

SECURE CONTROL PROTOCOLS FOR  
RESOURCE-CONSTRAINED EMBEDDED SYSTEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jin Kyu Koo

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2012

Purdue University

West Lafayette, Indiana

This dissertation is dedicated to my family.

## ACKNOWLEDGMENTS

First and foremost, I would like to express sincere gratitude to my academic advisors, Prof. Saurabh Bagchi and Prof. Xiaojun Lin. Prof. Bagchi always treated me as a colleague, and pushed me to try more meaningful work with full trust in my capability. His encouragement and constructive comments were the cure for several obstacles during Ph.D. study. The passion that he showed me at nights when we sat together and worked intensely to finish a paper would be the one that I will never forget in my research career. Prof. Lin gave me a chance of training myself to think more rigorously throughout my Ph.D. study. His keen insight was a huge help to point out my mistakes and to find the first step to a solution in various research activities. I am sure that his guidance upgraded my state of maturity as a researcher.

I also want to thank my committee members, Prof. Sonia Fahmy and Prof. Ninghui Li for taking time to serve in my committee despite their busy schedules. I appreciate all their valuable comments and helpful criticisms.

This dissertation would never been finished without my dear family's unconditional support. I would like to give my deepest gratitude to each and every member in my family. Special thanks go to my wife, Jiwon and my son, Jonghyun. I appreciate Jiwon's support in taking this journey with me. Jonghyun's smile was the best remedy against every down moment.

Last but not least, I would like to thank the Korea Institute for Advancement of Technology (KIAT) for providing the financial support in part for my Ph.D. work.

## TABLE OF CONTENTS

	Page
LIST OF TABLES . . . . .	vii
LIST OF FIGURES . . . . .	viii
ABSTRACT . . . . .	xi
1 INTRODUCTION . . . . .	1
1.1 Embedded Systems and Their Security . . . . .	1
1.2 Summary of Contributions . . . . .	2
1.3 Document Outline . . . . .	4
2 RELATED WORK . . . . .	5
2.1 Clock Synchronization in Sensor Networks . . . . .	5
2.2 Timely Event Reporting in Sensor Networks . . . . .	8
2.3 Privacy Protection and Cost Saving in Smart Grids . . . . .	9
3 CLOCK SYNCHRONIZATION IN SENSOR NETWORKS . . . . .	11
3.1 CSOnet . . . . .	13
3.1.1 CSOnet Architecture . . . . .	13
3.1.2 CSOnet Hardware . . . . .	16
3.2 Can FTSP be Used to Synchronize CSOnet? . . . . .	19
3.3 Proposed Protocol . . . . .	23
3.3.1 Operational Scenario . . . . .	23
3.3.2 Synchronization Protocol . . . . .	25
3.3.3 Failure Handling . . . . .	28
3.3.4 Packet Sequences and State Management . . . . .	29
3.3.5 Fast Recovery . . . . .	30
3.3.6 Choice of Important Parameters . . . . .	31
3.4 Experiments . . . . .	32

	Page
3.4.1	Experimental Methodology . . . . . 32
3.4.2	Network-wide Synchronization Time . . . . . 34
3.4.3	Synchronization Error . . . . . 38
3.5	Discussion . . . . . 40
4	TIMELY EVENT REPORTING IN SENSOR NETWORKS . . . . . 44
4.1	Problem Statement . . . . . 46
4.2	Straw-Man Protocols . . . . . 49
4.3	Proposed protocol: SEM . . . . . 53
4.3.1	Detail of SEM . . . . . 54
4.3.2	Overhead Analysis . . . . . 61
4.4	Miscellaneous Issues . . . . . 62
4.5	Experiments . . . . . 64
5	PRIVACY PROTECTION AND COST SAVING IN SMART GRIDS . . . . . 68
5.1	System Model . . . . . 70
5.2	Solution Approach I: Basic Formulation . . . . . 72
5.2.1	Mapping between $X(n)$ and $Y(n)$ . . . . . 72
5.2.2	Strategy for charging/discharging the battery . . . . . 74
5.2.3	Basic approach . . . . . 75
5.2.4	Simulation study for the basic approach . . . . . 78
5.3	Solution Approach II: Advanced Formulation . . . . . 80
5.3.1	Advanced approach: PRIVATUS . . . . . 80
5.3.2	Simulation study for PRIVATUS . . . . . 83
5.4	Per-day Energy Usage Flattening . . . . . 84
5.4.1	Total energy usage different across days . . . . . 84
5.4.2	Solution summary . . . . . 85
5.4.3	Virtual battery state . . . . . 86
5.4.4	Battery capacity . . . . . 87
5.4.5	Per-day usage flattening algorithm . . . . . 88

	Page
5.4.6 Simulation study for per-day usage flattening . . . . .	90
5.4.7 Effects on cost saving . . . . .	92
5.5 Experiment . . . . .	93
5.5.1 Metrics and simulation parameters . . . . .	93
5.5.2 Information leakage and cost saving . . . . .	95
5.6 Discussion . . . . .	102
6 CONCLUSION . . . . .	105
7 FUTURE WORK . . . . .	108
7.1 Tree topology creation in HARMONIA . . . . .	108
7.2 SEM deployment into a testbed . . . . .	108
7.3 Handling practical issues in PRIVATUS . . . . .	109
LIST OF REFERENCES . . . . .	110
VITA . . . . .	114

## LIST OF TABLES

Table	Page
3.1 Values of parameters in HARMONIA used in the experiments. . . . .	33
3.2 Slopes of the linear relationship between network-wide synchronization time and synchronization period observed in our experiments. . . . .	38
3.3 One-hop synchronization error. . . . .	38

## LIST OF FIGURES

Figure	Page
3.1 South Bend Interceptor Sewer and CSO Diversion Structure. . . . .	15
3.2 CSOnet’s Hierarchical Architecture. . . . .	15
3.3 Overlaid map view of the largest of the 36 CSO areas in South Bend. It shows the four different kinds of nodes - Instrumentation node (INode in yellow), Relay node (RNode in red), Gateway node (GNode in green), and Actuator node (Anode in gray). The blue box is a unit with a RNode and a GNode. . . . .	17
3.4 The duty cycle of a Chasqui node showing the awake period ( $T_a = 6$ seconds in the deployment) and the sleep period, which together constitute a slot ( $T_w = 5$ minutes in the deployment). The beginning of a round is marked by the base station initiating a new synchronization process. . . . .	18
3.5 FTSP’s problem with linear regression when working with sleep-wake operation. . . . .	21
3.6 Sleep-wake operation and its relationship to the synchronization protocol. . . . .	24
3.7 Illustration of the synchronization protocol. . . . .	27
3.8 The signaling from the MaxStream radio to the microcontroller. The signal on the transmit and on the receive side are used to take timestamps which are used in HARMONIA. . . . .	28
3.9 Network topologies used for our experiments. . . . .	34
3.10 Average network-wide synchronization time of HARMONIA. . . . .	37
3.11 Average network-wide synchronization time of FTSP. . . . .	37
3.12 Synchronization error of HARMONIA for the different nodes in Topology 1. . . . .	40
3.13 Synchronization errors of HARMONIA with different values of $t_{gap}$ in Topology 1. . . . .	40
3.14 Synchronization error of FTSP in Topology 1. . . . .	41
3.15 Synchronization error of FTSP in Topology 2. . . . .	41
4.1 Multi-hop routing paths to collect events. . . . .	47



Figure	Page
4.2 Line network model to collect events. . . . .	50
4.3 Staggered timeout. . . . .	52
4.4 Event gathering circle (EGC). . . . .	54
4.5 An example to show that a compromised node may hold the PT for more than $B$ time units without being identified. . . . .	56
4.6 Threshold $T_{th}$ . . . . .	57
4.7 Time margin $T_m$ . . . . .	59
4.8 Nodes placement for experiments. . . . .	64
4.9 False-alarm rate according to the number of retransmissions for the ARQ mechanism. . . . .	65
4.10 Detection rate according to the delay introduced by a compromised node. . . . .	66
5.1 System model. . . . .	70
5.2 An example of the probabilistic symbol mapping between $X(n)$ and $Y(n)$ in the corner cases when $K = 20$ and $M = 4$ ((a)-(c): empty or near-empty battery; (d)-(f): full or near-full battery). The symbol '*' in $P_Y(n)$ represents the element that can be non-zero. . . . .	73
5.3 Desired battery state profile. . . . .	74
5.4 An example to derive the dynamic programming framework. . . . .	76
5.5 Simulation results for the basic approach. . . . .	79
5.6 Penalty areas. . . . .	80
5.7 Simulation results for PRIVATUS. . . . .	84
5.8 Example of average daily usage across days ( $P = 7$ ). In order to flatten the the daily consumption, the days 1,2, and 4 need to use more energy by the amount indicated by the red arrow, and the days 3,5,6, and 7 are required to lessen the consumption by the amount indicated by the blue arrow. . . . .	85
5.9 The methods to keep energy and use the kept energy. . . . .	87
5.10 Virtual battery state and corresponding actual battery state. . . . .	88
5.11 Simulation results for PRIVATUS's per-day usage flattening. . . . .	90
5.12 Per-day average usage changes. In the figure, 'before' represents the per-day average use before flattening ( <i>i.e.</i> , $U(d)$ ), and 'after' means the per-day average use after flattening. . . . .	91

Figure	Page
5.13 Cost saving reduction due to the per-day usage flattening when $P = 7$ . The ratio $r_s$ is defined as $r_s = \frac{K_v u - \frac{m}{P} U_k^{max}}{K_v u + (m-1) U_k^{max}}$ , which implies the ratio of the maximum possible cost savings with and without the per-day usage flattening. . . . .	93
5.14 Information leakage when $K = 20$ and $m = 1$ . . . . .	95
5.15 Effects of sequence length $m$ and capacity $K$ in PRIVATUS ( $\alpha = 2$ ; $\beta = 1$ ). . . . .	96
5.16 Effects of $\alpha$ and $\beta$ in PRIVATUS ( $K = 20$ ). . . . .	98
5.17 Information leakage comparison between PRIVATUS with $\alpha = 2$ and $\beta = 1$ (legend: ‘prop’) and an existing scheme [23] (legend: ‘conv’), when $K = 20$ and $m = 2$ . The higher is $L_{(n,2)}^1$ , the worse is the information leakage. . . . .	98
5.18 Cost saving comparison between PRIVATUS and an existing scheme [23]. Here, we set $u = 0.2143\text{kWh}$ and $R_H = \$0.033/u = \$0.155/\text{kWh}$ . This results in the average daily usage ( <i>i.e.</i> , $E(\sum_{n=1}^{n_H} X(n))$ ) equal to $30\text{kWh}$ . . . . .	99
5.19 Effects of the estimation error for the distribution of $X(n)$ when $K = 20$ . . . . .	100
5.20 Information leakage ( $m = 2$ , and $s = 1$ ) with the per-day usage flattening, where $K = 30$ , $\alpha = 1$ , $\beta = 1$ , and $P = 7$ . Days 1, 2, and 4 are of type 1. . . . .	101

## ABSTRACT

Koo, Jin Kyu Ph.D., Purdue University, August 2012. Secure Control Protocols for Resource-Constrained Embedded Systems. Major Professors: Saurabh Bagchi and Xiaojun Lin.

Embedded systems are increasingly being deployed into the world around us. For example, they are used to monitor the environment around us, measure and control the electrical grid, and control vehicles on the road. As they are integrated in the real world, their security becomes increasingly important. However, due to their lower cost, energy constraints and slow computation speed, maintaining security for these systems is usually very challenging.

In this work, we study a range of the important security issues in the operation of embedded systems, which includes reliable synchronization, timely event reporting, and privacy-preserving data transmission. First, we propose a fast and reliable clock synchronization protocol for wireless sensor networks, called CSOnet, which is for wastewater monitoring and is deployed city-wide in a mid-sized US city, South Bend, Indiana. The nodes in CSOnet have a low duty cycle (2% in current deployment) and use an external clock, called the Real Time Clock (RTC), for triggering the sleep and the wake-up. The RTC has a very low drift (2 ppm) over the wide range of temperature fluctuations that the CSOnet nodes operate at, and it has low power consumption (0.66 mW). However, there are two challenges to using RTC for synchronization. First, RTC has a coarse time granularity of only 1 second. Therefore, it is insufficient to synchronize the RTC itself, which would lead to a synchronization error of up to 1 second. Such a large error would be unacceptable for the low duty cycle operation when each node stays awake for only 6 seconds in a 5-minute time window. The second challenge is that the synchronization has to be extremely fast

because ideally the entire network should be synchronized during the 6 second wake-up period. We address these challenges by designing a synchronization protocol called HARMONIA. It has three design innovations. First, it uses the fine-granular micro-controller clock to achieve synchronization of the RTC, such that the synchronization error, despite the coarse granularity of the RTC, is in the micro-second range. Second, HARMONIA pipelines the synchronization messages through the network resulting in fast synchronization of the entire network. Third, HARMONIA provides failure handling for transient node- and link-failures such that the network is not overburdened with synchronization messages. Further, the recovery is done locally. We evaluate HARMONIA on CSOnet nodes and compare the two metrics of synchronization error and synchronization speed with the flooding time synchronization protocol (FTSP). It performs only slightly worse in the former metric and significantly better in the latter metric

Second, we design a timely event reporting protocol for event monitoring, which is a common application of wireless sensor networks. For event monitoring, a number of sensor nodes are deployed to monitor some phenomenon. When an event is detected, the sensor nodes report it to a base station (BS), where a network operator can take appropriate action using the event report. In this paper, we are interested in scenarios where the event must be reported within a time bound to the BS, even under the case that some sensors need multiple hops to reach the BS. However, such a reporting process can be attacked by compromised nodes in the middle that drop, modify, or delay the event report. To solve such a problem, we propose SEM, a secure event monitoring protocol against arbitrary malicious attacks by Byzantine adversary nodes that may collude among themselves. SEM can provide the following provable security guarantees. As long as the compromised nodes want to stay undetected, a legitimate sensor node can report an event to the BS within a bounded time. If the compromised nodes prevent the event from being reported to the BS within the bounded time, the BS can identify a small set of nodes that is guaranteed to contain

at least one compromised node. To the best of our knowledge, no prior work in the literature can provide such guarantees.

Third, we introduce a privacy-preserving mechanism for smart meters in the smart grids. In smart power grids, a smart meter placed at the customer endpoint reports fine-grained usage information to utility providers. Based on this information, the providers can perform demand prediction and set on-demand pricing. However, such a fine-grain report also threatens user privacy, since users' specific activity or behavior patterns can be deduced from the fine-granular meter readings. To resolve this issue, we design PRIVATUS, a privacy-protection mechanism that take advantage of a rechargeable battery. In PRIVATUS, the meter reading reported to the utility is probabilistically independent of the actual usage at any given time instant. PRIVATUS also considerably reduces the correlation between the meter readings and the actual usage pattern over time windows. Further, using stochastic dynamic programming, PRIVATUS charges/discharges the battery in the optimal way to maximize savings in the energy cost, by taking advantage of different price zones.

# 1. INTRODUCTION

## 1.1 Embedded Systems and Their Security

An embedded system is a kind of specialized computer that is designed to execute a particular function. The embedded system is in contrast to a general-purpose computer that is designed to be flexible to satisfy a wide range of end-user needs. The embedded system often has a real-time constraint, *i.e.*, the operational deadline from an event to system response. Embedded systems control many devices in our lives, such as smart phones and car navigation.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the cost of the product. However, in the course of the efforts to reduce the cost, the security of the embedded system may be compromised. This is mainly because getting a market advantage for price-sensitive products has a higher priority than making the product secure, considering that there is no quantitative measure of security before the product is deployed. For example, many low-end embedded systems still use a 8-bit microcontroller that operates at several MHz clock frequency, and has a highly limited RAM space such as 16KB. These kinds of systems have a limited room for security overhead: they take too long time to generate a digital signature, or do not even have a memory available to store a cryptographic key.

Many embedded systems are battery-powered, and thus have significant energy constraints. This energy constraint is also another obstacle in achieving secure control of the embedded systems, because sophisticated security features are usually computation-intensive and thus consume a considerable amount of energy. For example, a TinyECC cryptographic suite takes a few seconds to generate or verify a digital signature in a 16-bit microcontroller working at 8MHz clock frequency. Considering that a typical message transmission takes only a few microsecond at the same plat-

form, this kind of expensive cryptography scheme might be something that should not be heavily used.

The challenges unique to embedded systems require new approaches to security in all aspects of control mechanisms of the embedded systems. In this work, we study a range of the important security issues in the operation of embedded systems, and provide situation-tailored solutions.

## 1.2 Summary of Contributions

The focus of this work is on the design of secure control protocols for embedded systems. Since the devices in most deployment scenarios for embedded systems have constrained resources such as bandwidth, energy, or processing power, the design of secure control protocols often turns out to be more challenging than in a general purpose system. Overcoming the challenges, we provide situation-tailored solutions for given problems with security in mind. Specifically, this work deals with three different topics: clock synchronization in the sensor networks, timely event reporting in sensor networks, and privacy-preserving data transmission in smart grids.

