Capability and Fidelity of Mote-class Wireless Sniffers

Jordan Cote, Bing Wang, Wei Zeng, Zhijie Shi
Computer Science & Engineering Department
University of Connecticut, Storrs, CT 06269 USA

Abstract—**Monitoring the health of a sensor network is important for maintaining the health and normal operation of the network. For large-scale cost-effective monitoring, using inexpensive motes as sniffers is an attractive choice. In this paper, we quantify the capability and fidelity of mote-class sniffers for sensor network monitoring. In particular, we experimentally quantify the sustainable workload and the accuracy of delay and loss measurements by these types of sniffers. We find that (1) a sniffer can monitor traffic at the rate of 60 packets per second with little buffer overflow, (2) per-hop loss measurements from sniffers exhibit variations but are comparable to those at the receiver and (3) per-hop delay measurements from a sniffer are accurate (the errors are up to 300 μs). Our results indicate that measurement quality by mote-class sniffers is satisfactory for many monitoring purposes.**

I. Introduction

Wireless sensor networks are used for many applications, including environmental monitoring, health care, emergency or medical response and military reconnaissance. Monitoring the health of a sensor network to quickly detect faulty behaviors and take corrective actions is important for maintaining the health and effective operation of the network. Excessive delay in a sensor network may result in sensor data being irrelevant by the time it reaches the destination.

Existing studies use "in-band" and "out-of-band" approaches for sensor network monitoring. In-band monitoring uses sensor nodes to monitor themselves and their neighbors (e.g., [1], [2], [3], [4], [5], [6]). Out-of-band monitoring uses dedicated monitoring nodes that are attached to sensor nodes [7], [8] or placed inside the sensor network to monitor nearby sensor nodes [9], [10], [11].

Out-of-band monitoring has several advantages over in-band monitoring: (1) it does not require instrumenting sensor nodes, and hence requires no change to the application; (2) it does not consume scarce resources (e.g., CPU, memory, storage) of the sensor nodes; and (3) it provides a convenient way to monitor per-hop delays and detect abnormal delays without the need for clock synchronization [11], which is critical for time-sensitive applications such as health monitoring and military intelligence[1]. For cost-effective large-scale out-of-band monitoring, using inexpensive motes as sniffers is an attractive choice. On the other hand, since motes are simple embedded devices with stringent resources, the amount of workload a sniffer can handle and the accuracy of the monitoring results is of critical importance.

[1]Although measuring per-hop delay is straightforward when clocks are synchronized, clock synchronization schemes typically require a large number of message exchanges, which can be very resource consuming [12].

In this paper, we evaluate the capability and fidelity of mote-class sniffers (henceforth simply referred to as sniffers). In particular, we answer the following questions: *What workload can a sniffer sustain? How accurate are the loss and delay measurements from a sniffer?* Through a combination of experimental and analytical study, we find that (1) a sniffer can monitor traffic at the rate of 60 packets per second with little buffer overflow, (2) per-hop loss measurements from sniffers exhibit variations but are comparable to those at the receiver, and (3) per-hop delay measurements from a sniffer are accurate (the errors are up to 300 μs). To the best of our knowledge, our study is the first quantifying the capability and fidelity of motes for monitoring wireless sensor networks.

As related work, the study in [11] proposes a monitoring architecture that uses sniffers to monitor delays in a wireless sensor network. The measurements, however, are coarse-grained (with granularity in milliseconds), and are compared to measurements obtained from sensor nodes (by instrumenting those nodes). In this study, measurements from the sniffers are of much finer granularity (to 30.5 μs), and the results are compared to those from a logical analyzer with accuracy to 100 ns. Furthermore, this study covers a broader scope including quantifying the sustainable workload and the accuracy of delay and loss measurements. In this regard, our study also differs from [13] which focuses on characterizing per-hop and end-to-end delays in a sensor network. Several studies investigate the accuracy and fidelity of IEEE 802.11 sniffers [14], [15]. Our study differs from them in that we focus on mote-class sniffers using IEEE 802.15.4.

The rest of the paper is organized as follows. Section II describes our methodology. Section III presents our results starting with sustainable workload, followed by the accuracy of loss and delay measurements. Finally, Section IV concludes the paper.

