Delay Monitoring for Wireless Sensor Networks: An Architecture Using Air Sniffers

Wei Zeng*, Xian Chen*, Yoo-Ah Kim*, Zhengming Bu†, Wei Wei‡, Bing Wang*, Zhijie Jerry Shi*
*Computer Science & Engineering Department, University of Connecticut
†Department of Electrical Engineering, Ba Yin Vocational and Technical College, Xinjiang, China
‡Department of Computer Science, University of Massachusetts, Amherst

Abstract—**Wireless sensor networks have been used for many delay-sensitive applications, e.g., emergency response and plant automation. In such networks, delay measurement is important for a number of reasons, e.g., real-time control of the networked system, and abnormal delay detection. In this paper, we propose a measurement architecture using distributed air sniffers, which provides convenient delay measurement, and requires no clock synchronization or instrumentation at the sensor nodes. One challenge in deploying this architecture is how to place the sniffers for efficient delay measurement. We prove the sniffer placement problem is NP-hard and develop two algorithms to solve it. Using a combination of small-scale testbed experiments and large-scale simulation, we demonstrate that our architecture leads to accurate delay monitoring and is effective in detecting abnormal delays, and furthermore, the number of sniffers required by our sniffer placement algorithms is close to the minimum required value.**

I. INTRODUCTION

Wireless sensor networks have been used for many delay-sensitive applications, e.g., emergency response, plant automation and control, and health care. In such networks, measuring the delays inside the network 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 [1]. It is also important for detecting abnormal delays 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 simple: 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. Clock synchronization is, however, a challenging task in large-scale sensor networks. Although numerous solutions have been proposed (see survey [2] and the references therein), they typically require a large number of message exchanges, which consume the scarce energy of the sensor nodes. One way to eliminate the need of clock synchronization is using half of the RTT between two nodes as the one-way delay. This, however, may lead to inaccurate estimates given the asymmetric communication in sensor networks [3].

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 (e.g., CPU, memory, network bandwidth) of sensor nodes. On the other hand, since deploying sniffers incurs additional deployment cost, one key challenge in designing this architecture is how to place the sniffers to minimize this cost. We hence formulate and solve a *sniffer placement problem* which places the sniffers so that (1) each pair of sensor nodes that can transmit to each other is monitored by one sniffer, (2) each sniffer monitors no more than w pairs of nodes (sniffers are simple embedded devices for large-scale deployment, and hence have limited capabilities), and (3) the number of sniffers is minimized. We prove that the sniffer placement problem is NP-hard and develop two algorithms to solve it. One is an approximation algorithm that utilizes max-flow formulation; the other is a simple heuristic algorithm.

We evaluate the feasibility and effectiveness of the proposed architecture using a combination of small-scale testbed experiments and large-scale simulation. The small-scale experiments demonstrate that our architecture obtains accurate delay measurements and is effective in detecting abnormal delays. The large-scale simulation evaluates the performance of the two sniffer placement algorithms, and show that the number of sniffers required by both algorithms is close to the minimum required value.

As related work, several studies (e.g., [4], [5]) have successfully utilized sniffers in infrastructure-based wireless LANs for network management and characterization. Single-hop infrastructure-based wireless LANs, however, differ fundamentally from multi-hop wireless sensor networks as in our setting. Furthermore, none of the these studies addresses how to place sniffers for delay measurement. Several recent studies use sniffers in wireless sensor networks [6], [7]. Their focuses are on code debugging and performance monitoring, not on monitoring delays and placing sniffers as in our study. Our max-flow based algorithm is inspired by [8], which determines how to choose centers from a set of nodes in a network (each center serves a group of nodes). However, our problem differs from that in [8] in important ways. First, in our problem, the monitoring is over pairs of sensor nodes that can transmit to each other, while the monitoring in [8] is over individual
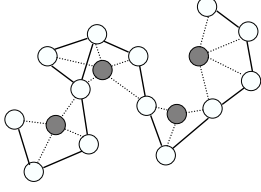
Fig. 1. Measurement architecture using sniffers for a wireless sensor network. The white and shaded nodes represent sensor nodes and sniffers respectively.

nodes. Furthermore, in [8], the set of nodes (and hence the candidate centers) is given beforehand. In our problem, sniffers can be placed at any point in the sensor network (and hence there are infinitely many candidate locations).

