Accurate and Robust Time Reconstruction for Deployed Sensor Networks

Wei Dong, *Member, IEEE*, Jie Yu, Jiliang Wang, *Member, IEEE*, Xuefeng Zhang, Yi Gao, *Member, IEEE*, Chun Chen, *Member, IEEE*, and Jiajun Bu, *Member, IEEE, ACM*

***Abstract*—The notion of global time is of great importance for many sensor network applications. Time reconstruction methods aim to reconstruct the global time with respect to a reference clock. To achieve microsecond accuracy, MAC-layer timestamping is required for recording packet transmission and reception times. The timestamps, however, can be invalid due to multiple reasons, such as imperfect system designs, wireless corruptions, or timing attacks, etc. In this paper, we propose ART, an accurate and robust time reconstruction approach to detecting invalid timestamps and recovering the needed information. ART is much more accurate and robust than threshold-based approach, especially in dynamic networks with inherently varying propagation delays. We evaluate our approach in both testbed and a real-world deployment. Results show that: 1) ART achieves a high detection accuracy with low false-positive rate and low false-negative rate; 2) ART achieves a high recovery accuracy of less than 2 ms on average, much more accurate than previously reported results.**

***Index Terms*—Reference clock, sensor networks, time reconstruction.**

I. Introduction

THE NOTION of global time is of great importance for many sensor network applications. Many scientific data are useful only if the collected measurements have accurate global timestamps [1], [2]. With the global time, it is also apparent for the measurement of routing delay, which is a key metric for many network protocols [3].

Traditionally, the global time is setup by a time synchronization protocol, such as FTSP [4], TPSN [5], GTSP [6], etc. These protocols often incur control-plane overhead since they require exchanging timesync messages periodically. Recently, Ferrari *et al.* propose low-power synchronous protocols, Glossy [7] and LWB [8], which exploit constructive interference for fast network floods. They can implicitly perform time synchronization with a very high accuracy. However, constructive interference based flooding suffers from the scalability problem [9].

Manuscript received February 19, 2014; revised October 09, 2014; accepted July 09, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Liu. Date of publication August 19, 2015; date of current version August 16, 2016. This work was supported by the National Science Foundation of China under Grants No. 61472360, No. 61202359, and No. 61373166; the Fundamental Research Funds for the Central Universities, Zhejiang Commonwealth Project 2015C33077, and the Zhejiang Provincial Platform of IoT Technology under Grant 2013E60005.

W. Dong, J. Yu, X. Zhang, Y. Gao, C. Chen, and J. Bu are with the Zhejiang Provincial Key Laboratory of Service Robot, College of Computer Science, Zhejiang University, Hangzhou 310027, China (e-mail: dongw@zju.edu.cn).

J. Wang is with the School of Software and Tsinghua National Lab for Information Science and Technology (TNLIST), Tsinghua University, Beijing 100084, China (e-mail: jiliang@greenorbs.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2456214

Rather than setting up a global time among all network nodes, the global time can also be reconstructed offline by instrumenting the protocol stack and analyzing the received packets at the sink [10], [11], eliminating the need for timesync message exchanges in the network. MAC-layer timestamping technique [12] is typically employed in order to achieve microsecond measurement accuracy. The key idea of MAC-layer timestamping is to record the timestamps at time instants closest to the actual transmissions or receptions so that protocol processing delays can be eliminated.

MAC-layer timestamping, however, can introduce errors due to two main reasons. First, the timestamping errors happen frequently when the traffic is bursty with low power listening (LPL) mechanisms in TinyOS. The community suspects that the errors are mainly caused by mismatches between the timestamps and the corresponding packets. Second, the timestamps can be corrupted during wireless transmissions. Even worse, those corruptions cannot be detected since the timestamps are not protected by the CRC in the current CC2420 protocol stack in order not to introduce CRC computation delays when embedding the timestamps into the packets before transmissions.

For these reasons, our current work aims to propose an effective approach to processing and analyzing the collected packet traces so that invalid timestamps can be detected and recovered. While a better system design will certainly facilitate interpreting the data at the higher layer, we argue that offline processing and analysis are valuable for improving data acquisition quality. First, it allows to clean historical artifacts in data derived from an initial and imperfect system version. This is very valuable since more sensor data can be utilized despite early imperfections in the realization of a sensor system [10]. Second, even a well designed system may suffer from fundamental limits [10], e.g., packet corruptions during wireless transmissions. Third, data integrity validation is a valuable tool even if a system is designed and/or operating correctly. Lastly, our approach is also useful for tolerating timing attacks in sensor networks. For example, an attacker may compromise and abuse a sensor node to send incorrect timestamps, or an attacker may modify and fake the timestamps, causing incorrect estimate of the timing information [13], [14]. Our approach is robust against such timing attacks.

Our approach tries to find out the properties valid packets (i.e., packets with valid timestamps) have and utilize these properties for identifying valid packets. The basic idea of our approach is based on the fact that valid packets from the same source node are conforming to each other considering the

1063-6692 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

constraint on the relative clock drift. Random errors in invalid packets will break such a constraint with high probability, i.e., an invalid packet will be nonconforming to a valid/an invalid packet with high probability. The valid packet identification problem can thus be reduced to the maximum clique problem that is NP-hard in a general graph. To solve this problem, we employ a strict conformance relation, transforming the original problem to a problem that can be solved in polynomial time. After we have identified the valid packets, we are able to utilize them for error recovery.

The contributions of our work are summarized as follows.

- We propose ART, an accurate and robust approach to detecting and recovering invalid timestamps. Our approach is much more accurate and robust than the threshold-based approach, especially in dynamic networks with inherently varying propagation delays.
- We formulate the problem of identifying valid packets as the maximum clique problem in a graph. By utilizing the transitive property of packet conformance, we transform the problem to the longest path problem in a directed acyclic graph that can be solved in polynomial time. We prove the correctness of our algorithm.
- We evaluate ART in both testbed and a real-world deployment. Results show that: 1) ART achieves a high detection accuracy with low false-positive rate and low false-negative rate; 2) ART achieves a high recovery accuracy of less than 2 ms on average.

The rest of this paper is structured as follows. Section II discusses related work. Section III presents the design details on how to reconstruct accurate global time in spite of errors, including the packet timestamping method (Section III-A), the error detection algorithm (Section III-B), and the error recovery algorithm (Section III-C). Section IV introduces the implementation. Section V shows the evaluation results. Finally, Section VI concludes this paper.

II. Related Work

Many sensor network systems have been deployed in recent years [1], [2], [15]–[17]. The PermaSense project [1] aims to model physical processes related to high-alphine permafrost. The SensorScope project [15] deploys several WSN systems from EPFL's campus to high-mountain sites. The GreenOrbs project [17], [18] includes more than 300 nodes to monitor environmental conditions in the forest. The CitySee project [2] is deployed in an urban area for measuring the carbon absorbance and emission in different zones.

Many scientific data are useful only if the collected measurements have accurate global timestamps. On the other hand, many real-world deployments reported a high percentage of packet losses, packet corruptions, and data inconsistencies [19]. In particular, it is well recognized that the TinyOS CC2420 MAC-layer timestamping mechanism suffers from bogus readings when the incoming traffic is bursty with LPL mechanisms.[1] While system design and implementation at the low level will effectively improve state-of-the-arts, a good implementation is both time-consuming and labor-intensive [20]. On the other hand, offline data analysis at a high level can be useful for detecting and recovering certain errors and has the potential to reveal the fundamental limits of a system design. It shifts the complexity from the sensor network to the PC, resulting in little overhead on the deployed system.

Time reconstruction can be achieved in two ways. First, timesync protocols can be used to setup a global time across the entire network. Second, the global time can also be reconstructed without a timesync protocol by instrumenting and analyzing the received packets at the sink. With timesync protocols, each node knows the global time. Without timesync protocols, each node does not know the global time. The global time, however, can still be reconstructed at the sink.

Many timesync protocols, such as TPSN [5], FTSP [4], and GTSP [6], require exchanging timesync messages periodically in order to maintain a global time. Maintaining an accurate and stable clock among all nodes in a deployed system is challenging considering practical issues such as instability in clock frequency, error propagation, etc. [21], [22]. Moreover, running an additional timesync protocol incurs additional complexity and run-time overhead. In particular, broadcasting additional timesync messages on top of asynchronous low-power MAC protocols is shown to be costly [23]. Recently, Ferrari *et al.* proposed Glossy, which combines flooding and time synchronization by exploiting constructive interference of radio packets. Low-Power Wireless Bus (LWB) extends Glossy by supporting one-to-many, many-to-one, and many-to-many traffic. It maps all traffic demands on fast network floods, and globally schedules every flood. The use of constructive interference can be very efficient for many sensor networks. Glossy and LWB can flood packets within a few milliseconds. Results show that Glossy can achieve an average time synchronization error below one microsecond. However, constructive interference based flooding suffers the scalability problem. The packet reception performance of intermediate nodes degrades significantly as the density or the size of the network increases [9]. Glossy is validated in networks with no more than 8 hops. However, our approach targets for large-scale sensor networks, e.g., GreenOrbs and CitySee with more than 20 hops.

Time reconstruction methods reconstruct the event time at a particular node with respect to a reference clock (e.g., the clock of the sink node). A few research works utilize regularities in environments to reconstruct the time. For example, Lukac *et al.* [24] use microseismics for global time reconstruction, while Gupchup *et al.* [25] use sunlight measurements. Phoenix [26] is another recent work dealing with offline time reconstruction. However, it requires periodically exchanging timing information within the sensor network. Keller *et al.* [10] reconstruct the temporal order and generation time of packets by instrumenting and analyzing the received data packets. Similar to Keller *et al.* [10], our work aims to reconstruct accurate packet generation time by analyzing received packets offline. Our work has two main differences from [10]. First, our work employs MAC-layer timestamping to achieve microsecond accuracy. Second and more importantly, we propose detection and recovery algorithms to specifically deal with invalid MAC-layer timestamps experienced in real-world deployments.

