$.

The probability of E_5 is the probability that the neighboring sensor has not received UPD from the target in C_1. Thus,

$$\Pr[E_5] = \exp\left(-\frac{(1-P_f)\mathrm{e}^{-(2d\tau_{up}/TW)}}{T}C_1\right).$$

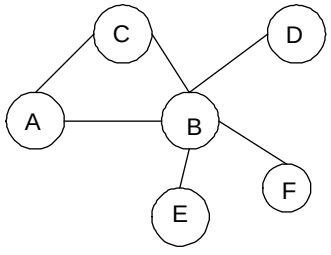

Fig. A2. A basic scenario.

Continuing with the same example with sensors A, B and C in Fig. A2, AQR-ARJ communication between two sensors may fail because the AQR failed or the ARJ failed when AQR was successful. Recall that we consider only the collisions of UPD packets. If C wants to successfully receive AQR from B, other neighbors of C including C itself cannot transmit UPD when B is transmitting AQR. Therefore, the probability that C successfully receives AQR from B is the probability that there are no other transmissions in a period of length $(\tau_{up}+\tau_{aq})/W$ times the probability of no failure due to noise. Since the transmission of UPDs of C and C's neighbors excluding B is modeled as a single Poisson process with rate d/T, the probability AQR succeeds is

$$(1-P_f)\exp\left(-\frac{d(\tau_{up}+\tau_{aq})}{TW}\right).$$

Similarly, given an AQR succeeds, an ARJ succeeds, with probability

$$(1-P_f)\exp\left(-\frac{d(\tau_{up}+\tau_{ar})}{TW}\right).$$

Therefore, between two sensor exchanging AQR and ARJ, we have the following:

$$\begin{aligned}\Pr[E_6|\bar{E}_5] &= \Pr[\text{AQR fails}] + \Pr[\text{AQR succeeds}]\\ &\quad \Pr[\text{ARJ fails}|\text{AQR succeeds}]\\ &= 1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right).\end{aligned}$$

The probability that a sensor does not successfully receive an ARJ from the target itself is given by

$$\Pr[E_3] = 1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right).$$

Thus

$$\begin{aligned}\Pr[E_2|E_1] \approx{}& \left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right]\\ &\times\left\{\exp\left(-\frac{(1-P_f)\mathrm{e}^{-(2d\tau_{up}/TW)}}{T}C_1\right)\right.\\ &+\left(1-\exp\left(-\frac{(1-P_f)\mathrm{e}^{-(2d\tau_{up}/TW)}}{T}C_1\right)\right)\\ &\left.\times\left[1-(1-P_f)^2\exp\left(-\frac{d(\tau_{aq}+\tau_{ar}+2\tau_{up})}{TW}\right)\right]\right\}^{d'}.\end{aligned}$$

The average number of common neighbors to two neighboring sensors can be calculated (see Appendix A.5) to be $d'=0.6885d$. This leads to Result 3 given in Section 4.

The derivation of the response delay for the ARJO scheme is similar to what we showed with the basic scheme. One scenario that needs special attention is shown in Fig. A3. Here when Sensor A does not successfully receive the last UPD from i, $C_1(i)$ will expire eventually, which results in the alarm checking process. As shown, when $C_1(i)$

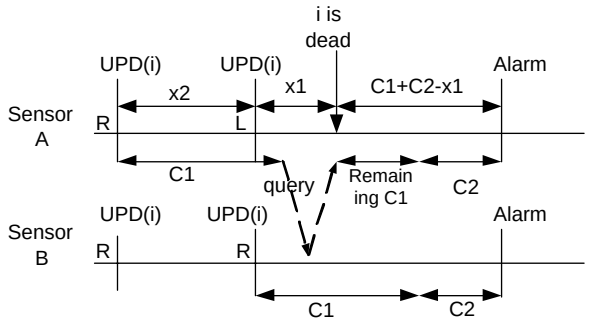

Fig. A3. A possible scenario. 'R' means that UPD is received from *i*. 'L' means that UPD is lost.

in A expires, A will consult its neighbor B. If B receives the last UPD from *i* and the transmissions of AQR(i) and ARJ(i) are successful, A resets its $C_1(i)$ to the remaining $C_1(i)$ value sent by B. Thus, the response delay under this scenario is $C_1+C_2-x_1$.

