Online Protocol Verification in Wireless Sensor Networks via Non-intrusive Behavior Profiling

Yangfan Zhou[1,2], Xinyu Chen[1], Michael R. Lyu[1,2], and Jiangchuan Liu[3]

[1] Shenzhen Research Institute, The Chinese U. of Hong Kong, Shenzhen, China
[2] Dept. of Comp. Sci. and & Eng., The Chinese U. of Hong Kong, Shatin, Hong Kong
[3] School of Computing Sci., Simon Fraser U., Burnaby, BC, Canada

Abstract. Wireless communication protocols are centric to Wireless Sensor Network (WSN) applications. However, WSN protocols are prune to defects, even after their field deployments. A convenient tool that can facilitate the detection of post-deployment protocol defects is of great importance to WSN practitioners. This paper presents `Probe-I` (sensor network `Pro`tocol `be`havior `I`nspector), a novel tool to obtain, visualize, and verify the behaviors of WSN protocols after their field deployments. `Probe-I` collects the protocol behaviors in a non-intrusive manner, *i.e.*, via passively listening to the packet exchanges in the target network. Then with a role-oriented behavior modeling approach, `Probe-I` models the protocol behaviors node by node based on the sniffed packets, which well reflects how the target protocol performs in each node. This allows the WSN practitioners to readily see if the target protocol behaves as intended by simply verifying the correctness of the behavior metrics in a simple, baseline test. Finally, the verified metrics allow `Probe-I` to automatically check the protocol behaviors from time to time during the network lifetime. The suggested behavior discrepancy can unveil potential protocol defects. We apply `Probe-I` to verify two WSN data collection protocols, and find their design defects. It shows that `Probe-I` can substantially facilitate WSN protocol verification.

1 Introduction

Wireless communication protocols are centric to wireless sensor networks (WSNs) in reporting the physical information of interest. The successful application of a WSN largely relies on whether its protocols can work as intended. However, recent publications have reported that various protocol defects are frequently encountered in field deployments, leading to their failures [1,2]. Trustworthy protocol remains a critical concern towards the extensive deployments of WSNs. Unfortunately, discovering protocol defects after deployment is a very challenging task. It is hard to identify the subtle symptoms of a defect before it causes notable problems that may lead to fatal system failures.

This paper presents `Probe-I` (sensor network `Pro`tocol `be`havior `I`nspector), a novel tool to unveil post-deployment protocol defects in WSNs. A WSN practitioner can load `Probe-I` into a mobile device and carry it to the deployment field.

X. Wang et al. (Eds.): WASA 2012, LNCS 7405, pp. 100–111, 2012.

`Probe-I` can then profile the runtime of a protocol, learn its behavior models, and automatically produce alarms when suspicious protocol defect symptoms are found. To this end, `Probe-I` incorporates two key components: a *non-intrusive* mechanism to collect the protocol runtime data and accurately model the protocol behaviors, and an anomalous behavior detection approach that can identify protocol defect symptoms from the tremendous behavior data.

Key to `Probe-I` in modeling protocol behaviors is that the packet exchanging profiles of a protocol can well reflect its behaviors, since packet exchanging is centric to WSN protocols in nature. Hence, leveraging on the broadcasting nature of wireless communications, `Probe-I` equips a wireless interface (*e.g.*, that on a compatible sensor node) compatible with that adopted in the target network, and passively eavesdrops the packets in the air. A profiling approach specifically tailored for WSN protocols is then employed to model the behaviors of the target protocol based on the sniffed packets. Thus, unlike instrumentation-based tools (*e.g.*, EnviroLog [3] and Declarative Tracepoints [4]) that will inevitably intrude the executions of the target protocol, `Probe-I` requires no modifications to both the software and hardware of the target sensor nodes. Most importantly, it will not alter the original executions of the target protocol. This provides it nice fidelity and no overhead in capturing the protocol behaviors.