In the clock synchronization problem, we introduce our design experience for a clock synchronization protocol used in a large-scale sensor network, called CSOnet, which is deployed in the city of South Bend, Indiana for monitoring combined sewer overflow events. The distinctive challenges for synchronization in CSOnet were as follows. First, the synchronization had to be fast since the network only stayed awake for 6 seconds every 5 minutes and the projected scale of the network is large, of the order of a few hundred nodes. Second, the nodes in the CSOnet used the real time clock (RTC), which is external to the microcontroller chip, for the trigger for wake up. This is due to the RTC's low drift over the large temperature range to which the nodes are exposed. Its power consumption is also very low. However, the RTC has a coarse granularity of only 1 second. Thus, we have the situation that wake up is controlled by a clock whose granularity is so low that it is not sufficient to synchronize the clock,

given that the duty cycle is low. We found that no existing time synchronization protocol addressed these challenges, motivating us to design and develop our protocol called HARMONIA. HARMONIA is designed and implemented in TinyOS. It has three primary design innovations. First, it uses the high-resolution microcontroller clock to synchronize the low-resolution RTC. Second, the synchronization-related hand-shake between two adjacent nodes happens in two phases through a single message in each phase. However, HARMONIA pipelines the two phases, with a node acting as a source of the first phase message before it has itself received the second phase message. This design is important in achieving a rapid synchronization of the entire network. Third, reliability is built into HARMONIA to handle transient node- and link-failures. The goal is to localize the effect of a failure and not overburden the network with synchronization-related messages.

We next turn to the problem of timely event reporting. We focus on the fact that it is a difficult task to secure the event reporting process when the monitoring network is under attack. The sensor nodes are inherently vulnerable to attacks because they are usually deployed in non-protected environments. The adversary can often easily access the sensor nodes, and may even compromise them by reprogramming. Once some sensor nodes in a monitoring sensor network are compromised, they may prevent other legitimate sensor nodes from reporting information in a timely manner. For this reason, we propose SEM, a secure event monitoring protocol that can work even when there exist compromised nodes in the network. We are interested in a multi-hop network scenario where all sensor nodes except the base-station node can be compromised. The compromised nodes can launch arbitrary attacks in a Byzantine manner, such as dropping, modifying, and delaying the event report. They may also arbitrarily collude among themselves. Even in such a hostile environment, SEM can provide the following provable security guarantees: (1) As long as the compromised nodes want to avoid being detected, a legitimate sensor node can report an event to the base station within a bounded time. (2) If the compromised nodes launch an attack that causes the event report not to arrive at the base station within the



bounded time, the base station can identify a small set of nodes that is guaranteed to contain at least one compromised node.

In the last topic, we try to resolve a privacy issue in the smart grid, which is reportedly a key obstacle to faster deployment of the smart grid. Smart meters are key component of the smart grid. Up to three-fourths of the homes in the United States are expected to install the smart meter in the next decade. The smart meter provides the utility companies ways to collect the measurement readings for energy usage in a fine granularity (e.g., once in a few minutes). However, by gathering hundreds of data points even in a day, the utility companies may reconstruct much of our daily lives – when we wake up, when we go out for work, and when we come back after work. They might sell or accidentally reveal this information to marketing companies. To fix this issue, we design PRIVATUS, an algorithm by which the energy usage profile reported to the utilities looks independent of the actual energy usage profile. This is done by putting a rechargeable battery between the smart meter and the appliances. Our algorithm also achieves additional cost saving by optimally controlling the charge/discharge processes, based on stochastic dynamic programming.

### **1.3 Document Outline**

The rest of this document is organized as follows. The related works of HARMONIA, SEM, and PRIVATUS are summarized in Chapter 2. Chapter 3 presents HARMONIA, the clock synchronization protocol, tailored for the CSOnet. Chapter 4 describes our secure event monitoring protocol, SEM. In Chapter 5, we introduce our privacy-preserving solution PRIVATUS for smart grids. Conclusion of this work is given in Chapter 6. Finally, Chapter 7 describes the future research issue.

## 2. RELATED WORK

### 2.1 Clock Synchronization in Sensor Networks

Clock synchronization has long been a subject of study in wired networks. Network Time Protocol (NTP) [1] and global positioning system (GPS) receivers are popularly used for synchronization. However, there are significant challenges in applying them to wireless sensor networks, such as high power consumption, accuracy of only milliseconds, and unavailability of synchronization signals indoors. We also need to consider that the hardware clocks on the individual nodes may experience significant drifts. This happens chiefly due to manufacturing variations in the different crystals, the temperature fluctuations the nodes (and consequently the crystals) are exposed to, and aging of the crystals. Tight budget concerns in the design of the sensor nodes rule out the use of the highly accurate oven controlled crystal oscillator (OCXO) or high-end temperature compensated crystal oscillator (TCXO). Also, the multi-hop nature of the sensor network precludes the use of client-server solutions, which most of the solutions from the landline world fall in.

Therefore, there has been active research in time synchronization in the sensor network community. We refer the reader to [2] for a good coverage of the early work in this field and here we focus on the more recent work. At the high level, HARMONIA is motivated by the unmet need for synchronizing networks that are sleep-wake enabled and that have low duty cycle. The real-world constraints of the Chasqui node introduce the additional challenge for HARMONIA to synchronize a low resolution real time clock. These challenges are orthogonal to those addressed by the existing work that we survey here.

The Timing-sync Protocol for Sensor Networks (TPSN) [3] aims to provide network-wide time synchronization. The TPSN algorithm elects a root node and builds a

spanning tree of the network during an initial discovery phase. The synchronization phase proceeds in rounds with the children node in the tree being synchronized to their parents through a two-way message handshake in each round. Each node embeds its local clock's readings in the two-way message handshake and through it the child node can calculate the propagation delay and its clock offset relative to its parent's. TPSN introduced the idea of MAC layer time-stamping. However, TPSN does not compensate for clock drifts which makes frequent resynchronization necessary. In addition, TPSN requires the two-way handshake to complete between a parent-child pair before the synchronization can propagate further in the network.

The Flooding Time Synchronization Protocol (FTSP) [4] has already covered it in some detail in Section 3.2.

The Rapid Time Synchronization (RATS) [5] is also a MAC layer time-stamping based protocol. In RATS, a root floods a message carrying an event time. On receiving this message, nodes calculate the elapsed time since the event occurrence using a simple time-stamping primitive called Elapsed Time on Arrival (ETA) based on the MAC layer time-stamping technique. By subtracting the elapsed time from the receiving time, nodes convert the event time from the root's local time to its local time. This process may look similar to HARMONIA's SYNC/SYNCD flooding. However, in HARMONIA, as a SYNCD message propagates through a network, each node calculates the relative difference in the MCC readings between the BS and itself. By doing so, each node estimates the current value of the MCC at the BS.

The Reachback Firefly Algorithm (RFA) for clock synchronization [6] is inspired by the way neurons and fireflies spontaneously synchronize. Each node periodically generates a pulse (message) and observes pulses from other nodes to adjust its own firing phase. RFA only provides synchronicity—nodes agree on the firing phases but do not have a common notion of time. RFA is likely to take a long time to get all the nodes to be firing synchronously and therefore will likely not be suitable for our application.

In [7], the authors propose a way to estimate the drifts in the clocks of two nodes caused by the environment-dependent variations. The authors introduce the notion of a software compensated crystal oscillator (SCXO). In an SCXO, the differential drift between the crystals of two nearby nodes is used to estimate the drift in the crystal of one of the nodes. The solution comprises of a one time calibration phase and a runtime measurement and compensation phase. SCXO achieves mean effective clock stability of 1.6 ppm over a temperature range of  $-40^{\circ}\text{C}$  to  $75^{\circ}\text{C}$ . This would allow us to increase the period between synchronizations of CSOnet. The authors provide a practical implementation of the SCXO work in [8] and describe a Crystal Compensated Crystal based Timer (XCXT), a new way of compensating a pair of crystals which achieves a 1.2ppm precision over a temperature range of  $-10$  to  $60^{\circ}\text{C}$  while using only 1.27mW. The solution relies on a node having two crystal inputs and two timer units (TMote Sky is their demonstration platform). To improve the power consumption the authors describe two approaches. The first is to simply duty cycle one of the crystals. The second approach is to use two crystals, one fast and the other slow. The fast crystal (8 MHz crystal of the MSP430 microcontroller in their demonstration) is used if fine granularity time is needed. The second slower crystal (32 kHz in their demonstration) is used while the system is in sleep. Both crystals compensate for each other's drift and together form a highly stable timer unit. This last hardware design feature, in a context quite different from that of the Chasqui nodes, shares a similarity with the Chasqui design of two clocks. However, this is used by the authors for achieving power savings.

A recent development in the field is gradient based clock synchronization [9]. In this the authors present the design to minimize the clock offset between neighboring nodes. The motivation is that other time synchronization protocols synchronize clocks based on some topology, whether assumed or created as part of the synchronization protocol. Two geographically nearby nodes may be distant in this topology. Therefore, existing protocols, while trying to ensure a small synchronization error globally in the network, may cause the synchronization error in a local neighbor-

hood to be appreciable. Therefore, the authors design the protocol to have very low synchronization error in local neighborhoods.

## 2.2 Timely Event Reporting in Sensor Networks

Event monitoring applications of WSNs have been researched for a variety of scenarios, including military surveillance and forest-fire detection. The common research issue of event monitoring is energy efficiency and lifetime maximization of sensor networks. Several schemes are proposed to address the optimization of sensing coverage (*e.g.*, [10–12]), the goal of which is to monitor the system of interest using the minimal amount of resources. Another direction to improve energy efficiency is to balance the energy consumption over the sensor nodes (*e.g.*, [11, 13]), since unbalanced energy dissipation causes some nodes to die faster than others, thus reducing the network lifetime. However, most of existing protocols are designed without security in mind.

Recently, event monitoring in the presence of compromised nodes began to receive attention [14]. The authors in [14] assume the Man-in-the-middle attackers that can prevent, delay, or manipulate the event report from a legitimate sensor node. Their approach to defend against the compromised nodes is to make the sensor nodes flood the event report over the entire network. This method will work when there exists at least one “legitimate route” from each reporting node to the BS that does not contain any compromised node. However, if the adversary simply compromises all the neighboring sensor nodes of the BS, thus isolating the BS, this method cannot provide any guarantee.

To the best of our knowledge, no existing solution can work in hostile environments where the compromised nodes may block all the routing paths coming into the BS, thereby leaving no chance for a legitimate node to report an event in time.

### 2.3 Privacy Protection and Cost Saving in Smart Grids

There has been extensive research about privacy protection in the area of database systems, where the goal is to provide statistical information (such as sum, average, or maximum) without revealing sensitive information about individuals. The common approach to achieve this goal is data perturbation [15–17]. The perturbing data can be simply done by swapping values between records [15], and adding noise into the record [16]. Anonymizing attributes of the datasets is another form of the data perturbation [17]. An example of this category is the concept of  $k$ -anonymity [17] by which information contained in a record of the database cannot be distinguished from at least  $k - 1$  other records. However, none of methods in this area is directly applicable to hide the privacy information in the meter readings from the smart meters, because the utility companies do have to know precise meter reading records for billing purpose.

Recently, many studies raised the privacy concern in the smart grid both from a technical perspective and from a legal perspective [18–21]. These works suggest enforcement of privacy properties based on organizational means, codes of conduct and regulations, subject to current legislations. However, only a few works have been proposed so far on the design of technical solutions to handle the privacy issue in the smart grid.

Rial *et. al.* [22] proposed a privacy-preserving metering system, where the energy bill for a specific period is calculated by the user and then sent to the utility company. This system allows the user not to report the fine-granularity meter readings. However, it limits the power grid operator’s capability such as demand prediction.

Kalogridis *et. al.* [23] and McLaughlin *et. al.* [24] used a rechargeable battery to perform low-pass filtering over the load profile. Their algorithm forces the battery to charge (or discharge) a certain amount of energy if possible, when the required load is smaller (or larger) than the previously metered load. Thus, the high-frequency variation on energy usage profile is not visible to the smart meter. This approach can help

eliminate load signatures that indicate which appliance is being used. However, the low-frequency components of a load profile are still revealed without any protection. Further, the proposed solution did not consider the cost-saving opportunity of using the rechargeable battery.

Another work using the rechargeable battery is proposed by Varodayan *et. al.* [25]. They considered a simple binary-state battery model, where the battery is probabilistically charged by drawing the energy from the grid and discharged to feed the appliances. However, in their model, the charging and discharging processes at a given time instant are not independent of each other. This leads to a high level of information leakage (at least 0.5 bit for one-bit information). The authors also failed to consider the possible saving in the electricity cost by using the rechargeable battery.

Our work also adopts the rechargeable battery to protect the user privacy, but we design a mechanism by which the charging and discharging processes are guaranteed to be independent of each other at a given time instant. Further, our design also considers to reduce the correlation between the sequences of the charging and discharging processes over multiple time instants (instead of just for a single time instant). This makes it difficult for the adversary to make a meaningful guess on the user behavior by observing the sequence of meter readings. In addition, our design ensures that the way of charging the battery is optimal in the sense that we can maximize the average saving in the energy cost. This is achieved by controlling the charging process by dynamic programming [26].

### 3. CLOCK SYNCHRONIZATION IN SENSOR NETWORKS

Wireless sensor actuator networks or WSANs consist of computer controlled sensors and actuators that communicate over a wireless (usually RF) communication network. WSANs use sensed data to power actuators which can then affect the sensed environment. The resulting changes in that environment can then be sensed by the network. This forms a distributed feedback loop that has the potential for efficiently controlling geographically distributed processes at a scale that was previously unthinkable. A metropolitan scale (city wide) WSAN, called CSOnet, is currently being built by a partnership of private (EmNet, LLC), public (City of South Bend), and academic (Purdue University and University of Notre Dame) agencies. The WSAN is being built to control the frequency of combined sewer overflow (CSO) events in a mid sized U.S. city (South Bend, Indiana). More than 700 cities in the U.S. have sewer systems that combine sanitary and storm water flows in the same system. During rain storms, wastewater flows can easily overload these combined sewer systems, thereby causing operators to dump the excess water into the nearest river or stream. The discharge is called a CSO event [27]. The problem addressed by CSOnet represents a major public health and environmental issue faced by many U.S. cities. At present, the system consists of 150 wireless sensor nodes monitoring 111 locations in the South Bend sewer system.

The CSOnet deploys nodes in the sewage channels for sensing, on top of traffic poles for relaying, and at major traffic intersections to act as gateways to the cellular network, by which the sensed data is uploaded to a backend server. The nodes are called Chasqui nodes, which are based on the Crossbow Mica2 mote design, but expand on it to add a longer range and faster radio, and significant to our problem, a Real Time Clock (RTC) with an extremely low drift of 2 ppm. The Chasqui nodes



are meant for long-term operation without the need to change batteries. Therefore, a natural design point is to have low duty cycle operation of the network. In the current deployment, each node stays awake for 6 seconds in a 5 minute period, leading to a 2% duty cycle. This led us to the requirement of accurate time synchronization for the Chasqui nodes.

The distinctive challenges for synchronization in CSOnet were three-fold. First, the synchronization had to be fast since the network only stayed awake for 6 seconds at a time and the projected scale of the network is large, of the order of a few hundred nodes. Second, the Chasqui nodes used the RTC, which is external to the microcontroller chip, for the trigger for wake up. This is due to the RTC's low drift over the large temperature range to which the nodes are exposed—from  $-13^{\circ}\text{F}$  to  $122^{\circ}\text{F}$ . However, crystals used for clocks have a tradeoff in three dimensions—drift, granularity, and power consumption. The power consumption has also to be kept very low and hence the RTC sacrifices the granularity that is exposed to the programmer—it has a coarse granularity of only 1 second. Thus, we have the situation that wake up is controlled by a clock whose granularity is so low that it is not sufficient to synchronize the clock, given that the duty cycle is low. Third, the high power, long range, and high speed radio used is a MaxStream 115.2 kbps radio where the firmware is not available for modification. Thus, we cannot use a common technique used in time synchronization protocols—MAC layer time-stamping. Additionally, MAC layer time-stamping with high speed radios poses problems as documented in [28,29]. While we have posed these challenges in the context of CSOnet, we believe they are more general than that. Abstracting out the details, these challenges to a time synchronization protocol will be posed by any WSN that has large scale, low duty cycle operation, proprietary radio stack, and crystals that make a natural tradeoff between drift, granularity, and power consumption.

We found that no existing time synchronization protocol addressed these challenges motivating us to design and develop our protocol called HARMONIA. HARMONIA is designed and implemented in TinyOS and executes on the Chasqui nodes.

It has *three primary design innovations*. *First*, it has an algorithm to use the high resolution microcontroller clock to synchronize the low resolution RTC. *Second*, the synchronization-related hand-shake between two adjacent nodes happens in two phases through a single message in each phase. However, HARMONIA pipelines the two phases, with a node acting as a source of the first phase message before it has itself received the second phase message. This design is important in achieving a rapid synchronization of the entire network. *Third*, reliability is built into HARMONIA to handle transient node and link failures. The goal is to localize the effect of a failure and not overburden the network with synchronization-related messages.

To evaluate HARMONIA, we create small-scale linear and tree topologies with Chasqui nodes, with each node running the CSOnet application and having a low 2% duty cycle. We evaluate the time to synchronize the network and the synchronization error between any two pair of nodes. We compare this to FTSP running on Mica2 nodes. While a comparative evaluation on the same hardware platform would have been desirable, each protocol relies critically on some specific hardware feature. The results validate our design goal that HARMONIA is faster than FTSP, while sacrificing synchronization error. A representative result is that HARMONIA is 8.7X and 12.1X faster than FTSP for a 5 hop linear network depending on the setting of FTSP, and with a period of 300ms for synchronization messages. The average one-hop synchronization error of FTSP is only  $1.5\mu\text{s}$ , while that of HARMONIA is  $16.77\mu\text{s}$ .