[1]See bug reports: Bug in CC2420 timestamp at TinyOS mailing list (http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2007-October/028892.html); PacketTimeStamp CC2420 bug (http://docs.tinyos.net/tinywiki/index.php/PacketTimeStamp_CC2420_bug); CC2420 and microsecond precision timestamps at TinyOS mailing list (https://www.millennium.berkeley.edu/pipermail/tinyos-help/2010-August/047507.html).

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

Keller *et al.* [27] propose Multihop Network Tomography, MNT, to reconstruct the per-packet routing path and the bounds of the arriving times at intermediate nodes. The reconstruction of the routing path assumes the availability of the global packet generation time, i.e., MNT relies on the information provided by [10]. We believe that the more accurate timestamps provided by ART can further improve MNT's reconstruction performance [27]. Our previous work [11] performs measurements and analysis on the delay performance in a deployed sensor network. It focuses on analyzing factors impacting the delay performance and the implications to protocol designs without details on how invalid timestamps can be detected and recovered in an accurate and robust manner. Our current work addresses the issue on how the delay metric can be accurately reconstructed in a real deployed network.

III. Design

We assume a data collection sensor network in which all (or a subset of) nodes collect sensing data and forward the data to the sink node via multihop wireless. The sink node connects to the rest of the network via low-power wireless. The sink node can be attached to a host computer via serial connection. We would like to reconstruct the data generation time with respect to the sink node. For our application, knowing the data generation time with respect to the sink node is sufficient since we only care about the relative time difference of the sensed events. We note that the time can be further converted to the time of the host computer when required. This conversion is complementary to our work and is well solved by existing approaches [28], which can achieve an average synchronization accuracy of less than 10 μs.

One important design consideration is that the reconstruction approach should be applicable for large-scale sensor networks with LPL mechanisms. In the widely used TinyOS system, LPL is the standard mechanism to achieve low duty-cycling, which is very important to prolong the lifetime of a deployed sensor system. CTP is one popular data collection protocol in TinyOS. CTP can be combined with LPL to achieve low-power operations (without application-level changes). With LPL, each node periodically polls the channel for a short duration (e.g., 11 ms) in each cycle. If energy is detected, the node stays awake for another short duration (defaults to 100 ms in TinyOS). Otherwise, the node enters the sleep state and resamples the channel in the next cycle (i.e., 500 ms later). When a node stays awake, it can receive packets. It can also reply with ACKs if required. To transmit a data packet, the sender first transmits a long packetized preamble to wake up the receiver (i.e., the parent node if CTP is employed). For unicast with link-layer ACK, the sender can stop the transmission immediately when an ACK is received. For broadcast, the preamble lasts for the cycle duration.

We do not assume a specific data generation model, i.e., data can be generated and sent either periodically or randomly. Data packets can follow different paths to the sink from a specific sender (see detailed explanation in the paragraph after Theorem 1). We do not require time synchronization among sensor nodes.

Fig. 1 gives an overview of our approach, ART. There are three steps. First, a packet timestamping component is installed on each sensor node for estimating the network sojourn time (Section III-A). Second, ART employs a graph theoretical approach for detecting valid/invalid timestamps in the received packets (Section III-B). Third, ART utilizes the valid timestamps for time reconstruction for packets with invalid timestamps (Section III-C).

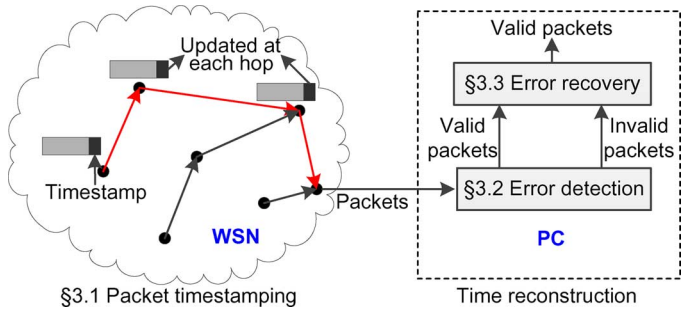

Fig. 1. ART overview.

Before describing the design details, we first introduce the following notations:

- Packet. A received packet at the sink, p, is represented by a tuple: $p = (s, k, \hat{sk})$, where s is the packet generation time measured at the source node (in ms), k is the packet reception time measured at the sink node (in ms), and $\hat{sk}$ is the estimated packet generation time with respect to the sink's clock.
- Clock model. A local clock $\tau = (1+\rho)t + \beta$ where ρ denotes the clock drift, t denotes the reference time (we use sink's time as the reference time in this paper), and β denotes the offset. The clock drift is bounded by $[-\rho_m, \rho_m]$. For example, the TelosB datasheet reports a maximum drift of 40 ppm with respect to the reference clock [29]. We set $\rho_m = 80$ ppm considering clock drifts of the source node and the sink node.
- Transmission delay. We denote d as the real transmission delay that is not observable and $\hat{d}$ as the measured transmission delay, which satisfies

$$(1-\rho_m)\cdot d \le \hat{d} \le (1+\rho_m)\cdot d \Leftrightarrow |d-\hat{d}| \le \rho_m \cdot d. \quad (1)$$

  This is because the measured delay is estimated by the network sojourn time that is measured using each forwarder's local clock [see (4)]. Therefore, the real delay is bounded by the measured delay

$$\frac{\hat{d}}{1+\rho_m} \le d \le \frac{\hat{d}}{1-\rho_m}. \quad (2)$$

- Packet generation time in sink's clock (i.e., global packet generation time). We denote sk as the real global packet generation time that is not observable and $\hat{sk}$ as the measured global packet generation time. Clearly, $d = k - sk$ and $\hat{d} = k - \hat{sk}$. Combining (1), we have

$$|sk - \hat{sk}| \le \rho_m \cdot d. \quad (3)$$

A. Packet Timestamping

In this section, we present the packet timestamping mechanism to reconstruct the packet generation time with respect to the sink's clock, i.e., $\hat{sk}$. For this purpose, we instrument the data packet. In each data packet, there are two 4-B fields:

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

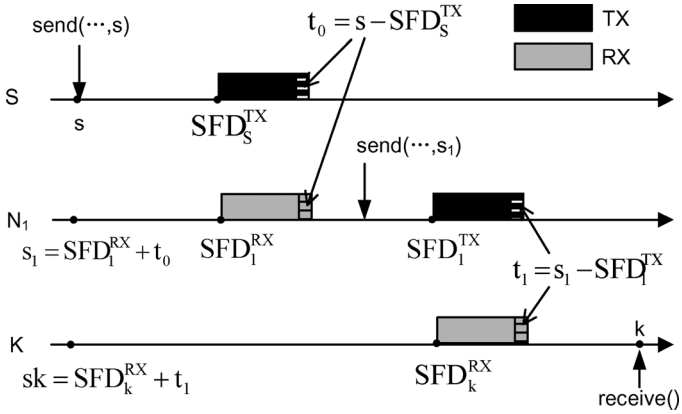

Fig. 2. Measurement of the global packet generation time (sk) for a two-hop transmission path (S, N_1, K).

- Local packet generation time s. It is the local time that the packet is passed from the application layer to the network layer (e.g., CTP [30]).
- MAC-layer timestamp that is accumulated hop by hop and is used to convert the *local* packet generation time to the *global* packet generation time with respect to the sink's clock.

The measurement of $\hat{sk}$ uses MAC-layer timestamping. During packet transmission, when a preamble of a packet is transmitted, a start frame delimiter (SFD) interrupt will be generated immediately. During packet reception, when the preamble of a packet is received, an SFD interrupt will also be generated immediately.

We assume a transmission path of $(S, N_1, \ldots, N_m, K)$, where S is the source node and K is the sink node. We explain the measurement of $\hat{sk}$ based on the MAC-layer timestamping mechanism as follows [see Fig. 2 for an example of two-hop transmission path (S, N_1, K)].

First, the packet generation time s is passed to the radio stack when the node is sending the packet. When the packet is actually transmitted (i.e., the SFD interrupt is signaled), a local timestamp SFD_S^{TX} is taken and the difference of s and SFD_S^{TX} (i.e., $t_0 = s - SFD_S^{TX}$) is appended to the data packet and transmitted over the radio.

Second, when receiving the timestamped packet at a forwarder N_i with timestamp t_{i-1}, the packet reception time is recorded as SFD_i^{RX} (when the SFD interrupt is signaled) in the local clock. The packet generation time with respect to the forwarder's local clock is inferred as $s_i = SFD_i^{RX} + t_{i-1}$. (We assume the transmit SFD and the receive SFD happen at the same time as radio propagation time over short distance is negligible.) When the forwarded packet is actually transmitted (i.e., the SFD interrupt is signaled), a local timestamp SFD_i^{TX} is taken, and the difference of the packet generation time s_i and SFD_i^{TX} (i.e., $t_i = s_i - SFD_i^{TX} = SFD_i^{RX} + t_{i-1} - SFD_i^{TX}$) is appended to the data packet and transmitted over the radio.

Finally, when the sink node receives the packet with timestamp t_m, the packet generation time with respect to the sink's clock is inferred as $\hat{sk} = SFD_K^{RX} + t_m$.

We can see that

$$\hat{sk} = SFD_K^{RX} - \sum_{i=1}^{m} (SFD_i^{TX} - SFD_i^{RX}) - (SFD_S^{TX} - s). \tag{4}$$

Let us denote $\hat{sj}_S$ as the sojourn time at S: $\hat{sj}_S = SFD_S^{TX} - s$, $\hat{sj}_K$ as the sojourn time at K: $\hat{sj}_K = k - SFD_K^{RX}$, and $\hat{sj}_i$ as the sojourn time at N_i: $\hat{sj}_i = SFD_i^{TX} - SFD_i^{RX}$. Then $\hat{sk} = SFD_K^{RX} - \sum_i^m \hat{sj}_i - \hat{sj}_s$. That is, the packet generation time with respect to the sink's clock is given by subtracting the estimated sojourn time of a packet at $(S, N_1, \ldots, N_m)$ from the arrival timestamp SFD_K^{RX}. The transmission delay can thus be estimated as $\hat{d} = \hat{sj}_S + \sum_i^m \hat{sj}_i + \hat{sj}_K = k - \hat{sk}$.

