Ad Hoc Networks 13 (2014) 549–559

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Delay monitoring for wireless sensor networks: An architecture using air sniffers

Wei Zeng [a,*], Jordan Cote [a], Xian Chen [a], Yoo-Ah Kim [b], Wei Wei [a], Kyoungwon Suh [c], Bing Wang [a], Zhijie Jerry Shi [a]

[a] Computer Science & Engineering Department, University of Connecticut, Storrs, CT, United States
[b] National Center for Biotechnology Information, National Institutes of Health, Bethesda, MD, United States
[c] School of Information Technology, Illinois State University, Normal, IL, United States

ARTICLE INFO

Article history:
Received 11 September 2012
Received in revised form 13 March 2013
Accepted 15 October 2013
Available online 24 October 2013

Keywords:
Delay monitoring
Abnormal delay detection
Wireless sensor networks
Sensor network management

ABSTRACT

Wireless sensor networks have been used for many delay-sensitive and safety–critical applications, e.g., emergency response and plant automation. For such applications, delay measurement inside the sensor networks is important for real-time monitoring and control of the networked system, and abnormal delay detection. In this paper, we propose a measurement architecture using distributed air sniffers. This approach provides convenient delay measurement, and requires no clock synchronization or instrumentation at the sensor nodes. Since using sniffers incurs additional deployment cost, we investigate two aspects to reduce deployment cost: (1) using inexpensive mote-class sniffers and (2) carefully placing the sniffers to minimize the number of sniffers that are needed. Specifically, we experimentally quantify the capability and fidelity of mote-class sniffers for delay measurement, and show that they provide satisfactory monitoring performance. We further formulate and solve a sniffer placement problem that minimizes the number of sniffers while taking account of the workload constraints of the sniffers, and show that the number of sniffers under our sniffer placement algorithms is only a small fraction of the number of sensor nodes in the network. Last, we demonstrate the effectiveness of our architecture for abnormal delay detection using experiments in a testbed.

1. Introduction

Wireless sensor networks have been used for many delay-sensitive and safety–critical applications, e.g., emergency response, plant automation and control, and health care. For such applications, measuring delays inside the wireless sensor networks is important for a number of reasons. It is important for real-time control: control strategies for the networked system need to be designed and adjusted based on communication delays [32]. It is also important for detecting abnormal delays (e.g., large delays caused by congestion or malfunctioning nodes) so that they can be corrected to maintain the normal operation of the network.

When nodes in a network have synchronized clocks, obtaining the delay from one node to another is straightforward: the sender places a timestamp when sending a packet, the receiver places a timestamp when receiving the packet, and the difference of the two timestamps is one instance of the delay. Existing clock synchronization approaches are however slow to converge and may lead to a large number of message exchanges (see Section 2.1). When the clocks of the sensor nodes are not synchronized,

* Corresponding author. Tel.: +1 860 420 9912.
E-mail addresses: Wei.Zeng@engr.uconn.edu (W. Zeng), jordocote@gmail.com (J. Cote), Xian.Chen@engr.uconn.edu (X. Chen), Yoo-ah.Kim@nih.hhs.gov (Y.-A. Kim), weiwei@engr.uconn.edu (W. Wei), kwsuh@ilstu.edu (K. Suh), bing@engr.uconn.edu (B. Wang), zshi@engr.uconn.edu (Z.J. Shi).

1570-8705/$ - see front matter
http://dx.doi.org/10.1016/j.adhoc.2013.10.008

one way for delay measurement is by instrumenting sensor nodes [30]. This approach, however, may not be feasible and consumes scarce resources of the sensor nodes (see Section 2.1).

In this paper, we propose an architecture that uses air sniffers for delay measurement in wireless sensor networks. The sniffers are placed at distributed locations, each passively listening to packet transmissions in its neighborhood and recording the time when hearing a transmission. We demonstrate that this architecture provides a convenient way to monitor delays and detect abnormal delays inside a wireless sensor network. It has the advantages that it does not require clock synchronization or instrumenting the sensor nodes to measure delays, and hence does not consume scarce resources of the sensor nodes.

Using sniffers, however, incurs additional deployment cost. This additional cost can be justified for mission-critical sensor networks (e.g., emergency response, plant automation and control). In addition, we investigate two aspects to reduce the additional deployment cost: (1) using inexpensive mote-class sniffers and (2) carefully placing the sniffers to minimize the number of sniffers that are needed. Mote-class sniffers, however, are simple embedded devices with stringent resources. Therefore, it is important to understand their capabilities and the accuracy of their monitoring results. Through a combination of experimental and analytical study, we quantify the sustainable workload and fidelity of mote-class sniffers. We find that a sniffer can monitor traffic at the rate of 60 packets per second with little buffer overflow and the per-hop delay measurements from a sniffer are accurate (the errors are up to 300 μs). Therefore, mote-class sniffers are suitable for many monitoring purposes. For sniffer deployment, we formulate and solve a sniffer placement problem that minimizes the number of sniffers while taking account of the workload constraints of the sniffers. Using extensive simulation, we show that the number of required sniffers under our sniffer placement algorithms is only a small fraction of the number of sensor nodes in the network. Last, we demonstrate the effectiveness of our architecture for abnormal delay detection through experiments in a testbed.

The rest of the paper is organized as follows. Section 2 describes the proposed delay measurement architecture using sniffers. Section 3 quantifies the capability and fidelity of mote-class sniffers. Section 4 formulates and solves the sniffer placement problem. Section 5 demonstrates the effectiveness of our architecture for abnormal delay detection. Finally, Section 6 reviews related work, and Section 7 concludes this paper.

2. Delay monitoring

Consider a static sensor network that is used to support a delay-sensitive and mission-critical application. Hence it is important to monitor delays and detect abnormal delays in real-time inside the network. We next first describe existing approaches for delay measurement, and then detail our proposed approach.

2.1. Existing approaches

One approach to monitor delays inside a wireless network is through clock synchronization. Once the clocks of the nodes are synchronized, a sender places a timestamp when transmitting a packet, and a receiver can obtain the delay from the sender as simply the difference from when the packet is received and when the packet is transmitted (which is the timestamp carried by the packet). Despite many efforts (see survey [25] and the references therein), clock synchronization, however, remains a challenging task in large-scale sensor networks. Existing clock synchronization approaches are slow to converge and may require a large number of message exchanges. For instance, for a 20-node network, the-state-of-the-art clock synchronization algorithms FTSP and GTSP [16,26] take roughly 30 min until the algorithms converge and need to re-synchronize to reach the desired precision. During the synchronization period, each node needs to send at least one message, leading to a large number of message exchanges in a large-scale network. One way to eliminate the need of clock synchronization is using half of the RTT between two nodes as the one-way delay. This approach, however, requires the sensor nodes to measure RTTs. Furthermore, it may lead to inaccurate estimates due to asymmetric communications in sensor networks [12,33,18].