## 3.1 CSOnet

### 3.1.1 CSOnet Architecture

CSOnet’s architecture was designed to be a set of local WSAFs that connect to an existing wide area network (WAN) through gateway devices. CSOnet can therefore be viewed as a heterogeneous sensor-actuator network. It consists of four types of devices: (i) Instrumentation Node or INode: these nodes are responsible for retrieving the measurement of a given environmental variable, processing that data

and forwarding the data to the destination gateway through a radio transceiver. (ii) Relay Node or RNode: these nodes aid in forwarding data collected by INodes that are more than one-hop away from the gateway node. The RNodes only serve to enhance the connectivity in the wireless network. (iii) Gateway Node or GNode: these nodes serve as gateways between the WSA used to gather data from the INodes and a Wide Area Network (WAN) which allows remote users easy access to CSOnet's data. (iv) Actuator Node or ANode: these nodes are connected to valves (actuators) that are used to hold back water in the sewer system.

To appreciate the challenges posed to a synchronization protocol, we first need to describe the system that is controlled by CSOnet. Figure 3.1 shows a sewer system in which combined sewer trunk lines (sanitation and storm water flows) feed into a large *interceptor sewer*. Prior to 1974, municipal combined sewer lines dumped directly into rivers and streams. Under the Clean Water Act, cities were forced to treat the water from these combined sewer lines before they were released into a river or stream. One common way to meet this regulatory burden was to build an interceptor sewer along the river. This sewer would intercept the flow from the combined sewer trunk lines and convey that flow to a wastewater treatment plant (WWTP). Under dry weather conditions the flows were small enough to be handled by the WWTP. Under wet weather conditions (storms), the flows often overwhelmed the WWTP's capacity, thereby forcing operators to dump the excess directly into the river or stream. Such discharges constitute the CSO events described earlier.

From Figure 3.1 we can see that the combined sewer trunk lines and interceptor sewer connect at a *CSO diversion structure*. This is the point where we can apply control. This means that the natural place to put ANodes is at the CSO diversion points. These ANodes would then adjust the amount of water diverted into the interceptor sewer line based on an adaptive threshold that is a function of the current flows into the system. The GNode serves as a gateway between this particular WSA and neighboring WSAs up and down the interceptor line. Figure 3.2 illustrates this system architecture with 2 different WSAs controlling the two diversion structures

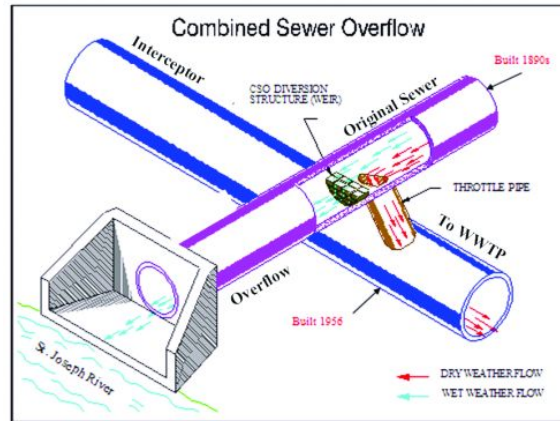


Fig. 3.1. South Bend Interceptor Sewer and CSO Diversion Structure.

into the interceptor line. GNodes at these diversion structures and the WWTP are used to exchange control information in a way that allows coordinated flow control across the city's entire sewer system.

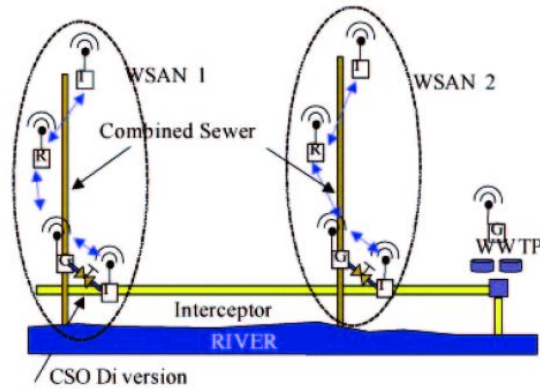


Fig. 3.2. CSOnet's Hierarchical Architecture.

### 3.1.2 CSOnet Hardware

The basic building block of CSOnet’s WSA is a more rugged version of the Mica2 processor module called the Chasqui wireless sensor node. The Chasqui node started with the original embedded node designs developed by U.C. Berkeley. EmNet, LLC enhanced the radio subsystem and sensor/actuator interface subsystems of this earlier design. The Chasqui node uses a 115 kbps MaxStream radio operating at 900 MHz. It uses frequency hopping spread spectrum (FHSS) signaling to reduce the radio’s sensitivity to interference. The radio has a larger maximum transmission power (1 watt) than the conventional Chipcon radio. Consequently, the Chasqui node has a range of over 700 meters in urban environments and up to a 5 km range for line-of-sight connections. The longer range of the Chasqui processor fits well with the distances required by the CSOnet application. The MAC layer of the radio is implemented in proprietary firmware that is closed source. However, a feature significant to our synchronization protocol, is that the radio sends a signal a fixed offset time after the first bit being sent out on the wireless channel and also a signal when the first bit is received from the wireless channel. This signal is used to trigger an interrupt followed by executing part of HARMONIA’s algorithm.

To give a sense of the deployment for which our HARMONIA is targeted, we provide in Figure 3.3 an overlaid map view of the largest of the 36 CSO areas in South Bend, which covers an area of 3758 acres. It has 7 RNodes, 3 INodes, 2 GNodes and 1 ANode, that controls an automated valve at the basin. Notice that the network of Rnodes is almost linear. Due to the requirements of the application that the network needs to span a large geographical area, the Rnodes provide relaying functionality, and the radio has a long range, the network in most parts is almost linear. This is a driver for some design decisions in HARMONIA, which we will discuss in Section 3.5.

In spite of the higher transmission power required by the MaxStream module, careful design of the CSOnet middleware and hardware allows the WSANs based on the Chasqui node to operate for several years before changing batteries. The Chasqui

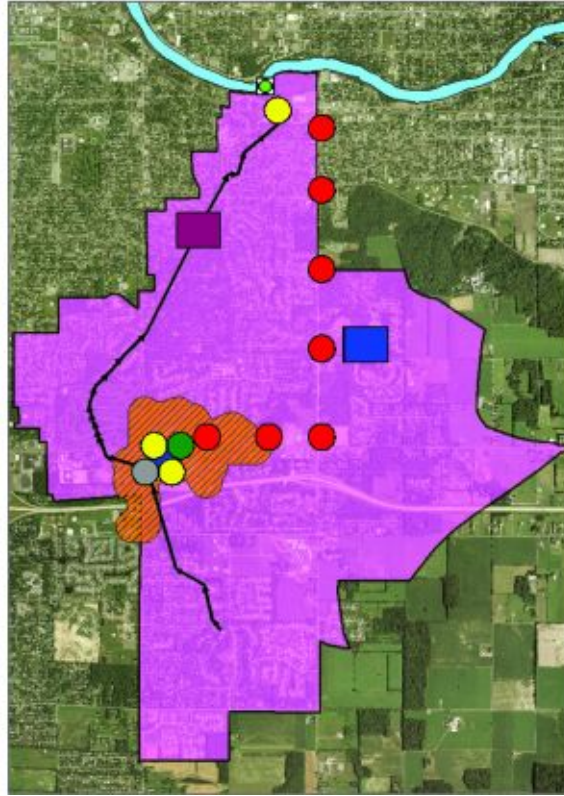


Fig. 3.3. Overlaid map view of the largest of the 36 CSO areas in South Bend. It shows the four different kinds of nodes - Instrumentation node (INode in yellow), Relay node (RNode in red), Gateway node (GNode in green), and Actuator node (ANode in gray). The blue box is a unit with a RNode and a GNode.

node consumes up to 5W when fully active and drops down to 0.14mW in sleep mode. Long battery life can be effectively achieved by using low duty cycles. All the nodes in CSOnet wake up at the beginning of every  $T_w$  seconds defined a slot, and stay awake only for the first  $T_a$  seconds of each slot based on the RTC. Here,  $T_a$  is much smaller than  $T_w$  to save battery power. In current deployment, those are set to  $T_a = 6$  seconds and  $T_w = 300$  seconds, resulting in a 2% duty cycle.

These values are possible due to the nature of the phenomenon that the WSN is meant to monitor—such events last for more than 5 minutes. The biggest limitation to efficient communication in low duty cycle systems is precise synchronization. Typ-

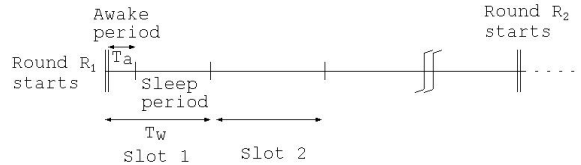


Fig. 3.4. The duty cycle of a Chasqui node showing the awake period ( $T_a = 6$  seconds in the deployment) and the sleep period, which together constitute a slot ( $T_w = 5$  minutes in the deployment). The beginning of a round is marked by the base station initiating a new synchronization process.

ical crystal tolerances such as the one used in the Mica2 platform are on the order of 40 ppm yielding drifts of up to 3.456 seconds per day. Extreme temperature differentials can be seen in the CSOnet application: nodes inside the sewer system are at a relatively constant temperature of around  $10^\circ\text{C}$  year round while nodes mounted on traffic poles can experience temperatures ranging between  $-20^\circ\text{C}$  and  $50^\circ\text{C}$ . Experiments at these temperatures showed drifts of up to 3 seconds per day using regular crystals. While synchronization algorithms can periodically reset the drift error between nodes, they also consume precious energy resources. Therefore, the Chasqui node uses a precision RTC provided by the Maxim DS3231 chip [30]. Using this, the nodes can coordinate their active and sleep cycles with sufficient precision to reliably function at a 2% duty cycle. The Chasqui node implements a precision RTC with a typical drift of only 2 ppm giving CSOnet tight synchronism between synchronization updates. Our calculation, based on experimental results for HARMONIA's synchronization error, shows that the Chasqui nodes can reliably function with periodic synchronization updates in HARMONIA every 13 hours (see Section 4.5). With such a duty cycle, the CSOnet applications based on the Chasqui processor node have a service life in excess of three years with a 4 cell lithium battery pack.

### 3.2 Can FTSP be Used to Synchronize CSOnet?

The FTSP protocol [4] represents the state-of-the-art in synchronization protocols and compensates for most sources of time variability, thus achieving highly accurate synchronization. It does not rely on any network topology. A root is elected, based on node IDs, and it initiates the synchronization by periodically broadcasting a synchronization message. After some initial startup time when the caches are being populated, each node periodically broadcasts to its neighbors its local estimate of the time at the root node. FTSP uses a single broadcast message, rather than a two-way handshake, to establish synchronization points between the sender and the receiver. FTSP’s design eliminates many sources of synchronization error, notably the interrupt handling time and the encoding/decoding time. It also uses MAC layer time-stamping. Each node uses a linear regression table to estimate the offsets between the local clock and that of the root node. The performance of the protocol—the synchronization error and the time to synchronize the network—is dependent on the number of points that are used to create the regression line. This technique enables each node to estimate its drift with respect to another node and compensate for it.

If we say that the synchronization packet flooding period is  $P$ , the number of points needed to draw the regression line is  $N_R$  ( $N_R = 8$  by default), and the maximum number of hops in the network from the root is  $N$ , then FTSP takes approximately  $N_R P N$  time to synchronize the whole network [4]. This is because only after a node finishes the linear regression by receiving the  $N_R$  synchronization packets, it can start to flood the estimate of the global time through a local broadcast. Moreover, if the root fails and a new root needs to be re-elected, this takes  $P N / 2$  time on an average. This is for the average case where the new root is at a distance  $N / 2$  from the old root. Such time requirements of the FTSP make it challenging to apply it to CSOnet synchronization since nodes in the CSOnet stays awake only for 6 seconds every wakeup and many parts of the network are in effect connected in a linear topology. For example, even if we set the value of  $P$  quite short (compared to values used in the



experiments in [4]) as  $P = 300\text{ms}$  and sacrifice the performance of linear regression by taking the minimum two points, we can synchronize at most a 20-hop network from the root within the 6 seconds. Practically this number will be much smaller because nodes can communicate with each other for even less than 6 seconds due to the drift in RTC when a synchronization protocol is initiated. For example, we are targeting to synchronize the whole network within 2 seconds for this reason.

*Let us consider two straw man proposals to adapt FTSP to our problem.* First, we use FTSP to synchronize the microcontroller clock (MCC) since it has a fine granularity ( $0.125\mu\text{s}$  for the Mica2) and can benefit from the small synchronization error achievable with FTSP. However, the MCC does not run during the time the Chasqui node is asleep and the sleep-wake is guided by the RTC. Therefore, synchronizing the MCC will not serve our purpose.

Second (and alternately to the first), we synchronize the RTC since the RTC continues to tick through the microcontroller’s sleep period. Then we can relax the requirement that the entire network needs to be synchronized within the awake period of one slot. Rather the synchronization packets needed for regression can be collected over multiple slots and the RTC synchronized with them. However, the RTC has a coarse granularity of only 1 second and therefore, despite the small synchronization error of FTSP, the clock may differ by up to 1 second. This would be unsuitable for the low duty cycle CSOnet.

The two straw man proposals suggest the approach that we take in design of HARMONIA. The approach simply put is to synchronize the MCC first and then change the RTC to a globally determined value at the same time based on the synchronized MCC. This achieves both finely granular value for the time used in synchronization algorithm and synchronism of RTC for sleep-wake. Therefore, applying FTSP to this model boils down to first synchronizing the MCC using FTSP and then adjusting the RTC based on this synchronized MCC. However, this runs in to the slow network-wide synchronization problem of the FTSP explained at the beginning of this section. This argument crucially depends on the following observation—the synchronization of the

MCC has to happen within *one awake period of one slot*. This cannot be staggered over multiple slots.

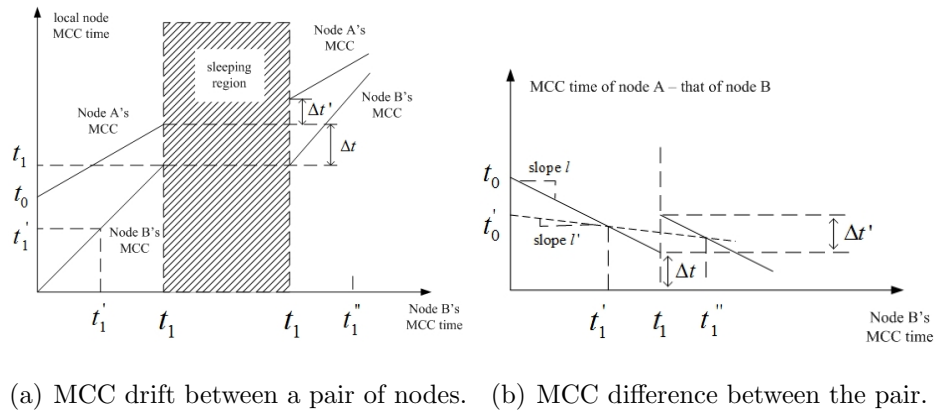


Fig. 3.5. FTSP's problem with linear regression when working with sleep-wake operation.

The reason is explained by Figure 3.5. Consider that node B is trying to synchronize itself to the clock of node A. In Figure 3.5(a), we see two lines one corresponds to node B's MCC measured with respect to node B's MCC — obviously this is a 45° line from the origin; the second corresponds to node A's MCC again measured with respect to node B's MCC. The two clocks have different frequencies and hence the difference in slope between the two lines. Node A's clock also has an offset—time  $t_0$  in the figure. In Figure 3.5(b), we see the MCC difference between nodes A and B with respect to node B's MCC. Ideally, node B should be able to estimate the difference in drift between A's MCC and its own MCC. Thus, in Figure 3.5(b), it should be able to estimate the slope  $l$ . According to FTSP, if FTSP had completed within the wake period, it would indeed have been able to estimate the slope. However, since the synchronization does not complete within the wake period, node B hits against the onset of sleep, time  $t_1$  in Figure 3.5(a). At this time the offset that node A's MCC has over node B's MCC is  $\Delta t$ . However, nodes A and B wake up after their sleep based on a trigger from their respective RTCs. The RTCs also have different frequencies. Therefore, nodes A and B wake up at slightly different times, say node

A wakes up before node B. Then the offset at node B’s MCC time  $t_1$  suddenly jumps from  $\Delta t$  to  $\Delta t + \Delta t'$ . In other words, the curve in Figure 3.5(b) has a discontinuity at time  $t_1$ . Now, consider what happens if node B had staggered its regression points across the two awake periods. Node B would then have estimated, using FTSP, that the slope of the relative MCC difference is  $l'$  (Figure 3.5(b)), rather than the correct slope of  $l$ . There is no fixed relation between  $l'$  and  $l$ —it depends on the arbitrary order and difference in time between the wake-up of nodes A and B.

The nub of the argument then is that the linear regression should be finished within each awake period. For networks of the size of CSOnet, FTSP out-of-the-box cannot achieve this as we will show in the experiments section. Yet a third strawman proposal to modify FTSP to suit our needs is as follows. Use FTSP to synchronize the MCC clocks across a large network by periodically keeping the nodes turned on for more than 6 seconds. Then, synchronize the RTC clocks by using the MCC clocks. The increase in the duty cycle will have to be done rarely (once every 13 hours in our network as per the calculation in Section 4.5.3). However, the problem with this approach would be that during the synchronization process, there is a large number of messages that are sent down from the root throughout the network. With a period of 300ms for synchronization messages in FTSP, the synchronization process takes a long time - greater than 8 seconds for a 5-hop diameter network in our experiments (see Figure 3.11) and which increases linearly with the number of hops. During this period, it is quite likely that data messages flowing up toward the base station (BS) will collide and have a low reliability. Considering that CSOnet is meant to detect rare and critical events, such reduced reliability during the periodic synchronization events would be unacceptable.