The rest of the paper is organized as follows. Section II describes the proposed delay measurement architecture using sniffers. Section III formulates the sniffer placement problem, and Section IV presents two algorithms to solve it. Section V evaluates the feasibility and effectiveness of our architecture. Finally, Section VI concludes the paper and presents future work.

II. DELAY MEASUREMENT USING SNIFFERS

Consider a static sensor network that is used to support a delay-sensitive application. Since the application is delay-sensitive, it is important to measure packet transmission delays inside the network so that real-time control strategies can be adjusted (e.g., in plant automation and control), and abnormal delays can be detected and corrected in a timely manner.

We do not assume the clocks at the sensor nodes are synchronized since clock synchronization requires a large number of message exchanges, and consumes precious energy of the sensor network. Without clock synchronization, obtaining packet transmission delay from one node to another is a challenging task. We propose an architecture that uses sniffers for delay measurement. As we shall see, this architecture provides convenient ways to monitor delays and detect abnormal delays inside the network.

In the architecture, a set of sniffers are deployed at distributed locations inside the sensor network (we discuss where to place sniffers in detail in Section III). Each sniffer has two network interfaces (as in [6], [7]). 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 from this interface do not interfere with the traffic inside the sensor network. Fig. 1 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. Without loss of generality, consider a network path with a hop connecting sensor nodes, A to B. We will describe how to measure delays and detect abnormal delays from A to B using sniffers. 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.

A. Delay monitoring

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 [1]) and detect abnormal delays (as we shall see in Section II-B). 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). As the sniffer obtains

---

[1]This is a common scenario in wireless sensor networks: for many monitoring applications, sources transmit packets periodically, and sequence number is a common technique to differentiate packets from a source.

a sequence of relative delays from A to B, it can adjust the delays by removing clock skew and offset in an online manner (e.g., using the technique in [9]). For the i-th 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 i-th packet from A to B shifted by a constant.

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 interface that operates on the channel not interfering with sensor nodes). 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 [10]. Consider a sequence of delays, $\{d_i\}_{i=1}^n$, where d_i is the i-th 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 $\hat{\sigma}^2 = W_n/(n-1)$.

B. 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 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 [11], [12]. 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 V-A.

C. Summary

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 [7]. For instance, a sniffer may raise an alarm when it stops hearing from a sensor node for a while. We only focus on monitoring delays in this paper.

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. As we shall see (in Section V), our delay monitoring and abnormal delay detection methodologies using this architecture indeed provide accurate results. On the other hand, a key challenge in deploying this architecture is how to place the sniffers for effective monitoring. We formulate and solve *sniffer placement problem* in Sections III and IV, respectively.

III. SNIFFER PLACEMENT PROBLEM

In our proposed architecture, sniffer placement needs to satisfy several constraints. First, as we have seen in Section II, each sniffer needs to monitor the transmission of a *pair* of sensor nodes to obtain per-hop delays. Secondly, each sniffer may only be able to monitor a limited number of sensor node pairs (we assume sniffers are simple embedded devices for large-scale deployment). Last, since deploying sniffers incurs additional cost, it is desirable to minimize the number of sniffers.

We formulate the sniffer placement problem as follows. It places the sniffers inside a sensor netowrk so that (1) any two sensor nodes that can transmit to each other is monitored by at least one sniffer, (2) each sniffer monitors at most w pair of nodes, and (3) the total number of sniffers is minimized. The first constraint assumes that any pair of nodes that can transmit to each other can potentially communicate in the sensor network (and hence needs to be monitored). This is because the network topology changes with traffic demands, and dynamic routing protocols (such as [13]) can render dynamic topologies. The second constraint takes account of the limited capability of the sniffers. We refer to w as the *maximum allowed workload* or *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 aims at minimizing the cost for deploying sniffers.

More formally, consider a wireless sensor network deployed in a two-dimensional area, and let r_i denote the transmission range of sensor node n_i. As in many studies, we assume that the *coverage region* of n_i, R_i, is a circular area, centered at the node, with the radius of r_i. We assume that a sniffer placed at any point in the coverage region of n_i can overhear the transmission from n_i. Furthermore, we assume that $r_i = r, \forall i$. Therefore, two nodes can transmit to each other if their distance is less than r, and a sniffer can overhear a sensor node if its distance to the sensor node is less than r. The goal of the sniffer placement problem is to determine the locations of the sniffers, and assign pairs of sensor nodes to sniffers to satisfy the three requirements stated earlier. When a sniffer s monitors a pair of nodes, n_i and n_j, we denote it as $\varphi(n_i, n_j) = s$, and refer to φ as the *assignment function*.