Although the protocol behaviors can be profiled, manually inspecting the data to identify the potential defect symptoms becomes a daunting task, which may be extremely labor intensive. `Probe-I` addresses this challenge with a two-step approach. First, it allows a WSN practitioner to perform a baseline test, where the protocol behaviors are easy to be verified. After the correctness of the protocol is confirmed in the baseline test, `Probe-I` saves the *verified* behavior data to the mobile device. During the system runtime, the WSN practitioner can from time to time bring the device again to the network field. The newly collected protocol behavior data will be compared with the verified data obtained in the baseline test. The discrepancy of the two set of data indicates suspicious protocol behaviors. `Probe-I` will then issue an alert, suggesting further inspection of the protocol implementation.

The rest of the paper is organized as follows. We overview the design of `Probe-I` in Section 2. Section 3 elaborates how `Probe-I` collects the protocol runtime data of a WSN protocol and models its behaviors. In Section 4, we discuss the details on detecting defect symptoms in `Probe-I`. Two case studies are provided in Section 5 to demonstrate the effectiveness of `Probe-I`. Section 6 presents the related work. We conclude this paper in Section 7.

2 Overview of `Probe-I`

Figure 1 shows the concept of our mobile device assisted non-intrusive approach, namely, `Probe-I`, to discover post-deployment protocol defects. The packet exchanges of the target WSN can be eavesdropped with a compatible wireless interfacing device, namely, a *sniffer*, connected to a more powerful mobile device (*e.g.*, a tablet computer). A convenient choice of such a sniffer is a compatible

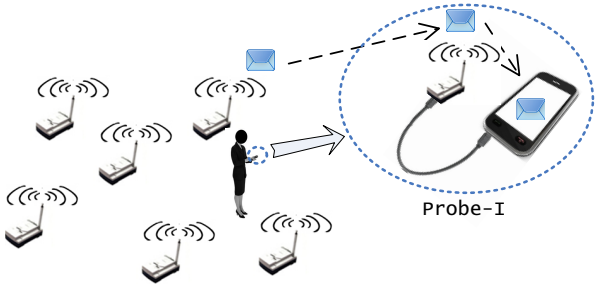

Fig. 1. Probe-I concept

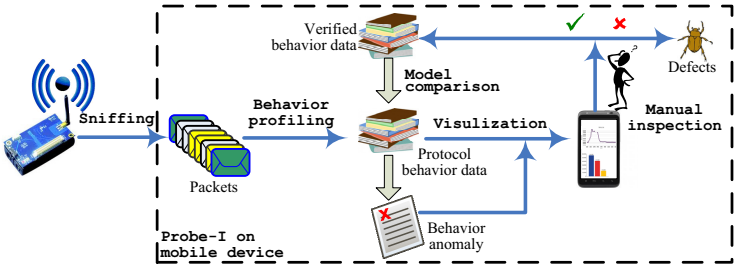

Fig. 2. System overview of Probe-I

sensor node. Exploiting the computational and visualization capabilities of the mobile device, Probe-I models and shows the protocol behaviors. Suspicious behaviors can be detected, which help discover post-deployment protocol defects.

Such a new conceptual design is feasible with the recent advancement in mobile computing. For example, the current version of Android [5] enables mobile devices to connect through a USB (Universal Serial Bus) cable to a peripheral device via the on-the-go (OTG) mode or the host mode. It is convenient to connect a mobile device to a sniffer to obtain the packets it has captured.

Figure 2 overviews the Probe-I design. Probe-I can collect the protocol behaviors in a simple test scenario and plot them. A WSN practitioner can then readily examine whether the protocol behaves as intended. We call such a verification process a *baseline test*. If the correctness of the protocol in the baseline test is verified, the behavior data can be saved in the mobile device for further verification processes, namely, the *runtime tests*: During the network lifetime, the WSN practitioner can from time to time carry the mobile device into the network field to verify the protocol. Each time when a node is accessed, Probe-I can collect its behaviors, and compare them with those collected in the baseline test. The discrepancy means that the protocol behaviors are different from the baseline test unexpectedly, which, as a result, indicates potential defect symptoms. Hence, such discrepancy will be shown to the WSN practitioner.

Next, we will discuss how Probe-I collects and models the protocol behaviors in Section 3, and how Probe-I finds behavior discrepancy in Section 4.