When the clocks of the sensor nodes are not synchronized, one way for delay measurement is by instrumenting the nodes [30]. More specifically, consider the delay on a network hop from sensor node *A* to *B*. This delay contains two components: the delay at *A* and the radio propagation delay for sending a packet from *A* to *B*. Since the second component is negligible, the delay is approximately the delay at *A*, which can be obtained by instrumenting *A* to record two timestamps, one is when *A* starts to transmit a packet (or receives a packet when it is an intermediate node) and the other is when *A* receives a signal that this packet is actually sent out into the air. This approach, however, requires modifying the source code for each sensor node. It is infeasible when source code is not available. In addition, requiring sensor nodes to monitor delays and detect abnormal delays consumes scarce resources (including CPU, memory, network bandwidth, and energy) of the sensor nodes, which may affect the intended functionality of the sensor network. (In fact, this approach is only used in an offline manner in [30].)

2.2. Proposed approach

In our proposed architecture, a set of sniffers are deployed at distributed locations inside the sensor network (we discuss where to place sniffers in detail in Section 4). Each sniffer has two network interfaces (as in [9,19]). One interface operates on the same channel as that of the sensor nodes, and is used to listen to packet transmissions from nearby sensor nodes. The other interface operates on a non-interfering channel, and is used to communicate with other sniffers and a server (e.g., for reporting abnormal delays). The reason for using a non-interfering channel is that packet transmissions using this channel do not interfere with the traffic inside the sensor network. Fig. 1

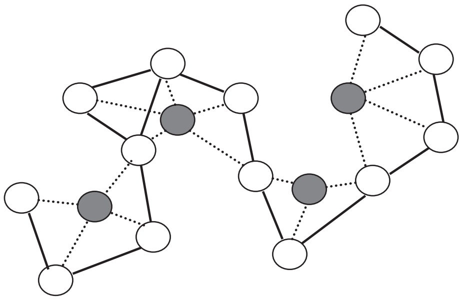


Fig. 1. Measurement architecture using sniffers for a wireless sensor network. The white and shaded nodes represent sensor nodes and sniffers respectively.

illustrates this architecture, where the white nodes represent sensor nodes, and the shaded nodes represent sniffers. In the figure, two sensor nodes are connected by an edge if they can transmit to each other; a sniffer is connected to a sensor node (using a dashed line) if the sniffer can hear the transmission from that node.

We next describe methodologies for per-hop delay monitoring and abnormal delay detection using the above architecture. Consider an arbitrary network hop from node A to B. Our description considers two cases: (1) A is an intermediate node: it receives packets from an upstream node and then forwards them to B and (2) A is a source: it does not receive any incoming packet; instead, it generates packets and forwards them to B.

2.2.1. Delay monitoring using air sniffers

When A is an intermediate node, obtaining packet transmission delay from A to B using sniffers is straightforward. Suppose an upstream node sends a packet to A, and a sniffer overhears this transmission and records the reception time as t. Once receiving the packet, A forwards it to B, and the sniffer overhears this transmission and records the reception time as t'. Then the transmission delay of the packet from A to B is $d = t' - t$. This is because when ignoring radio propagation delay (which is negligible since the transmission range in a sensor network is tens or hundreds of meters while the radio propagation speed is approximately 3×10^8 meters per second), A receives the packet at t and B receives the packet at t'. Since t is also the time point when A starts to transmit the packet to B (A starts to forward the packet immediately after receiving it), $t' - t$ represents the delay from sending the packet from A to B. Note that, in the above method, since d is determined by the relative difference of t' and t, the sniffer's clock does not need to be set to the correct wall clock time to obtain accurate measurement of d.

When A is a source and no packets are transmitted to A, using sniffers does not obtain the absolute delay from A to B. However, we can easily obtain relative delays from A to B, which can be used to obtain delay variance (which is important for realtime control [32]) and detect abnormal delays (as we shall see in Section 2.2.2). More specifically, consider a common scenario where sources transmit packets periodically and embed an application-level sequence number to each packet[1]. In such a scenario, for a packet with sequence number i, the packet sending time at A, t_i, is $i\tau + t_0$, where t_0 is a constant and τ is the period of the transmission at the source. Since a sniffer does not know t_0, it does not know t_i. However, when the sniffer overhears the packet transmitted from A to B at time t'_i, it can treat $t'_i - i\tau$ as a relative delay for this packet (we assume the sniffer knows the period, τ, and obtains the sequence number, i, from the overheard packet). The quantity, $t'_i - i\tau$, is a relative delay because (1) it ignores the constant t_0 and (2) $i\tau$ and t'_i are according to the clocks of A and B, respectively, which are not synchronized (and hence may have clock skew and offset). While obtaining relative delays from A to B, the sniffer can adjust the delays by removing clock skew and offset in an online manner (e.g., using the technique in [31]). For the ith packet, let d_i be the adjusted delay after removing clock skew and offset in $t'_i - i\tau$. Then d_i is the absolute delay of the ith packet from A to B shifted by a constant. The above method assumes periodic transmission from the source and the period τ is known by the sniffers. If this is not the case, the sniffer needs to know the interval between two packet transmissions (which can be obtained from the application, e.g., by adding transmission time of a packet in the payload). Let τ_i denote the interval between sending the ith and $(i+1)$th packet. Then the relative delay for the ith packet is $t'_i - \sum_{j=1}^{i} \tau_j$, which can be adjusted to remove clock skew and offset as described earlier.

As per-hop delays (absolute or relative delays) are being obtained, depending on the requirements of the application, a sniffer may (selectively) transmit the delays to other sniffers and/or to a server using the non-interfering interface. Or it may only obtain statistics of the delays, and transmit these statistics. Two basic statistics, mean and standard deviation, can be obtained using the following method at little computation and storage overhead [13]. Consider a sequence of delays, $\{d_i\}_{i=1}^{n}$, where d_i is the ith delay measurement. Let $\hat{\mu}$ and $\hat{\sigma}$ denote respectively the current estimates of the mean and standard deviation. They are updated when a new delay measurement is obtained. Define $S_n = \sum_{i=1}^{n} d_i$, and $W_n = \sum_{i=1}^{n} (d_i - \hat{\mu})^2$. After obtaining the latest delay observation, d_n, S_n and W_n are updated as:

$$S_n = S_{n-1} + d_n,$$
$$W_n = W_{n-1} + ((n-1)d_n - S_{n-1})^2 / (n(n-1)).$$

Then the mean and standard deviation are updated as $\hat{\mu} = S_n/n$, and $\hat{\sigma}^2 = W_n/(n-1)$.

2.2.2. Abnormal delay detection

Abnormal delay detection can be modeled as a change-point detection problem: when the distribution of the delays changes (we assume the original delays are normal), we say the delays become abnormal. When A is an intermediate node, as shown earlier, a sniffer obtains a sequence of absolute delays from A to B, and can apply an online change-point detection algorithm to these delays

[1] We claim this as a common scenario since in many monitoring applications, sources transmit packets periodically, and use sequence numbers to differentiate the packets.