We can see that using MAC-layer timestamps (i.e., record the timestamps at the SFD interrupts) can yield accurate measurement results since the sojourn time essentially captures the nondeterministic delays at local nodes including software routines and MAC backoffs. It is worth noting that the above description is also valid for networks with LPL: The packet depicted in Fig. 2 corresponds to the first received packet after the sender wakes up the receiver.

MAC-layer timestamps (i.e., t_i), however, are prone to measurement errors especially when LPL is employed. The default TinyOS CC2420 protocol stack saves the actual reception timestamps in a queue in the SFD interrupt before the packet is fully received. After the SFD interrupt, the protocol stack waits until the entire packet is received into the radio chip's FIFO buffer. Only after the entire packet has been received, the protocol stack starts downloading the packet to a message_t structure in the MCU's RAM. When the download of the packet is completed and the CRC check is passed, the corresponding timestamp is removed from the queue and placed in the metadata of the message_t structure. It is possible that another packet is received into the radio chip's FIFO buffer while the MCU is downloading the packet into RAM. The downloading procedure may be preempted, causing complex interleaving executions. This can potentially cause mismatch between the packet and the timestamp corresponding to the SFD event of the packet. If there are reception failures, e.g., the length byte is corrupted, or the CRC check fails, etc., the corresponding timestamp needs to be removed from the queue. If the timestamp is not at the head of the queue, the incorrect removal will cause mismatches for all the timestamps already in the queue.

It is also important to note that packet corruptions typically occur in wireless networks due to signal fading and interference. To make the situation even worse, the 4-B timestamp is not protected by the CRC in the current CC2420 protocol stack in order not to introduce CRC computation delays when embedding the timestamp into the packet before transmission. Thus, corrupted timestamps cannot be detected by the CRC mechanism.

In the above cases, the estimated value of $\hat{sk}$ cannot be trusted. Therefore, we need to filter out those invalid measurements and design efficient algorithms to reconstruct valid estimates of $\hat{sk}$.

B. Error Detection

A simple threshold-based approach is to setup a maximum delay threshold d_m to cutoff invalid packets. Valid packets are those with $\hat{sk} : 0 < k - \hat{sk} \leq d_m$. The selection of d_m is, however, nontrivial considering varying transmission delays in dynamic routing protocols.

We avoid the selection of d_m by employing another approach. Our approach tries to find out the properties valid packets have and utilize these properties for identifying valid packets. The

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

basic idea of our approach is based on the fact that valid packets from the same source node are conforming to each other in the sense that the difference of s and the difference of $\hat{sk}$ should be very close considering sufficiently small clock drifts. Random errors in invalid packets will break such a relationship with high probability, i.e., an invalid packet will not be conforming to a valid/an invalid packet with high probability.

In the following paragraphs, we will use subscripts to differentiate different packets from the same source node (when required). We use a smaller subscript to denote a packet with earlier packet generation time, i.e., for two packets p_i, p_j from the same source node, $i < j$ implies that $s_i < s_j$. Two valid packets from the same source node satisfy a certain constraint. Conceptually, the difference of s and the difference of $\hat{sk}$ should be very close considering a sufficiently small clock drift, i.e., $s_2 - s_1 \approx \hat{sk}_2 - \hat{sk}_1$. Concretely, the following theorem describes this constraint exactly.

Theorem 1: For two valid packets $p_1 = (s_1, k_1, \hat{sk}_1)$ and $p_2 = (s_2, k_2, \hat{sk}_2)$ from source node S ($s_1 < s_2$), they satisfy the following constraint:

$$\frac{\Delta s}{1+\rho_m} - \frac{\rho_m}{1-\rho_m} \cdot \Sigma\hat{d} \le \Delta\hat{sk} \le \frac{\Delta s}{1-\rho_m} + \frac{\rho_m}{1-\rho_m} \cdot \Sigma\hat{d} \quad (5)$$

where $\Delta s = s_2 - s_1$, $\Delta\hat{sk} = \hat{sk}_2 - \hat{sk}_1$, and $\Sigma\hat{d} = \hat{d}_1 + \hat{d}_2$.

In the Appendix, we give the formal proof for the theorem. There are some important points we would like to make clear. First, although we use subscripts 1, 2 in Theorem 1, we do not require these packets to be consecutive in the sequence number. It also holds true for the following definitions, theorems and corollaries. Second, packets p_1 and p_2 might reach the sink over different paths. In this case, Theorem 1 still holds true. Different routing paths will impact the delays that are considered in (5) by the term $\sum \hat{d}$: the longer the traveling time the packet stays in the network, the more probable that the sojourn time at each node $\hat{sj}_i$ will deviate due to clock drifts, impacting the accuracy of $\hat{sk}$. We also consider the *maximum* clock drift ρ_m specified by the hardware: clock drift at any instant (including sudden changes in clock drift) is supposed to keep within ρ_m. Therefore, Theorem 1 is a *necessary* condition for the two packets to be valid.

Definition 1: Two packets p_1 and p_2 from source node S are conforming (denoted as $p_1 \simeq p_2$) iff (5) is satisfied.

We can see that packet conformance is a necessary condition for packet correctness. Consider an ideal condition in which all the packets from a source node are valid and they form the set P. According to Theorem 1, any two packets in P are conforming, i.e., $\forall p_i, p_j \in P, p_i \simeq p_j$. With a graph representation $G = (V, E)$ where V represents the set of received packets from a source node (we will use packets and vertices interchangeably), and $e = (p_i, p_j) \in E$ iff $p_i \simeq p_j$. If all packets are valid, G is a clique.

Assume a random error occurs, then the probability that an invalid packet conforms to a valid packet is low. Consider a case illustrated in Fig. 3. For two valid packets, Δsk must fall in the range of $\left[\frac{\Delta s}{1+\rho_m} - \varepsilon, \frac{\Delta s}{1-\rho_m} + \varepsilon\right]$ where $\varepsilon = \frac{\rho_m}{1-\rho_m} \cdot \Sigma\hat{d}$. If there happens an abrupt change in $\hat{sk}_2$, say sk_2 now becomes $sk_2' = sk_2 - C$ where C is a constant, Δsk will move to the point

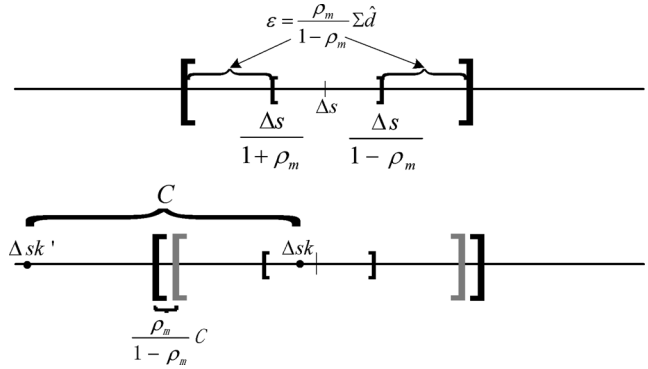

Fig. 3. Error detection using packet conformance [i.e., (5)].

$\Delta sk' = \Delta sk - C$. At the same time, the specified range will increase slightly considering a small clock drift. Therefore, it is highly probable that Δsk will move to the left of the specified range, and the two packets will no longer conform to each other. It means that G is no longer a clique.

The probability that two invalid packets conform to each other is also low if the errors happen randomly. The reasoning is similar as stated above: The change in Δsk is usually much larger than the increase of the range. Therefore, we can use packet conformance for error detection. Specifically, *it is reasonable to consider that the valid packets will form the maximum clique in G.*

Based on the above idea, a straightforward method to identify valid packets is to find the maximum clique in G. However, it is NP-hard to find the maximum clique in a general graph [31]. The execution overhead will be prohibitively large considering a large number of received packets.

We try to solve the valid packets detection problem (i.e., maximum clique problem) by redefining a graph G having good properties. Problems on certain graphs have algorithms with polynomial execution time.

In particular, we redefine the edges in $G(V, E)$: instead of using the conformance relation $\simeq$ defined in Definition 1, we use a more "strict" conformance relation $\sim$ defined in Definition 2, i.e., $e = (p_i, p_j) \in E$ iff $p_i \sim p_j$.

Definition 2: Two packets p_1 and p_2 from source node S are strictly conforming (denoted as $p_1 \sim p_2$) iff

$$\frac{\Delta s}{1+\rho_m} \le \Delta\hat{sk} \le \frac{\Delta s}{1-\rho_m}. \quad (6)$$

We can see that the strict conformance relation slightly shortens the specified range by ε [compared to (5)]. Without loss of generality, we consider $\varepsilon > 0$ since we can immediately identify an invalid packet with $\hat{d} < 0$. Therefore, we only need to check packets having $\hat{d} > 0$ and $\varepsilon > 0$. The relation between conforming and strict conforming is apparent: Two strictly conforming packets must conform; the opposite, however, does not necessarily hold true.

In theory, using (6) for identifying valid packets, it is probable that there will be false negatives (i.e., valid packets misidentified as invalid) since there is probability that two valid packets conform but do not strictly conform. Such probabilities are very small under certain conditions.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

If $\Delta\hat{sk}$ is uniformly and randomly distributed in its maximum allowable range specified in (5), the probability that two valid packets conform but do not strictly conform (denoted as P_{fn}) can be derived as follows (see the Appendix for the derivation).

Theorem 2: If $\Delta\hat{sk}$ is uniformly and randomly distributed in its maximum allowable range specified in (5), the probability that two valid packets conform but do not strictly conform satisfies

$$P_{\text{fn}} \leq \frac{2}{2 + \frac{\Delta s_m}{d_m(1-\rho_m^2)}} \tag{7}$$

where d_m is the maximum delay in the network and Δs_m is the minimum transmission interval between two packets originated from the source node.

Theorem 2 implies that our approach will be more robust in a network with relatively low transmission rate and short propagation delays. For instance, in the CitySee network with transmission period of 10 min and a maximum network delay of 10 s, the probability that two packets conform but do not strictly conform is smaller than 3.2%.