We prove that the sniffer placement problem is NP-hard (proof is found in the Appendix). In Section IV, we develop efficient approximate algorithms to solve this problem.

IV. SNIFFER PLACEMENT ALGORITHMS

In this section, we first propose a pre-processing algorithm that determines candidate sniffer locations (since a sniffer can be placed anywhere in a sensor network, the number of
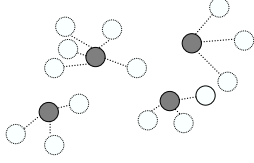
Fig. 2. Illustration of virtual graph.

Fig. 3. Illustration of the max-flow formulation.

candidate sniffer locations is infinite). We then present two algorithms to solve the sniffer placement problem.

A. Determining candidate sniffer locations

We develop the following algorithm to determine a set of candidate sniffer locations, denoted as L. Initially, L is empty. We then consider each pair of sensor nodes, n_i and n_j, in the network. If n_i and n_j can transmit to each other (i.e., their distance is less than r), 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.

Algorithm 1 Determine Candidate Sniffer Locations

1: $L = \emptyset$
2: **for** $\forall n_i, n_j, i \neq j$ **do**
3:    **if** n_i and n_j can transmit to each other **then**
4:       The boundaries of R_i and R_j intersect at two points, denoted as p_1 and p_2
5:       $L = L \cup \{p_1, p_2\}$
6:    **end if**
7: **end for**

We next show that the above algorithm for determining candidate locations is sufficient. That is, suppose S^* is the set of sniffers in an optimal solution. Then, for any sniffer $s \in S^*$, we can find a candidate location in L that corresponds to the location of s.

Theorem 1: $\forall s \in S^*$, 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.

Proof: Without loss of generality, suppose the set of sensor node pairs that are monitored by s is $X = \{(n_i, n_j)\}$. Then 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. Then there exist i, j such that $(n_i, n_j) \in X$, and one intersection point of the boundaries of 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, thus proving our claim. ∎

B. Sniffer placement algorithms

We develop two algorithms for sniffer placement. Both algorithms use Algorithm 1 to determine a set of candidate sniffer locations, and place a sniffer at each candidate location to construct a candidate sniff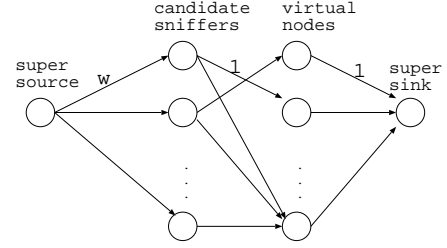er set, S_c. Furthermore, both algorithms consider a *virtual graph* constructed as follows. We first map each pair of sensor nodes that can transmit to each other to a *virtual node*. We then connect a virtual node and a candidate sniffer using a *virtual edge* if the sniffer can monitor the pair of sensor nodes that correspond to the virtual node. Fig. 2 shows an example virtual graph (it is the virtual graph for the example in Fig. 1), where the white dashed nodes and shaded nodes represent respectively the virtual nodes and candidate sniffers, and the dashed lines represent the virtual edges. Let V denote the set of virtual nodes. Let $\varphi(v) = s$ denote that sniffer $s \in S_c$ monitors virtual node $v \in V$. It is equivalent to the assignment function $\varphi(n_i, n_j) = s$, where n_i and n_j are the pair of sensor nodes corresponding to v. Therefore, we only need to assign sniffers to the virtual nodes to obtain assignment in the original problem.

We next present the algorithms for sniffer placement in detail. Both algorithms run in iterations to determine a set of sniffers, $S \subseteq S_c$, and the assignment function, φ. Initially, the set of sniffers, S, is empty. In each iteration, the algorithms add a sniffer 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. We refer to them as *Max-flow* and *Max-degree* Sniffer Placement Algorithm, respectively.