3 Protocol Behavior Profiling and Modeling

3.1 Profiling Protocol Behaviors

When a protocol defect is triggered, it will change the correct behaviors of the protocol, resulting in a malfunction or a performance degradation. Since packet exchanges are centric to a protocol, the malfunction or performance degradation of a protocol will generally cause the packet exchanging behaviors to deviate from the normal. Examples include packet loss, large packet delay, and low packet throughput. Hence, we can verify the protocol via a "black box" approach, *i.e.*, by monitoring the packet exchanging behaviors.

Note that it is possible that the sniffer successfully receives a packet intended to a node, while the node *per se* fails to receive the packet, or *vice versa*. Such inconsistency will make the sniffer get distorted knowledge of the protocol. To avoid it, `Probe-I` focuses on one node at a time (namely, the *target node* u) by putting the sniffer close to u. Thus, it can obtain high-fidelity packet receiving events of the target node. In this way, `Probe-I` observes the behaviors of the protocol running on u by monitoring the packets that involve u (*i.e.*, those intended for u and those sent by u).

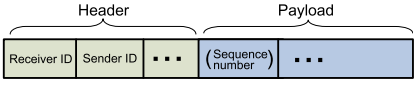

Fig. 3. Typical packet structure of WSNs

To do this, `Probe-I` should be packet content-aware. It obtains the sender and receiver information by analyzing the packet header. Figure 3 shows the typical packet structure of WSNs, where a packet p consists of a header and a payload. Let $p.data$ denote the payload. In the header, the *sender ID* is the node that sends the packet (*i.e.*, the *sender*). The *receiver ID* is its intended recipient (*i.e.*, the *receiver*). When a node sends or relays a packet, it will update the *sender ID* to its own ID, and the *receiver ID* to its next-hop neighbor. Let $p.src$ and $p.dest$ denote the sender and receiver of p. Note that such a packet structure is generally adopted in typical WSN protocols. For example, the *Active Message* packet format bears such a structure, which is generally used in the protocols for TinyOS applications [6] (*e.g.*, Collection Tree Protocol (CTP) [7]).

Provided the packet format information, `Probe-I` can then parse the packets it has eavesdropped during t. For a sniffed packet p, if either $p.src$ or $p.dest$ is node u, `Probe-I` will save p (together with the capturing time, denoted by $p.time$). Thus, during the monitoring period t, `Probe-I` can obtain a sequence of packets that are sent to or sent by u in a chronological order of their capturing time. Let $\mathcal{P}_t(u)$ denote such a sequence. `Probe-I` then models the protocol behaviors running on u during t based on $\mathcal{P}_t(u)$, which is illustrated next.

3.2 Role-Oriented Protocol Behavior Modeling

Data packet flows in a WSN typically follow two types, data collection and data dissemination. The former is generally for obtaining the readings from the sensor nodes, while the latter for distributing information to the sensor nodes. Considering the major purpose of WSNs is typically for obtaining the sensor readings, we focus on modeling the behaviors of data collection protocols in this paper. Data dissemination can be deemed as the reverse traffic of data collection, and therefore can be modeled with a similar approach.

There are three kinds of nodes involved in a typical data collection protocol, specifically, *source*, *sink*, and *relay*, as shown in Figure 4. A source node generates a packet (*e.g.*, a packet carrying the sensor readings of the node), a sink node is the intended final destination of the packet, and a relay is an intermediate node that helps forward the packet to its next-hop neighbor towards the sink.

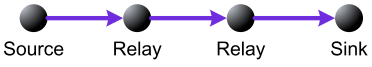

Fig. 4. A simple scenario where three roles of nodes are shown

`Probe-I` identifies that a target node u is a source if it captures packets from u which are not previously received by u. We say such a packet p is *generated* by node u. A target node u is a sink if `Probe-I` captures packets intended for u which will not again be sent out by u. We say such a packet p is *collected* by node u. Finally, `Probe-I` can know that a target node u is a relay if it captures packets intended for u which is again sent out by u. We say such a packet p is *relayed* by node u. The above notions are formally described as follows.

source - if $\exists\ p \in \mathcal{P}_t(u)$ with $p.src = u$ and $\nexists\ q$ with $q.dest = u$ and $q.data = p.data$
sink - if $\exists\ p \in \mathcal{P}_t(u)$ with $p.dest = u$ and $\nexists\ q$ with $q.src = u$ and $q.data = p.data$
relay - if $\exists\ p$ and $q \in \mathcal{P}_t(u)$ with $p.dest = u$, $q.src = u$, and $q.data = p.data$

Naturally, nodes with different roles (*i.e.*, source, sink, or relay) have different protocol behavior specifics, which should be modeled separately. What follows discusses our role-oriented behavior modeling considerations.