The reality will be even more optimistic since $\Delta\hat{sk}$ will be in a smaller range than that specified in (5) [mostly within the range specified in (6)] considering that the relative clock drifts along the routing path are evenly distributed. Suppose fast clocks and slow clocks are evenly distributed along the routing path. $\hat{d} = \hat{sj}_S + \hat{sj}_K + \sum_{i=1}^{m} \hat{sj}_i$ will be close to d no matter how long the routing path is. This is because sojourn time errors at certain fast/slow nodes will be compensated by other slow/fast nodes. In other words, (1) can actually be made more stringent with a smaller bound. It is also true for (3) and (5). Hence, most packet pairs will reside in a range smaller than that specified in (5).

In Section V, we will show that more than 99.99% conforming packet pairs are strictly conforming for a deployed sensor network.

The benefits of using the strict conformance relation in defining the edges in G is that it will make an equivalent directed acyclic graph *transitive*. It is known that the maximum clique problem in a transitive graph has polynomial-time algorithms [32].

With Definition 2, we have the following theorem (see the Appendix for the proof):

Theorem 3: Transitive relation: If packets p_2 and p_1 are strictly conforming, p_3 and p_2 are strictly conforming ($s_1 < s_2 < s_3$), then p_3 and p_1 are strictly conforming, i.e.,

$$p_2 \sim p_1 \&\& p_3 \sim p_2 \Rightarrow p_3 \sim p_1.$$

We say that the transitive relation is weak in the sense that it holds true for a given order of packets. In other words, $p_1 \sim p_3$ and $p_2 \sim p_3$ do not necessarily yield $p_1 \sim p_2$.

Corollary 1:

$$p_1 \sim p_2, p_2 \sim p_3, \ldots, p_{n-1} \sim p_n (s_1 < s_2 < \cdots < s_n) \Rightarrow p_1 \sim p_n.$$

We define a directed acyclic graph $G' = (V, E')$ where V is the set of packets from a source node and $e' = (p_i, p_j) \in E'$ iff $p_i \sim p_j$ and $s_i > s_j$ (i.e., a recent packet will point to an early packet). The undirected version, $G = (V, E)$, is the corresponding graph by turning directed edges into undirected edges. Fig. 4(a) and (b) shows the undirected graph G and the directed graph G'.

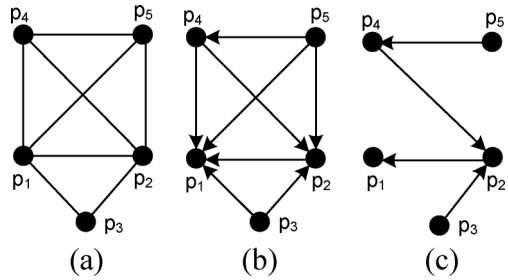

Fig. 4. Graph representation using strict packet conformance [i.e., (6)]: (a) Undirected graph G in which an edge represents the two endpoints (i.e., packets) are strictly conforming. (b) Directed acyclic graph G' in which a recent packet points to an early packet. (c) Directed acyclic graph in which the implicit edges are removed away. For example, since the edges (p_4, p_2) and (p_2, p_1) imply $p_4 \sim p_1$, so the edge (p_4, p_1) does not appear in the graph.

Algorithm 1 Algorithm for constructing the directed acyclic graph G′

Input: P: the set of packets of a give source node
Output: G': directed acyclic graph

```
1:  procedure CONSTRUCT
2:    for i: 0 ≤ i < |P| do
3:      for j: 0 ≤ j < i do
4:        if CONFORM(p_j, p_i) then
5:          Adj[p_i] = Adj[p_i] ∪ p_j

6:  procedure CONFORM(p_j, p_i)
7:    // check if two pkts are strictly conforming
8:    if p_j ~ p_i according to (6) then
9:      return TRUE
10:   else
11:     return FALSE
```

In the Appendix, we prove that the nodes comprising the longest path in the directed acyclic graph G' will form the maximum clique in the corresponding undirected graph G.

Theorem 4: The nodes comprising the longest path of the directed acyclic graph G' will form the maximum clique in the corresponding undirected graph G.

Therefore, we only need to find the longest path for identifying valid packets. It is shown that the longest path problem in a directed acyclic graph can be solved in polynomial time using dynamic programming [31], [32].

Algorithm 1 shows the pseudocode for constructing the directed acyclic graph. We use the adjacency list representation of the graph: $\text{Adj}[p_i]$ stores all earlier packets strictly conforming to p_i.

Algorithm 2 shows the pseudocode for finding the longest path. The procedure TOPORDER() returns a topological order of vertices using depth first search. Since this algorithm requires edges point in the opposite direction: Each node is visited only after all its predecessors have been visited, the topological order is obtained by reversing the visited order. For example, a valid

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Algorithm for identifying valid packets

```
Input: G': directed acyclic graph
Output: C: the set of valid packets
procedure LONGEST-PATH
  for v in G' do
    len[v] := 0; pre[v] := −1;
  for v in topOrder(G') do
    for each edge (v, w) in G' do
      if len[w] <= len[v] + 1 then
        len[w] = len[v] + 1;
        pre[w] = v;
  find v that has the max len[v]
  for i = v; i! = −1; i = pre[i] do
    push p_i to C
  return reverse C

procedure topOrder(G)
  L = Empty list that will contain the sorted elements
  S = Set of all nodes with incoming edges
  for each node n in S do
    dfs(n)
  procedure dfs(node n)
    if n has not been visited yet then
      mark n as visited
      for each node m with an edge from n to m do
        dfs(m)
      add n to L
  return reverse L
```

topological order for Fig. 4 is p_5, p_4, p_3, p_2, p_1 (the topological order can also be p_5, p_3, p_4, p_2, p_1 or p_3, p_5, p_4, p_2, p_1). The longest path algorithm visits the nodes in topological order. For each visiting node v, the algorithm checks each outgoing edge to w and records in len[w] the maximum path length seen so far. The algorithm also uses the pre[] array to backtrace the previous node.

The working details of Algorithm 2 can be illustrated using the example shown in Fig. 4.

1) For $v = p_5$, the algorithm checks p_1, p_2, p_4. $\text{len}[p_1] = \text{len}[p_2] = \text{len}[p_4] = 1$, and $\text{pre}[p_1] = \text{pre}[p_2] = \text{pre}[p_4] = p_5$.
2) For $v = p_4$, the algorithm checks p_1, p_2. $\text{len}[p_1] = \text{len}[p_2] = 2$, and $\text{pre}[p_1] = \text{pre}[p_2] = p_4$.
3) For $v = p_3$, the algorithm checks p_2. $\text{len}[p_2]$ and $\text{pre}[p_2]$ do not change since the path from p_3 is not longer.
4) For $v = p_2$, the algorithm checks p_1. $\text{len}[p_1] = 3$ and $\text{pre}[p_1] = p_2$.
5) For $v = p_1$, no actions are performed since p_1 has no outgoing edges.

Hence, the maximum path length is $\text{len}[p_1] = 3$, and the path is (p_5, p_4, p_2, p_1).

In the Appendix, we formally prove that the above algorithm outputs the longest path in G'.

Theorem 5: Algorithm 2 outputs the longest path in the directed acyclic graph G′.

The complexity of the algorithm is $O(T + |V| + |E| + |V|)$ where $T = O(|V| + |E|)$ is the complexity of the topological sort algorithm. The complexity of the algorithm can be further reduced if we eliminate implicit edges implied by the transitive relation. For example, Fig. 4(c) shows the directed acyclic graph without implicit edges. However, the reduction of implicit edges will cause additional overhead in constructing the graph. Hence, we do not eliminate implicit edges in our implementation.

C. Error Recovery

After we have detected the set of valid packets, we are able to recover the global packet generation time sk for the invalid packets. The basic idea is simple: For each invalid packet, we find the nearest two valid packets right before and after it. Then, we use a linear clock model to get an estimate of sk.

For a valid packet, according to (3) and (2), we have

$$|sk - \hat{sk}| \leq \rho_m \cdot d \leq \frac{\rho_m}{1-\rho_m}\hat{d}.$$

Therefore, the global packet generation time sk satisfies

$$\hat{sk} - \frac{\rho_m}{1-\rho_m} \cdot \hat{d} \leq sk \leq \hat{sk} + \frac{\rho_m}{1-\rho_m} \cdot \hat{d}.$$

That is

$$sk^l = \hat{sk} - \frac{\rho_m}{1-\rho_m} \cdot \hat{d} \text{ and } sk^u = \hat{sk} + \frac{\rho_m}{1-\rho_m} \cdot \hat{d} \quad (8)$$

where sk^l and sk^u denote the lower and upper bounds of sk.

Consider a packet p with invalid $\hat{sk}$, we cannot get an accurate estimate of sk based on $\hat{sk}$. To get an estimate of sk, we look for the nearest valid packets p_1 and p_3 with $s_1 < s < s_3$, and try to utilize those two valid packets to get an estimate.

According to the clock model, we have

$$\frac{s - s_1}{sk - sk_1} = \frac{s_3 - s_1}{sk_3 - sk_1} = 1 + \rho.$$

Therefore

$$sk = \frac{s - s_1}{s_3 - s_1} sk_3 + \frac{s_3 - s}{s_3 - s_1} sk_1.$$

Combining (8), we can get the lower and upper bounds of sk as follows:

$$\begin{aligned} sk^l &= \frac{s - s_1}{s_3 - s_1}\left(\hat{sk}_3 - \frac{\rho_m}{1-\rho_m}\hat{d}_3\right) \\ &\quad + \frac{s_3 - s}{s_3 - s_1}\left(\hat{sk}_1 - \frac{\rho_m}{1-\rho_m}\hat{d}_1\right) \\ sk^u &= \frac{s - s_1}{s_3 - s_1}\left(\hat{sk}_3 + \frac{\rho_m}{1-\rho_m}\hat{d}_3\right) \\ &\quad + \frac{s_3 - s}{s_3 - s_1}\left(\hat{sk}_1 + \frac{\rho_m}{1-\rho_m}\hat{d}_1\right). \end{aligned} \quad (9)$$

Hence, we estimate the value of sk as:

$$\hat{sk} = \frac{sk^l + sk^u}{2} = \frac{s - s_1}{s_3 - s_1}\hat{sk}_3 + \frac{s_3 - s}{s_3 - s_1}\hat{sk}_1. \quad (10)$$

Algorithm 3 shows the pseudocode for recovering the sk field based on (10).