II. Methodology

We experimentally evaluate the capability and fidelity of mote-class sniffers using a testbed. In the testbed, sensor nodes and sniffers are TelosB motes that use CC2420 wireless transceivers (IEEE 802.15.4), and run on TinyOS 2.1.0. We first describe experimental settings for measuring sustainable workload, and then describe the settings for evaluating the accuracy of loss and delay measurements.

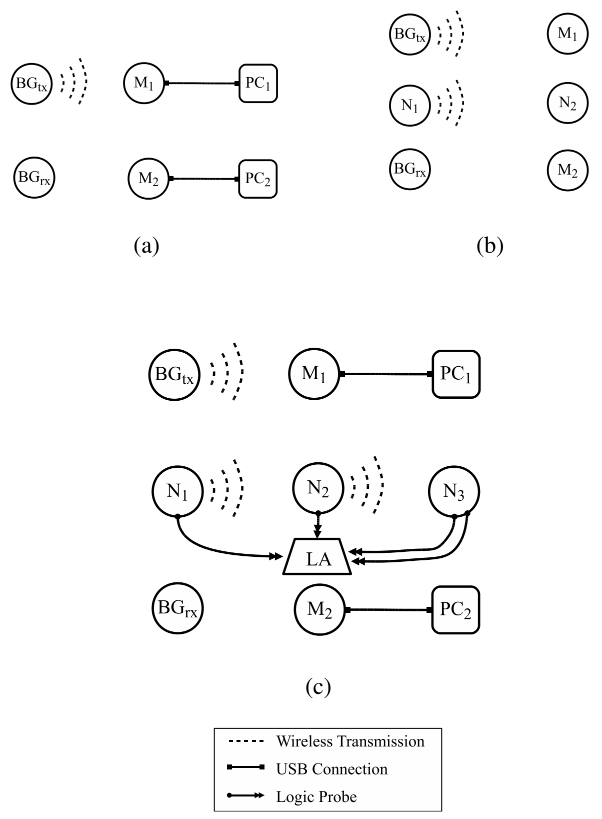

Fig. 1: Experimental testbed layouts: (a) setting to measure sustainable workload, (b) setting to evaluate the fidelity of loss measurements, and (c) setting to evaluate the fidelity of delay measurements. In the figure, M_1 and M_2 are sniffers, BG_{TX}, BG_{RX}, N_1, N_2 and N_3 are motes, PC_1 and PC_2 are computers, and LA is a logical analyzer.

A. Sustainable Workload Measurements

Fig. 1(a) shows the experimental setting for measuring sustainable workload at the sniffers. It consists of a transmitter and receiver, BG_{TX} and BG_{RX}, two sniffers, M_1 and M_2, and two PCs. Each sniffer is connected to a PC via USB. The transmitter sends packets to the receiver with inter-sending times following an exponential distribution (i.e., creating a Poisson arrival process at the sniffers). This traffic is referred to as *background* traffic. It simulates traffic from a large number of virtual nodes. These virtual nodes may be generating packets or simply relaying already transmitted packets to other nodes. To ensure exponential transmission interval, retransmission and CSMA (Carrier Sense Multiple Access) were disabled in BG_{TX}. The minimum sending time of the motes imposes a lower bound on the effective distribution of inter-packet arrival times.

The sniffers passively listen to packet transmissions in their neighborhoods. Upon overhearing a packet from N_1 or BG_{TX}, a sniffer modifies the timestamp field to contain the current time and passes the packet over USB into a data log stored at the connected PC. 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. The granularity of the timestamps is approximately 30.5 μs (by using a 32 kHz clock). 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).

TABLE I: Network packet payload structure.