1) Source: For a source node, an important consideration is the number of packets it has generated in a given period of time. In this regard, `Probe-I` divides the monitoring period into many disjoint time intervals, each with a fix length τ. Then it considers the packets g_i $(i = 1, 2, ..., \lfloor \frac{t}{\tau} \rfloor)$ generated by u in every time interval as the metrics that reflect the behaviors of the protocol running at u.

2) Sink: For a sink node, the number of packets it can collect in a given period of time is an important parameter. Hence, similarly, `Probe-I` also divides the monitoring period into disjoint time intervals, each with length τ. Then it considers the packets c_i $(i = 1, 2, ..., \lfloor \frac{t}{\tau} \rfloor)$ collected by u in every time interval as the metrics that reflect the protocol behaviors at u.

Moreover, if packet sequence number is available in the packet structure (see Figure 3), it is then feasible to check the end-to-end packet loss rate. Again, packet loss rate is measured in each of the intervals with length τ.

3) Relay: Critical to a packet relay process is how long a packet has been staying in the relay node. This indicates the hop-by-hop delay, and contributes in sum to the end-to-end delay of the packet. Therefore, `Probe-I` obtains the time between when a packet arrives at node u and when the packet leaves the node. Specifically, consider packets p and q in $\mathcal{P}_t(u)$ with $p.dest{=}q.src{=}u$ and $q.data{=}p.data$. Then the relay delay is $q.time\text{-}p.time$. `Probe-I` uses such delays of all forwarded packets in t to model the protocol behaviors in t.

Note that it is possible that a packet p may be resent if the packet cannot be successfully delivered. Hence, in the above considerations, q is the packet with the largest $q.time$ in all packets with source field src identical to u and payload field $data$ identical to $p.data$. In other words, we only consider the last (successful) relay attempt of a packet.

Moreover, to describe such *retransmissions*, for the source and the relay nodes, `Probe-I` records the number of transmission attempts for each packet being sent. This metric can capture the link quality.

Finally, `Probe-I` also measures the *protocol overhead* for all nodes. Specifically, it divides the monitoring period into disjoint time intervals, each with length τ, and calculates the ratio between the number of data packets and that of control packets in each interval.

4 Detecting the Anomalous Protocol Behaviors

Now we discuss how `Probe-I` compares the verified behaviors (*i.e.*, those collected in the baseline test) with those collected in a runtime test. The protocol behaviors of a node may change in two aspects, *i.e.*, role or behavior metrics discussed in Section 3. We illustrate them as follows.

4.1 Role Changes

We consider the role change of a node because it may reflect dramatic changes of the protocol behaviors. For example, when a node recognized as a relay in the baseline test is found to be a sink in a runtime test, it means that the node has not relayed some received packets as it should have done. This indicates unexpected packet drops for the relay node. Hence, such a model violation should be presented to the WSN practitioner.

Finally, note that it is straightforward to detect the role change of a node with the role identification approach described in Section 3.2.

4.2 Discrepancy in Behavior Data

`Probe-I` compares the protocol behaviors in terms of their data distributions. Specifically, given each protocol behavior metric, we suggest that for the two

sets of corresponding behavior data $\mathcal{B}$ and $\mathcal{R}$ collected in the baseline test and the runtime test respectively, their distributions should be compared. If their distributions have no significant difference, `Probe-I` considers that the protocol behaviors in the runtime test is similar to those in the baseline test. As a result, the protocol functions correctly in the runtime test.

`Probe-I` detects the discrepancy of the behavior data in $\mathcal{B}$ and $\mathcal{R}$ with a statistical hypothesis test approach as follows. First, `Probe-I` assigns the samples in $\mathcal{B}$ into k different bins according to their values. We consider two cases: 1) The data of the performance metric are continuous; 2) They are discrete. The relay delay is an example of the first case, while the transmission times is an example of the second case.