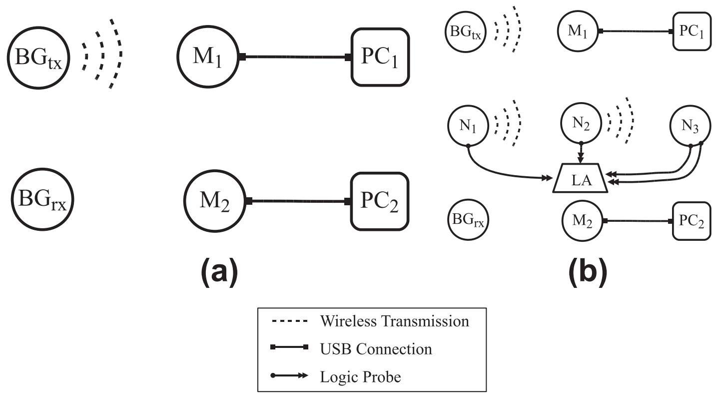


Fig. 2. Experimental settings: (a) setting to measure sustainable workload, (b) setting to evaluate the fidelity of delay measurements. In the figure, M_1 and M_2 are sniffers, BG_{TX}, BG_{RX}, N_1, N_2 and N_3 are motes, PC_1 and PC_2 are computers, and LA is a logical analyzer.

to detect a change point. When A is a source, a sniffer obtains a sequence of relative adjusted delays from A to B. Since these delays only differ from the absolute delays by a constant, the sniffer can still apply an online change-point detection algorithm to these delays to detect a change point. Many techniques have been developed for online change-point detection [3,4]. Different online detection techniques may prove effective for different abnormal scenarios. We illustrate how we detect abnormal delays that are caused by congestion in Section 5.

2.3. Summary

In summary, our proposed architecture uses existing traffic inside the sensor network for delay measurement. It is simple, requiring no clock synchronization or instrumentation at the sensor nodes. The sniffers placed for delay measurement can also be used for other purposes. For instance, they can monitor sensor nodes and discover abnormal nodal behaviors [19,6]. They can also be used for intrusion detection [27]. We only focus on monitoring delays in this paper.

Deploying the proposed monitoring architecture incurs additional deployment cost. This cost can be justified in mission-critical networks (e.g., sensor networks used for emergency response, plant automation and control). Furthermore, the cost can be reduced by using inexpensive mote-class sniffers and carefully placing the sniffers to minimize the number of sniffers that are needed, two aspects that we will investigate in Sections 3 and 4, respectively.

3. Capability and fidelity of mote-class sniffers

We experimentally evaluate the capability and fidelity of mote-class sniffers for delay monitoring. In particular, the sniffers we use are TelosB motes that use CC2420 wireless transceivers (IEEE 802.15.4), and run on TinyOS 2.1.0. We first evaluate their sustainable workload, and then evaluate the accuracy of their delay measurements.

3.1. Sustainable workload measurements

Fig. 2(a) shows the experimental setting for measuring sustainable workload at the sniffers. It consists of a transmitter, BG_{TX}, a receiver, BG_{RX}, two sniffers, M_1 and M_2, and two PCs. Each sniffer passively listens to packet transmissions in its neighborhood, and once overhearing a packet, records the current time as a payload in the packet and passes the packet over USB into a data log stored at the connected PC.[2] Using two sniffers allows us to validate whether the measurements by the sniffers are consistent (to avoid measurement errors caused by hardware or software inconsistencies of the sniffers).

Our goal is to measure the workload that can be sustained by the sniffers. For this purpose, we let BG_{TX} transmit packets to BG_{RX} following a Poisson process with a rate increasing from 5 packets to 60 packets per second, and measure the corresponding loss rate at the sniffers. The losses at the sniffer are mainly due to receiver buffer overflow since there is a single traffic source in our testbed and the testbed is in an isolated lab with little other sources of interference. The receiver buffer uses FIFO (first-in-first-out) scheduling. Table 1 records the number of lost packets in each experiment (the loss count for the two sniffers is the same.). We observe very few losses even when the average sending rate is 60 packets per second, indicating that the sniffer is reliable for capturing the traffic received at this rate.

To gain additional insights, we use a queuing model to approximate the number of losses at the sniffer. Our measurements show that the processing time of a packet at the sniffer is close to a constant of 4.4 ms.[3] Because the arrival process follows a Poisson distribution, the processing time at the sniffer is constant, and the buffer at the sniffer can hold up to three packets (the buffer size is 128 bytes and each packet is 38 bytes), we model the sniffer as an $M/D/1/3$ queue. We then obtain the probability of buffer overflow from the queuing model [5]. The analytical results from

[2] The sniffer cannot use internal flash memory to store the data log because it takes roughly 20 ms to commit a packet to flash, while the inter-arrival time of packets at the sniffer can be smaller than 20 ms.

[3] The processing time of a packet at the sniffer is the duration from when a packet arrives at the buffer to when the packet is removed from the buffer. It is the difference of two timestamps: the first is taken when the CC2420 wireless chip sets the SFD (start of frame delimiter) pin high, and the second timestamp is taken when the receive event is triggered by the CC2420 driver code.

Table 1
Sustainable workload measurement results. The results at M_1 and M_2 are the same. The measured inter-sending time does not coincide exactly with the value we set due to random delays at the sender.

Inter-sending time (ms)	Number of packets	Lost packets (measurement)	Lost packets (analysis)
16.2	23,278	11	10.84
16.2	27,303	9	12.72
25.9	12,289	2	0.7
30.8	41,498	4	1.1
30.9	10,402	0	0.27
40.7	30,820	2	0.24
51.3	26,053	3	0.08
99.25	12,515	0	0
197.9	6280	0	0

the model match well with the experimental results for various sending rates as shown in Table 1.

3.2. Fidelity of delay measurement

We evaluate the accuracy of delay measurements from the sniffers by comparing them with high-fidelity measurements from a logic analyzer. Fig. 2(b) shows the experimental setting. Node N_1 sends a packet to N_3 through N_2 every second using Collection Tree Protocol (CTP) [11]. Using the methodology in Section 2.2.1, sniffers M_1 and M_2 listen to packet transmissions in their neighborhood, and obtain the delays on the two hops, (N_1,N_2) and (N_2,N_3), from the overheard traffic, and transmit this information via USB to PC_1 and PC_2, respectively, in an online fashion. The logic analyzer, a 34-channel Intronix LA1034 device, is connected to nodes N_1, N_2, and N_3 via probes (specifically, it is connected to the MCU pin 2.6 of the MSP430 microprocessor of each node as in [2]). It records timing information to obtain accurate per-hop delays as follows. Consider a packet sent from N_1 to N_3. N_1 raises the pin to a logical high when it begins to transmit, and lowers it to a logical low when it finishes transmitting; N_2 raises the pin when the application layer has finished receiving the packet, and lowers it when completing forwarding; N_3 also raises the pin when the application layer finishes receiving the packet, and lowers it when the packet information is committed to flash memory. Fig. 3 shows the logical pattern for these pins, where t_1 and t_1' represent respectively the time when a packet is being sent from N_1 at the application level and when it is done transmitting; t_2 and t_2' represent respectively the time when N_2 finishes receiving the packet at the application level and finishes forwarding; t_3 and t_3' represent respectively the time when N_3 finishes receiving the packet at the application level and finishes committing to the flash memory. All the timing events are transmitted to a PC that is connected to the logical analyzer via USB. Using the recorded timestamps, we can easily obtain per-hop delays. They are the amount of time for the sender to transmit a packet and the receiver to receive it. More specifically, we use $(t_2 - t_1)$ as the delay on the first hop, and use $(t_3 - t_2)$ as the delay on the second hop.