Bytes	Data
2	Original Sender ID
2	Current Sender ID
2	Sequence Number
1	Number of Hops Traveled
4	Timestamp

Our goal is to measure the workload that can be sustained by the sniffers. For this purpose, we gradually increase the sending rate of BG_{TX} from 5 packets to around 60 packets per second (by decreasing the mean inter-sending time from 200 ms to 15 ms), and measure the corresponding loss rate at the sniffer. The losses at the sniffer are mainly due to receiver buffer overflow (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. It overflows when a packet arrives and cannot be accommodated in the buffer. Interestingly, we find that, according to the TinyOS source code, when buffer overflow occurs, the buffer is flushed along with all the frames that are in the buffer. We verify that indeed multiple packets are lost when the buffer overflows through a program that signals when the overflow event occurs. This flushing behavior differs from common queuing models, a point we will return to in Section III.

B. Loss Measurements

Fig. 1(b) shows the experimental setting for evaluating the fidelity of loss measurements by the sniffers. It extends the setting in Fig. 1(a) to include two network nodes, N_1 and N_2. These two nodes form a one-hop network running Collection Tree Protocol (CTP) [16], a widely used routing protocol for data collection in sensor networks. Node N_1 transmits one packet per second to the sink N_2; each packet carries an 11-byte payload with the fields shown in Table I. We refer to the traffic from N_1 to N_2 as *foreground* traffic. The two monitors, M_1 and M_2 passively log the foreground and background traffic, and count the number of packets lost in the foreground traffic. In this case, we do not connect the sniffers to PCs as in Fig. 1(a) to eliminate the overhead of sending logs from the sniffers to the PCs. Losses are determined by the absence of expected sequence numbers. At the end of the experiment, we download the number of foreground packet losses observed by the sniffers, and we compare them to the number of losses observed by the receiver N_2. We vary the

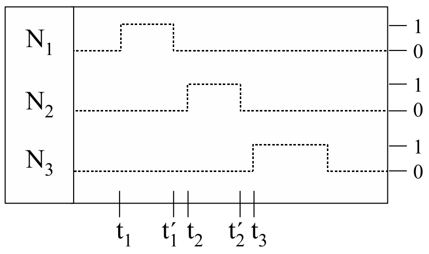

Fig. 2: MCU pin logical timing pattern for the two-hop network as shown in Fig.1(c).

intensity of background traffic to evaluate the accuracy of loss measurements by the sniffers under different workloads.

C. Delay Measurement

For a network path, a sniffer can obtain a relative delay on the first hop and the absolute delays on all later hops [11]. We experimentally evaluate the fidelity of the delay measurements using the setting shown in Fig. 1(c). This setting adds one more sensor node, N_3, and a logic analyzer to the setting in Fig. 1(b). The three network nodes, N_1, N_2 and N_3, form a two-hop network running CTP. Node N_1 sends one packet per second to N_3 through N_2. All three nodes are connected to the logic analyzer via probes, and the logical analyzer is connected to a PC using USB. Sniffers M_1 and M_2 obtain the delays on the first hop (N_1, N_2), and the second hop (N_2, N_3) from the overheard foreground traffic and transmit this information via USB to PC_1 and PC_2, respectively, in an online fashion. By varying the rate of background traffic, we evaluate the accuracy of delay measurements by the sniffers under different workloads.

The logic analyzer is a 34-channel Intronix LA1034 device. It provides a sampling rate of 10 MHz, giving 100 ns accuracy, much finer than the accuracy of the sniffers. Furthermore, by connecting the analyzer to multiple motes, we obtain timestamp events from the motes based on a single clock. These timestamps, as to be described, can be conveniently used to obtain objective and accurate per-hop delays. We therefore use the delay measurements from the logical analyzer as the ground truth to evaluate the delay measurements from the sniffers.