1) The data are continuous: Suppose b_{max} and b_{min} are the samples with largest and smallest values in $\mathcal{B}$ respectively. Then, $(-\infty, +\infty)$ is divided into k intervals, where $[b_{min} + \frac{b_{max}-b_{min}}{2(k-1)}, b_{max} - \frac{b_{max}-b_{min}}{2(k-1)}]$ is divided into k-2 intervals with equal size $\frac{b_{max}-b_{min}}{k-1}$, and two tail intervals are $(-\infty, b_{min} + \frac{b_{max}-b_{min}}{2(k-1)}]$ and $(b_{max} - \frac{b_{max}-b_{min}}{2(k-1)}, +\infty)$. A bin is then assigned to each interval. A sample is put into a bin if the value of the sample falls into the corresponding interval of the bin. Figure 5(a) shows an example of how to divide $(-\infty, +\infty)$ into k=5 intervals, while Figure 5(b) shows the resulting bins corresponding to the intervals.

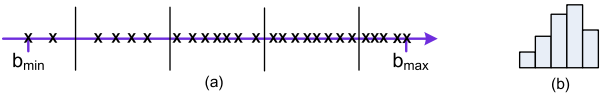

Fig. 5. An example showing how to put the samples (each denoted by a 'x') into 5 bins

2) The data are discrete: Suppose there are k different values of the samples in $\mathcal{B}$. A bin is then assigned to each value. A sample is put into a bin if the value of the sample is equal to the corresponding value of the bin.

Thus, for either case, all the samples in $\mathcal{B}$ can be put into the bins. Let B_1, B_2, ..., B_k denote the number of samples in the bins respectively. Then, for all the samples in $\mathcal{R}$ collected in the runtime test, they will also be put into k bins, according to the same value intervals as those for $\mathcal{B}$ (for the continuous data case), or according to the same values as those for $\mathcal{B}$ (for the discrete data case). Let R_1, R_2, ..., R_k denote the number of samples in the bins respectively.

Note that for the discrete-data case, it is possible that a sample in $\mathcal{R}$ cannot be put into any of the k bins. In other words, its value does not match the values of any samples in $\mathcal{B}$. In this case, such a sample is an outlier. This indicates an anomaly in the protocol behaviors in the runtime test, comparing with the baseline test. Hence, `Probe-I` will report such discrepancy in behavior data immediately and suggest further inspection of the protocol implementation.

Otherwise (*i.e.*, all the samples in $\mathcal{R}$ can be put into the bins), let us suppose the number of the data samples in $\mathcal{R}$ is n. If the data samples in $\mathcal{R}$ follow the distribution of the data samples in $\mathcal{B}$, the *expected* number of the samples in each bin i, denoted by $\overline{R}_i$, should be:

$$\overline{R}_i = \frac{B_i}{\sum_{i=1}^{k} B_i} \cdot n \tag{1}$$

If the expected number of samples in any bin in either tail is less than 5, the bin is pooled with a neighboring bin, until the count in each extreme bin is at least 5. Suppose in the end, there are m resulting bins. Let R'_1, R'_2, ..., R'_m denote the numbers of samples in $\mathcal{R}$ that are put in the m bins respectively, and $\overline{R}'_1, \overline{R}'_2, ..., \overline{R}'_m$ denote the expected numbers of samples in $\mathcal{R}$ that should be in the bins respectively.

`Probe-I` then adopts Pearson's chi-squared (χ^2) hypothesis test to test the goodness of fit of the two sets of data (*i.e.*, $[R'_1, R'_2, ..., R'_m]$ and $[\overline{R}'_1, \overline{R}'_2, ..., \overline{R}'_m]$) in terms of their distributions [8]. Its *null hypothesis* is that the distribution of $[R'_1, R'_2, ..., R'_m]$ is consistent with the expected distribution, *i.e.* that of $[\overline{R}'_1, \overline{R}'_2, ..., \overline{R}'_m]$, while the *alternative hypothesis* is that it is not.