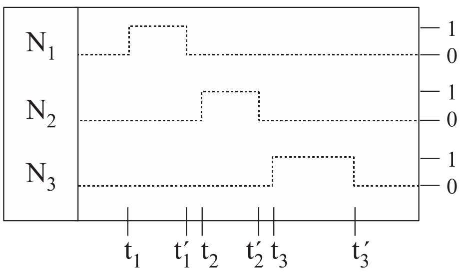


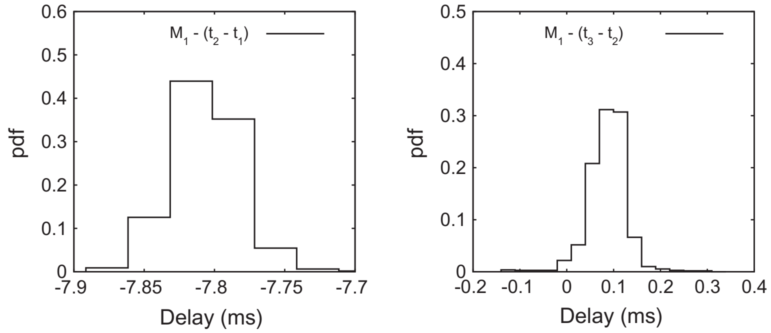
Fig. 3. MCU pin logical timing pattern for the two-hop network as shown in Fig. 2(b).

The logic analyzer has sampling rate of 10 MHz, providing 100 ns accuracy, much finer than the granularity of 30.5 μs that are obtained using 32 kHz clocks at the sniffers. We therefore use the delay measurements from the logical analyzer as the ground truth to evaluate the delay measurements from the sniffers. In addition to the traffic on network hops (N_1,N_2) and (N_2,N_3), node BG_{TX} sends packets to BG_{RX} following a Poisson process, referred to as background traffic (used to simulate traffic from a set of sensor nodes as in [2]), which is captured by the sniffers as well. By varying the rate of background traffic from 5 to 60 packets per second, we evaluate the accuracy of delay measurements by the sniffers under different workloads.

Let measurement error be the difference of delay measurement from a sniffer and the logical analyzer. We next present measurement results from M_1 when there is no background traffic (the results when there is background traffic and the results from M_2 are similar). Fig. 4 plots the distribution of the measurement errors on the first hop. From the figure, we see that the difference is indeed close to a constant (the distribution is concentrated in a narrow range of around 190 μs, from −7.89 ms to −7.7 ms). Fig. 4(b) plots the distribution of the measurement error on the second hop. We observe that the errors are up to 300 μs, indicating that the delay measurements from the sniffer are very accurate. We also observe that the errors are biased towards being positive (i.e., the delays measured by the sniffer are typically larger than the corresponding delays from the logical analyzer). This is because the sniffer needs to process each captured packet (e.g., adding timestamp, placing it into a USB packet, and transmitting it to the PC), which incurs additional delay. When this delay occurs after receiving the first hop transmission, the mote may not be able to finish before the second hop transmission arrives. In this case, the delay is artificially increased because the microprocessor is busy.

4. Sniffer placement

In the previous section, we have shown that simple inexpensive mote-class sniffers can provide satisfactory delay measurement. We can further reduce the deployment cost of our proposed monitoring architecture by minimizing the number of sniffers that are needed. In the following, we first formulate and solve a sniffer placement problem, and then explore the number of needed sniffers using extensive simulation.



(a) First hop. (b) Second hop.

Fig. 4. Distribution of the delay measurement errors from sniffer M_1.

4.1. Problem formulation

The sniffer placement problem determines the locations of the sniffers that (i) the pair of sensor nodes of each network hop is monitored by at least one sniffer, (ii) each sniffer monitors at most w pairs of nodes, and (iii) the total number of sniffers is minimized. The first constraint ensures that the delays of all network hops are monitored[4]. A sniffer needs to monitor the two nodes on a network hop simultaneously to obtain delay measurements. The second constraint takes account of the workload of the sniffers, referred to as *sniffer workload constraint*. If a sniffer overhears the transmission from more than w pairs of nodes, it only processes the packets from w pairs. The last constraint minimizes the number of sniffers that are needed to minimize deployment cost.

4.2. Sniffer placement algorithms

We solve the sniffer placement problem in two steps. First, we propose a pre-processing algorithm that determines candidate sniffer locations. Second, for the given set of candidate sniffer locations, we select a subset of locations and place a sniffer at each of these locations.

4.2.1. Determining candidate locations

We consider both regular and irregular radio ranges [34]. In both cases, let l_i denote the location (i.e., the coordinate) of sensor node n_i. Let R_i and r_i denote the *coverage region* and *transmission range* of v_i respectively. We assume any node in the coverage region, R_i, can hear the transmission of n_i. When the radio range is regular, R_i is a circular area centered at l_i with the radius of r_i. Otherwise, we assume that R_i is a polygon [34] and the average distance from l_i to the vertices of the polygon is r_i.

Regular Radio Range. Let L denote the set of candidate sniffer locations. Initially, L is empty. We then consider each network hop. Suppose the two nodes of a network hop are n_i and n_j, that is, n_i transmits to n_j and/or n_j transmits to n_i. We then add candidate sniffer locations to L depending on the relationship of R_i and R_j. If $R_i \subset R_j$, we add the location of n_i, l_i, as a candidate location (we may use any location in R_i as a candidate location; for simplicity, we use l_i). Similarly, if $R_j \subset R_i$, we add l_j as a candidate location. If neither of the above holds, then the boundaries of their coverage regions, R_i and R_j, must intersect at two points, and we add these two intersection points to L. Algorithm 1 summarizes this algorithm. The following theorem shows that the above algorithm for determining candidate locations is sufficient.

Algorithm 1. Determine Candidate Sniffer Locations

$L = \emptyset$
for $\forall n_i, n_j, i \neq j$ that are on a network hop **do**
  if $R_i \subset R_j$ **then**
    $L = L \cup \{l_i\}$
  else if $R_j \subset R_i$ **then**
    $L = L \cup \{l_j\}$
  else
    The boundaries of R_i and R_j intersect at two points, denoted as p_1 and p_2
    $L = L \cup \{p_1, p_2\}$
  end if
end if

Theorem 1. *For any optimal solution S^*, there exists a corresponding subset $S \subseteq L$ so that $|S^*| = |S|$ and S covers all node pairs on the network hops.*

Proof. We prove the above theorem by showing that $\forall s \in S^*$, there exists a location $l \in L$ so that the set of sensor node pairs monitored by s can be monitored by a sniffer located at l. Without loss of generality, suppose the set of sensor node pairs that are monitored by s is $X = \{(n_i, n_j)\}$. For ease of exposition, let Y denote the set of sensor nodes that are in X. That is, $Y = \{n_i | \exists n_j, (n_i, n_j) \in X \text{ or } (n_j, n_i) \in X\}$. Since s monitors all the node pairs in X, s must be in the intersection region of R_i and R_j, $\forall (n_i, n_j) \in X$. Let B denote the boundary of this intersection region. We next consider two cases. In the first case, there exist i, j such that $(n_i, n_j) \in X$ and one intersection point of the boundaries of

[4] When the route in the sensor network changes dynamically [28], the union of network hops in all the routes are monitored.