1) Max-flow Sniffer Placement: This algorithm uses a max-flow formulation and is inspired by [8]. We construct a max-flow graph 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 (i.e., w pairs of sensor nodes). Each sensor node is connected to the super sink with the capacity of 1. Fig. 3 illustrates the max-flow graph thus constructed. 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. In the following, we refer to a max-flow graph thus constructed as $G(S_c, V, E, w, 1, 1)$, where the first two

elements represent the sets of candidate sniffers and virtual nodes, respectively; the third element represents the set of edges that connects candidate sniffers and virtual nodes; the last three elements represent the capacity of an edge from the super source to a sniffer, the capacity of an edge from a sniffer to a virtual node, and the capacity of an edge from a virtual node to the super sink, respectively.

Algorithm 2 Max-Flow Sniffer Placement

1: Place a sniffer at each candidate location to construct a candidate sniffer set S_c
2: $S = \emptyset$, $E = \emptyset$
3: **for** $\forall s \in S_c, \forall v \in V$ **do**
4:    **if** s can monitor v **then**
5:       $E = E \cup \{(s, v)\}$
6:    **end if**
7: **end for**
8: **repeat**
9:    $S_c = S_c \setminus S$
10:    **for** $\forall s \in S_c$ **do**
11:       Construct max-flow graph $G(S \cup \{s\}, V, E, w, 1, 1)$
12:       Let f_s denote the maximum integral flow of G
13:    **end for**
14:    $s = \arg\max_{s \in S_c} f_s$
15:    $S = S \cup \{s\}$
16: **until** all virtual nodes are monitored
17: **for** $\forall s \in S, \forall v \in V$ **do**
18:    **if** $f_s(s, v) > 0$ **then**
19:       $\varphi(v) = s$
20:    **end if**
21: **end for**
22: Return (S, φ)

The Max-flow Sniffer Placement algorithm is presented in Algorithm 2. Line 1 places a sniffer at each candidate location to construct a candidate sniffer set, S_c. Line 2 initializes the sniffer set, S, to be an empty set. Lines 3-7 add a set of edges, E, between candidate sniffers and virtual nodes: it adds an edge (s, v) when $s \in S_c$ can monitor $v \in V$. In each iteration (lines 9-15), the algorithm selects a candidate sniffer $s \in S_c$ and $s \notin S$ so that $S \cup \{s\}$ produces the maximum integral flow in the max-flow graph $G(S \cup \{s\}, V, E, w, 1, 1)$, and adds this sniffer into the sniffer set. This process continues until all virtual nodes are monitored. Last, lines 17-21 record the assignment function.

Following the result in [8], we have the following approximation ratio result (the proof is similar to that in [8]; detailed proof is omitted):

Theorem 2: The Max-flow Sniffer Placement algorithm has approximation ratio of $\ln |V|$, where V is the set of virtual nodes.

2) Max-degree Sniffer Placement: Algorithm 3 describes this algorithm. Line 1 places a sniffer at each candidate location to construct candidate sniffer set, S_c. Line 2 initializes the sniffer set, S, to be an empty set. Lines 4-16 describe the

operations in one iteration. In each iteration, the algorithm first constructs graph $G(S_c \cup V, E)$, where S_c and V are the current set of candidate sniffers and virtual nodes, respectively, $(s, v) \in E$ if s can monitor v, $\forall s \in S_c, \forall v \in V$. It then adds the sniffer with the maximum degree into the sniffer set (hence the name Max-degree Sniffer Placement Algorithm). 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). Afterwards, it adjusts S_c and V: line 13 removes s from S_c, and line 14 removes all virtual nodes that are monitored from V. The iteration continues until all virtual nodes are monitored.

Algorithm 3 Max-Degree Sniffer Placement

1: Place one sniffer at each candidate location to construct candidate sniffer set S_c
2: $S = \emptyset$
3: **repeat**
4:    Construct graph $G(S_c \cup V, E)$, edge $(s, v) \in E$ if s can monitor $v \in V$, $\forall s \in S_c, \forall v \in V$
5:    Suppose that $s \in S_c$ has the maximum degree
6:    $S = S \cup \{s\}$
7:    $N(s) = \{v \mid s$ can monitor $v, v \in V\}$
8:    **if** $|N(s)| \leq w$ **then**
9:       $\varphi(v) = s, \forall v \in N(s)$
10:    **else**
11:       Assign sniffer s to w virtual nodes in $N(s)$ that have the lowest degrees
12:    **end if**
13:    $S_c = S_c \setminus \{s\}$
14:    Remove all virtual nodes that are monitored from V
15: **until** all virtual nodes are monitored
16: Return (S, φ)

V. PERFORMANCE EVALUATION

In this section, we evaluate the feasibility and effectiveness of our delay measurement architecture. Our evaluation is on two aspects: one on evaluating the methodologies for delay monitoring and abnormal delay detection; the other on evaluating the sniffer placement algorithms.