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3 Algorithm for recovering invalid packets

Input: $E = P \setminus C$: the set of all invalid packets from a single source node
Output: E': the set of invalid packet with recovered sk

```
1: procedure RECOVER
2:     for p ∈ E do
3:         Find nearest p_1 and p_3 with s_1 < s < s_3
4:         // Now p_1 and p_3 have been found
5:         Compute sk according to (10)
```

Now that we have successfully obtained an estimate of the global packet generation time for almost all packets, we can further derive two important system metrics:

- Packet transmission delay. The packet transmission delay can be estimated as $\hat{d} = k - \hat{sk}$.
- Clock drift between two successive packets. The drift estimates can be derived

$$\rho^l = \frac{\Delta s}{\Delta sk^u} - 1 \quad \text{and} \quad \rho^u = \frac{\Delta s}{\Delta sk^l} - 1$$

where ρ_l and ρ_u denote the lower and upper bounds of the drift. We estimate the drift as: $\hat{\rho} = \frac{\rho^l + \rho^u}{2}$.

There is probability that s and k may occasionally be invalid. For example, if s is invalid, the recovered $\hat{sk}$ will also be invalid. We ensure the correctness by checking the recovered drift $\hat{\rho}$, which should satisfy $\hat{\rho} \in [-\rho_m, \rho_m]$. If k is invalid, we can use nearby packets for detection and estimation. We want to stress that in practice the error probability of s and k is very low while the error probability of sk is high, especially with low power listening techniques.

IV. Implementation

We implement the packet timestamping mechanism (Section III-A) in the CTP protocol based on TinyOS 2.1.1. The core algorithm for detecting and recovering errors (Sections III-B and III-C) consists of 585 lines of code in Perl.

The detection and recovery algorithm constructs a separate graph for each source node. The running overhead of the algorithm can be large for a large number of receiving packets. We optimize the performance by dividing the receiving packets into fixed-sized windows (in terms of number of packets) and execute the algorithm in each window to speed up the execution time.

There is a tradeoff in determining the window size w. If w is too small, the result will be inaccurate. For instance, if $w = 1$, all packets will be considered valid, resulting in incorrect detection results. On the other hand, if w is too large, the running overhead for the detection algorithm ($O(n^2)$) will be large (the recovery algorithm runs much faster since the time complexity is $O(n)$, where n is the number of received packets).

We perform testbed experiments with 24 TelosB nodes (as well as in a large-scale deployed network; see Section V), running the modified CTP protocol with a packet transmission period of 1 min. We collect a total of 66, 168 packets, which translates to about 2700 packets from each node. We then apply two versions of the detection algorithms: 1) parameterized detection with window size w; 2) full detection on all received packets for each source node (i.e., about 2700). We consider the detection results of the full detection algorithm to be ground truth.

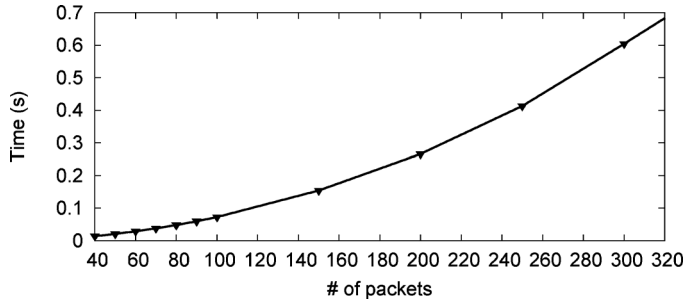

Fig. 5. Execution time (s) in each window of size w.

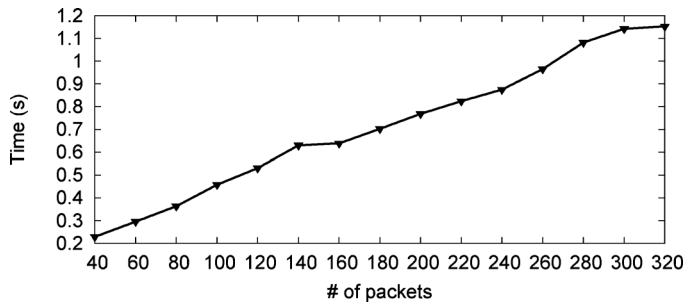

Fig. 6. Total execution time (s) varied with w.

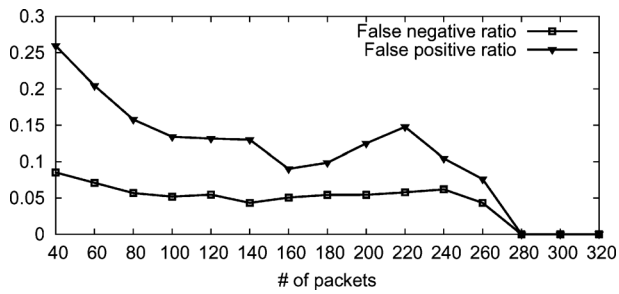

Fig. 7. False positive and false negative varied with w.

Fig. 5 shows the execution time for one source node in each window with different sizes. We can see that the time overhead is quadratic with the number of received packets in each window. Fig. 6 shows the total execution time for one source node using parameterized detection. We can see that a small window size yields small execution overhead because the time complexity becomes $O(w^2) \cdot n/w = O(wn)$. Fig. 7 shows the false positive rate (i.e., invalid packets misidentified as valid) and false negative rate (i.e., valid packets misidentified as invalid) for the parameterized detection algorithm. We see that parameterized detection with $w = 300$ yields the same result as the full detection algorithm while incurring a relatively small execution overhead of 1.15 s for a single source node. In the following evaluations, we will use a window size of 300.

V. Evaluation

We apply ART in CitySee, a large-scale deployed sensor network in urban areas. We evaluate the time reconstruction performance for 764 541 packets collected from one subnet with more than 280 nodes from July 19 to 25, 2011. In addition, we also use our own testbed with 24 TelosB nodes (see Fig. 8) to

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

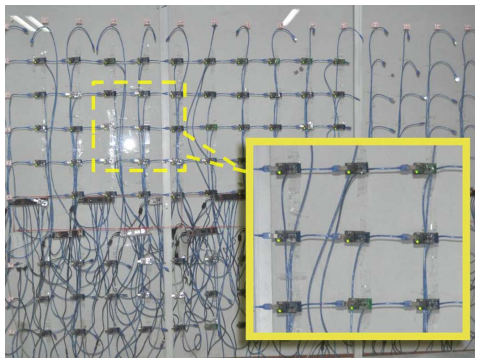

Fig. 8. Testbed.

reveal more system insights of ART when multiple parameters vary.

CitySee: The main applications of CitySee include measuring the carbon absorbance and emissions in different zones of a city, supporting the government's policy decision in energy saving and emission reduction, and offering citizens with convenient daily living services. In CitySee, each sensor node reads the sensor data, measures and records system status, and delivers packets to the sink with a period of 10 min. The application uses the TinyOS LPL MAC protocol (with a sleep interval of 500 ms) for achieving low duty-cycling, the CTP routing protocol [30] for multihop routing, and the Drip dissemination protocol for disseminating and configuring key system parameters.

In this paper, we analyze data packets for a subnet of more than 280 nodes from July 19 to 25, 2011.

Evaluation Methodology: We use the packet timestamping mechanism to instrument the data packets by attaching two 4-B fields, i.e., a local packet generation time and a MAC-layer timestamp (as described in Section III-A). The sink node additionally records the packet reception time.

After collecting packets with fields specified in Section III-A, we analyze the collected packets at the sink. First, we filter out corrupted and duplicate packets by checking a few packet contents. Second, we apply the error detection and recovery algorithm to each individual node. Finally, we can get an estimation of the global packet generation time sk, the packet transmission delay d, and the clock drift ρ between two successive packets.

In the testbed experiments, we rely on external events triggered by the arrival times of special probing packets (transmitted with the maximum transmission power) to test the reconstruction accuracy. On reception of such a time probe, each node writes both its hardware and logical time to the external flash memory. At the end of the experiment, the measurement results stored in the flash memory of the nodes are transferred to the PC for further analysis. The accuracy can be inferred since the logical times of one time probe correspond to the same reception event.

We evaluate our approach in the following sections. Section V-B shows the results for our detection algorithm. We use a metric of "drift violations" to compare the quality of our result to that obtained by a simple threshold-based approach. Section V-C shows the robustness of our detection and recovery algorithm by varying both the percentage of errors and the magnitude of the error. We examine the detection accuracy in terms of false positives (i.e., the number of invalid packets misidentified as valid) and false negatives (i.e., the number of valid packets misidentified as invalid). We examine the recovery algorithm in terms of the absolute error (we know the correct value since we artificially inject errors). Section V-D shows the corrected measurements using our algorithms. Finally, Section V-E compares ART to traditional time synchronization.

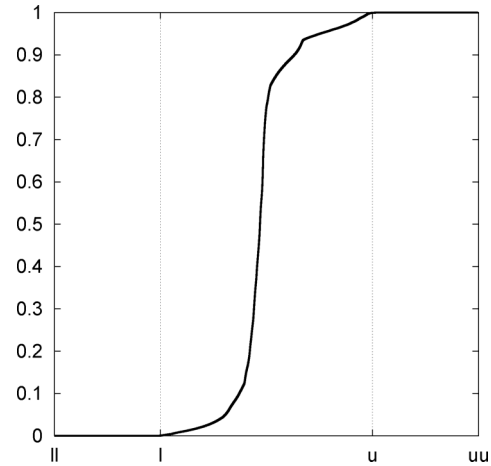

Fig. 9. CDF of the $\Delta\hat{sk}$ normalized in the range specified in (5) and (6). $[\mathrm{l}, \mathrm{u}] = \left[\frac{\Delta s}{1-\rho_m}, \frac{\Delta s}{1+\rho_m}\right]$, $[\mathrm{ll}, \mathrm{uu}] = [\mathrm{l} - \epsilon, \mathrm{u} + \epsilon]$.