Moreover, a problem common to all these FTSP extensions is that they will not handle efficiently the case that a node (or a sub-tree) comes out of failure and wants itself (or the set of nodes in the sub-tree) to be synchronized. The extensions will flood the synchronization messages all through the network. Hence, the need for a new synchronization protocol, hence HARMONIA.

### 3.3 Proposed Protocol

#### 3.3.1 Operational Scenario

The CSOnet is connected for data dissemination and collection in a tree topology whose root is a BS. The topology is created by stateless gradient-based routing [31]. Each node in the network has a gradient number that is an indication of how close the node is to the destination. Since there might be several destinations, each node stores one gradient number per destination in the network. HARMONIA will also use the tree topology.

Recollect that all nodes in the CSOnet wake up at the beginning of every  $T_w$  seconds defined as a slot, and stay awake only for the first  $T_a$  seconds of each slot. The BS initiates the synchronization procedure in certain slots. We say a new *round* of synchronization is started when that happens. The BS may decide when to initiate this based on a fixed period, for example, through calculation of the worst case drift of the RTC, or some indication that the network has gone out of synchronism, for example, inferring from a drop in the received data rate.

We first provide a conceptual view of how HARMONIA works, hiding the technical details. Note that there are two clocks in the picture - a MCC and the RTC. The MCC has a high drift but high resolution, and it also does not tick when the node is sleeping. The RTC has a low drift but low resolution, such that synchronizing the RTC alone will have a large synchronization error (up to 1 second) and thus will not be sufficient for our requirements. Our goal is to synchronize RTCs of all nodes accurately enough to ensure all nodes in the network wake up at the same time.

The BS initiates the synchronization once a certain time has elapsed since waking up. Synchronization happens in cascaded stages where the synchronization proceeds along the tree topology with the BS acting as the root node. The interaction between a node and its children happens in two phases. A pipelining effect is achieved between multiple levels of the tree by having a node perform the first phase of the synchronization with its children even though it has not *completed* its own synchronization,

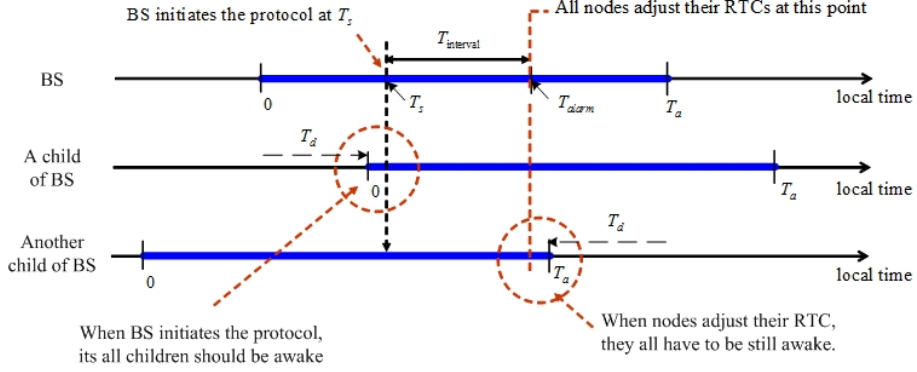


Fig. 3.6. Sleep-wake operation and its relationship to the synchronization protocol.

i.e., it is yet to complete its second phase. After a node has received the two phases from its parent, a node is considered synchronized with respect to its parent. It then sets an alarm using its MCC. The drift in the MCC during this alarm interval contributes to the synchronization error in HARMONIA, in addition to other factors. The synchronization achieves the effect that the alarms of all the nodes in the network will go off at the same time, modulo the synchronization error. When the alarm goes off, a node sets its RTC's second hand to a value determined by globally known parameters. Since all the nodes do this at the same time and since sleep-wake happens according to the RTC value, this implies that the entire network is synchronized for its sleep-wake.

Once the BS decides to synchronize a network, it begins the protocol  $T_s$  seconds after it wakes up as depicted in Figure 3.6. The value of  $T_s$  should be chosen to ensure that all the children of the BS have already woken up so as not to miss any synchronization-related messages from the BS. In addition, since all nodes adjust their RTC at  $T_{alarm} = T_s + T_{interval}$  they have to stay awake until the RTC is adjusted. Thus the values for  $T_s$  and  $T_{interval}$  must be taken to satisfy the following conditions:

$$T_s > T_d \quad \text{and} \quad T_s + T_{interval} < T_a - T_d, \quad (3.1)$$

where  $T_d$  denotes the maximum offset in RTC that has built up between a parent and a child node since the previous synchronization. However the second condition is not as critical if we enforce the design that a node delays going to sleep till its alarm has expired. Additionally, the value  $T_{interval}$  used at the BS should be large enough that all the nodes in the network have gone through both synchronization phases and are ready to set their RTCs. However, there is a desire to keep  $T_{interval}$  small since the drift in the MCC during this interval contributes to the synchronization error.

### 3.3.2 Synchronization Protocol

Our goal is to synchronize RTCs of all nodes to ensure all nodes in the network wake up at the same time. However, since RTC has only 1-second resolution, if we adjust any node’s RTC on the basis of another node’s, there could be at worst a 1-second synchronization error between the two nodes. In order to reduce this kind of uncontrollable synchronization error, we adopt another timer in our protocol, which uses a MCC provided by the Atmel Atmega128L, a microcontoller used in Chasqui notes. The MCC provides much finer resolution than the RTC, operating at the frequency of 8 MHz. However it cannot be used directly as a system clock since it does not run when a node is sleeping. Therefore the core part of HARMONIA is about how to use the MCC to set the RTC to the same value, at the same time. Here “same time” must be defined within a high resolution, identical to that of the MCC. From now on, the value of the MCC is expressed using lowercase  $t$  not to be confused with the value of the RTC, which is being expressed using uppercase  $T$ .

When the BS initiates the protocol, it sets an alarm to go off after  $T_{interval}$ . To achieve this, it sets the MCC timer that goes off at  $t_{alarm}$ . For example, for  $T_{interval} = 2$  seconds and for a 8 MHz MCC, it will set the timer to expire after  $16 \times 10^6$  ticks. When the alarm fires, the RTC’s second hand is set to the value  $T_{alarm} = T_s + T_{interval}$ . Right after setting the alarm, the BS gets to be the first to do the following two-phase

message transmission. This is repeated recursively by each node with its children through the network.

**Phase 1:** SYNC packet transmission and reception

- Transmission: A parent sends to its children a synchronization initiator packet called SYNC that carries  $t_{alarm}$ . The parent records  $t_p$  the local time at which its radio chip starts to transmit the first bit of the SYNC through an antenna.
- Reception: Each child records  $t_c$  the local time at which its radio chip starts to receive the first bit of the SYNC.

**Phase 2:** SYNCD packet transmission and reception

- Transmission: A parent sends to its children a synchronization data packet called SYNCD carrying the  $t_p$  and  $t_{dif}$ , where the  $t_{dif}$  is the offset between its MCC and the BS's. For the BS, the  $t_{dif}$  is always set to zero.
- Reception: After receiving the SYNCD, each child of the parent updates its  $t_{dif}$  as  $t_{dif} = t_{dif}^{rcv} + t_c - t_p$  (the "rcv" indicates it is the value received by the node), and sets an alarm that goes off at  $(t_{alarm} + t_{dif})$ .

Here each SYNC(SYNCD) packet is sent after a backoff time taken randomly from a uniform distribution over  $[0, t_{bf}]$ , where  $t_{bf}$  denotes the maximum backoff time. This is to avoid contention in the synchronization packets among neighbors.

Figure 3.7 depicts the above two-phase synchronization packet transmissions and receptions performed from the BS to two-level lower hierarchy. Every node in the network becomes aware of  $t_{alarm}$ , the time at which the BS expires its alarm by receiving SYNC packet from its parent. However, since all nodes' MCC may not be synchronized, each node needs to figure out the offset in the MCC between itself and BS to make its alarm go off at the same physical time as at the BS. This is done by the SYNCD packet propagation: When a node receives the SYNCD from its parent, the SYNCD lets it know the offset between the parent and the BS, that is,

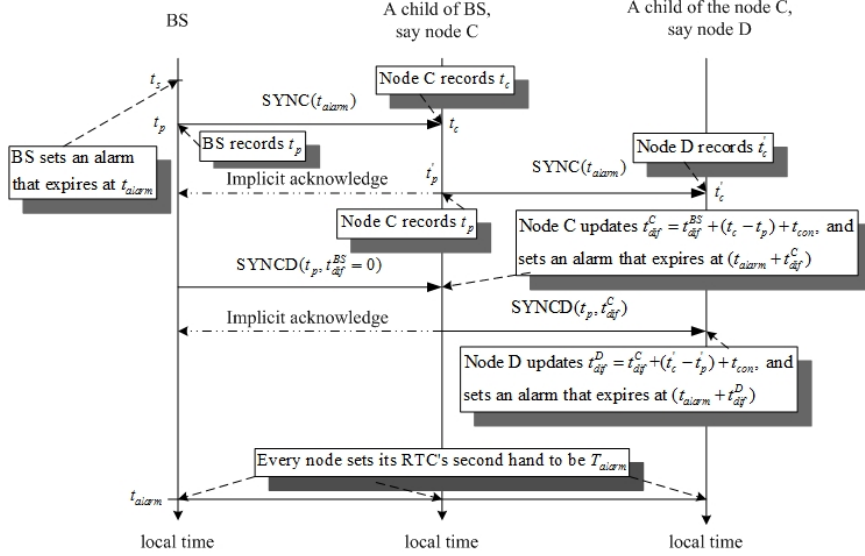


Fig. 3.7. Illustration of the synchronization protocol.

$t_{dif}$ . Thus the node can calculate the offset between itself and the BS by adding the offset between itself and its parent to the received  $t_{dif}$ . It would be obvious from the above description that HARMONIA does not compensate for the difference in drifts in the MCCs or the RTCs of two nodes, nor for the jitter in the interrupt handling times for the interrupts arising from the MaxStream signals.

Note that the  $t_p$  and  $t_c$  in the Phase 1 are recorded in a similar way that MAC layer time-stamping technique gets timestamps, but unlike in the MAC layer time-stamping, the value of  $t_p$  is transmitted in a different packet—SYNCD, not SYNC. This is because the MaxStream radio MAC firmware is not modifiable and we cannot embed the  $t_p$  into the SYNC.

### MaxStream Signaling on Bit Transmission and Reception

In our description above, we simplified the issue of signaling from the MaxStream radio to the microcontroller. In reality, what happens is depicted in Figure 3.8. On the transmitter side, the radio generates a pulse of width  $T_{TL}$  and on the receiver side, the radio generates a pulse of width  $T_{RL}$ . Trigger to the Chasqui microcontroller happen respectively on the rising edge and the falling edge. There is a time difference,



say  $t_{pulsedif}$ , between when the event is time-stamped at the transmitter and at the receiver end, since  $T_{TL} > T_{RL}$ . According to the MaxStream 9XTend OEM RF Module specification [32],  $t_{pulsedif}$  is ideally  $190\mu s$ . We reflect this using a parameter  $t_{con}$  and thus  $t_{dif}$  in phase 2 reception is updated as  $t_{dif} = t_{dif}^{rcv} + t_c - t_p + t_{con}$ . Here  $t_{con}$  is a constant intended to compensate a synchronization error offset obtained when without it. By this, we can compensate a signal propagation delay and a handling time for the interrupts by the MaxStream radio as well as  $t_{pulsedif}$ . We explain in Section 4.5 how  $t_{con}$  is experimentally measured.

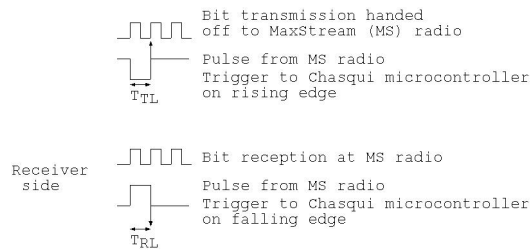


Fig. 3.8. The signaling from the MaxStream radio to the microcontroller. The signal on the transmit and on the receive side are used to take time-stamps which are used in HARMONIA.

### 3.3.3 Failure Handling

In this section, we discuss how HARMONIA can handle transient failures in either links or nodes. A node needs to detect the loss of any synchronization packet. For this it uses overhearing of its child's synchronization packet as an implicit acknowledgement (ACK).

After a node sends SYNC to its children, it sets a timer which goes off after  $t_{out}$  time within which it expects to overhear all its children sending SYNC to their own children. If the node does not overhear the SYNC packet(s) from one or more children, this is taken as an indication of failure and it sends the SYNC again. The

protocol has a bound  $N_{max}$  for which the process will be tried, within a slot, before declaring failure.

A parent node begins sending the SYNC<sub>CD</sub> packet to its children only after it has been assured that all its children have received the Sync packet, or that there has been a failure. The same technique is used by the node to detect and to handle failure in SYNC<sub>CD</sub>.

### 3.3.4 Packet Sequences and State Management

In HARMONIA, since retransmissions can occur, we need a way to allow the nodes which have already received a packet to disregard the same type of packet subsequently. Practically those state variables are managed in the following manner. Every node has two different kinds of round sequences: One is round sequence as a parent denoted by  $N_{rp}$  and the other is round sequence as a child which is  $N_{rc}$ . Those are both initially set to zero. Whenever BS initiates a new round of synchronization procedure, it increases the  $N_{rp}$  by 1. All SYNC and SYNC<sub>CD</sub> packets carry the node's  $N_{rp}$  value. A child accepts a synchronization packet with  $N_{rp}$  greater than or equal to its current  $N_{rc}$  if the packet is from its parent. When a node receives a  $N_{rp}$  value from its parent, it updates its own  $N_{rp}$  value to be the received one. It updates its  $N_{rc}$  value as  $N_{rc} = N_{rp} + 1$  after it sends SYNC<sub>CD</sub>. This will help the node disregard re-sends of the SYNC from its parent. All the round sequences are dealt with by doing modular arithmetic in implementation to handle variable overflow.

In addition to the round sequences, the SYNC packet should carry another type of sequence number for the ARQ operation denoted by  $N_{trial}$  which represents how many times the SYNC transmission has been tried so far including the current one. A parent does not know at what value of  $N_{trial}$  the SYNC packet will be received at each child. Therefore, it has to record all the instants at which the SYNC is sent with the corresponding value of  $N_{trial}$ , and send all these information in the SYNC<sub>CD</sub> packet. Each child remembers the value of  $N_{trial}$  in the SYNC it received, and finds

the corresponding time of sending the SYNC when it receives the SYNC.D. It uses this time to calculate  $t_{dif}$  in reception step of the Phase 2. For example, if node  $A$  had to send two SYNCs to satisfy its two children  $C_1$  and  $C_2$ . The times corresponding to these two sends are  $t_{p1}$  and  $t_{p2}$ . Then when the node  $A$  sends the SYNC.D, it has fields: *Trial 1:  $t_{p1}$ ; Trial 2:  $t_{p2}$ .*

### 3.3.5 Fast Recovery

In spite of trying  $N_{max}$  number of times within a slot, a node may be unable to synchronize all its children. Let us say node  $A$  is in such a situation. For this case, we introduce the feature of fast recovery. The fast recovery allows the node  $A$  to proactively initiate the synchronization procedure in the next slot, targeted only to its descendant sub-tree, using the value of updated  $N_{rp}$ . Thus node  $A$  does not have to wait for the BS to initiate the next synchronization round. Let us consider one of node  $A$ 's child nodes  $C_1$ . The fast recovery can happen because of any of the three reasons: (i) node  $C_1$  did not even receive the SYNC; (ii) node  $C_1$  received the SYNC but for some reason did not finish getting synchronized in the slot; (iii) node  $C_1$  is synchronized, but the implicit ACK has been lost to node  $A$ , or node  $A$  is trying to synchronize a sibling of node  $C_1$ . For case (i), no special treatment of the state variables  $N_{rp}$  or  $N_{rc}$  is needed for node  $C_1$  since these had not been incremented (the SYNC was not even received). For case (ii), node  $C_1$  decrements its  $N_{rc}$  before going to sleep so that it will accept the SYNC in the next slot. For case (iii), node  $C_1$  disregards the synchronization message and sends an explicit ACK to node  $A$  by transmitting a message called SYNCA. A node tries the fast recovery at most  $N_{maxtrial}$  times.

The fast recovery concept is powerful enough to handle the situation that a large network cannot all be synchronized in one slot. Rather the synchronization proceeds with as much of the network being synchronized initially as possible, and the unsynchronized parts of the network being handled through fast recovery.

### 3.3.6 Choice of Important Parameters

Here we discuss the tradeoffs in choosing the most important parameters in HARMONIA.

1.  $T_s$ : This is the time the BS waits after waking up to initiate the synchronization messages. This value has to be large enough to accommodate clock drifts that have built up between a parent and its child node. This is to ensure that the child node is awake to receive the synchronization message. But, it must be small enough that the synchronization can complete in the awake period of one slot. We find for the CSOnet a value of 2 seconds is reasonable.
2.  $N_{max}$ : This is the maximum number of times a node tries to synchronize its children nodes within a slot. A larger value will increase the reliability of the synchronization process, within one slot. However, it cannot be so large that the node arrives at the time to sleep within the slot before it has exhausted all  $N_{max}$  tries. Also there is a resource consumption that goes up with increasing values of  $N_{max}$ . This depends upon the frequency of transient failures in the network. We find a value of 3 works well for us.
3.  $T_{interval}$ : This is the time after which an alarm will be triggered to set the RTC, all together all through the network. This value should be large enough to give time for the entire network to be synchronized. However, the drift in the MCC in this time contributes to the synchronization error; therefore, it should be kept small. The value will depend on the scale of the network and we should set it to the smallest possible value that meets the above condition.
4.  $t_{bf}, t_{out}$ : The first is the backoff before sending a SYNC or SYNCD, the second is the time between the two phases. We have the condition  $t_{out} > t_{bf} + t_{proc}$ , where  $t_{proc}$  is the small time used in processing the synchronization message. This condition is required since otherwise a node may mistake that its SYNC message to its child has been lost when in reality the child was backing off before sending it along. The parameter  $t_{bf}$  should be chosen based on the network density, a higher density requiring a larger

value. The smaller the value of  $t_{bf}$  is, the faster will be the synchronization time of HARMONIA. For our case with a network density of 6 neighbors, we find  $t_{bf} = 100\text{ms}$  does not cause appreciable collisions. However, we are yet to do thorough experimentation to determine its setting.