A. Evaluation of measurement methodologies

We use testbed experiments to evaluate our methodologies for delay monitoring and abnormal delay detection. Our testbed consists of eight TelosB motes, as illustrated in Fig. 4. All the motes use B-MAC [14], 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. 4. Correspondingly, the power level at each
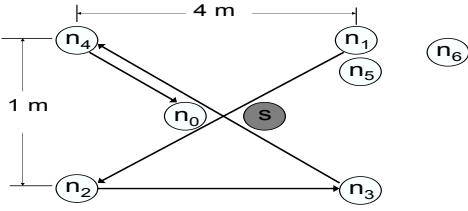
Fig. 4. Testbed setting: node n_0 is the sink, node s is a sniffer.

mote is set to a low level (it is set to 3, i.e., -25 dBm). Node n_0 is the sink. Node s is the single sniffer in the testbed. It can overhead packet transmissions from all the nodes in the testbed. Last, 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.

1) Delay monitoring: To evaluate the delay monitoring methodology presented in Section II-A, we compare per-hop delays obtained using this methodology with those obtained by instrumenting the sensor nodes. More specifically, consider the delay on a network hop from sensor node A to B. This per-hop delay contains two components: the delay at node A and the radio propagation delay for sending a packet from A to B. When ignoring the latter (which is negligible), the per-hop delay equals to 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. Then the difference of these two timestamps is one instance of the delay at A.

We next present evaluation results. In our testbed, we let n_3 be a source, and transmits 1500 packets via n_4 to the sink, n_0. This scenario leads to two network hops: (n_3, n_4), where n_3 is a source, and (n_4, n_0), where n_4 is an intermediate node. We instrument n_3 and n_4 to obtain the delays on hops (n_3, n_4) and (n_4, n_0) respectively, and compare them with the measurements at the sniffer.

For hop (n_3, n_4), since n_3 is a source, the sniffer can only obtain relative delays (it regards the transmission time of the packet with sequence number i as $i\tau$, $\tau = 2$ seconds, and uses the method in [9] to remove clock skew and offset in the delays on the fly). As described in Section II-A, these relative delays differ from the absolute delays by a constant. From our measurements, the relative delays from the sniffer have mean of 5.0 ms and standard deviation of 2.8 ms; the delays obtained by instrumenting n_3 have mean of 12.0 ms and standard deviation of 2.8 ms. Thus, we confirm that the delays from the sniffer and from n_3 indeed have the same standard deviation (since they are off by a constant). Furthermore, for each packet, we obtain the difference of the delay measurements from the sniffer and from n_3. We observe that these differences are indeed close to a constant: they are (7 ± 1) ms, and the error of ± 1 ms are due to the time granularity of 1 ms at the

motes.

For hop (n_4, n_0), since n_4 is an intermediate node, the sniffer can obtain the absolute delays on this hop. We indeed observe that the delays from the sniffer have mean and standard deviation close to those from n_4: the means are 13.2 and 12.8 ms, respectively, and standard deviations are both 2.8 ms. For each packet, we obtain the difference of the delay measurements from the sniffer and from n_4. We observe that 98.4% of packets have differences of 0, 1, or -1 ms, verifying the accuracy of the measurements from the sniffer (again the error of ± 1 ms are due to the time granularity of 1 ms at the motes). For the 1.6% of the packets with larger differences, we suspect that they are caused by measurement noise at either the sniffer or the sensor node.

2) Abnormal delay detection: We next evaluate the effectiveness of the methodology that detects abnormal delays (see Section II-B). 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 look at 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 leads to congestion and hence abnormal delays.