TABLE I
DETECTION RESULTS

	Testbed	CitySee
Unfiltered packet trace		
Correct pkts (i.e. total pkts)	3,629	764,541
Drift violations	2,180(60.1%)	542,824(71.0%)
Threshold-based approach		
Correct pkts	2,401(66.2%)	334,104(43.7%)
Drift violations	579(24.1%)	85,864 (25.7%)
Our approach		
Correct pkts	2,020(55.7%)	257,650(33.7%)
Drift violations	0(0%)	0(0%)

A. Detection Results

Before we show the detection result, we first verify that the probability that packet pairs conform but do not strictly conform is extremely low in our deployed system. Fig. 9 shows the CDF of the $\Delta\hat{sk}$ normalized in the range $\left[\frac{\Delta s}{1+\rho_m} - \epsilon, \frac{\Delta s}{1-\rho_m} + \epsilon\right]$ [i.e., specified in (6)]. Using both traces from the testbed and CitySee, we find that the fraction of packet pairs conform but do not strictly conform only occupies less than 0.01% (the ratio of invalid packets, however, can be as large as 60% as observed in Table I). Therefore, using strictly conformance for detection can yield accurate results.

Fig. 10 gives a visual illustration on the directed acyclic graphs for nodes 9 and 20 in the testbed experiment. Each vertex in the graph represents a received packet. Packets (vertices) are ordered such that early packets are in the inner part,

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

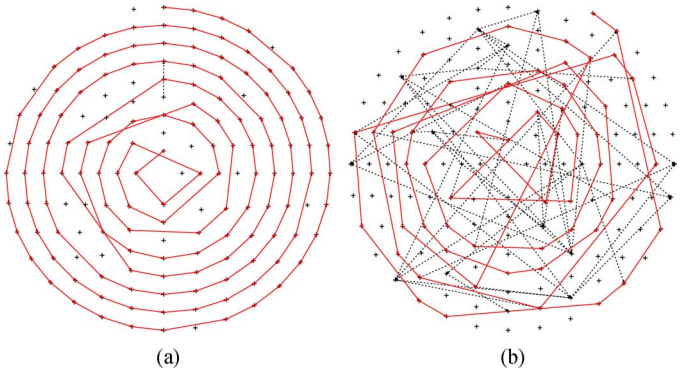

Fig. 10. Packet conformance graph for (a) node 9 and (b) node 20 in the testbed. The solid and dotted lines denote packet conformance relation. The longest path is illustrated by red solid lines.

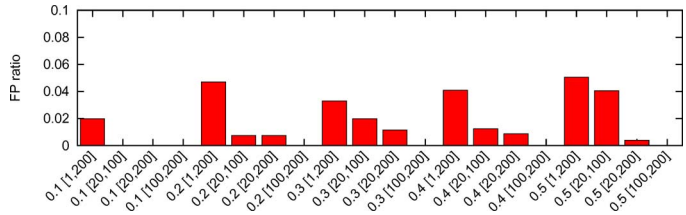

Fig. 11. False positive rates. The experiment settings are indicated in the labels of the x-axis in the format $e[a, b]$, where e is the injected error rate and $[a, b]$ indicates the interval of the increase.

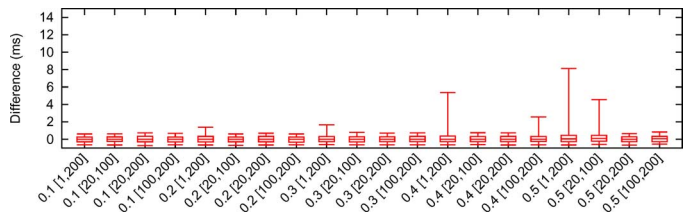

Fig. 12. Difference of the estimated global packet generation time and the "real" global packet generation time. Experiment settings are indicated in the x-axis labels as in Fig. 11. In each experiment, the figure shows the median, the 25% percentile, the 75% percentile, the 5% percentile, and the 95% percentile.

while recent packets are in the outer part. For clear illustration, the implicit edges are removed away. Nodes comprising the longest path are identified as valid packets. A few isolated packets have a poor conformance to other packets, indicating that they are invalid with high probability. Packets from node 9 are mostly valid, while node 16 experiences much more errors.

Table I shows our detection results. In simple threshold-based approach described in the start of Section III-B, we use $d_m = 5000$ ms for the testbed and $d_m = 10\,000$ ms for CitySee, considering the duty cycle of 500 ms and maximum hop counts of 10 and 20, respectively. We believe that most valid packets shall have delays falling in this range.

We use a metric called drift violations to compare the detection quality of different approaches. The number of drift violations is computed as follows. For a sequence of received packets from a single source node, we count the number of times two successive packets are not conforming according to (5). Then, we sum up all the numbers for all nodes. Clearly, if all packets are valid, the number of violations must be 0 [(5) is a necessary condition for two valid packets]. On the other hand, if there exist errors, the number of violations will be large. Hence, we can use this metric for comparing the quality of detection result.

Table I shows that the (unfiltered) packet traces consist of 3629 and 764 541 packets, for the testbed and CitySee, respectively. With the simple threshold-based approach to cutoff invalid packets, only 66.2% and 43.7% packets are accepted (as valid packets). We see that there are large false positives since our approach only accepts 55.7% and 33.7% packets.

It is obvious that there is no violation using our approach, while the number of violations using the threshold-based approach is 24.1% and 25.7% for the testbed and CitySee, respectively. Hence, our approach yields the highest detection quality.

B. Detection and Recovery Accuracy

A direct measurement on the accuracy of detection and recovery is challenging since we lack ground truth. We examine the robustness of our detection and recovery algorithm by "artificially injecting" errors and varying the percentage and magnitude of errors.

For a specified percentage of all identified valid packets, we intentionally increase the value of $\hat{sk}$ at random. Our detection algorithm takes these modified traces as inputs. We would like to see if our algorithm can detect such intentionally injected errors.

We find in testbed and real deployment that real errors are roughly uniformly distributed in a large range (see Fig. 13). Large errors can be definitely detected by our algorithm. For example, with packet transmission period of 10 min, there are two consecutive packets ($s_1 = 0$, k_1, $\hat{sk}_1 = 1000$), ($s_2 = 600$, k_2, $\hat{sk}_2 = 1600 + \theta$) where θ denotes the introduced error. If $\theta \geq 100$ ms $= 0.1$ s, $\Delta\hat{sk} = \hat{sk}_2 - \hat{sk}_1 = 600 + \theta$ will be outside the range $\left[\frac{\Delta s}{1+\rho_m} \approx 599.95, \frac{\Delta s}{1-\rho_m} \approx 600.05\right]$. Hence, these two packets are no longer considered conforming. In order to see the robustness of our algorithm, we intentionally keep the increase in a small range, i.e., ≤ 200 ms. We conduct experiments under the following parameter settings. The percentages of injected errors are selected to be 10%, 20%, 30%, 40%, 50%. We increase $\hat{sk}$ by a value randomly distributed in the range of $[a, b]$ ms. Hence, the range reflects the magnitude of the error. The ranges are selected to be [1, 200], [20, 100], [20, 200], [100, 200]. Intuitively, the larger the error (e.g., [100, 200]), the easier can it be detected by our algorithm.

Fig. 11 shows the false positive rate of our algorithm (i.e., invalid packets misidentified as valid) for the total number of $5 \times 4 = 20$ experiments. We do not show results for the threshold-based approach since it cannot identify errors with small increases (i.e., they will be definitely accepted as valid). We can see several facts. 1) With the same percentage of errors (e.g., 10%), the false positive rate decreases as the error becomes larger. This is obvious since large errors will more likely break the drift constraint. 2) With the same magnitude of errors (e.g., [1, 200]), the false positive rate increases as the error percentage increases. 3) In all the cases, the false positive rate is very low. It is worth noting that a high false positive rate is only possible when the increase is very small, e.g., in the range of [1, 200]. This rarely happens in practice.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

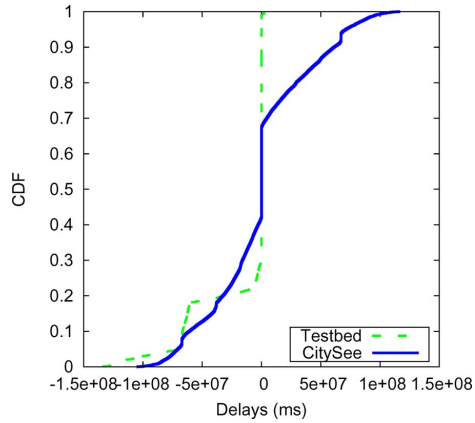

Fig. 13. Delays (unfiltered) for two datasets.

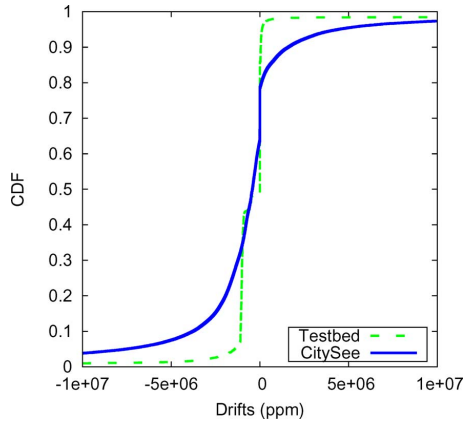

Fig. 15. Drifts (unfiltered) for two datasets.

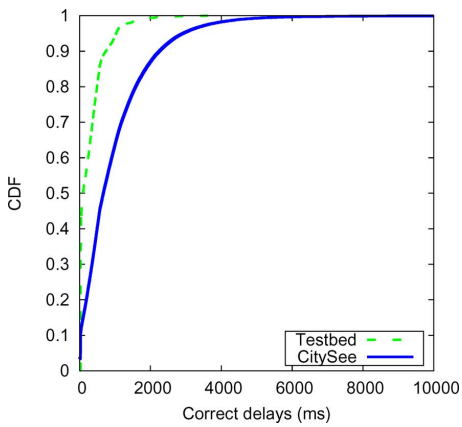

Fig. 14. Recovered delays for two datasets.