## 3.4 Experiments

### 3.4.1 Experimental Methodology

We tested HARMONIA focusing on network-wide synchronization time and synchronization error with three different network topologies shown in Figure 3.9. However, in our experiments, in all three topologies, the nodes are actually placed within a short distance. This is to make the experiments feasible from a logistic standpoint. Therefore, we use software topology control to define the neighbor relations between the nodes. Thus, if node  $i$  is not connected to node  $j$  in the topology, it disregards all packets it receives from node  $j$  and vice-versa. Note that this still causes contention that would not be present in the actual network.

#### Metrics

We define the *synchronization error* for a node as the difference in its estimate of the BS's local time from the actual local time at the BS. For HARMONIA, synchronization error for node  $i$  corresponds to the difference in time when the BS and the node adjusts its RTC. We measure this error right after node  $i$  has been synchronized. For FTSP, a polling node queries the network nodes with a fixed period (3 seconds in our experiments). On being polled, a node  $i$  responds with its estimate of the BS's local time and at that instant the BS's own local time is also measured. The difference gives the synchronization error. Thus, for FTSP, there can be a delay of up to 3 seconds from the synchronization to the measurement.

We define the network-wide *synchronization time* as the time from when a round of the synchronization protocol begins to when all nodes in the network get all the packets required to make an estimate of the BS's local time and then have finished

the processing of the packets. In HARMONIA, the time ends when the last node has received SYNCED and done the processing (update its  $t_{dif}$ ) based on SYNCED.

For the experiments with HARMONIA, the microcontroller is programmed to generate a rectangular pulse at Pin 7 and Pin 10 on the Chasqui board at the instants when we have to pinpoint to calculate the synchronization error and the network-wide synchronization time. These two pins are connected to an oscilloscope. Specifically, a node generates the pulse at Pin 7 when it receives SYNCED for the first time in a round of synchronization procedure and has completed the attendant processing. A node generates a pulse at Pin 10 when it adjusts its RTC to get synchrony back. In case of BS, it generates a pulse at Pin 7 whenever it initiates a round of the synchronization procedure. Therefore, the synchronization error between a pair of nodes is the time gap in the rising edge of the pulse generated at Pin 10 and the network-wide synchronization time is measured by taking the time gap at Pin 7 between the BS and the last node to generate the pulse.

Table 3.1  
Values of parameters in HARMONIA used in the experiments.

$T_a$	$T_w$	$T_s$	$T_{alarm}$	$t_{bf}$	$t_{con}$	$N_{max}$
6s	5min	2s	4s	100ms	250 $\mu$ s	3

All the experimental results are statistics calculated from at least 10 points—in many cases, it is more; the 10 runs are used when experimental errors caused us to reject other runs. We have run experiments for HARMONIA for four different values for  $t_{out}$  ( $t_{out} = 150, 200, 250,$  and  $300$  (ms)) choosing other parameters as in Table 3.1. Regarding how to measure the value of  $t_{con}$ , we need to think about what the potential sources are for the synchronization error in HARMONIA: (i) the propagation delay; (ii) the frequency difference in MCC of each node, accumulated between the time the alarm is set to when the alarm fires, and (iii) the handling time for the

interrupts that the radio chip signals to record  $t_p$  and  $t_c$  with the SYNC packet. Let us use the uncorrected equation for synchronization:  $t_{dif} = t_{dif} + t_c - t_p$ . Then, the absolute value of the synchronization error between a sender (node  $i$ ) and a receiver (node  $j$ )  $E$  can be expressed as  $E = t_{con} + F$ , where if  $F$  is the error due to (ii),  $t_{con}$  covers the error due to (i), (iii), and  $t_{pulse\ dif}$ . We then measure the absolute value  $E'$  of the synchronization error with node  $j$  as sender and node  $i$  as receiver. Then  $E' = t_{con} - F$ . Hence, we can obtain  $t_{con}$  as  $t_{con} = (E + E')/2$ . Averaging over a number of experiments, we select  $t_{con}$  as  $250\mu s$ .

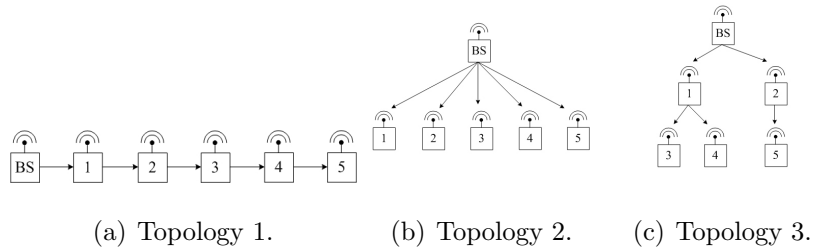


Fig. 3.9. Network topologies used for our experiments.

### 3.4.2 Network-wide Synchronization Time

Our main objective is to synchronize a network of sensor nodes running on a very low duty-cycle *quickly*—within the time period for which they remain awake—keeping the synchronization error among the nodes within a tolerable limit. Hence synchronization time is the primary metric for us.

For the experiment with HARMONIA, we vary the time between a SYNC and a SYNCED message, denoted as  $t_{gap}$ . This time is given by a time-out at the sender side ( $t_{out}$ ) followed by a back-off at the sender side (chosen in a random uniform manner from  $[0, t_{bf}]$ ). Therefore, the expected value of  $t_{gap} = t_{out} + t_{bf}/2$ . This calculation of  $t_{gap}$  assumes there is no retransmission. In our experiments, there are collisions and retransmissions, and the synchronization time value for HARMONIA is measured in

the presence of such events. It is only that the average value of  $t_{gap}$  would be higher in that case from what is plotted.

Figure 3.10 shows that HARMONIA can synchronize the three networks within several hundred milliseconds for all the chosen parameters. We can see from the figure that the synchronization time increases quite slowly with  $t_{gap}$  compared to FTSP as can be seen by comparing the result shown in Figure 3.11. Since HARMONIA pipelines the SYNC and SYNC-D transmissions, a node does not have to finish getting synchronized before it can act as a source of synchronization messages. Thus, the network-wide synchronization time is kept small. When there is no retransmission, increment in the network-wide synchronization time at each hop is due to the backoff, not the timeout. In this situation, the total synchronization time can be modeled as  $c+h \times b$ , where  $c$  is the constant cost due to the timeout at the BS and  $b$  is the variable cost which depends on the back-off and is multiplied by the number of hops  $h$ . Using this, we can roughly estimate how many hops HARMONIA can synchronize within a single slot. As an example, let us consider the case of  $t_{gap} = 200\text{ms}$  in Topology 1. Since  $c = t_{out} = 150\text{ms}$  and  $h = 5$  in this case, we have  $b = (673.5 - 150)/5 = 104.7$  (ms). On the other hand, HARMONIA needs to finish all the procedure within  $T_{interval}$  (currently set to 2 seconds). Using  $b = 104.7\text{ms}$ , we can therefore calculate  $h = (2000 - 150)/104.7 = 17.67$  (hops), which can be an estimate of the maximum hops in linear topology that can be synchronized within a slot. However, considering that the result in Figure 3.10 was obtained in a collision-prone environment (all nodes are in one-hop distance), the value of  $b$  will be much lower than 104.7ms in reality (nodes are sparsely deployed) and thus we can expect that the limit of the hops that can be synchronized within a slot would be larger than 17 hops with  $t_{gap} = 200\text{ms}$ . The remainder of the network that cannot be synchronized within one slot will be synchronized in the next slot, according to the fast recovery mechanism.

We ran some testbed experiments using Mica2 motes for the topologies shown in Figure 3.9 using FTSP to see if it can achieve our goal and also to compare FTSP with HARMONIA. In FTSP, each node periodically broadcasts the synchronization



packet (say with a period  $P$ ) containing the MAC layer time-stamp of the instant when the packet is sent. A node needs to receive  $N_R$  (8 by default) number of such packets to apply linear regression (to account for the clock drift) and get synchronized with the *root* node. Since the network-wide synchronization time, say  $T_N$ , is directly proportional to  $P$  and  $N_R$ , we reduced the values of these parameters as much as we could to see how fast FTSP can synchronize the network. For linear regression,  $N_R$  has to be at least 2. We found that the TinyOS timer does not fire when we reduced  $P$  below 10ms and therefore the minimum value for which we have the reading is  $P = 10\text{ms}$ .

Figure 3.11 shows the network-wide synchronization time for FTSP for the three topologies as a function of  $N_R$  and  $P$ . From this figure, we can see that  $T_N$  is even larger than  $N_R P N$  in reality where synchronization packets can collide. Except for one-hop network of Topology 2, the network-wide synchronization time is quite large for our purpose because we need to synchronize the network within at most 6 seconds when the nodes are awake. Furthermore, this figure also shows that  $T_N$  increases with the increase in the number of hops in the network. Thus FTSP out-of-the-box would not be suitable for deployment in CSOnet due to its performance in terms of network-wide synchronization time.

Although we do not provide the network-wide synchronization time for larger values of the synchronization period, note that Figure 3.11 shows that it increases linearly with the synchronization period. The slope of this linear relationship depends upon various factors like network topology, link reliabilities among the nodes, etc. Table 3.2 shows the slope of these lines along with the y-intercept value using linear regression.

First off, comparison between HARMONIA and FTSP would ideally have been done on the same platform. However, critical features of the protocols are dependent on the features of the specific hardware. Thus, HARMONIA depends on the signals from the MaxStream radio while FTSP depends on MAC layer time-stamping available in the Mica2 radio stack. Nevertheless, we see that the network-wide synchronization

time for HARMONIA is of the order of a few seconds in FTSP and it is in the order of a few hundreds of milliseconds in HARMONIA. For example, with Topology 1, which most closely resembles CSOnet topology, with  $t_{gap}=200\text{ms}$  and equivalently,  $P=200\text{ms}$ , FTSP is 7.4X and 9.8X slower than HARMONIA, for number of regression points 2 and 8 respectively. The improvement of HARMONIA increases with increasing values of the period. The improvement is 8.7X and 12.1X for  $t_{gap}=P=300\text{ms}$ . Note that the equivalence between  $t_{gap}$  and  $P$  is not perfect. In FTSP,  $P$  denotes a fixed period; in HARMONIA,  $t_{gap}$  is an expected value and this represents the gap between SYNC and SYNCNCD messages and not a period.

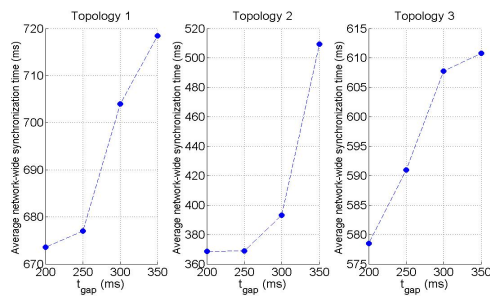


Fig. 3.10. Average network-wide synchronization time of HARMONIA.

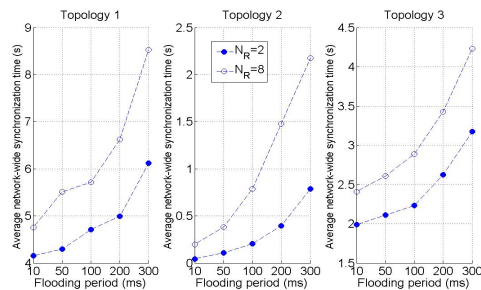


Fig. 3.11. Average network-wide synchronization time of FTSP.

Table 3.2

Slopes of the linear relationship between network-wide synchronization time and synchronization period observed in our experiments.

	Topology 1		Topology 2		Topology 3	
	$N_R=2$	$N_R=8$	$N_R=2$	$N_R=8$	$N_R=2$	$N_R=8$
slope (s/ms)	0.0054	0.0110	0.0015	0.0060	0.0031	0.0052
y-intercept (s)	4.00	4.64	0.02	0.09	1.89	2.30

Table 3.3

One-hop synchronization error.

	HARMONIA	FTSP ( $N_R = 8$ )
average	$16.77\mu s$	$1.5\mu s$
max	$38\mu s$	$3\mu s$

### 3.4.3 Synchronization Error

First, we measure the synchronization error in HARMONIA and FTSP in a single-hop network. In this, there are only two nodes. For this, HARMONIA results in an average synchronization error of  $16.77\mu s$ , while FTSP results in  $1.5\mu s$  as shown in Table 3.3.

Thus FTSP outperforms HARMONIA in terms of synchronization error. There are two primary contributory factors. First, we do not compensate for the differential drifts in the MCCs of two nodes. Note however that we are exposed to this effect only during the period  $T_{interval}$ . Second, we do not account for the jitter in interrupt handling that occurs when the MaxStream radio gives a signal on message transmission and on message reception. However, since the RTC has a high precision oscillator (with a drift of only 2 ppm), the synchronization error achieved by HARMONIA still means CSOnet can operate for extended periods of time between synchronizations. A simple computation for this can be formulated as follows. Consider that in CSOnet,

for a safety margin, we do not want any two nodes to be out of synchrony by more than 2 seconds. The current CSOnet deployment has a diameter of 20 hops. Therefore, in the worst case, a parent and a child node can be allowed to go out of synchrony by no more than  $2/20 = 0.1$  second. This is the worst case considering that the clock drifts between any parent-child pair are in the same direction and therefore the errors add up. Now, to calculate the frequency of HARMONIA’s synchronization rounds, we solve the following equation:  $38\mu\text{s} + 2\mu\text{s}/\text{second} \times x \text{ second} = 0.1 \text{ second}$  (we use the measured value of maximum synchronization error of HARMONIA as  $38\mu\text{s}$  and the fact that the RTC has a maximum drift of 2 ppm). Solving this equation, we get that HARMONIA must initiate a synchronization round every 13.88 hours in the worst case.

How HARMONIA will work in multi-hop networks can be seen from Figure 3.12, where the synchronization error at node  $i$  is the absolute value of the synchronization error between node  $i$  and the BS. We use the default value of  $t_{out} = 200\text{ms}$ . The synchronization error decreases from node 1 to node 2 in Topology 1. This can be explained by the fact that the relative synchronization error between the BS and node 1 has the opposite sign to that between node 1 and node 2. The sign of the synchronization error between a pair of nodes depends on the relative frequencies of the clocks of the two nodes and could be either positive or negative. Thus, the synchronization error in HARMONIA will not continuously build up as the number of hops from the BS increases. We can confirm this from the result of Topology 3, where node 3 has smaller synchronization error than node 1. We can also see from Figure 3.13 that the time gap between SYNC and SYNCED does not have a strong impact on the synchronization error. This is expected — the synchronization error will go up with  $T_{interval}$  and with message load that would cause a higher rate of interrupts at a node. With a really small value of  $t_{gap}$ , the second effect could be seen, but this was not observed during the experiments.

From Figures 3.14 and 3.15, we see that the synchronization error in FTSP is very small (the results for Topology 3 are omitted since they are similar to that of

Topology 2). The error tends to increase when the synchronization messages are sent too quickly (faster than 100ms) except for the one-hop network (Topology 2). However, the error is always within the tolerable limit for CSOnet. Also as the number of regression points is increased, the synchronization error decreases, as expected.

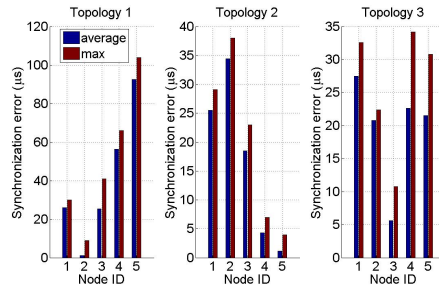


Fig. 3.12. Synchronization error of HARMONIA for the different nodes in Topology 1.

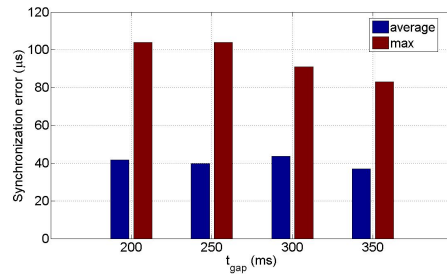


Fig. 3.13. Synchronization errors of HARMONIA with different values of  $t_{gap}$  in Topology 1.

### 3.5 Discussion

#### On-demand synchronization

HARMONIA can be easily extended to handle on-demand synchronization in which a node requests its parent for initiating synchronization. It sends a SYNC\_REQ packet which causes the parent to send the SYNC packet thereby initiating the first phase of

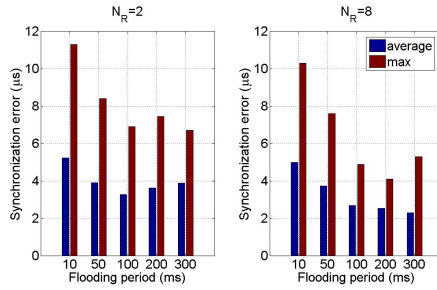


Fig. 3.14. Synchronization error of FTSP in Topology 1.