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 II-A that incurs little storage and computation overhead. We explore two change-point detection methods. The first change-point detection 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 [12]. In particular, we define $\tilde{d}_i = d_i - a$, where d_i denotes the i-th 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

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

where $(x)^+$ equals to x when $x \geq 0$, and equals to 0 otherwise. This method updates y_i after each delay observation and raises an alarm when $y_i \geq h$, 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 $1,000$ delay sequences 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 delays. 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 a delay sequence, our change-point detection methods stop and raise an alarm after detecting that the delay has become abnormal. A detection is *successful* if it is within the range of abnormal delays; a detection is a *false alarm* if it is within the range of normal delays; and a *false negative* occurs if no alarm is raised at the end of a delay sequence. Our performance metrics are detection ratio, false positive ratio, false negative ratio, and detection delay (the delay from the change point to when an alarm is raised).

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 33 delay observations.

B. Evaluation of sniffer placement algorithms

We evaluate the performance of the two placement algorithms using simulation (for large-scale evaluation). We consider 100 sensor nodes deployed in a 500 m $\times$ 500 m area using uniform random or grid 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 $\times$ 50 m), and hence the node distribution is more even than that in uniform random deployment. The transmission range of all the sensor nodes is the same, varied from 80 to 140 meters (corresponding to the transmission range of mote-class sensor nodes; we choose the minimum transmission range of 80 m because the network is disconnected when using a lower value). A sniffer is allowed to monitor at most w pairs of sensor nodes. 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 for all the settings, the Max-Flow based algorithm only slightly outperforms the Max-Degree based algorithm: the maximum relative difference is 7% and 12% under uniform random and grid random deployment, respectively. Furthermore, for both algorithms, the results under uniform random and grid random deployments are similar. We
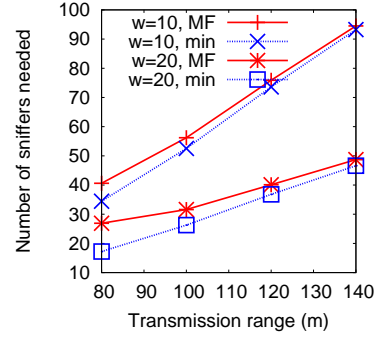

Fig. 5. Number of needed sniffers versus transmission range under Max-flow based algorithm and random uniform deployment.
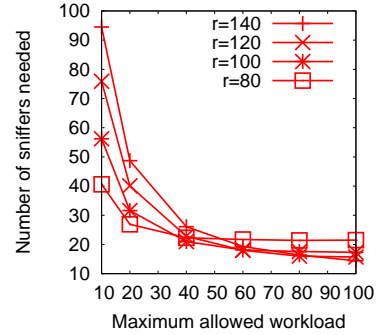

Fig. 6. Number of needed sniffers versus maximum allowed workload, w, under Max-flow based algorithm and random uniform deployment.

next only present the results of the Max-Flow based algorithm under uniform random deployment.

Fig. 5 plots the number of needed sniffers as the transmission range increases from 80 to 140 meters, $w = 10$, or 20. The results from the Max-flow based algorithm and the minimum number of needed sniffers are plotted in the figure (for a given transmission range, the minimum number of needed sniffers is the average number of virtual nodes divided by w, i.e., when all sniffers monitor w virtual nodes). We observe that the results from our algorithm are close to the minimum required values. For smaller transmission ranges, our results differ slightly more from the minimum values than for larger transmission ranges. This is because when the transmission range is small, the network is not sufficiently dense for the sniffers to achieve the maximum allowed workload; as the transmission range increases, the network becomes denser and more sniffers achieve the maximum allowed workload. This is confirmed by workload distribution from our algorithm. For instance, when $w = 20$, the fraction of sniffers that have the maximum workload increases from 36% to 90% as the transmission range increases from 80 to 140 m.

We also observe from Fig. 5 that for $w = 10$ and 20, the number of needed sniffers increases with transmission range. This is because a larger transmission range leads to a denser network with more sensor node pairs (and more virtual nodes) to be monitored, which leads to more needed sniffers. The above results are for small values of w; we

observe an opposite trend for larger values of w. Fig. 6 depicts the impact of the maximum allowed workload, w, on the number of needed sniffers. We observe that, although for small values of w, a larger transmission range leads to more needed sniffers; as w increases, the number of needed sniffers decreases more dramatically for a larger transmission range. In particular, when $w = 40$, all transmission ranges lead to a similar number of needed sniffers, and afterwards, a larger transmission range can lead to less needed sniffers. This is because a larger transmission range leads to a denser network, which, although leads to more virtual nodes to be monitored, can take advantage of larger allowed workloads to reduce the number of needed sniffers.