We next describe how the logical analyzer obtains timestamps and per-hop delays in more detail. The Logicport application is used with the logic analyzer to record the timestamps of packet events in the sensor nodes. This is accomplished by using the MCU pin 2.6 of the MSP430 microprocessor on the motes (as in [13]). More specifically, 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 a packet is received and lowers it when it completes transmission. For N_2, the CTP code was augmented in order to accomplish this. In particular, the *SubSend.sendDone* event was modified. For N_3, the pin is also raised upon receiving a packet, and lowered again when the packet information is committed to flash memory. Fig. 2 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 receives the packet and finishes forwarding; t_3 and t'_3 represent respectively the time when N_3 receives the packet 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 at the logical analyzer, we can easily obtain per-hop delays. More specifically, we use $(t'_1 - t_1)$ as the delay on the first hop (since radio propagation delay is negligible, t'_1 is also the time that the packet reaches N_2). For the second hop, we can use $(t'_2 - t'_1)$, $(t_3 - t_2)$, or $(t'_2 - t_2)$ as the delay on that hop. M_1 and M_2 can obtain a relative delay on the first hop, which can be used to detect abnormal delays [11]. This relative delay differs from the absolute delay by a constant. M_1 and M_2 can obtain the absolute delay on the second hop as follows. Suppose the sniffer overhears the transmission of a packet from N_1 at time s_1 (according to the sniffer's local clock), and overhears the forwarding of the packet from N_2 at time s_2. Then it obtains the delay on the second hop as $(s_2 - s_1)$.[2] In Section III-C, we evaluate the measured delays from the sniffers with the various delays measured from the logical analyzer.

Each data log file at the logical analyzer contains a series of timestamps along with the associated state of all probes at that time. Logicport automatically "compresses" the entries so that the only entries in the log are ones for which some signal has changed. Note that although each packet carries a unique sequence number, the logic analyzer only records logical highs and lows, and cannot record the sequence number for a given packet event. On the other hand, since the logical analyzer is connected to N_1 (the source node) through probes, we expect it to record the transmission of all the packets from N_1, which can than be used to sequence the timing logs. To verify this, we program the sink, N_3 to raise a separate pin (MCU pin 2.3) on even packet numbers only (based on the sequence number embedded in the packets). Each log file is checked to be in order by comparing the expected sequence number with the *even* signal. If the signal does not match, a warning is issued to indicate that a packet was lost from N_1. During compilation of our resultant data set, this warning did not occur. The expected signal pattern is then checked with regular expressions to retrieve timestamps. If the pattern is not satisfied, then the analysis code tries to determine what happened. If N_3's signal was never raised, then the packet was lost. It is then checked whether or not N_1's signal remained high until the end of the log. If so, then it did not complete sending the packet and the loss is marked at hop one. Otherwise, the packet is marked lost on the second hop.

[2]This is because, since radio propagation delay is negligible, N_2 receives the packet at s_1, starts to forward it at s_1, and N_3 receives the packet at s_2, and hence $(s_2 - s_1)$ represents the delay from sending the packet from N_2 to N_3.

III. Results

We now present experimental results on sustainable workload and accuracy of loss and delay observations at the sniffers. Each experiment runs for 20 minutes. Unless otherwise specified, we use the measurements from sniffer M_1 (the measurements from M_2 are similar).

A. Sustainable Workload

We first quantify how much workload a sniffer can sustain. For this purpose, we increase the average sending rates from 5 to around 60 packets per second in the setting shown in Fig. 1(a). Table II[3] records the number of lost packets in each experiment (the third column). 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. Furthermore, the loss count for both sniffers is the same.

The losses seem to be due to buffer overflows at the sniffer. To gain additional insights, we use a queuing model to approximate the number of losses at the sniffer. To achieve this, we first measure the processing time of a packet at the sniffer. In particular, we want to know the length of time from when a packet arrives at the buffer to when the packet is removed from the buffer. We use a modified sniffer program to clock this length of time by subtracting 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. Our measurements show that this duration is typically $4.4\ ms$ with little variance. We next calculate the maximum number of packets that can fit inside the receive buffer at the sniffer (which serves as the queue length). Our background packets carry a payload of 20 bytes. With the added MAC and PHY layer data of 18 bytes [13], the total size of one packet stored in the RXFIFO buffer is 38 bytes. Given that the size of the RXFIFO buffer is 128 bytes on a Chipcon CC2420 wireless transceiver, only three packets may reside in the queue before it overflows.

Because the arrival process follows a Poisson distribution, the processing time at the sniffer is constant, and the buffer at the sniffer can hold three packets, we model the sniffer as an $M/D/1/3$ queue. We then obtain the probability of buffer overflow from the queuing model [17]. The analytical results from the model are also shown in Table II (the fourth column). They tend to match well with the experimental results for varying sending rates. It may seem that the experiment should produce more losses, since the RXFIFO buffer is flushed on overflow - causing two more losses each time. However, this is overcome by the fact that the model's state does not return to the empty state of the queue after loss (it remains in state 3). Because of this, losses are more likely in the model until the service unit has time to clear the queue.