R_i and R_j, denoted as l, is on B. Then l can monitor all the pairs in X, and $l \in L$ by Algorithm 1. In the second case, we cannot find i, j such that $(n_i, n_j) \in X$ and one intersection point of the boundaries of R_i and R_j is on B. Then there must exist one sensor node n_i so that $n_i \in Y$ and $R_i \subset R_j, \forall n_j \in Y$, $j \neq i$. In this case, by Algorithm 1, a sniffer located at the location of n_i, $l_i \in L$, can monitor all the pairs in X. Summarizing the above two cases, we have proved our claim. □

Irregular Radio Range. When the radio range is irregular, we assume the coverage region of a node is a polygon, which can be obtained based on Received Signal Strength (RSS) measurements in different directions of the node [34]. Our algorithm for determining candidate monitor locations is similar to Algorithm 1. The only difference is that since R_i and R_j are polygons, when they intersect, they may intersect at multiple points (more than two) or an infinite number of points (i.e., their intersection forms an edge). For the former case, we include the multiple points into L; for the latter case, we include the two end points of the edge into L. Therefore, the total number of candidate monitor positions is finite. We can again show that the above algorithm is sufficient; the proof is similar to that for Theorem 1 and is omitted in the interest of space.

4.2.2. Placing sniffers

For a given a set of candidate sniffer locations, we place a candidate sniffer at each candidate location to construct a candidate sniffer set, S_c. Consider all the network hops. The pair of nodes on each network hop needs to be monitored. We transform the node pair monitoring problem to a node monitoring problem by constructing a virtual graph. The vertices of the virtual graph are $V \cup S_c$, where V is the set of *virtual nodes*, each corresponding to a node pair that needs to be monitored. A virtual node is connected to a candidate sniffer using a *virtual edge* if the candidate sniffer can monitor the pair of sensor nodes that corresponds to the virtual node. In this way, monitoring the set of node pairs is equivalent to monitoring the set of virtual nodes in the virtual graph. Fig. 5 shows an example of the virtual graph, where the white dashed nodes and shaded nodes represent respectively the virtual nodes and candidate sniffer locations, and the dashed lines represent the virtual edges. It is the virtual graph for the example in Fig. 1.

Choosing sniffers from the set of candidate sniffers and determining the assignment function for each sniffer (i.e., determining the set of virtual nodes to be monitored by a sniffer) can be solved using the two algorithms that are developed for node monitoring in [6]. Both algorithms run in iterations. Initially, the set of sniffers, S, is empty. In each iteration, the algorithms add a sniffer from the candidate sniffer set, S_c, into S. The iteration continues until all virtual nodes are monitored. These two algorithms differ in that one is based on a max-flow formulation, and the other uses a simple heuristic, referred to as *Max-flow* and *Max-degree* sniffer placement algorithms, respectively. For completeness, the two algorithms are described briefly in the Appendix.

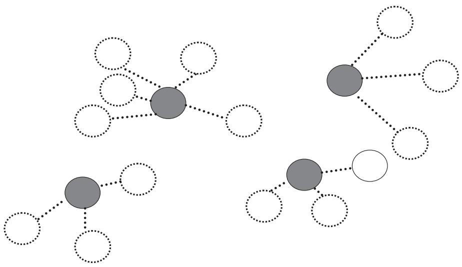


Fig. 5. Illustration of virtual graph.

4.3. Performance evaluation

We consider 100 sensor nodes deployed in a 500 m × 500 m area using uniform random, grid uniform or non-uniform deployment. In uniform random deployment, the sensor nodes are deployed uniformly at random in the area. In grid uniform deployment, one sensor node is uniformly randomly placed in each grid (of 50 m × 50 m), and hence the node distribution is more even than that in uniform random deployment. In non-uniform deployment, the entire region is divided into four sub-regions, the top left and bottom right regions have much higher node density than the other two regions (the two denser regions have 35 sensor nodes while the other two regions have 15 sensor nodes). Furthermore, we also place a region head in the center of each region. The region heads are connected to each other; the nodes in a region are uniformly deployed, and connected to their region head. We assume all nodes transmit sensed data to a sink in the center of the area. The routing is either static or dynamic. Under static routing, the routes from the sensors to the sink form a routing tree. The number of branches in the tree is uniformly distributed in $[1, B]$, where $B = 10$ or 5. Under dynamic routing, the routes are chosen dynamically from two routing trees. Therefore, all the routes in the two routing trees need to be monitored.

The radio range of a sensor node is regular or irregular. Under regular radio range, the coverage region of a sensor node is circular, and all the sensor nodes have the same transmission range, which is varied from 100 to 200 m (corresponding to the transmission range of mote-class sensor nodes; we choose the minimum transmission range of 100 m because the network is disconnected when using a lower value). Under irregular radio range, the coverage region is a polygon with 7 to 16 vertices, and all the sensor nodes have the same average transmission range, which is varied from 100 to 200 m. A sniffer is allowed to monitor at most w pairs of sensor nodes. Assuming that each sensor node needs to transmit sensed data every one second or two seconds, we set $w = 30$ or 60 correspondingly based on the measurement results in Section 3. The performance metric we use is the *number of sniffers needed*. For each setting, we make 10 independent runs using randomly generated seeds. The results below are averaged over 10 runs; the 95% confidence intervals are tight and hence omitted.

We find that the performances of the Max-flow and Max-degree based algorithms are similar. Furthermore, the results under different deployments, regular or irregular radio range are similar. We next only present the results of the Max-flow based algorithm with irregular radio range under uniform random deployment. Figs. 6(a and b) plot the number of needed sniffers under static and dynamic routings, respectively when $B = 5$ (the number of needed sniffers under $B = 10$ is slightly less). In both figures, we observe that the number of needed sniffers decreases as the transmission range increases. This is because as the transmission range increases, more virtual nodes can be monitored by a sniffer, confirmed by higher workloads at the sniffers from our results. The number of needed sniffers is slightly less when $w = 60$ than that when $w = 30$. In both cases, the number of needed sniffers is small (varying from 5 to 14 for the various transmission ranges), indicating a small overhead in deploying the sniffers.