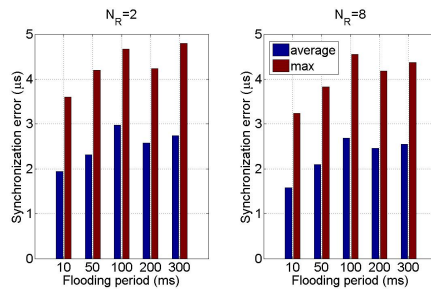


Fig. 3.15. Synchronization error of FTSP in Topology 2.

the two phase protocol. The child will send the request if it has not been synchronized for greater than some multiple of the duration of a round. This threshold time is such that if the node does not get synchronized then *complete asynchrony* may result, meaning the node's  $T_a$  wake period completely misses the wake period of a neighbor. With the on-demand synchronization, the node initiating the request and the sub-tree rooted at that node will be synchronized. This function would be important when a node or a link recovers from a failure and the synchronization process had occurred during the failure duration.

### Issues with MAC layer time-stamping

MAC layer time-stamping is quite widely used in synchronization protocols, e.g., TPSN and FTSP. In CSOnet's Chasqui node, MAC-layer time-stamping was not possible due to the proprietary closed-source nature of the MAC protocol. However,

even if it had been possible, there are some cautions to using the technique. On the receiving side, as soon as a (synchronization) packet comes in, it is time-stamped at the MAC layer and put in a queue. A queue is required since for fast radios, more than one packet may come in before being consumed by the synchronization protocol. However, the packet itself may be discarded by the receiver if it fails the CRC check. Then, in the absence of identifying information attached to the time-stamp, the receiver has no way of discarding the timestamp that corresponds to the discarded packet. This issue was hinted at in [29] and in subsequent postings on the TinyOS help forum [28].

### **Synchronization in sparse (almost) linear networks**

The CSOnet is almost linear in most parts when we consider the Rnodes as the network nodes. This means that HARMONIA can have a low back-off time since a parent has one or only a few children nodes. However, in our experiments, we have the Chasqui nodes placed on a table close by to each other and use software topology control. This increases the likelihood of collisions. Additionally, MaxStream radio typically sends larger-sized packets than the Chipcon radios making the packets more susceptible to collisions. The maximum packet size in MaxStream is 2048 Bytes while in CC2420 it is 128 Bytes. Therefore, in actual deployment, we expect that HARMONIA will have a lower synchronization time since it will incur smaller back-off times.

### **Reliance on topology**

HARMONIA relies on some other middleware service (like the stateless gradient-based routing in the case of CSOnet) to get the knowledge about the tree structure used for communication. FTSP does not require this knowledge. Although at first glance this may appear to be a drawback of HARMONIA, we believe this prerequisite about the knowledge of the topological structure is essential to tradeoff generality for synchronization speed. It is because of this knowledge of the topological structure that a node  $n_1$  can quickly start synchronizing its children after receiving the synchronization packet from its parent. It just needs to backoff a short random time depending

upon the number of nodes present at the same depth of the tree as  $n_1$  to prevent collision. Without such knowledge, after a node receives a synchronization packet, it has to conservatively estimate the backoff time or wait for a timer with a sufficiently long interval to fire before starting to broadcast its own synchronization packet. Although HARMONIA relies on the knowledge of the tree structure, it works with any such structure as long as it gets this information from some other middleware application. During the network operation, if the tree structure is changed due to node/link failure, HARMONIA will work with the new topology by adjusting the backoff period of a node.

### **Handling permanent failures**

If the external service that creates the topology runs relatively infrequently and a node fails permanently or for a long time, the subtree rooted at the failed node may lose synchrony. However, HARMONIA can be adjusted such that a node does not need to wait for the topology service to reconstruct the tree if its parent has a permanent failure or a failure that persists for a long time. In such a situation, the children of the failed node can select a new parent by broadcasting the *ReqToChangeParent* packet containing the information about the depth of this node in the tree. Since many nodes at different depths of the tree can receive this request, they may concurrently try to be the parent of the requesting node. To avoid this, each node replies to this request if its depth is smaller than that of the requesting node (i.e. if it is higher up in the tree than the requesting node) after a random interval proportional to the difference between its depth and that of the requesting node. This causes the nodes which are in the closest tier *above* the requesting node in the tree to respond to the request first and become the parent. If another node overhears this response, it will suppress its response. This will allow the sub-tree to be synchronized before the topology has been repaired.



## 4. TIMELY EVENT REPORTING IN SENSOR NETWORKS

So far, we have focused on developing a reliable protocol for wireless sensor networks (WSNs). In this chapter, we now look into security threats of WSNs. One of the important application scenarios for WSNs that have a prominent security threat is the domain of event monitoring systems [10, 33]. In event monitoring, the WSN is deployed over a region where some phenomenon is to be monitored. For example, a number of sensor nodes could be deployed over a battlefield to detect enemy intrusion. When the sensor nodes detect the event being monitored, the event is reported to a base station (BS), which can then be used by a network operator to take appropriate action.

Significant work to date has focused on making data gathering (equivalently, event monitoring) energy efficient. However, in some scenarios, energy consumption is not an issue because the nodes can be plugged to a source of energy. An important case for this is provided by monitoring in the smart grid, *e.g.*, to detect if a powerline has been shut off [34]. Here, the sensor nodes are installed on electric poles. A wireless surveillance system using infrared (IR) beam sensors is another example for such scenarios, since many of commercially available wireless IR beam sensor nodes are plugged into existing electrical outlets [35]. In many such deployments, there exist financial incentives to disrupt the event monitoring, which includes the possibility of delaying the receipt of the monitored event at the BS.

However, it is a difficult task to secure the event reporting process when the monitoring network is under the attack. The sensor nodes are inherently vulnerable to attacks because they are usually deployed in non-protected environments. The adversary can often easily access the sensor nodes, and may even compromise them by reprogramming. Once some sensor nodes in a monitoring sensor network are compro-

mised, they may prevent other legitimate sensor nodes from reporting information in a timely manner. On the other hand, if the BS fails to get a critical event report due to such a malicious activity, the result could be catastrophic. For example, during a powerline shutoff in smart grids, failure in taking action in time may lead to an electrical blackout over a large area.

For this reason, we propose SEM, a secure event monitoring protocol that can work even when there exist compromised nodes in the network. We are interested in a multi-hop network scenario where all sensor nodes except the BS node can be compromised. The compromised nodes can launch arbitrary attacks in a Byzantine manner, such as dropping, modifying, and delaying the event report. They may also arbitrarily collude among themselves. Even in such a hostile environment, SEM can provide the following provable security guarantees:

- As long as the compromised nodes want to avoid being detected, a legitimate sensor node can report an event to the BS within a bounded time.
- If the compromised nodes launch an attack that causes the event report not to arrive at the BS within the bounded time, the BS can identify a small set of nodes that is guaranteed to contain at least one compromised node.

We believe that in many practical scenarios, the adversary has the incentive to keep the compromised nodes from being detected, since otherwise, the network operator will be able to remove or reprogram the detected compromised node one by one, eventually defeating the attack. Hence, the above security guarantees are meaningful and useful to attain.

Our design of SEM is parsimonious in its usage of resources - both in the usage of expensive computations for cryptographic operations, and in generating additional network traffic. Specifically, SEM makes a distinction between a normal operation mode and a diagnosis mode (when some attack has been detected and culprit nodes are sought to be identified), by which SEM only uses the indispensable level of asymmetric cryptography in the normal operations.

The problem that we address here has been solved before but with the following caveats. Much of the early work focused on detecting malicious actions done on packets by routing nodes, through the mechanism of overhearing [36]. However, none of this work can handle colluding Byzantine adversaries. Subsequent work - ODSBR [37] and PAAI [38] - can handle colluding Byzantine adversaries and can, with careful tuning of timeouts, be made to handle delay attacks. However, they require use of onion-manner asymmetric signatures for *all* their transmissions. This makes such schemes infeasible for resource-constrained wireless networks because they take too much time, bandwidth, and packet length.

The remainder of this chapter is organized as follows. Section 2.2 gives an overview of the previous works for event monitoring. In Section 4.1, we formally state our objective, assumptions, and notations. In Section 4.2, we discuss what approach we have to take to achieve our objective. In Sections 4.3 and 4.4, we present SEM in detail, and discuss the relevant miscellaneous issues. We provide experiment results for SEM in Section 4.5.

## 4.1 Problem Statement

We consider a multi-hop wireless sensor network that consists of a base station (BS) node and a number of sensor nodes. A sensor node is in charge of sensing a delay-sensitive event like a power line shutoff. A network operator monitors the sensor network through the BS that attempts to collect the events (if any) from the sensor nodes. It is important that if an event occurs at a sensor node, the BS gets informed of it as soon as possible in order for the network operator to take action in time.

Since some sensor nodes are more than one hop away from the BS, the events sensed at such nodes are reported to the BS through a multi-hop routing path as shown in Figure 4.1. However, if a node in the middle of the routing path is compromised, the compromised node may drop/modify the event report, or delay it for a very long time. This problem cannot be resolved even if a legitimate node sends

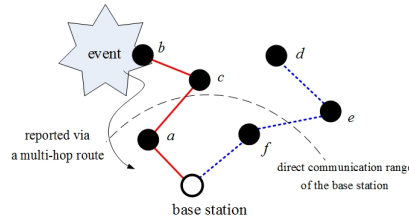


Fig. 4.1. Multi-hop routing paths to collect events.

the event by flooding over the entire network, because the BS may be encircled by compromised nodes. For example, in Figure 4.1, if nodes  $a$  and  $f$  are compromised, no other sensor nodes can successfully report an event to the BS even by flooding. This is often easy to achieve because networks have significant fan-in close to the BS, leading to only a few nodes close to the BS.

**Objective:** The objective of this chapter is to overcome the aforementioned difficulty in monitoring delay-sensitive events. We want to devise a protocol that provides the following provable security guarantees. (1) As long as the compromised nodes want to stay undetected, a legitimate node can report an event to the BS within  $P$  time units; (2) If the compromised nodes launch an attack that causes the event report from a legitimate node not to reach the BS within  $P$  time units, the BS can identify a small set of nodes that is guaranteed to contain at least one compromised node.

Note that we do not guarantee that all sensor nodes will report a detected event to the BS within  $P$  time units: once a node is compromised, the event occurred at the compromised node may not be reported. Intuitively, if the reporting node is itself compromised, there is no way to get informed of the event that occurs at the node. However, if a phenomenon is sensed by multiple sensor nodes, say  $n$  sensors, and at least one of them is legitimate, it is still guaranteed to be reported to the BS within  $P$  time units by our protocol. To make our protocol out of use, the adversary needs to compromise all  $n$  sensor nodes. In many practical scenarios, the value of  $n$  may be large (*e.g.*, all sensor nodes around an invasion route by the

enemy movement). In contrast, without our protocol, the adversary only needs to compromise the neighboring nodes of the BS to nullify the monitoring system. Thus, security benefit from our protocol is significant.

**Assumptions:** (1) Only the BS can be trusted, *i.e.*, any sensor node can be compromised and then behave maliciously. The compromised nodes may arbitrarily collude among themselves. (2) Byzantine adversary model is considered, *i.e.*, the compromised nodes can take arbitrary malicious actions. For example, the compromised node may drop, modify, or delay the event reports. Further, they may launch a jamming attack to prevent some nodes in the network from communicating. However, for ease of presentation, we assume for the time being that there is no jamming attack. We will relax this assumption in Section 4.4. (3) All links are bi-directional. (4) The time to transmit a packet across one hop is bounded above by  $B$  time units. (5) We assume that transient packet loss can be recovered by a lower-layer automatic repeat request (ARQ) mechanism. Further, we assume that nodes do not fail unless they are compromised. We will also relax these assumptions in Section 4.4. (6) A node shares a secret key with each of its neighboring nodes, which will be used for a symmetric cipher.

**Notations and Definitions:** (1)  $[X_1, \dots, X_n]$  returns the concatenation of the input strings  $X_1, \dots, X_n$ . (2) We use  $N(a)$  to denote 'node  $a$ '. (3)  $S_a(X)$  returns a signed message for the input string  $X$  made by  $N(a)$  that is defined as  $[X, \text{SIG}_a(X)]$ , where  $\text{SIG}_a(X)$  is a signature created on the input string  $X$  by  $N(a)$  using its private key. (3)  $C_{a_1, a_2}(X)$  returns a ciphered message for the input string  $X$  made by  $N(a_1)$  using the secret key shared with  $N(a_2)$ . (4)  $ID(a)$  denotes the identification (ID) of  $N(a)$ . (6)  $SET(a_1, \dots, a_n)$  denotes the set of nodes  $N(a_1), \dots, N(a_n)$ . (7) A suspicious set is a set of nodes that includes at least one compromised node in it.

## 4.2 Straw-Man Protocols

In this section, we first consider two straw-man protocols, and discuss the issues of each protocol. This discussion will motivate the proposed protocol in the next section to achieve our objective.

**Straw-man protocol  $A_1$ :** As we have seen in Section 4.1, if we simply let the sensor nodes send a report to the BS when they detect an event, the compromised node in the middle of route may drop or tamper with this event report. Thus, the BS may not know if the event has happened. To resolve this issue, the protocol  $A_1$  requires the BS to get a report from a sensor node every  $P/2$  time units. Since a node sends a report to the BS every  $P/2$  time units, the BS can be notified of an event within  $P$  time units in the worst case after the event. If the BS does not get a new report within  $P/2$  time units from a certain node, it can recognize that some compromised node has dropped or delayed the report from the node.

Here, the nodes have to send a report periodically even without any event to report. Further, the report should be signed by the reporting node, since otherwise the compromised node may alter the report. Hence, this protocol incurs a large overhead. What is worse with the protocol  $A_1$  is that the BS cannot identify who the malicious node is after detecting a dropped or delayed report. This is because the BS has to rely on other nodes' opinions to locate the culprit, but the BS cannot trust anybody except itself. Further, after the BS detects a malicious activity, the compromised node may start acting normally. Thus, without an addition defense mechanism in place, the BS cannot identify who is the malicious node after detection.

**Straw-man protocol  $A_2$ :** The weakness of the protocol  $A_1$  leads to us to think about an improved protocol by which the BS can detect a malicious behavior and collect critical evidence at the same time. We note that we can make such a protocol by adopting the acknowledgment (ACK) mechanism with staggered timeout, which is used in ODSBR [37]. We refer to this protocol as protocol  $A_2$ .

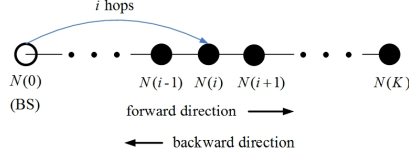


Fig. 4.2. Line network model to collect events.

The protocol  $A_2$  organizes sensor nodes to multiple line networks that originate from the BS. For example, in Figure 4.1, the BS,  $N(a)$ ,  $N(b)$ , and  $N(c)$  forms a line network, while  $N(d)$ ,  $N(e)$ ,  $N(f)$ , and the BS forms another line network. In the protocol  $A_2$ , the BS collects events from each line network separately. Thus, without loss of generality, we consider one line network model of  $K$  hops as shown in Figure 4.2. In this model, we denote by  $N(0)$  the BS, and by  $N(i)$  the sensor node at a  $i$ -hop distance from the BS. Define  $I = \{0, 1, \dots, K\}$ . We refer to the direction from  $N(0)$  to  $N(K)$  as the forward direction, and the opposite as the backward direction.

In the protocol  $A_2$ ,  $N(0)$  (*i.e.*, the BS) sends out a probe token (PT) in the forward direction through the line network. The PT is defined as

$$\text{PT} = S_0([ID(0), R, Q]), \quad (4.1)$$

where  $R$  denotes the routing information of the line network model in Figure 4.2, and  $Q$  is an integer variable, called round sequence. Every time  $N(0)$  sends out the PT, it increases the value of  $Q$  by 1 to prevent the old PT from being re-used in replay attacks. When  $N(K)$  receives the PT, it generates an ACK packet. On receiving the ACK,  $N(i)$  for any  $i \in I \setminus \{0, K\}$  forwards the ACK in the backward direction. If  $N(i)$  has an event to report, it can embed the report into the ACK. Hence, if  $N(0)$  receives the ACK from  $N(K)$ , then no packet drop has occurred, and  $N(0)$  can be informed of any embedded event. It is easy to see that if each round is completed within  $P/2$  time units,  $N(i)$  for any  $i \in I \setminus \{0\}$  can report a sensed event to  $N(0)$  within  $P$  time units.

To detect a malicious node that drops/delays either the PT or the ACK,  $N(i)$  for any  $i \in I \setminus \{K\}$  starts an ACK timer  $t_a(i)$  immediately after transmitting the PT, whose timeout value is set to  $B_a(i)$ . If the ACK has not been received from  $N(i+1)$  before  $t_a(i)$  goes off,  $N(i)$  gives up waiting and generates its own ACK packet. Ideally, only when  $N(i+1)$  is a compromised node that drops either the PT or the ACK, or delays it beyond a certain limit,  $N(0)$  receives the ACK generated by  $N(i)$ . Note that to ensure this property, if  $N(i+1)$  is legitimate,  $N(i+1)$  should be able to send the ACK to  $N(i)$  before  $t_a(i)$  expires. Due to this reason, the nodes calculate the maximum time required for the PT to traverse from themselves to  $N(K)$  and for the ACK to return to the current position along the reverse route. Then,  $N(i)$  for any  $i \in I \setminus \{K\}$  sets the timeout  $B_a(i)$  to this maximum value, which is equal to

$$B_a(i) = (K - i - 1)B + (K - i)B. \quad (4.2)$$

Note that  $B_a(i)$  is larger than  $B_a(i+1)$  by  $2B$ . As Figure 4.3 illustrates, this allows  $N(i+1)$  to send  $N(i)$  the ACK before  $t_a(i)$  expires, which can be either the one that  $N(i+1)$  has received from  $N(i+2)$ , or the one that  $N(i+1)$  generates on its own. More formally, we can state this property as the following lemma.