Last, we observe from Fig. 6 a diminishing gain when increasing w for all transmission ranges: the number of needed sniffers decreases dramatically at first, and then less dramatically afterwards. The above results indicate that the capability of the sniffers need to be carefully chosen: their maximum allowed workload needs to be sufficiently large so that the number of needed sniffers is small, while deploying very capable sniffers may not be cost effective because of the diminishing gains.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an architecture that uses distributed sniffers for delay measurement in wireless sensor networks. We demonstrated that this architecture provides convenient ways for delay monitoring and abnormal delay detection. Furthermore, we developed two algorithms for sniffer placement inside the architecture. Using a combination of testbed experiments and simulation, we demonstrated that our architecture leads to accurate delay monitoring, and is effective in detecting abnormal delays. Furthermore, the number of sniffers required by our sniffer placement algorithms is close to the minimum required value.

As future work, we plan to expand our testbed for a larger scale evaluation of our measurement methodologies. For sniffer placement, we plan to consider more practical issues, e.g., in the presence of obstacles, a sniffer may not be able to hear packet transmissions from a sensor node even when it is within the transmission range of that node.

ACKNOWLEDGMENT

APPENDIX

Theorem 3: The sniffer placement problem is NP-hard.

Proof: We prove this theorem by reducing a known NP-hard problem, geometric K-center problem [15] to the sniffer placement problem. In geometric K-center problem, we are given a constant K, a set of nodes to be served, a set of candidate centers, and the distance from a center to a node (the distances satisfy triangular inequality). Let V denote the set of nodes, C denote the set of centers, and $d(v, c)$ denote the distance from v to c, $v \in V$, $c \in C$. The goal is to find a subset of centers, $C' \subseteq C$ and $|C'| = K$, so that $\max_{v \in V} \min_{c \in C'} d(v, c)$ is minimized.

Our reduction is by showing that if we have an optimal algorithm, A_s, for the sniffer placement problem, then we can devise an optimal algorithm, A_K, for the K-center problem. In the K-center problem, suppose $|V| = n$ and $|C| = m$. We order the distance from a node to a center in non-decreasing order and denote them as $d_1 \le d_2 \ldots \le d_{mn}$. Then for a given d_i, the corresponding sniffer placement problem is as follows. The sensor node set is V, the candidate sniffer set is C, and a sniffer can monitor a sensor node if and only if its distance to the sensor node is within d_i. This instance can be solved using A_k, and let S_i denote the set of sniffers in the optimal solution. We devise an algorithm, A_K, for the K-center problem as follows. It uses A_s to solve the sniffer placement problem by using increasingly larger d_i's. That is, it starts with d_1, and then uses d_2, and so on. The minimum d_i so that $|S_i| \le K$ is an optimal solution for the K-center problem ($C' = S_i$, $\max_{v \in V} \min_{c \in C'} d(v, c) = d_i$). We hence have proved that the sniffer placement problem is NP-hard. ■

REFERENCES

[1] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *IEEE Control Systems Magazine*, vol. 21, pp. 84–99, 2001.

[2] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, 2004.

[3] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *SenSys*, 2003.

[4] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer, "Understanding congestion in IEEE 802.11b wireless networks," in *IMC*, 2005.

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

[6] F. Dressler, R. Nebel, and A. Awad, "Distributed passive monitoring in sensor networks," in *Proc. of IEEE INFOCOM*, 2007. poster.

[7] M. Ringwald, K. Romer, and A. Vitaletti, "Passive inspection of sensor networks," in *DCOSS*, 2007.

[8] J. Bar-Ilan, G. Kortzars, and D. Peleg, "How to allocate network centers?," *J. Algorithms*, vol. 15, 1993.

[9] L. Zhang, Z. Liu, and C. Xia, "Clock synchronization algorithms for network measurements," in *Proc. of IEEE INFOCOM*, 2002.

[10] D. M. Hawkins, P. Qiu, and C.-W. Kang, "The changepoint model for statistical process control," *Journal of Quality Technology*, vol. 35, October 2003.

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

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

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

[14] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys*, 2004.

[15] T. Feder and D. Greene, "Optimal algorithms for approximate clustering," in *STOC*, pp. 434–444, 1988.