5. Abnormal delay detection

In this section, we put everything together and demonstrate experimentally how to detect abnormal delays using our proposed monitoring architecture. Our testbed consists of eight TelosB motes, as illustrated in Fig. 7. All the motes use B-MAC [17], the default MAC protocol in TinyOS. Due to limited space (the testbed is deployed in an office), we separate the sensor nodes in a few meters, as marked in Fig. 7. Correspondingly, the power level at each mote is set to a low level (it is set to 3, i.e., −25 dBm). Node n_0 is the sink. The transmission range of each mote is in tens of meters. Using the sniffer placement algorithm in Section 4, we only need a single sniffer to overhear packet transmissions from all the nodes in the testbed. For convenience, we place the single sniffer, s, in the middle of the testbed.

Abnormal delays in a sensor network can be due to many reasons. We focus on abnormal delays caused by congestion in the network. In particular, we consider two scenarios: (1) *parallel sources*, where nodes n_1 and n_5 are sources, both sending packets via nodes n_2, n_3, and n_4 to the sink and (2) *tandem sources*, where n_1 and n_2 are sources, n_1 sends its packets via nodes n_2, n_3, and n_4 to the sink, and n_2 sends its packets via nodes n_3 and n_4 to the sink. In both scenarios, we emulate the occurrence of abnormal delays as follows. At the beginning, the transmissions of the two sources are not synchronized. Then after a certain time point, they are synchronized by sending a synchronization signal from node n_6 to the two sources, which

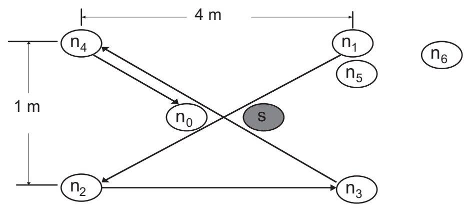


Fig. 7. Testbed setting: node n_0 is the sink, node s is a sniffer

leads to congestion and hence abnormal delays. In both scenarios, a source sends a packet every two seconds; each packet carries an application-level sequence number. For ease of experiments, we fix the route from a source to the sink.

For each source, the sniffer obtains per-hop delays (the first-hop delays are relative delays), and maintains the current estimates of the mean and standard deviation of the delays. Let $\hat{\mu}$ and $\hat{\sigma}$ denote respectively the current estimates of the mean and standard deviation of the delays on a hop. They are updated using the method in Section 2.2.1 that incurs little storage and computation overhead. We explore two change-point detection methods. The first method raises an alarm after observing two consecutive delays that are larger than $\hat{\mu} + 3\hat{\sigma}$ (we use two consecutive large delays instead of a single one to reduce false alarms). The second method is a non-parametric CUSUM method [4]. In particular, we define $\tilde{d}_i = d_i - a$, where d_i denotes the ith delay observation, and a is chosen so that $\tilde{d}_i$ is negative (with high probability) before a change point (we use $a = \hat{\mu} + 3\hat{\sigma}$). Let

$$y_i = (y_{i-1} + \tilde{d}_i)^+, \quad y_0 = 0,$$

where $(x)^+ = \max(x, 0)$. This method updates y_i after each delay observation and raises an alarm when $y_i \geqslant h$, where $h > 0$ is a threshold, and we use $h = 1.25\hat{\sigma}$.

To systematically evaluate the performance of our abnormal-delay detection methods, in both scenarios (i.e., parallel and tandem sources), for each source, we construct multiple sequences of delay observations on each hop as follows. We first run experiments when the two sources are not synchronized, and obtain a sequence of 10,000 delays on each hop, which represents normal de-

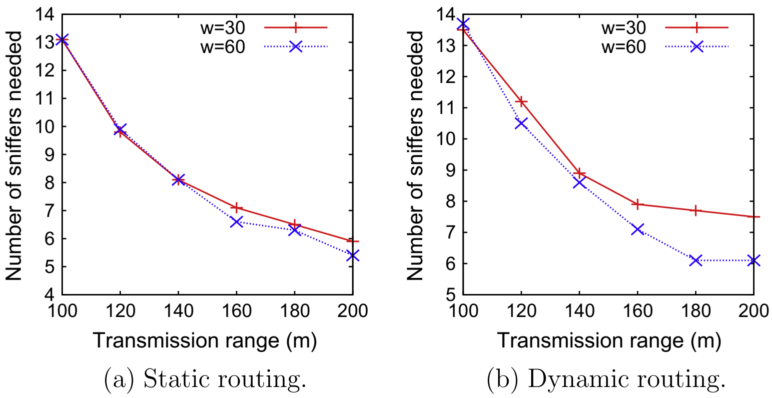


Fig. 6. Number of needed sniffers versus transmission range under Maxflow based algorithm for irregular radio range and uniform random deployment, $B = 5$.

lays. We then run experiments when the two sources are synchronized to obtain a sequence of 10,000 delays on each hop, which represents abnormal delays. Afterwards, we construct delay sequences using samples from the normal and abnormal delay observations. In particular, each sequence contains 250 normal delay observations (chosen from the normal delay observation sequence, starting from a random position) followed by 500 abnormal delay observations (chosen similarly from the abnormal delay observation sequence).

For each hop, we construct 1000 delay sequences as above. For a delay sequence, our change-point detection methods stop and raise an alarm after detecting that the delay has become abnormal. For each delay sequence, the result of a change-point detection method falls into one of the following three categories: it is *successful* if the detection is within the range of abnormal delays; it is a *false alarm* if the detection is within the range of normal delays; and it is a *false negative* if no alarm is raised at the end of the delay sequence. We define detection ratio (DR) of a change-point detection method as the number of delay sequences with successful detection over the total number of delay sequences (i.e., 1000 in our setting). Similarly, we define false positive ratio (FPR) and false negative ratio (FNR). Our performance metrics are DR, FPR, FNR and detection delay (DD), i.e., the delay (in terms of the number of delay observations) from the change point to when an alarm is raised.

Table 2 shows the evaluation results for source n_1 in the two scenarios (the results for another source have similar trend). We observe the two change-point detection methods are both effective. For both methods, the sniffer successfully detects that the hop delays become abnormal: for all the hops, the detection ratios are close to 1 (above 98.3%), the false positive ratio is close to 0 (less than 0.1%), and the false negative ratio is close to 0 (less than 1.7%). Furthermore, the detection delay is short: it ranges from 7 to 38 delay observations.

6. Related work

Existing studies propose placing dedicated sniffers in sensor networks for code debugging [9], performance monitoring [19], development support [10], network management [22], and sensor node health monitoring [6]. Our study differs from them in that we use monitors for delay monitoring and abnormal delay detection. The sniffer placement problem in our study is related to [6]. Specifically, the Max-flow and Max-degree based algorithms for placing sniffers in the virtual graph are from [6].

Passive monitoring through dedicated sniffers has been used in other types of wireless networks. For instance, it has been successfully used in single-hop infrastructure-based wireless LANs (e.g. [1,29,14,15,7,23]), wireless mesh networks [24] and wireless ad hoc networks [27]. Their focuses are on network management, monitoring, characterization and intrusion detection. None of them is on per-hop delay monitoring or abnormal delay detection as in our study. In addition, they do not consider how to place sniffers.