For the χ^2 test, the value of the test-statistic is calculated as [8]:

$$\chi^2 = \sum_{i=1}^{m} \frac{(R'_i - \overline{R}'_i)^2}{\overline{R}'_i} \tag{2}$$

The χ^2 statistic can then be used to calculate a *p-value* by comparing the value of the statistic to a χ^2 distribution with the number of degrees of freedom equal to m-1. The p-value represents the uncertainty in the claim that the null hypothesis is false. `Probe-I` considers that when the p-value is larger than 0.1, a conventional significance level threshold, the null hypothesis will not be rejected. In other words, we will consider that the protocol behaviors in the baseline test are different from those in the runtime test if the difference in the distributions of $\mathcal{B}$ and $\mathcal{R}$ is statistically significant, *i.e.*, the probability that the alternative hypothesis is true is larger than 90%. In this case, `Probe-I` will output such a protocol behavior anomaly detected in the runtime test, and suggest further inspection of the protocol implementation.

Finally, note that the statistical hypothesis test is non-parametric, which does not require any *a priori* knowledge of the distribution. This fits our problem domain since we do not know how a performance metric should actually distribute for the target protocol.

5 Evaluation

To show the effectiveness of `Probe-I` in modeling and verifying WSN protocols, we provide two representative case studies in this section. We will examine how protocol behavior anomalies caused by design defects can be conveniently identified with `Probe-I`. The target WSN protocols in these two case studies are based on the codes distributed with TinyOS [6].

We implement `Probe-I` with Java. Such a platform independent implementation makes it convenient to port it to various mobile devices. In our experimental studies, for convenience consideration, we use a laptop computer as the mobile device and a sensor node as the sniff to eavesdrop the packets exchanged in our target WSNs. The computer is connected with the sniff with a USB cable.

5.1 Case Study I: Data Forwarding

In our first case study, we verify a lightweight multi-hop packet forwarding protocol based on `BlinkToRadio` distributed with TinyOS [6]. The target WSN contains three sensor nodes. The correct behaviors of the target protocol are simple: Node 2 will generate 25 packets per second, and send each packet to node 1. Node 1, upon receiving a packet from node 2, will forward the packet immediately to node 0. Node 0 will collect the packets intended for itself. In other words, node 0 is a sink, node 1 is a relay, and node 2 is a source.

After the network is deployed, we perform a baseline test. The sniff is put close the three nodes one by one. Since the baseline test should be a simple verifiable test, for each target node, the monitoring period is short (nearly 30 seconds). `Probe-I` then collects the packets sent to or sent by each target node, and models the protocol behaviors running on each node. τ is set 1 second. `Probe-I` then identify the role of each node successfully. For each node, the corresponding behavior metrics are also consistent with our design purpose. Hence, the correctness of the protocol is confirmed in the baseline test.

We then again access the network with `Probe-I` to perform a runtime test. This time we let `Probe-I` monitor each node for a longer period of time (nearly 3 minutes). `Probe-I` finds no behavior anomaly for the sink node and the source node. Also, for the relay node, `Probe-I` does not find the data of the behavior metrics (*i.e.*, the relay delay and the transmission attempts) collected in both tests have significant discrepancy. However, it issues an alert showing that node 1 has two roles: a new but unintended role *sink*, in addition to its designed role *relay*. This is obviously a fault, which means node 1 must have received some packets and have not sent them out.

We then inspect the protocol implementation for the relay node. Starting from the codes that handle a packet receiving event, we instantly find that a received packet can be actively dropped in function `AMSend.send` due to a busy flag. The flag is set when the node is in the process of sending a packet. This means before a previously-received packet has been sent, another packet arrives unexpectedly, causing the protocol to drop the new arrival packet. To correct this fault, the protocol should employ a buffer to cache packets until the previously-received packet has been sent.

Note that such a fault is only triggered occasionally, and causes an occasional packet loss, which tends to be neglected. We have shown that `Probe-I` can however effectively model the protocol behaviors and successfully detect such a subtle protocol defect via identifying role changes of sensor nodes.

5.2 Case Study II: Collection Tree Protocol

In this case study, we test a more sophisticated routing protocol CTP (Collection Tree Protocol) [9]. CTP is frequently employed to transfer sensor readings to sinks. We intend to examine whether CTP performs well in mobile networks. The target WSN contains four sensor nodes. One is the sink, while three are sources that will generate one packet per second. The packets from the sources

will be conveyed to the sink possibly via other sensor nodes. Hence, a source node may also serve as a relay. Since the packet rate is low, we set τ 10 seconds in our tests, and let `Probe-I` monitor each sensor node for nearly 3 minutes.