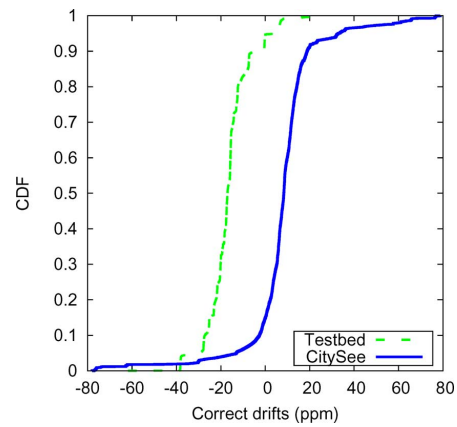

Fig. 16. Recovered drifts for two datasets.

We next examine the false negatives (i.e., valid packet misidentified as invalid). A direct measure of false negative is difficult since we lack ground-truth knowledge. We check if there are possibilities that there are misidentified invalid packets by examining their conformance [according to (5)] to the identified valid packets (these valid packets are also manually verified). If the misidentified invalid packets are valid, they will conform to all other valid packets since (5) is a necessary condition. We account for the number of invalid packets having conformance to all valid packets. We find that there are no such packets in all packet traces we examined, indicating there are no false negatives.

To see the accuracy of our recovery algorithm, we examine the absolute error in terms of the difference of the recovered sk and the real sk, which is originally valid in these experiments. Fig. 12 shows the difference for the total 18 experiments. For each experiment, we show the median value, the 5% percentile, the 95% percentile, the 25% percentile, and the 75% percentile. We can see that our approach can achieve an accuracy of less than 2 ms on average. Compared to accuracy result of 2.8 s in previous work [10] (which utilizes a second-level timer), our result is three orders of magnitude more accurate.

C. Reconstructed Results

We can obtain two important network metrics using the data fields (s, k, $\hat{sk}$) in packets: routing delay ($\hat{d} = k - \hat{sk}$) and time drift between two received packets. However, without error detection and recovery, there will be large fractions of invalid timestamps as shown in Figs. 13 and 15. As $\hat{sk}$ is corrupted, the measured delay and drift are invalid. Fig. 13 shows very large delays that are impossible in our network. Fig. 15 shows very large time drifts that far exceed the specified maximum value in the TelosB datasheet. The above results clearly indicate the measured results must be further processed to obtain accurate results.

With our time reconstruction approach, we can transform the erroneous metrics shown in Figs. 13 and 15 into the correct ones. Fig. 14 shows the CDF of recovered delays using our approach. The result is consistent with our expectation: The network delay should be on the order of seconds. The conclusion is similar for the drifts. Fig. 16 shows the CDF of recovered drifts using our approach. The recovered drifts fall in the expected range of $[-80\ \text{ppm},\ 80\ \text{ppm}]$. For a more detailed analysis and discussion on the measurement results, please refer to [11].

D. Comparison to Traditional Time Synchronization

We conduct experiments in a 24-node testbed to compare ART to FTSP [4]—a traditional time synchronization protocol for accurate time reconstruction. In the experiments, we use the CitySee program that incorporates the ART approach. For comparison, we add the FTSP protocol to the CitySee program (without ART) for time reconstruction using traditional time synchronization. The transmission power is set to -31.5 dbm in order to simulate multihop behaviors. Each node periodically

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

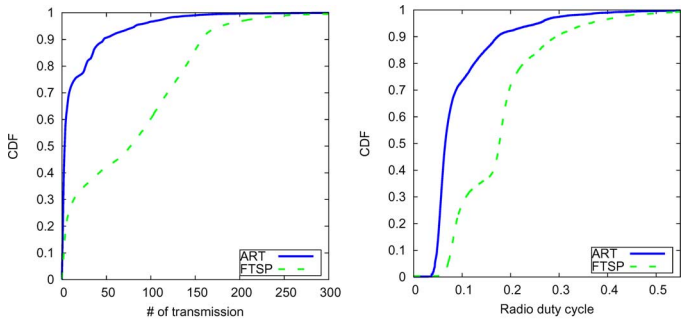

Fig. 17. Comparison of ART and FTSP. (a) Transmission overhead. (b) Radio duty cycle.

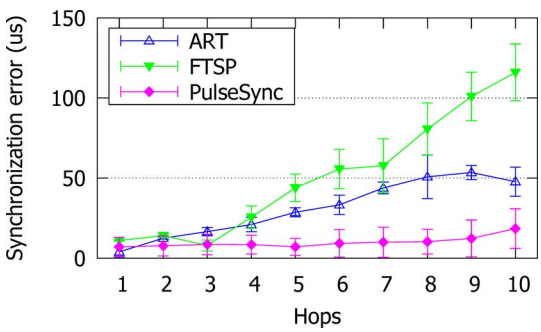

Fig. 18. Time reconstruction accuracy.

sends three packets to the sink node via multihop wireless (using the CTP protocol) with a period of 10 s. The beaconing interval of FTSP is also set to 10 s.

Fig. 17 shows the overhead of ART with FTSP in terms of the number of transmissions and the radio duty cycle for an experiment duration of 2 h. We can see that ART has a much smaller overhead than FTSP since it does not generate additional traffic for time synchronization. Therefore, the passive approach adopted by ART is suitable for deployed sensor networks for its low overhead and high accuracy.

Fig. 18 shows the accuracy with increasing hop count. We can see that ART achieves a higher accuracy than FTSP. Each FTSP node broadcasts timesync beacons with a fixed interval. For a node far away from the synchronization root, the synchronization errors amplify due to relatively large sojourn time at each node. This problem is largely mitigated by PulseSync, which distributes information on clock values as fast as possible [33]. We can see that PulseSync can achieve the highest accuracy with errors less than 25 μs even at hop count of 10. We think that the main benefit of ART lies in its low overhead: Unlike FTSP and PulseSync, it can reconstruct the global time at the sink node without periodically exchanging control-plane messages.

VI. Conclusion

In this paper, we propose ART, an accurate and robust approach for detecting and recovering invalid timestamps. We check the conformance of two packets for identifying valid packets. The valid packet identification problem can be reduced to the maximum clique problem that is NP-hard in a general graph. We thus employ a strict conformance relation, transforming the problem to the longest path problem in a directed acyclic graph that can be solved in polynomial time. We formally prove the correctness of our algorithm.

We evaluate ART in both testbed and real-world deployed systems. Evaluation results show that: 1) ART achieves a high detection accuracy with low false positive rate and low false negative rate; 2) ART achieves a high recovery accuracy of less than 2 ms on average.

Appendix

A. Proof of Theorem 1

For two valid packets, we get the following equations according to the clock model:

$$s_1 = (1+\rho)\cdot sk_1 + \beta$$
$$s_2 = (1+\rho)\cdot sk_2 + \beta.$$

Subtracting the above two equations yields

$$\Delta s = (1+\rho)\cdot \Delta sk. \tag{11}$$

Considering the drift bound, we have

$$\frac{\Delta s}{1+\rho_m} \le \Delta sk \le \frac{\Delta s}{1-\rho_m}. \tag{12}$$

However, we only have the estimated values of sk. Therefore, we need to further derive the relation between Δs and $\Delta\hat{sk}$.

According to (3), we have

$$|sk_1 - \hat{sk}_1| \le \rho_m \cdot d_1$$
$$|sk_2 - \hat{sk}_2| \le \rho_m \cdot d_2.$$

Subtracting the above two equations yields

$$|\Delta sk - \Delta\hat{sk}| \le \rho_m \cdot \Sigma d \tag{13}$$

where $\Sigma d = d_1 + d_2$.

Combining (13) with (12), we have

$$\frac{\Delta s}{1+\rho_m} - \rho_m\cdot\Sigma d \le \Delta\hat{sk} \le \frac{\Delta s}{1-\rho_m} + \rho_m\cdot\Sigma d. \tag{14}$$

Combining (13) and (2), we have:

$$|\Delta\hat{sk} - \Delta sk| \le \frac{\rho_m}{1-\rho_m}\cdot\Sigma\hat{d}.$$

Hence

$$\Delta sk - \frac{\rho_m}{1-\rho_m}\cdot\Sigma\hat{d} \le \Delta\hat{sk} \le \Delta sk + \frac{\rho_m}{1-\rho_m}\cdot\Sigma\hat{d}.$$

Combining (12), we have

$$\frac{\Delta s}{1+\rho_m} - \frac{\rho_m}{1-\rho_m}\cdot\Sigma\hat{d} \le \Delta\hat{sk} \le \frac{\Delta s}{1-\rho_m} + \frac{\rho_m}{1-\rho_m}\cdot\Sigma\hat{d}.$$

Hence, the theorem holds.

B. Proof of Theorem 2

The probability that two valid packets conform but do not strictly conform P_{fn} can be calculated by observing the corresponding ranges (see Fig. 3), i.e.,

$$P_{\text{fn}} \le \frac{2\epsilon}{\frac{\Delta s_m}{1-\rho_m} - \frac{\Delta s_m}{1+\rho_m} + 2\epsilon} = \frac{1}{1+\frac{\rho_m\Delta s_m}{\epsilon(1-\rho_m^2)}}. \tag{15}$$

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

We can see from (14) that if two packets are valid, ϵ can be as small as $\rho_m \sum d$ where $\sum d = d_1 + d_2$ denotes the real network delay (as opposed to $\frac{\rho_m}{1-\rho_m} \sum \hat{d}$ where $\sum \hat{d} = \hat{d}_1 + \hat{d}_2$ is the observed network delay). Since $\epsilon = \rho_m \sum d \leq 2\rho_m d_m$ where d_m is the maximum network delay, we can get

$$P_{\text{fn}} \leq \frac{1}{1 + \frac{\rho_m \Delta s_m}{2\rho_m d_m (1-\rho_m^2)}} = \frac{2}{2 + \frac{\Delta s_m}{d_m (1-\rho_m^2)}}. \quad (16)$$

C. Proof of Theorem 3

Since $p_2 \sim p_1 \&\& p_3 \sim p_2$, from the definition, we have

$$\frac{s_2 - s_1}{1+\rho_m} \leq \hat{sk}_2 - \hat{sk}_1 \leq \frac{s_2 - s_1}{1-\rho_m}$$
$$\frac{s_3 - s_2}{1+\rho_m} \leq \hat{sk}_3 - \hat{sk}_2 \leq \frac{s_3 - s_2}{1-\rho_m}.$$

Adding these two equations yields

$$\frac{s_3 - s_1}{1+\rho_m} \leq \hat{sk}_3 - \hat{sk}_1 \leq \frac{s_3 - s_1}{1-\rho_m}.$$

By the definition, $p_3 \sim p_1$. Therefore, the theorem holds.