[3]The table shows measured inter-sending time. Due to random delays at the sender, the measured delay does not coincide exactly with the value we set.

TABLE II: Sustainable workload measurement results.

Inter-sending time	**Total Packets**	**Lost Packets** ($M_1 = M_2$)	**Lost Packets (analysis)**
16.2 ms	23278	11	10.84
16.2 ms	27303	9	12.72
25.9 ms	12289	2	0.7
30.8 ms	41498	4	1.1
30.9 ms	10402	0	0.27
40.7 ms	30820	2	0.24
51.3 ms	26053	3	0.08
99.25 ms	12515	0	0
197.9 ms	6280	0	0

TABLE III: Number of foreground packet losses. (1101 total packets)

Inter-sending time (background)	**Lost Pkts.** (N_2)	**Lost Pkts.** (M_1)	**Lost Pkts.** (M_2)	**Collisions (analysis)**
200 ms	8	5	9	5
100 ms	0	8	8	10
50 ms	22	13	36	20

B. Accuracy of Loss Measurements

We next evaluate the accuracy of loss measurements by the sniffers. For this purpose, we increase the average sending rate of the background traffic from 5 to 20 packets per second in the setting shown in Fig. 1(b), and compare the number of foreground losses seen by the receiver and the two sniffers in each experiment. In addition, we calculate the number of collisions between foreground and background packets as follows. Let λ denote the average arrival rate of background traffic, and t denote the duration during which a foreground packet is propagating through the air. Since the generation of background packets follows a Poisson distribution, the probability that a foreground packet collides with background packets is $1 - e^{-\lambda t}$ (i.e., the probability that a background packet is transmitted during interval t), and hence the average number of collisions for n foreground packets is $n(1 - e^{-\lambda t})$. In our experiments, the total frame size of a foreground packet is $11 + 18 = 29$ bytes, or 232 bits. The maximum transmission rate of the CC2420 is 250 $kbps$. We therefore approximate t as $232/250\ ms$. Table III shows the number of foreground packet losses at the receiver (i.e., node N_2) and the sniffers, and the number of estimated collisions. We observe that the numbers of losses observed by the receiver and the two sniffers are comparable. Furthermore, they are close to the number of estimated collisions, indicating that foreground losses are mainly due to packet collisions. On the other hand, we observe variations between the measurements of the two sniffers, and the receiver. This is due to many reasons, e.g., different location, different interference level, and different properties of the devices, as has been reported in the context of wireless LAN measurements [14], [15]. As expected, in general the number of lost packets increases as the rate of the

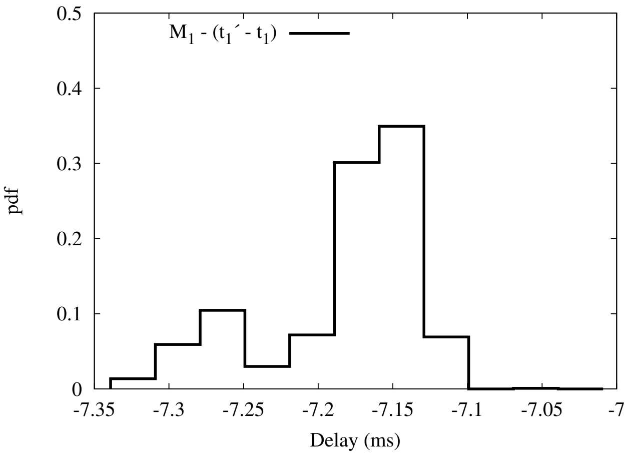

Fig. 3: Distribution of the first-hop delay measurement errors from sniffer M_1.