We quantify the capability and fidelity of mote-class sniffers for delay monitoring. This differs from existing studies that investigate the accuracy and fidelity of IEEE 802.11 sniffers [21,20]. Our study also covers a broader scope than the study in [2] that focuses on characterizing per-hop and end-to-end delays in a sensor network.

7. Conclusion

In this paper, we proposed an architecture that uses distributed sniffers for delay monitoring and abnormal delay detection in wireless sensor networks. To reduce deployment cost, we suggested using inexpensive mote-class sniffers and minimizing the number of sniffers that are needed. Specifically, we experimentally demonstrated that mote-class sniffers can provide satisfactory delay monitoring performance. Furthermore, we formulated and solved a sniffer placement problem to minimize the number of sniffers while taking account of the workload constraints of the sniffers. Extensive simulation results showed that the number of required sniffers under our sniffer placement algorithms is only a small fraction of the number of sensor nodes in the network. Last, we demonstrated the effectiveness of our architecture for abnormal delay detection through experiments in a testbed.

Acknowledgments

Preliminary results of this paper are presented in [30,8]. This work was supported in part by NSF CAREER Awards 0746841 and 0644188. It was additionally supported by

Table 2
Performance evaluation results of two abnormal-delay detection methods: outlier-based and CUSUM-based methods

	Tandem sources				Parallel sources			
	DR	FPR	FNR	DD	DR	FPR	FNR	DD
Outlier-based								
Hop1	0.999	0.001	0	7.413	1	0	0	13.272
Hop2	1	0	0	23.884	1	0	0	16.632
Hop3	0.987	0	0.013	36.263	1	0	0	22.416
Hop4	0.999	0	0.001	30.519	0.999	0.001	0	25.067
CUSUM-based								
Hop1	1	0	0	7.524	1	0	0	14.314
Hop2	1	0	0	25.504	1	0	0	16.582
Hop3	0.983	0	0.017	38.574	0.998	0	0.002	20.772
Hop4	0.997	0	0.003	32.024	0.996	0.001	0.003	26.835

DOE GAANN Program Award P200A090340. We would like to thank M. Zink for helpful discussions.

Appendix A. Algorithms to place sniffers

The Max-flow based sniffer placement algorithm is as follows. First, we construct a bipartite graph, where one set in the graph is the candidate sniffer set, S_c, and the other set is the virtual node set, V. A node $s \in S_c$ is connected to a node $v \in V$ if s can monitor v (i.e., s can overhear the transmission of the pair of sensor nodes corresponding to v). The capacity of edge (s, v) is 1. We further add a super source and a super sink. The super source is connected to each candidate sniffer with the capacity of w. This limits that a sniffer monitors at most w virtual nodes. Each sensor node is connected to the super sink with the capacity of 1. Let f denote the maximum integral flow of this graph. Then it is easy to see that all the virtual nodes are monitored if and only if $f = |V|$. Furthermore, the assignment function can be easily obtained from the max-flow solution: if the amount of flow from sniffer s to virtual node v is positive, i.e., $f(s, v) > 0$, we assign s to monitor v. The Max-flow based sniffer placement algorithm has approximation ratio of $\ln|V|$, where $|V|$ is the number of virtual nodes [6].

The main idea of the Max-degree based sniffer placement algorithm is as follows. In each iteration, it adds the sniffer that has the maximum degree in the virtual graph into the sniffer set. The intuition is that a candidate sniffer with a larger degree can monitor more virtual nodes, and hence may reduce the number of sniffers needed. More specifically, suppose s has the maximum degree. The algorithm adds s to the sniffer set, and assign s to monitor a set of virtual nodes that s can monitor, denoted as $N(s)$. If more than w virtual nodes are in $N(s)$, it assigns the w virtual nodes with the lowest degrees to s (the intuition is that virtual nodes with larger degrees may be able to be monitored by other candidate sniffers). The iteration continues until all virtual nodes are monitored.

References

[1] A. Adya, V. Bahl, R. Chandra, L. Qiu, Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks, in: Proc. of ACM MobiCom, September 2004.

[2] A. Ageev, D. Macii, D. Petri, Experimental characterization of communication latencies in wireless sensor networks, in: Symposium on Electrical Measurements and Instrumentation, Florence, Italy, April 2008, pp. 258–263.

[3] M. Basseville, I. Nikiforov, Detection of Abrupt Changes: Theory and Application, Prentice Hall, 1993.

[4] B.E. Brodsky, B.S. Darkhovsky, Nonparametric Methods in Change-Point Problems, Springer-Verlag, New York, 1993.

[5] O. Brun, J.-M. Garcia, Analytical solution of finite capacity M/D/1 queues, Journal of Applied Probability 37 (4) (2000) 1092–1098.

[6] X. Chen, Y.-A. Kim, B. Wang, W. Wei, Z.J. Shi, Y. Song, Fault-tolerant monitor placement for out-of-band wireless sensor network monitoring, Ad Hoc Networks 10 (1) (2012).

[7] Y.-C. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A.C. Snoeren, S. Savage, G.M. Voelker, Automating cross-layer diagnosis of enterprise wireless networks, in: Proc. of ACM SIGCOMM, Kyoto, Japan, August 2007.

[8] J. Cote, B. Wang, W. Zeng, Z. Shi, Capability and fidelity of moteclass wireless sniffers, in: IEEE GLOBECOM, December 2010.

[9] F. Dressler, R. Nebel, A. Awad, Distributed passive monitoring in sensor networks, in: Proc. of IEEE INFOCOM, May 2007. Poster.

[10] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, P. Blum, Deployment support network – a toolkit for the development of WSNs, in: European Conference on Wireless Sensor Networks, January 2007.

[11] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, A. Woo, TEP 123: The Collection Tree Protocol, 2006. <http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html>.

[12] 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. Technical Report UCLA/CSD-TR 02-0013, February 2002.

[13] D.M. Hawkins, P. Qiu, C.-W. Kang, The change point model for statistical process control, Journal of Quality Technology 35 (4) (2003).

[14] A.P. Jardosh, K.N. Ramachandran, K.C. Almeroth, E.M. Belding-Royer, Understanding congestion in IEEE 802.11b wireless networks, in: Proc. of ACM SIGCOMM Internet Measurement Conference (IMC), 2005.

[15] R. Mahajan, M. Rodrig, D. Wetherall, J. Zahorjan, Analyzing the MAC-level behavior of wireless networks in the wild, in: Proc. of ACM SIGCOMM, Pisa, Italy, September 2006.

[16] M. Maróti, B. Kusy, G. Simon, A. Lédeczi, The flooding time synchronization protocol, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys '04, ACM, New York, NY, USA, 2004, pp. 39–49.

[17] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: SenSys, 2004.

[18] N. Reijers, G. Halkes, K. Langendoen, Link layer measurements in sensor networks, in: MASS, Fort Lauderdale, FL, October 2004.

[19] M. Ringwald, K. Romer, A. Vitaletti, Passive inspection of sensor networks, in: Conference on Distributed Computing in Sensor Systems, June 2007.