D. Proof of Theorem 4

We first prove that the nodes comprising the longest path in G' form a clique in G. Assume the path is $(p_1, \ldots, p_n)$, $s_1 < \cdots < s_n$. It is easy to see that for any two packets in the path, p_i and p_j, $e = (p_i, p_j) \in E(G)$ according to the transitive relation given in Corollary 1. Therefore, the nodes in the longest path form a clique in G.

Next, we prove that the resulting clique is maximum. We show the correctness by contradiction. Assume there exists a maximum clique C' with $|C'| > |C|$ where C is the clique formed by nodes comprising the longest path. We order the vertices in C' by the corresponding local packet generation times. There must be a path from the first vertex in C' to the last vertex in C'. Therefore, there exists another path P' with length larger than $|C|$. This contradicts the assumption $|C|$ is the longest path length. Hence, the theorem holds.

E. Proof of Theorem 5

Assume the topological order of the graph is $(v_1, \ldots, v_n)$. We prove the theorem by induction.

For v_1, it is easy to see that the longest path length will be 0 since it has no incoming edges. We assume for $v \in \{v_1, \ldots v_{i-1}\}$, len[v] stores the longest path length. We will prove that for v_i, len[v] also stores the longest path length. This is true because the algorithm sets $\text{len}[v_i] = \max(\text{len}[v']) + 1$, where $v' \in \{v_1, \ldots, v_{i-1}\}$ is a predecessor of v_i in the topological order. Since $\text{len}[v']$ is the longest path length, $\text{len}[v_i]$ is also the longest path length.

REFERENCES

[1] A. Hasler, I. Talzi, C. Tschudin, and S. Gruber, "Wireless sensor networks in permafrost research—Concept, requirements, implementation and challenges," in *Proc. 9th Int. Conf. Permafrost*, 2008, pp. 669–674.

[2] X. Mao, X. Miao, Y. He, X.-Y. Li, and Y. Liu, "CitySee: Urban CO_2 monitoring with sensors," in *Proc. IEEE INFOCOM*, 2012, pp. 1611–1619.

[3] Y. Wang, M. C. Vuran, and S. Goddard, "Cross-layer analysis of the end-to-end delay distribution in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 305–318, Feb. 2012.

[4] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. ACM SenSys*, 2004, pp. 39–49.

[5] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. ACM SenSys*, 2003, pp. 138–149.

[6] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proc. ACM/IEEE IPSN*, 2009, pp. 37–48.

[7] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proc. ACM/IEEE IPSN*, 2011, pp. 73–84.

[8] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proc. ACM SenSys*, 2012, pp. 1–14.

[9] Y. Wang, Y. He, X. Mao, Y. Liu, and X. Li, "Exploiting constructive interference for scalable flooding in wireless networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1880–1889, Dec. 2013.

[10] M. Keller, L. Thiele, and J. Beutel, "Reconstruction of the correct temporal order of sensor network data," in *Proc. IPSN*, 2011, pp. 282–293.

[11] J. Wang, W. Dong, Z. Cao, and Y. Liu, "On the delay performance analysis in a large scale wireless sensor network," in *Proc. IEEE RTSS*, 2012, pp. 305–314.

[12] B. Kusy *et al.*, "Elapsed time on arrival: A simple and versatile primitive for canonical time synchronisation services," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 239–251, 2006.

[13] X. Hu, T. Park, and K. G. Shin, "Attack-tolerant time-synchronization in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2008, pp. 448–456.

[14] J. He, J. Chen, P. Cheng, and X. Cao, "Secure time synchronization in wireless sensor networks: A maximum consensus-based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1055–1065, Apr. 2014.

[15] G. Barrenetxea *et al.*, "SensorScope: Out-of-the-Box environmental monitoring," in *Proc. ACM/IEEE IPSN*, 2008, pp. 332–343.

[16] M. Ceriotti *et al.*, "Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels," in *Proc. ACM/IEEE IPSN*, 2011, pp. 187–198.

[17] Y. Liu *et al.*, "Does wireless sensor network scale? A measurement study on GreenOrbs," in *Proc. IEEE INFOCOM*, 2011, pp. 873–881.

[18] L. Mo *et al.*, "Canopy closure estimates with GreenOrbs: Sustainable sensing in the forest," in *Proc. ACM SenSys*, 2009, pp. 99–112.

[19] W. Dong, Y. Liu, Y. He, and T. Zhu, "Measurement and analysis on the packet delivery performance in a large scale sensor network," in *Proc. IEEE INFOCOM*, 2013, pp. 2679–2687.

[20] P. Levis, "Experiences from a decade of TinyOS development," in *Proc. USENIX OSDI*, 2012, pp. 207–220.

[21] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta, "A case against routing-integrated time synchronization," in *Proc. ACM SenSys*, 2010, pp. 267–280.

[22] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor networks," in *Proc. USENIX OSDI*, 2006, pp. 381–396.

[23] D. Puccinelli, M. Zuniga, S. Giordano, and P. J. Marron, "Broadcast-free collection protocol," in *Proc. ACM SenSys*, 2012, pp. 29–42.

[24] M. Lukac, P. Davis, R. Clayton, and D. Estrin, "Recovering temporal integrity with data driven time synchronization," in *Proc. ACM/IEEE IPSN*, 2009, pp. 61–72.

[25] J. Gupchup, R. Musăloiu-E, A. Szalay, and A. Terzis, "Sundial: Using sunlight to reconstruct global timestamps," in *Proc. EWSN*, 2009, pp. 183–198.

[26] J. Gupchup, D. Carlson, R. Musaloiu-Elefteri, A. S. Szalay, and A. Terzis, "Phoenix: An epidemic approach to time reconstruction," in *Proc. EWSN*, 2010, pp. 17–32.

[27] M. Keller, J. Beutel, and L. Thiele, "How was your journey? uncovering routing dynamics in deployed sensor networks with multi-hop network tomography," in *Proc. ACM SenSys*, 2012, pp. 15–28.

[28] I. Amundson, B. Kusy, P. Volgyesi, X. Koutsoukos, and A. Ledeczi, "Time synchronization in heterogeneous sensor networks," in *Proc. IEEE DCOSS*, 2008, pp. 17–31.

[29] T. Hao, R. Zhou, G. Xing, and M. Mutka, "WizSync: Exploiting Wi-Fi infrastructure for clock synchronization in wireless sensor networks," in *Proc. IEEE RTSS*, 2011, pp. 149–158.

[30] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. ACM SenSys*, 2009, pp. 1–14.

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.

[31] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness Bell Telephone Laboratories, Inc., 1979.
[32] S. Even, A. Pnueli, and A. Lempel, "Permutation graphs and transitive graphs," *J. ACM*, vol. 19, no. 3, pp. 400–410, Jul. 1972.
[33] C. Lenzen, P. Sommer, and R. Wattenhofer, "Pulsesync: An effcient and scalable clock synchronization protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 717–727, Jun. 2015.

Wei Dong (S'08–M'11) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 2005 and 2011, respectively.

He is currently an Associate Professor with the College of Computer Science, Zhejiang University. His research interests include network measurement, wireless and mobile computing, and sensor networks.

Jie Yu received the B.Eng. degree in software engineering from Shandong University, Jinan, China, in 2013.

He is currently a graduate student with Zhejiang University, Hangzhou, China. His research interests include network protocol design and wireless sensor networks.

Jiliang Wang (S'09–M'12) received the B.E. degree in computer science and technology from the University of Science and Technology of China, Hefei, China, in 2007, and the Ph.D. degree in computer science and engineering from Hong Kong University of Science and Technology, Hong Kong, in 2011.

He is currently an Assistant Professor with the School of Software and TNLIST, Tsinghua University, Beijing, China. His research interests include sensor and wireless networks, network measurement, and pervasive computing.

Xuefeng Zhang received the B.S. degree from Northeastern University, Shenyang, China, in 2011, and the Master's degree from Zhejiang University, Hangzhou, China, in 2014, both in computer science.

His research interests include sensor network protocol and wireless sensor network.

Mr. Zhang is a student member of CCF.

Yi Gao (S'09–M'15) received the B.S. degree in software engineering and Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2009 and 2014, respectively.

He is currently a Research Assistant Professor with Zhejiang University. From 2008 to 2009, he worked with the Information System College, Singapore Management University, Singapore, as an exchange student. From 2011 to 2012, he worked with McGill University, Montreal, QC, Canada, as a joint training research student. His research interests include protocols design and measurement in sensor networks.

Chun Chen (M'13) received the Bachelor of Mathematics degree from Xiamen University, Xiamen, China, in 1981, and the M.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1984 and 1990, respectively.

He is a Professor with the College of Computer Science and Director of the Institute of Computer Software, Zhejiang University. His research interests include embedded system, image processing, computer vision, and CAD/CAM.

Jiajun Bu (M'06) received the B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1995 and 2000, respectively.

He is a Professor with the College of Computer Science and the Deputy Dean of the School of Software Technology, Zhejiang University. His research interests include embedded system, mobile multimedia, and data mining.

Prof. Bu is a member of the Association for Computing Machinery (ACM).

Authorized licensed use limited to: Tsinghua University. Downloaded on February 03,2021 at 07:17:14 UTC from IEEE Xplore. Restrictions apply.