If B fails to receive the last UPD from *i*, the response delay would be $C_1+C_2-x_1-x_2$, same as in Fig. A1 (2). Note that A can consult with all its neighbors including B. Let *P* denote the probability that this consulting fails, which was derived earlier. If A successfully receives the last UPD from *i*, the response delay is $C_1+C_2-x_1$, same as in Fig. A1 (1). Again ignoring higher order terms we get *D* given in Result 3 in Section 4.

A.4. Performance of the alarm target only scheme (ATGO)

When a sensor transmits AQR(i) for a target *i*, only the sensor which is the target can reply with ARJ(i). It is easy to derive the probability that a sensor sends out a false alarm in this case by setting the number of common neighbors to zero in the previous result. The response delay is the same as in the basic scheme because the target cannot reply as it is dead.

A.5. Average number of common neighbors

Fig. A4 shows the transmission regions of two sensors A and B, each with transmission range of *R*. Assuming that the network density is fixed and that the distance *r* from a sensor to its neighbor is uniformly distributed in [0, *R*], The size of the shaded area may be shown to be

$$4\int_{r/2}^{R}\sqrt{R^2-x^2}\,\mathrm{d}x = 4\int_{\sin^{-1}\frac{r}{2R}}^{\pi/2} R^2\cos^2\theta\,\mathrm{d}\theta$$

$$= 2R^2\left[\theta+\frac{1}{2}\sin 2\theta\right]_{\sin^{-1}\frac{r}{2R}}^{\pi/2}$$

$$= 2R^2\left[\frac{\pi}{2}-\sin^{-1}\frac{r}{2R}-\frac{r}{2R}\sqrt{1-\left(\frac{r}{2R}\right)^2}\right].$$

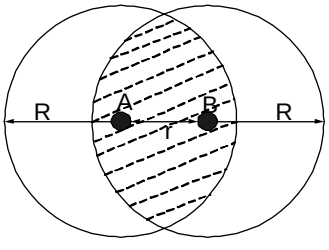

Fig. A4. Transmission regions of A and B.

Consequently, the average number of sensors in the shaded area = (average size of the shaded area) × density =

$$\frac{1}{R}\int_0^R\left[\pi R^2-2R^2\sin^{-1}\frac{r}{2R}-\frac{1}{2}r\sqrt{4R^2-r^2}\right]\mathrm{d}r\,\frac{d}{\pi R^2}$$

$$= 0.6885d.$$

References

[1] M. Bhardwaj, A.P. Chandrakasan, Bounding the lifetime of sensor networks via optimal role assignments, in: Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002.
[2] S. Chessa, P. Santi, Comparison based system-level fault diagnosis in ad hoc networks, IEEE 20th Symp. Reliable Distributed Systems (SRDS) 2001;.
[3] D. Ganesan, R. Govinda, S. Shenker, D. Estrin, Highly resilient, energy efficient multipath routing in wireless sensor networks, Mobile Comput. Commun. Rev. (MC2R) 1 (2) (2002).
[4] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, S. Wicker, Complex behavior at scale: an experimental study of low-power wireless sensor networks, in: UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013, 2002.
[5] W. Heinzelman, A. Chandrakasa, H. Balakrishnan, Energy-efficient communication protocols for wireless microsensor networks, in: Proceedings of Hawaiian International Conference on Systems Science, 2000.
[6] C.-F. Hsin, M. Liu, A distributed monitoring mechanism for wireless sensor networks, in: Proceedings of ACM Workshop on Wireless Security (WiSe), 2002.
[7] C. Intanagonwiwat, D. Estrin, R. Gonvindan, Impact of network density on data aggregation in wireless sensor networks, in: Proceedings of International Conference on Distributed Computing Systems (ICDCS), 2001.
[8] J. Kulik, W.R. Heinzelman, H. Balakrishnan, Negotiation-based protocols for disseminating information in wireless sensor networks, ACM Wireless Networks, 2002.
[9] J. Staddon, D. Balfanz, G. Durfee, Efficient tracing of failed nodes in sensor networks, in: Proceedings of First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), 2002.
[10] L. Subramanian, R.H. Katz, An architecture for building self-configurable systems, in: Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2000), 2000.

[11] A. Woo, D.E. Culler, A transmission control schemes for media access in sensor networks, in: Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 2001.
[12] F. Xue, P.R. Kumar, The number of neighbors needed for connectivity of wireless networks, Wireless Networks, 2003.
[13] W. Ye, J. Heidemann, D. Estrin, An energy-efficient mac protocol for wireless sensor networks, in: Proceedings of Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002.
[14] Y.J. Zhao, R. Govindan, D. Estrin, Residual energy scan for monitoring sensor networks, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'02), 2002.