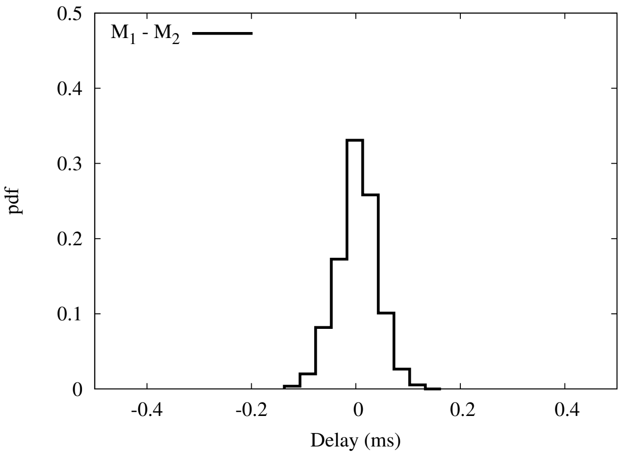

Fig. 5: Difference of delay measurements from M_1 and M_2 for the second hop.

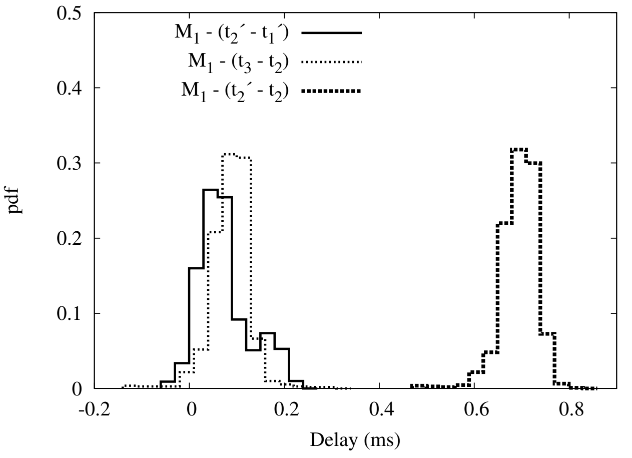

Fig. 4: Distribution of the second-hop delay measurement errors from sniffer M_1.

background traffic increases. However, this is not always the case, again indicating that loss measurements can be affected by many factors.

C. Accuracy of Delay Measurements

We evaluate the accuracy of per-hop delay measurements from the sniffer by comparing them with those obtained by the logical analyzer. More specifically, for a network hop, let $\{d_i\}$ denote the sequence of delay measurements from the sniffer, where d_i is the delay for the ith packet on this hop. Let $\{d'_i\}$ denote the corresponding delay measurements from the logical analyzer. Then the sequence $\{d_i - d'_i\}$ represents the measurement errors of the sniffer. In the following, we present measurement results with no background traffic. We also vary the average sending rate of the background traffic from 5 to 20 packets per second; the findings are similar to those without background traffic, indicating that the delay measurement accuracy is not much affected by the workload at the sniffer. Therefore, those graphs are omitted.

Fig. 3 plots the distribution of the measurement errors on the first hop, where the relative delay measurements by the sniffer are based on the approach in [11], and the delays from the logical analyzer are the duration from just before a packet is sent from N_1 to when it is actually sent into the air (i.e., $(t'_1 - t_1)$ in Fig. 2). From the figure, we see that the difference is indeed close to a constant (the distribution is concentrated in a narrow range of around 350 μs, from -7.35 ms to -7.0 ms).

On the second hop, we compare the delay measurements from the sniffer with three forms of measurements from the logical analyzer: (i) from when N_1 finishes transmitting the packet to when N_2 finishes transmitting the packet (since radio propagation delay can be ignored, the former corresponds to the time that N_2 receives the packet and the latter corresponds to the time that N_3 receives the packet; this is $(t'_2 - t'_1)$ in Fig. 2), (ii) from when N_2 receives / starts to transmit the packet to when N_3 receives the packet (i.e., $(t_3 - t_2)$ in Fig. 2), and (iii) from when N_2 receives / starts to transmit the packet to when it finishes transmitting the packet (i.e., $(t'_2 - t_2)$ in Fig. 2). Fig. 4 plots the distributions of the measurement errors. We observe that the delay measurements from the sniffer are closer to the first two forms of measurements from the logical analyzer (with errors up to 300 μs), while errors are larger compared to the third form of measurements. This is because the third form of measurements neglects the delay due to N_3's post-receive processing, and hence leads to lower (and less accurate) delay values. We also observe from Fig. 4 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.