**Lemma 1** *For any  $i \in I \setminus \{K\}$ , if both  $N(i)$  and  $N(i+1)$  are legitimate,  $N(i)$  has no reason to generate its own ACK.*

**Proof** Remember that nodes start the ACK timer right after transmitting the PT. Since  $N(i+1)$  holds the PT for  $B$  time units at most,  $t_a(i+1)$  starts within  $B$  time units after  $t_a(i)$  started. This implies that  $t_a(i+1)$  goes off at least  $B$  time units earlier than  $t_a(i)$  does, since  $B_a(i) = B_a(i+1) + 2B$ . Thus,  $N(i+1)$  can always send an ACK to  $N(i)$ , whether it is what  $N(i+1)$  generates on its own when  $t_a(i+1)$  expires, or it is what  $N(i+1)$  has received from  $N(i+2)$  before  $t_a(i+1)$  expires. Hence,  $N(i)$  should not be the node that generates its own ACK. ■



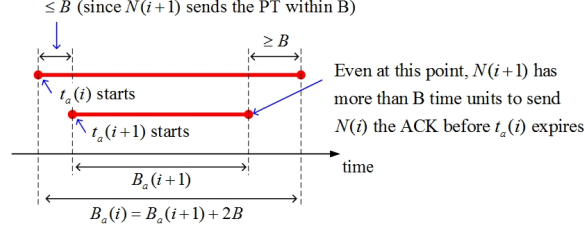


Fig. 4.3. Staggered timeout.

To prevent the compromised nodes from modifying the ACK (thereby falsely accusing legitimate nodes), or tampering with an event report in the ACK, the nodes are required to sign on the ACK. In detail, the ACK generated by  $N(i)$  is defined as

$$\text{ACK} = S_i([ID(i), Q, E]), \quad (4.3)$$

where  $E$  denotes the event report. If nodes do not have anything to report, they leave  $E$  as  $E = \text{NULL}$ . When  $N(i)$  forwards the ACK from  $N(i+1)$  to  $N(i-1)$  (*i.e.*, it does not generate its own ACK), the ACK sent by  $N(i)$ , denoted by  $\text{ACK}(i)$ , is defined as

$$\text{ACK}(i) = S_i([\text{ACK}(i+1), ID(i), E]). \quad (4.4)$$

Note that the ACK is signed by the nodes in an onion manner, *i.e.*, the ACK sent by a node encapsulates the ACK (if any) sent by higher-indexed nodes. Thus, a compromised node cannot modify or forge the ACK from any higher-indexed node, unless they collude with each other. As a precaution against forging the sender ID,  $N(i)$  sends  $C_{i,i+1}([ID(i), Q])$  along with the PT. Similarly,  $N(i)$  sends  $C_{i,i-1}([ID(i), Q])$  when sending the ACK to  $N(i-1)$ . Now, the protocol  $A_2$  can guarantee the following. Recall that a suspicious set contains at least one compromised node.

**Proposition 4.2.1** *If  $N(0)$  gets the ACK generated by  $N(i)$  for  $i \neq K$ ,  $SET(i, i+1)$  is a suspicious set.*

**Proof** Assume that  $SET(i, i+1)$  is not a suspicious set. Then,  $N(i)$  and  $N(i+1)$  are both legitimate. Since no compromised node can forge the signature of a legitimate

node, the ACK that  $N(0)$  has received is indeed generated by  $N(i)$ . This implies that  $N(i)$  has received the PT, and thus forwarded the PT to  $N(i+1)$ . Hence, by Lemma 1,  $N(i)$  cannot be the node that generate its own ACK. This is a contradiction. ■

Hence, we can see that the protocol  $A_2$  achieves our objective: it can collect the event reports from sensor nodes within  $P$  time units if there is no attack; Otherwise, it can identify a suspicious set that contains at least one compromised node. However, note that the protocol  $A_2$  always requires the sensor nodes to send a signed packet as in (4.4), *i.e.*, the protocol  $A_2$  needs to use an expensive ACK mechanism signed in an onion manner every  $P/2$  time units, whether or not an event occurs. This is important to provide the guarantee to identify a suspicious set when an event is not received in time. Since the occurrence of events is unpredictable and the PT generation by the BS has to be periodically done, the sensor nodes are required to sign on the ACK in an onion manner even though they may not have anything to report. However, this is very expensive in normal operations (when there is no attack) since the onion-manner signing technique causes a heavy computation overhead for the ACK. Further, it results in a large payload size for the ACK that often needs to be fragmented into multiple packets.

### 4.3 Proposed protocol: SEM

The issue with the protocol  $A_2$  is that it requires an expensive onion-manner signed ACK mechanism even in normal operations. Ideally, we would like to design a protocol that has low overhead in normal scenarios when there is no attack, and only invokes the heavier overhead when an attack is launched. For this purpose, we now propose our new protocol, called SEM. The approach of SEM is to first detect if something bad happens, *i.e.*, if there exists a compromised node that hinders the event collecting operation. If the event collecting operation looks fine, then SEM cancels the ACK timers at the sensor nodes. Thus, SEM uses the expensive onion-manner signed ACK mechanism only when some malicious activity that disturbs the

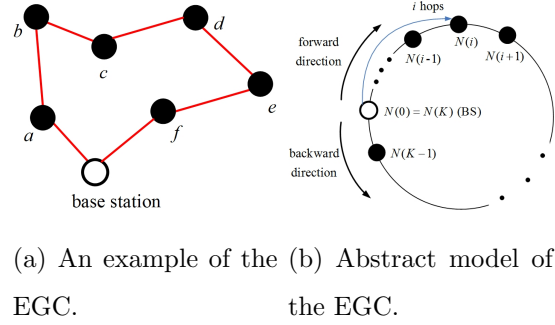


Fig. 4.4. Event gathering circle (EGC).

BS's event collection process is detected. By this, SEM can provide the same security guarantees as the protocol  $A_2$ , while eliminating the heavy communication overhead to send the ACK under normal legitimate scenarios.

Towards this end, SEM organizes the nodes into a circular network that passes through the BS as shown in Figure 4.4(a) (instead of the line network). We refer to the circular route as event gathering circle (EGC). If the BS finds a situation that it cannot include all the sensor nodes in the network into one single EGC, *e.g.*, due to scalability considerations, the BS can form multiple EGCs, and collect events from each EGC independently. As long as each sensor node belongs to at least one EGC, the BS can collect the event reports from the entire network. Thus, without loss of generality, we consider the case that there is only one EGC in the network that is modeled as a circular route of  $K$  hops as in Figure 4.4(b). Note that in this model,  $N(i)$  denotes the sensor node at a  $i$ -hop distance from the BS in the forward direction, thus implying that both  $N(0)$  and  $N(K)$  represent the BS.

#### 4.3.1 Detail of SEM

The basic idea of how SEM detects the existence of a compromised node is as follows. Every  $P/2$  time units (or less), the BS node  $N(0)$  checks the EGC in the same way as the protocol  $A_2$ . Namely,  $N(0)$  sends out the PT in the forward direction

of the EGC. When  $N(K)$  receives the PT, it generates a signed ACK packet. Nodes set up the timeout of the ACK timer similarly to (4.2). If  $N(i)$  for any  $i \in I \setminus \{0, K\}$  receives an ACK from  $N(i + 1)$  within the timeout,  $N(i)$  signs on the ACK and forwards it to  $N(i - 1)$ ; Otherwise,  $N(i)$  generates its own ACK. However, in SEM, if a sensor node  $N(i)$  has an event to report, it embeds the event report into the PT instead of the ACK as

$$\text{PT}(i) = S_i([\text{PT}(i - 1), ID(i), E]), \quad (4.5)$$

where  $\text{PT}(i)$  denotes the PT sent by  $N(i)$ . Note that  $N(i)$  embeds the event reports into the PT only when it has an event to report.<sup>1</sup> Since the event report is carried by the PT, the compromised node may want to drop or delay the PT to prevent an event from being reported in time to  $N(K)$ , which is also the BS. However, note that since  $N(0) = N(K)$ , the BS node  $N(K)$  can measure the circulation time  $t_c$  that is defined as the time value of the ACK timer  $t_a(0)$  when  $N(K)$  receives the the PT. If all the sensor nodes in the EGC are legitimate, the circulation time  $t_c$  should be less than or equal to  $(K - 1)B$ . Hence, if the circulation time  $t_c$  is larger than  $(K - 1)B$ , it is certain that there exists at least one compromised node in the EGC. However, even in such a case, as long as the value of  $t_c$  is no larger than some threshold time, say  $T_{th}$ , which is less than  $P/2$ , the BS may skip identifying the compromised node because the BS can still collect the event within  $P$  time units after the event occurred. Therefore, if  $t_c \leq T_{th} (< P/2)$ , the BS cancels the ACK timers at the sensor nodes, through which the BS prevents the nodes from doing the expensive ACK processing.

For this reason, we extend the timeout set up in (4.2) by a time margin  $T_m$ , in order to give enough time for the BS to cancel the ACK timers at the sensor nodes when  $t_c \leq T_{th}$ . Let  $B'_a(i)$  be the new timeout of the ACK timer  $t_a(i)$ . Then, we can express  $B'_a(i)$  as

$$B'_a(i) = B_a(i) + T_m. \quad (4.6)$$

---

<sup>1</sup>This feature introduces a new vulnerability that a compromised node may simply remove the event reports, since the BS has no way to know if there exists an event report before receiving it. We explain how to resolve this issue at the end of this section.

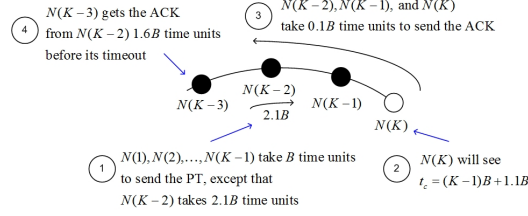


Fig. 4.5. An example to show that a compromised node may hold the PT for more than  $B$  time units without being identified.

In addition, we let  $N(K)$  start to send an ACK in the backward direction  $T_m$  time units after  $N(K)$  receives the PT. On the other hand, it is easy to see that even after such a change, Lemma 1 still holds, and thus Proposition 4.2.1 also still holds, *i.e.*, if  $N(0)$  gets the ACK generated by  $N(i)$  for  $i \neq K$ ,  $SET(i, i + 1)$  is a suspicious set. Canceling the ACK timers at the sensor nodes is done in such a way that  $N(0)$  starts a new round. In detail, if  $t_c \leq T_{th}$ , the BS node  $N(0)$  sends out the PT of the new round when the value of the ACK timer  $t_a(0)$  in the current round reaches  $T_{th}$ . This new round's PT acknowledges the old round's PT, and thus the sensor nodes can safely cancel their old ACK timer when receiving the new round's PT. Here, we should ensure that the value of  $T_m$  is long enough so that the new round's PT can arrive at the sensor nodes before their ACK timer started in the old round expires.

If  $t_c > T_{th}$ , the BS node  $N(0)$  stops circulating the PT, *i.e.*  $N(0)$  does not try to cancel the ACK timers at the sensor nodes. Thus,  $N(0)$  will receive the ACK signed by the sensor nodes in the onion manner. However, a wrong choice for the value of  $T_{th}$  may lead to a situation that the BS cannot identify a suspicious set even after receiving the ACK. To see this, consider the following example (see Figure 4.5), where we set the threshold  $T_{th}$  as  $T_{th} = (K - 1)B + B$ , *i.e.*, the BS will investigate the EGC using the ACK mechanism if the circulation time  $t_c$  is larger by a margin  $B$  over the legitimate maximum bound,  $(K - 1)B$  time units. Suppose that  $N(K - 2)$  is a compromised node, and it holds the PT for  $2.1B$  time units, while other nodes consume  $B$  time units to send the PT. Then,  $N(K)$  will find  $t_c > T_{th}$ , and thus

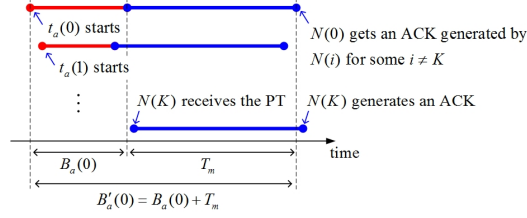


Fig. 4.6. Threshold  $T_{th}$ .

generate an ACK after  $T_m$  time units, without initiating a new round. However, to deliver the ACK, nodes may need less than  $B$  time units, depending on its current computation load and the channel condition. Suppose that every node takes just  $0.1B$  time units to send the ACK. In this situation, it is easy to check that  $N(K-2)$  can forward  $N(K-3)$  the ACK generated by  $N(K)$  before the ACK timer of  $N(K-3)$  expires. Figure 4.5 illustrates this situation. Therefore, the BS cannot find anything wrong with the compromised node  $N(K-2)$ , since it will get the ACK signed by all nodes in the EGC including  $N(K)$ . Namely, although  $N(K-2)$  holds the PT more than  $B$  time units, and the BS detects that  $t_c > T_{th}$ ,  $N(K-2)$  can avoid being identified.

Therefore, we now have to answer the following two questions:

- What should be the appropriate value for the threshold  $T_{th}$  such that if  $t_c > T_{th}$ , the BS can always identify a suspicious set?
- What should be the appropriate value for the time margin  $T_m$  by which the BS can cancel the ACK timers at the sensor nodes before they go off?

Note that if we can successfully answer these two questions, SEM can achieve our objective, and only use the expensive onion-manner signed ACK mechanism when we need to identify the compromised nodes.

In Proposition 4.3.1, we are looking to determine what is the smallest value of  $T_{th}$  that we can set and not run into a problem, namely, that the BS is unable to identify a suspicious set when  $t_c > T_{th}$ . Note that as  $T_{th}$  increases, the time required

for collecting event reports from a node to the BS goes up linearly (the exact relation is given in Corollary 1) and thus there is no incentive to make  $T_{th}$  large.

**Proposition 4.3.1** *If we set the threshold  $T_{th} = B_a(0)$ , where  $B_a(0) = (2K - 1)B$  (given by using (4.2) and setting  $i = 0$ ), the BS can always identify a suspicious set when  $t_c > T_{th}$ .*

**Proof** Suppose that  $N(0)$  receives the PT after  $t_c > T_{th}$ . Then, the time left in the ACK timer of  $N(0)$  is  $B'_a(0) - t_c = B_a(0) + T_m - t_c < B_a(0) + T_m - T_{th} = T_m$ . Namely, the time left in the ACK timer of  $N(0)$  is less than  $T_m$ . However, as mentioned below (4.6),  $N(K)$  starts to send its own ACK only  $T_m$  time units after it receives the PT. Therefore,  $N(0)$  will not receive the ACK generated by  $N(K)$ , and instead it will receive the ACK generated by some other node  $N(i)$  for  $i \neq K$ , as depicted in Figure 4.6. In that case, by Proposition 4.2.1,  $SET(i, i + 1)$  is a suspicious set. Note that  $N(0)$  will not receive multiple ACKs generated by multiple nodes. This is because of the protocol feature that if a node  $N(i)$  receives an ACK within  $B'_a(i)$  from  $N(i + 1)$ , then  $N(i)$  will put its own signature in an onion manner on that received from  $N(i + 1)$  and not generate its own ACK packet. ■

The goal of Proposition 4.3.2 is to answer the second of the two questions raised above. Proposition 4.3.2 gives us a lower bound to setting the time margin  $T_m$ . A larger value of  $T_m$  will mean, in the case of an attack, a longer time for a node to generate an ACK in the backward direction. This would mean a longer time for the BS to get the ACK and therefore to identify the suspicious set. Consequently, we would like to set  $T_m$  to be as small as possible.

**Proposition 4.3.2** *If we set the time margin  $T_m$  as  $T_m \geq 2T_{th}$ , then no legitimate sensor node will generate its own ACK when  $t_c \leq T_{th}$  for every round.*

**Proof** We prove this by focusing on a typical round  $Q$ . We seek to show that the earliest time at which the ACK timer of any node expires is *after* the latest time at which the ACK timer can be canceled at that node. Recollect that the ACK timer

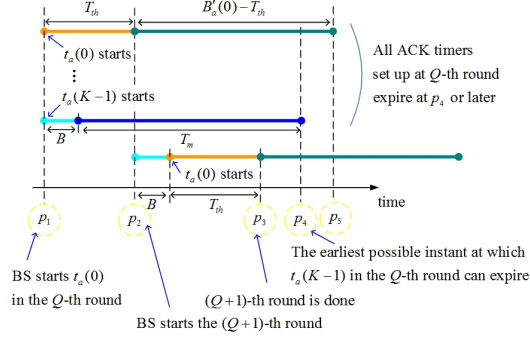


Fig. 4.7. Time margin  $T_m$ .

at a node is canceled by the BS circulating the PT for the next round and that PT reaching the node. Consider the situation in Figure 4.7, where the BS starts its ACK timer of the  $Q$ -th round at  $p_1$ , and it gets back the PT of the  $Q$ -th round within  $T_{th}$  time units. Then, the BS wants to cancel all nodes' ACK timers that were set up in the  $Q$ -th round, and initiates the  $(Q+1)$ -th round at  $p_2$ , which is  $T_{th}$  time units after  $p_1$ . The last node to receive the PT of the  $(Q+1)$ -th round will be  $N(K-1)$ . The time instant  $p_3$  is the latest time at which the PT of the  $(Q+1)$ -th round reaches back to the BS. Since  $t_c \leq T_{th}$  for both the  $Q$ -th and the  $(Q+1)$ -th rounds,  $p_3$  can be expressed as  $p_3 = p_1 + B + 2T_{th}$  (the extra term  $B$  is due to the fact that the BS can take up to  $B$  time units to transmit the PT in the  $(Q+1)$ -th round). Therefore, the node  $N(K-1)$  will receive the PT of the  $(Q+1)$ -th round (and will consequently cancel its ACK timer of the  $Q$ -th round) before  $p_3$ .