[20] D.C. Salyers, A.D. Striegel, C. Poellabauer, Wireless reliability: Rethinking 802.11 packet loss. In International Symposium on a World of Wireless, Mobile and Multimedia Networks, IEEE Computer Society, Washington, DC, USA, June 2008, pp. 1–4.

[21z] P. Serrano, M. Zink, J. Kurose, Assessing the fidelity of COTS 802.11 sniffers, in: Proc. IEEE INFOCOM, Rio de Janeiro, Brazil, April 2009.

[22] N. Sharma, J. Gummeson, D. Irwin, P. Shenoy, SRCP: Simple remote control for perpetual high-power sensor networks, in: European Conference on Wireless Sensor Networks, February 2009.

[23] A. Sheth, C. Doerr, D. Grunwald, R. Han, D.C. Sicker, MOJO: a distributed physical layer anomaly detection system for 802.11 WLANs, in: Proc. of ACM MobiSys, 2006, pp. 191–204.

[24] D.-H. Shin, S. Bagchi, Optimal monitoring in multi-channel multi-radio wireless mesh networks, in: Proc. of ACM Mobihoc, 2011.

[25] F. Sivrikaya, B. Yener, Time synchronization in sensor networks: a survey, IEEE Network 18 (4) (2004).

[26] P. Sommer, R. Wattenhofer, Gradient clock synchronization in wireless sensor networks, in: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, IPSN '09, IEEE Computer Society, Washington, DC, USA, pp. 37–48, 2009.

[27] D. Subhadrabandhu, S. Sarkar, F. Anjum, A framework for misuse detection in ad hoc networks – Part I, IEEE JSAC 24 (2) (2006) 274–289.

[28] A. Woo, T. Tong, D. Culler, Taming the underlying challenges of reliable multihop routing in sensor networks, in: SenSys, November 2003.

[29] J. Yeo, M. Youssef, A. Agrawala, A framework for wireless LAN monitoring and its applications, in: Proc. of ACM Workshop on Wireless Security (WiSe), 2004.

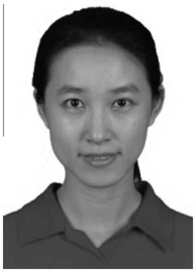
[30] W. Zeng, X. Chen, Y.-A. Kim, Z. Bu, W. Wei, B. Wang, Z.J. Shi, Delay monitoring for wireless sensor networks: an architecture using air sniffers, in: IEEE Military Communications Conference (Milcom), Boston, MA, October 2009.

[31] L. Zhang, Z. Liu, C. Xia, Clock synchronization algorithms for network measurements, in: Proc. of IEEE INFOCOM, 2002.

[32] W. Zhang, M.S. Branicky, S.M. Phillips, Stability of networked control systems, IEEE Control Systems Magazine 21 (2001) 84–99.

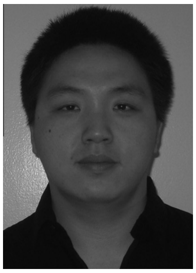
[33] J. Zhao, R. Govindan, Understanding packet delivery performance in dense wireless sensor networks, in: SenSys, 2003.

[34] G. Zhou, T. He, S. Krishnamurthy, J.A. Stankovic, Models and solutions for radio irregularity in wireless sensor networks, ACM Transactions on Sensor Networks 2 (2) (2006) 221–262.



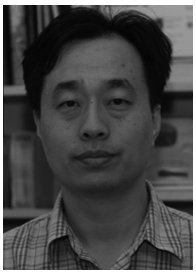
Wei Zeng received the bachelor and master degrees in Computer Science and Engineering from South China University of Technology, Guangzhou, China, in 2000 and 2003. Currently she is a Ph.D. student in the Computer Science and Engineering Department at the University of Connecticut, working with Professor Bing Wang. She is doing researches about network diagnosis, network measurement and network management for the wireless sensor networks.

Jordan Cote received his B.S. in Computer Science from the University of Connecticut in 2009. He began his Ph.D. studies at the same univiersity, researching wireless sensor network measurement with Professors Bing Wang and Jerry Shi. His current research interests include vehicular ad-hoc network security, cryptographic voting schemes, and group signatures.



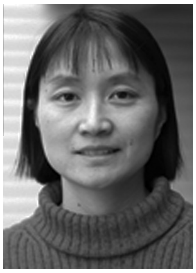
Xian Chen received his B.S. degree in Applied Mathematics from Beijing University of Aeronautics and Astronautics, China in 2003, and M.S. degree in Computer Science from Beijing JiaoTong University, China in 2006. He is currently a Ph.D. candidate in Computer Science Department at the University of Connecticut. His research interests are in the areas of computer networks, fault diagnosis and performance modeling.

Yoo-Ah Kim received her Ph.D. degree in Computer Science from the University of Maryland in 2005. She is currently a research fellow in National Center for Biotechnology Information (NCBI) at National Institute of Health. Before joining to NCBI, she was an assistant professor in the Department of Computer Science and Engineering at the University of Connecticut, Storrs, Connecticut. Her research interests include design and analysis of algorithms, bioinformatics, and parallel and networked systems.



Wei Wei received his B.S. degree in Applied Mathematics from Beijing University, China in 1992, and M.S. degree in Statistics from Texas A & M University in 2000. He then received M.S. degrees in Computer Science and Applied Mathematics, and a Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2004, 2004, and 2006 respectively. He is currently a Research Assistant Professor in the Computer Science & Engineering Department at the University of Connecticut. His research interests are in the areas of computer networks, distributed embedded systems and performance modeling.

Kyoungwon Suh received B.S. and M.S. degrees in Computer Engineering from Seoul National University in Korea in 1991 and 1993, respectively. He continued his studies in the Department of Computer Science at Rutgers University in NJ, where he earned a M.S. degree in 2000. In 2007, he received his Ph.D. degree in Computer Science from University of Massachusetts at Amherst. He is currently an associate professor in Illinois State University, Normal, IL. His research interests include mobile hand-held devices, wireless networks, network measurement and inference, network security, and multimedia content distribution. He is a member of ACM and IEEE.



Bing Wang received her B.S. degree in Computer Science from Nanjing University of Science & Technology, China in 1994, and M.S. degree in Computer Engineering from Institute of Computing Technology, Chinese Academy of Sciences in 1997. She then received M.S. degrees in Computer Science and Applied Mathematics, and a Ph.D. in Computer Science from the University of Massachusetts, Amherst in 2000, 2004, and 2005 respectively. Afterwards, she joined the Computer Science & Engineering Department at the University of Connecticut as an assistant professor. Her research interests are in Computer Networks, Multimedia, and Distributed Systems. She received NSF CAREER award in 2008.

Z. Jerry Shi is currently an Assistant Professor of Computer Science and Engineering at the University of Connecticut. He received his Ph.D. degree from Princeton University in 2004 and his M.S. and B.S. degrees from Tsinghua University, China, in 1996 and 1992, respectively. He is a member of IEEE and ACM. Dr. Shi received US National Science Foundation CAREER award in 2006. His current research interests include underwater sensor networks, sensor network security, hardware mechanisms for secure and reliable computing, side channel attacks and countermeasures, and primitives for cipher designs.