So far, we have only presented the results from sniffer M_1. We now compare the delay measurements of the two sniffers. Fig. 5 plots the distribution of the relative difference of the second-hop delays measured from the two sniffers. We observe a symmetric distribution in a narrow range of [-100,100] μs, indicating that the measurements at the two sniffers are close to each other. On the first hop, the delay measurements from these two sniffers are off by a constant (which is related to clock skews of these two sniffers) [11]. We verify that the relative difference is indeed in a narrow range, close to a constant (figure omitted).

IV. Conclusions

In this paper, we experimentally explored the capability and fidelity of mote-class sniffers for sensor network monitoring. We quantified the sustainable workload experimentally, and approximated the buffer overflow probability using a queuing model. Furthermore, we compared per-hop loss measurements by the sniffers and the receivers. Last, we compared fine-grain timing measurements from the sniffers to those from a logic analyzer to evaluate the accuracy of delay measurements. We find that a sniffer can monitor traffic at the rate of 60 packets per second with little buffer overflow. We also showed that per-hop loss measurements from sniffers exhibit variations but are comparable to those at the receiver and that per-hop delay measurements from a sniffer are accurate (with errors up to 300 μs). Our results indicate that measurement quality by mote-class sniffers is satisfactory for many monitoring purposes.

As future work, we will pursue in three directions: (1) conduct experiments over other mote platforms, e.g., MicaZ, Imote2, and compare the loss and delay characteristics as well as capability and fidelity of different mote platforms, (2) conduct experiments over large-scale networks (instead of using simulated background traffic), and (3) conduct similar experiments in conjunction with electromagnetic interference from other consumer or industrial devices.

V. Acknowledgments

This work was supported in part by NSF CAREER Awards 0746841 and 0644188. It was additionally supported by DOE GAANN Program Award P200A090340. We would like to thank W. Wei and M. Zink for helpful discussion, as well as the anonymous reviewers for their comments and suggestions.

References

[1] J. Zhao, R. Govindan, and D. Estrin, “Residual energy scans for monitoring wireless sensor networks,” in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, March 2002.

[2] G. Tolle and D. Culler, “Design of an application-cooperative management system for wireless sensor networks,” in *Proc. of European Workshop on Sensor Networks (EWSN)*, January 2005.

[3] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, “Sympathy for the sensor network debugger,” in *SenSys*, November 2005.

[4] S. Rost and H. Balakrishnan, “Memento: A health monitoring system for wireless sensor networks,” in *IEEE SECON*, September 2006.

[5] M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis, “Visibility: A new metric for protocol design,” in *SenSys*, November 2007.

[6] V. Krunic, E. Trumpler, and R. Han, “NodeMD: diagnosing node-level faults in remote wireless sensor systems,” in *MobiSys*, June 2007.

[7] M. Dyer, J. Beutel, T. Kalt, P. Oehen, L. Thiele, K. Martin, and P. Blum, “Deployment Support Network - a toolkit for the development of WSNs,” in *European Conference on Wireless Sensor Networks*, January 2007.

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

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

[10] M. Ringwald, K. Romer, and A. Vitaletti, “Passive inspection of sensor networks,” in *Conference on Distributed Computing in Sensor Systems*, June 2007.

[11] W. Zeng, X. Chen, Y.-A. Kim, Z. Bu, W. Wei, B. Wang, and Z. J. Shi, “Delay monitoring for wireless sensor networks: An architecture using air sniffers,” in *Proceedings of IEEE MILCOM*, (Boston, MA), October 2009.

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

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

[14] P. Serrano, M. Zink, and J. Kurose, “Assessing the fidelity of COTS 802.11 sniffers,” in *Proc. IEEE INFOCOM*, (Rio de Janeiro, Brazil), April 2009.

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

[16] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo, “Tep 123: The collection tree protocol,” August 2006. `http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html`.

[17] O. Brun and J.-M. Garcia, “Analytical solution of finite capacity M/D/1 queues,” *Journal of Applied Probability*, vol. 37, no. 4, pp. 1092–1098, 2000.