In our baseline test, all nodes are stationary. `Probe-I` correctly identifies the node roles. The behavior metrics are also correct. The baseline test has passed.

We then start a runtime test, where the sink and two nodes close to the sink are stationary (which may serve as relays) and the rest one node is mobile. CTP should be able to find another relay when the previous relay for the mobile node cannot be reached due to its mobility. `Probe-I` reports that the distributions of the number of packet transmission attempts and the overhead are inconsistent with the baseline test. In particular, `Probe-I` shows that the number of packet transmission attempts has a new value 30, *i.e.*, some packets are retransmitted for 30 times. By inspecting the CTP design, we find that a route is considered broken only when a packet cannot be successfully transmitted after 30 attempts. This may not be proper for mobile networks.

`Probe-I` reveals that CTP may not be a good choice for mobile networks, since routes may be reestablished frequently, incurring larger overhead and transmission failures (and packet loss). The route reestablishing procedure should be improved to cope with node mobility, which confirms the findings in [10].

This case study demonstrates how `Probe-I` can greatly facilitate the verification of a protocol in a new network scenario. Note that without such a tool, it would be quite labor-intensive to manually inspect the design of CTP to justify whether it is applicable in the new mobile scenario.

6 Related Work

Various research efforts have been put to enhance the reliability of WSN protocols. Many techniques, including simulation-based testing and troubleshooting approaches, network monitoring mechanisms, and debugging tools are proposed, which are surveyed in what follows.

TOSSIM [11] and Avrora [12] are two widely-adopted simulation tools for WSNs. Before field-deploying a protocol, WSN practitioners can resort to such simulation platforms to confirm its correct behavior. But a comprehensive simulation will generate tremendous protocol behavior data. Verifying the correctness of the protocol behavior largely depends on manual efforts, which is labor-intensive. T-Check [13] and KleeNet [14] are two simulation-based approaches that can find WSN bugs by exploring program states extensively. Sentomist [15] locates bug symptoms via finding outliers in application behaviors collected via simulations. However, high-fidelity simulation remains difficult, given the complexity of the real world and the unexpected working scenarios [16]. As a result, protocol defects may still escape from being detected in simulation platforms. `Probe-I`, in contrast, focuses on detecting protocol defects in deployed networks.

SNTS [17] deploys many additional sensor nodes in the target network field to collect the packets of the target WSN. These sensor nodes have to be collected manually to retrieve their collected packets. It only suits small experimental

network. PAD [18] attaches logs in regular data packets to help the base station diagnose network problems. The logging and piggyback mechanisms will inevitably disturb the original protocol behaviors.

Sympathy [19] introduces a diagnosis agent in the target sensor nodes to collects their run-time data, and transmitting them to the base station. Tools based on instrumentation (*e.g.*, EnviroLog [3] and Declarative Tracepoints [4]) have also been proposed to log the protocol behaviors in a sensor node during its runtime. PDA [20] inserts state hypotheses into the WSN codes. If they do not hold during runtime, alerts can be issued. These tools can help detect protocol defects. However, as real-time systems, WSN programs are sensitive to timing. Running on the same hardware, such behavior data collection and verification mechanisms will inevitably intrude the executions of the original codes. As a result, a defect may hide when such a mechanism turns on, but can still be triggered when it is disabled. Hence, these tools are still not adequate to eliminating protocol defects, not to mention the human efforts in reprogramming an existing WSN application to incorporate such tools. `Probe-I` avoids such inadequacy via a non-intrusive protocol behavior data collection approach.

7 Conclusion

This paper presents `Probe-I` (sensor network `Pro`tocol `be`havior `I`nspector), a mobile device assisted tool that can unveil post-deployment protocol defects in WSNs. `Probe-I` collects the protocol behaviors by passively listening to the packet exchanges with a mobile device equipped with a compatible wireless interface (*i.e.*, a USB-connected sensor node). Such a behavior data collection mechanism is non-intrusive, *i.e.*, it will not change the execution of the original WSN protocols. This can provide `Probe-I` nice fidelity in obtaining the real protocol behaviors, since the executions of WSN codes are sensitive to timing.

`Probe-I` then employs a node-by-node role-oriented approach to model the protocol behaviors based on the sniffed packets. Hence, it focuses on the protocol defects that can cause packet exchanges to deviate from the normal. With `Probe-I`, a WSN practitioner can verify whether a protocol works as well in field as it does in a simple baseline test. It can illustrate potential protocol defect symptoms, *i.e.*, the behavior discrepancy found in a runtime test. This can facilitate manual inspection of the protocol implementation to locate the root cause of such discrepancy. We successfully employs `Probe-I` to detect protocol design defects in two WSN data collection protocols, which shows its effectiveness.

Acknowledgements. This work was substantially supported by the National Natural Science Foundation of China (Project No. 61100077), the National Basic Research Program of China (973 Project No. 2011CB302603), the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Nos. CUHK 415311 and N_CUHK405/11). J. Liu's work was supported by a Canadian NSERC Discovery Grant, a Discovery Accelerator Supplements Award, an NSERC Engage Grant, a MITACS Project Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008).

References

1. Barrenetxea, G., Ingelrest, F., Schaefer, G., Vetterli, M.: The hitchhiker's guide to successful wireless sensor network deployments. In: Proc. of ACM SenSys (2008)
2. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and yield in a volcano monitoring sensor network. In: Proc. of OSDI (2006)
3. Luo, L., He, T., Zhou, G., Gu, L., Abdelzaher, T.F., Stankovic, J.A.: Achieving repeatability of asynchronous events in wireless sensor networks with EnviroLog. In: Proc. of the IEEE INFOCOM (2006)
4. Cao, Q., Abdelzaher, T., Stankovic, J., Whitehouse, K., Luo, L.: Declarative tracepoints: A programmable and application independent debugging system for wireless sensor networks. In: Proc. of ACM SenSys (2008)
5. Google Inc.: Android operating system, `http://www.android.com`
6. TinyOS Community Forum: TinyOS: An open-source OS for the networked sensor regime, `http://www.tinyos.net`
7. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proc. of the ACM SENSYS, pp. 1–14 (November 2009)
8. Greenwood, P.E., Nikulin, M.S.: A Guide to Chi-Squared Testing. Wiley (1996)
9. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proc. of ACM SenSys (2009)
10. Chipara, O., Lu, C., Bailey, T.C., Roman, G.C.: Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In: Proc. of ACM SenSys (2010)
11. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire tinyos applications. In: Proc. of the ACM SenSys (2003)
12. Titzer, B., Lee, D., Palsberg, J.: Avrora: Scalable sensor network simulation with precise timing. In: Proc. of the IEEE IPSN, pp. 477–482 (May 2005)
13. Li, P., Regehr, J.: T-Check: Bug finding for sensor networks. In: Proc. of IPSN (2010)
14. Sasnauskas, R., Landsiedel, O., Alizai, M.H., Weisez, C., Kowalewskiz, S., Wehrle, K.: KleeNet: Discovering insidious interaction bugs in wireless sensor networks before deployment. In: Proc. of the ACM/IEEE IPSN (2010)
15. Zhou, Y., Chen, X., Lyu, M., Liu, J.: Sentomist: Unveiling transient sensor network bugs via symptom mining. In: Proc. of the IEEE ICDCS (2010)
16. Stojmenovic, I.: Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. IEEE Comm. 46(12), 102–107 (2008)
17. Khan, M.M.H., Luo, L., Huang, C., Abdelzaher, T.: SNTS: Sensor network troubleshooting suite. In: Proc. of the IEEE DCOSS (2007)
18. Liu, K., Li, M., Liu, Y., Li, M., Guo, Z., Hong, F.: Passive diagnosis for wireless sensor networks. In: Proc. of the ACM SENSYS (2008)
19. Ramanathan, N., Chang, K., Kapur, R., Girod, L., Kohler, E., Estrin, D.: Sympathy for the sensor network debugger. In: Proc. of the ACM SENSYS (2005)
20. Rmer, K., Ma, J.: PDA: Passive distributed assertions for sensor networks. In: Proc. of IPSN (2009)