Now, consider when the ACK timers of the nodes are scheduled to expire. Since we are analyzing the earliest possible time when the ACK timers can expire, consider that the ACK timers of all the nodes start along with that of  $N(0)$  at  $p_1$ . The earliest expiry will be for node  $N(K-1)$ , and by (4.2) and (4.6) that will be at  $p_4$ , where  $p_4 = p_1 + B + T_m$ . This means that the ACK timers for all the nodes will expire at  $p_4$  or later. (As a side note, the ACK timer for  $N(0)$  will expire at  $p_5$ .) Thus, to guarantee that the latest time at which the ACK timer will get canceled at any node



is *before* the earliest time at which the ACK timer is scheduled to expire at any node, we need the following condition:  $p_3 \leq p_4$ , *i.e.*,  $B + 2T_{th} \leq B + T_m$ , which implies that  $T_m \geq 2T_{th}$ . ■

Note that in the proof of Proposition 4.3.2, if the BS cannot get the PT of the  $(Q + 1)$ -th round before  $p_3$ , some legitimate sensor node may generate its own ACK for the PT of the  $Q$ -th round. However, in this case, the ACK can be discarded by the nodes who have already received the PT of the  $(Q + 1)$ -th round, because they know that they have to send the ACK for the PT of the  $(Q + 1)$ -th round. Further, in this case, since the BS is expecting to receive an ACK, the statement of Proposition 4.3.2 still holds.

**Corollary 1** *If  $t_c \leq T_{th}$  in every round, which means that the compromised nodes (if any) stay undetected, a legitimate node can report an event within  $P$  time units, where  $P = 2(B + T_{th})$ .*

**Proof** By Proposition 4.3.1, the BS can finish each round within  $B + T_{th}$  time units when no attack is detected (here,  $B$  comes from the time that the BS takes to send the PT). Thus, each legitimate node in the EGC can report an event within  $2(B + T_{th})$  time units, *i.e.*,  $P = 2(B + T_{th})$ . The factor of 2 comes in because in the worst case, the event report may be generated by a node right after the PT has crossed that node. ■

In order to identify the compromised nodes that modify any contents in the PT, the sensor nodes verify the signatures in the PT. If a node  $N(i)$  finds any inauthentic signature in the received PT, it sends an event report as  $E = SE$  (Signature Error) in the PT. Then, when the BS gets back the PT, we can guarantee the following.

**Proposition 4.3.3** *If the BS receives an event report  $E = SE$  from  $N(i)$  for any  $i \in I \setminus \{0\}$ ,  $SET(i - 1, i)$  is a suspicious set.*

**Proof** Assume  $SET(i - 1, i)$  is not a suspicious set. Then, both  $N(i - 1)$  and  $N(i)$  are legitimate. Since  $N(i - 1)$  did not generate the event report  $E = SE$ , it must have received authentic signatures in the PT. Thus, the signatures in the PT that  $N(i)$  has received should also be authentic, which means that  $N(i)$  cannot be the node that generates the event report as  $E = SE$ . This is a contradiction. ■

On the other hand, if a normal node, say  $N(i)$ , sends an event report by piggybacking it as in (4.5), a compromised node between  $N(i)$  and  $N(K)$  may strip  $N(i)$ 's signature and event report from the PT, leaving only the PT originated by  $N(0)$ . This attack causes the BS to remain unaware of the event report. To prevent this attack, when the BS receives event reports, it puts the list  $L$  into the PT of a new round, which contains the IDs of nodes who sent the event reports in the previous round. By looking at the list  $L$ , each node should see the IDs of the nodes from which it has seen the event reports in the previous round. For example,  $N(i)$  should see its ID on the list  $L$  in a new round. Otherwise, it implies that somebody behind  $N(i)$  removed the event report from  $N(i)$ . If  $N(i)$  does not see its ID on  $L$ , it drops the PT and sends an ACK with a piggyback error (PE) message. When the BS receives a PE message, the BS may ask the nodes on the EGC if they have such an error, by using ODSBR. Based on the investigation result, we can find at least one compromised node out of the highest-indexed node among those who have such an error and its next-hop node. We omit the proof since it can be shown easily.

### 4.3.2 Overhead Analysis

We now calculate how much SEM can reduce the overhead from the protocol  $A_2$ , in terms of the number of packets required. The main difference in the overhead comes from the fact that in normal operations, SEM does not need the ACK signed in an onion manner. For this reason, we calculate the payload size of the ACK and thus the number of packets that we need for accommodating the payload.

Consider the protocol  $A_2$ , or SEM when an attack is detected. When a node sends an ACK, it generates its signature. The size of the signature is 22 bytes when we use TinyECC [39] that is a popular public key cryptography package for sensor platforms. Since the ACK also includes the sender ID, which is usually 2 bytes, each node needs at least 24 bytes to send an ACK. This means that when the ACK is conveyed from  $N(K)$  to  $N(0)$  through the EGC, the payload size increases by 24 bytes at each node. However, most of the commercial sensor nodes are IEEE 802.15.4 compliant devices, where the payload size of a packet should be less than 114 bytes. Therefore, a single packet can accommodate the ACK signed by up to four different nodes; If  $4m$  nodes sign on an ACK, where  $m$  is an integer, the ACK payload should be fragmented to  $m$  packets at a sender and reassembled at a receiver. Thus, it is easy to see that if  $K$  is a multiple of 4, the total number of packet transmissions for  $N(0)$  to receive the ACK from  $N(K)$  via the EGC can be expressed as  $K(K+4)/8$ , *i.e.*,  $O(K^2)$ . On the other hand, in both the protocol  $A_2$  and SEM, the PT can be sent in a single packet so that for each round, we need  $O(K)$  packets for the PT circulation.

Recall that in the protocol  $A_2$ , the BS has to receive the ACK, whether or not there happens a malicious activity. Thus, each round needs  $O(K + K^2)$  packets. However, SEM does not require the BS to receive the ACK in normal operations. Hence, the number of packet transmissions in normal mode is  $O(K)$ . As a result, SEM can reduce  $O(K^2)$  packets overhead from the protocol  $A_2$  in the normal mode, which corresponds to significant savings. Although SEM incurs similar overhead to the protocol  $A_2$  when an attack is detected, this cost is likely small compared to the overall system operation cost because our protocol can identify the compromised node, which can then be removed, returning the system back to the normal mode.

#### 4.4 Miscellaneous Issues

**Wormhole Attacks:** If two colluding compromised nodes  $N(i)$  and  $N(j)$  for any  $i, j \in I \setminus \{0\}$ ,  $j > i + 1$ , can talk to each other directly,  $N(i)$  can send the PT to  $N(j)$ .

If this is the case, the BS cannot get an event report from a node, say  $N(k)$ , between  $N(i)$  and  $N(j)$ . To prevent this, the BS may randomly select a node each round, and make it send a null event report by notifying the command in the PT. When  $N(k)$  is selected to do so,  $N(i)$  has to send the PT to  $N(i+1)$ , but then  $N(i+1)$  can find the round sequence  $Q$ , which is increased by more than 1 from the one that it has seen. This can be an indication that  $N(i)$  is a compromised node. If  $N(i)$  still directly forwards the PT to  $N(j)$  in this case,  $N(j+1)$  will find that the PT does not contain the null event report from  $N(k)$ , which can also be an indication that  $N(j)$  is compromised. By making any node who first finds such an indication to generate an error event report, we can provide the same security guarantee as in Proposition 4.3.3.

**Jamming Attacks:** Note that a jamming attack may cause a node to stop circulating the PT or the ACK. If  $N(i+1)$  is a victim of the jamming attack,  $N(i)$  will generate its own ACK. In this case, the BS thinks that  $SET(i, i+1)$  is a suspicious set, which is wrong. Therefore, when we also consider the possibility of a jamming attack, we replace  $SET(i, i+1)$  in the Proposition 4.2.1 with the set of the nodes within the jamming distance from the nodes  $N(i)$  and  $N(i+1)$ . Then, it is easy to see that the statement of the proposition is still valid.

**Persistent Failures:** By choosing a sufficiently large number of retransmissions in the ARQ mechanism, the packet loss probability due to transient node failures or link failures can be very small. However, it may not be zero in practice. Thus, we can still lose the PT or the ACK due to such transient failures, although chances are rare. This implies that in practice,  $SET(i, i+1)$  identified by Proposition 4.2.1 may not include any compromised node. However, in such a case,  $SET(i, i+1)$  can be regarded as a set of nodes that requires attention to repair. Thus, it is still useful to identify such a set from the network management point of view.

**Formation of the EGCs:** There would be many ways to form EGCs. The following is an example.

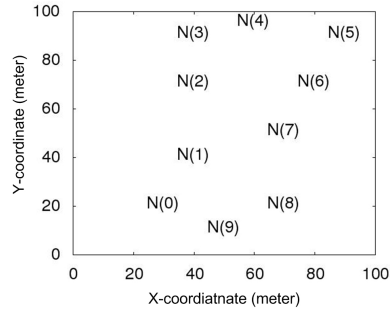


Fig. 4.8. Nodes placement for experiments.

(Step 1) Let  $S$  denote the set of nodes in a network. The BS node creates a topology map of  $S$ , for example, using a link state routing protocol. Initially, all nodes are not marked.

(Step 2) Suppose that  $X$  is the farthest node from the BS among the unmarked nodes in  $S$ . Using Dynamic Source Routing (DSR) protocol, the BS node requests a route to  $X$ . The response from  $X$  may include the round-trip path between  $X$  and the BS. This round-trip path forms a EGC. Mark the nodes in the EGC.

(Step 3) Repeat Step 2 until all nodes in  $S$  are marked.

As is commonly assumed in the literature, we assume that there is no compromised node at the beginning, and thus the formation of EGCs can be done with no attack at the beginning.

## 4.5 Experiments

We have implemented SEM using TOSSIM [40], a popular sensor network simulator based on TinyOS [41]. Our experiment network consists of 10 nodes, *i.e.*,  $K = 10$ , which forms a single EGC. The locations of the nodes are chosen as shown in Figure 4.8. The link gain between any two nodes is determined by a java tool included in TinyOS v2.1, called LinkLayerModel [42], which models path loss and log-normal shadowing. We use a path loss exponent of factor 2.2, and the standard deviation of

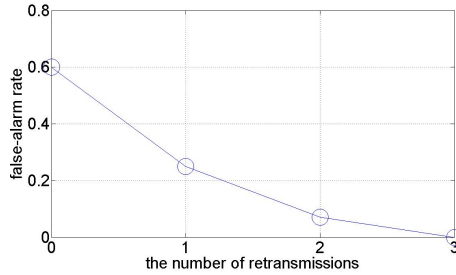


Fig. 4.9. False-alarm rate according to the number of retransmissions for the ARQ mechanism.

log-normal shadowing is 3dB. By this model, the average receiving power at one-hop distance is sufficiently high for correct reception. For digital signature, we assume to use TinyECC [39] so that it takes about 2 second to generate a signature and about 2.4 seconds to verify a signature in the TOSSIM environment. Including the signature overhead, the nodes hold the PT or the ACK less than 10 seconds, *i.e.*,  $B = 10$  seconds. From Propositions 4.3.1 and 4.3.2, we choose  $T_{th}$  and  $T_m$  as  $T_{th} = B_a(0) = 190$  seconds and  $T_m = 2T_{th} = 380$  seconds, respectively. Thus, in this configuration, we regard  $t_c < 190$  seconds as a legitimate case.

Figure 4.9 shows the false-alarm rate as we vary the number of retransmissions. Here, the false-alarm rate is defined as the inverse of the number of successive rounds until we lose the PT due to natural causes, for example, due to a bad communication channel. Since in our experiment, we limit the number of rounds to run as 100, if we do not lose the PT within 100 rounds, we report the false-alarm rate to be zero. The result in Figure 4.9 shows that when we do not use the ARQ mechanism (*i.e.*, no retransmission), the PT can be easily lost due to link failures. Hence, the false-alarm rate is high. However, when we allow the nodes to retransmit the PT up to three times on transmission failures, nearly all false-alarms are eliminated. Therefore, we conclude that at most three retransmissions would be a reasonable choice for the ARQ mechanism in our configuration.

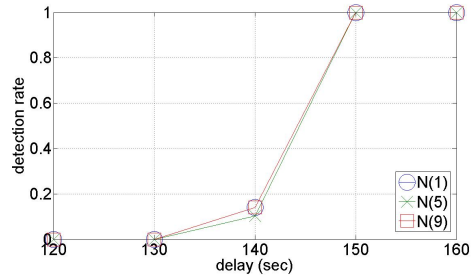


Fig. 4.10. Detection rate according to the delay introduced by a compromised node.

In Figure 4.10, we show the detection rate according to the amount of the delay incurred by a compromised node. For the experiment, we select three different positions for the compromised node ( $N(1)$ ,  $N(5)$ , and  $N(9)$ ). As we mentioned earlier, the compromised nodes may hold the PT longer than  $B = 10$  seconds without being detected, since the BS will not complain anything unless the circulation time  $t_c$  is larger than the threshold  $T_{th}$ . Figure 4.10 shows such a situation. For example, if a compromised node holds the PT for 120 seconds, the BS still receives the PT back within  $T_{th}$ , and thus the compromised node never gets detected. However, when the delay added by the compromised node is 140 seconds, the circulation time  $t_c$  may be larger than the threshold  $T_{th}$ , depending on how long the other nodes hold the PT. We can see that if the amount of the malicious delay is over 150 seconds, it is always detected. Here, remember that once the malicious activity is detected, the BS will not cancel the ACK timers at the sensor nodes, thus gathering an ACK from the nodes. Although we do not show in this figure, once detected, the compromised node is always identified in a form of a suspicious set, as we proved. Further, the amount of malicious delay that the compromised node can introduce without being detected is always the amount of time that is left over in  $T_{th}$  after the nodes actually spent their own time to transmit the PT. Thus, if we reduce the value of  $B$ , and thus reduce the value of  $T_{th}$ , then the compromised node must also decrease malicious delay to avoid

being detected. On the other hand, we can also see from the figure that the position of the compromised node does not affect the detection rate.



## 5. PRIVACY PROTECTION AND COST SAVING IN SMART GRIDS

A smart grid is a type of the electrical grid in which electricity delivery systems are equipped with computer-based remote control and automation, which can revolutionize the way that energy is generated and consumed. A key component of the smart grid is the use of the smart meters, which measure energy usage at a fine granularity (*e.g.*, once in a few minutes). However, by gathering hundreds of data points even in a day via the smart meter, the utility companies and third parties may learn a lot about our daily lives, *e.g.*, when we wake up, when we go out for work, and when we come back after work. In an industrial setting, this may be used to reveal details of the industrial process being used, or when a new process is adopted (which is achievable if the new machinery has electricity usage very distinct from prior machinery). Because of this privacy concern, there have been lawsuits to stop the installation of smart meters [43]. As a result, such privacy concerns have delayed the wide and quick deployment of smart grids.

There are a number of possible threat models for the above privacy risks. Given that we do need to report our energy usage profile to the utility company, the most important threat is that the metering data may be unwittingly disclosed from the utility company to third-party vendors. This problem is well illustrated in an article in MSNBC RedTape [44]. This article introduces a possible scenario with the smart grid that you get a discount with your power company at the cost that your auto insurance company may learn when you are home from the utility company. Additionally, due to possibly poor implementation of cryptography mechanisms, an eavesdropper on the wireless channel between the consumer's premises and the wireless network collection point may also determine the usage.

To resolve this issue, *the first objective of this paper is to make it difficult for an adversary to infer, based on the energy usage profile reported to the utilities, what is going on inside the house.* We achieve this objective by putting a rechargeable battery at the user-end point (*e.g.*, a home). The rechargeable battery acts like a buffer between the power grid and the end user in such a way that the actual energy usage pattern looks different from the energy usage pattern reported to the utility.

Additionally, the rechargeable battery provides us with an opportunity to lower the energy bill, by exploiting the time-of-use (TOU) pricing feature of smart grid, whereby electricity price varies according to pre-established time zones during a day. Basically, the cost-saving will be accomplished by charging the battery when the price is low and using the saved energy from the battery when the price is high. However, the two goals of privacy protection and cost saving are not always compatible with each other. *Our goal is therefore to achieve as much energy cost savings as possible, subject to privacy protection constraints.* To the best of our knowledge, we are the first to propose a mechanism that considers both privacy protection and cost saving simultaneously.

In this paper, we present PRIVATUS, our solution that *guarantees* that instantaneous values of the actual usage and the energy draw visible outside the home are independent in an information-theoretic sense. Further, the patterns of both of these variables are also designed to look dissimilar. We set up a dynamic programming problem that minimizes the energy cost while preserving the privacy guarantee mentioned above.

We evaluate our solution in terms of both the privacy information leakage and the cost saving, and compare it to a previous solution that masked high frequency variation in energy usage [23]. In our simulation environment, PRIVATUS can preserve at least 83% of the uncertainty of the actual usage sequences. In addition, PRIVATUS can achieve 72% of the theoretically-possible maximum cost saving with a 6.43kWh battery. This translates to a saving of \$16 per month in a typical residential pricing plan [45], assuming the average daily usage of 30kWh. We believe that this saving

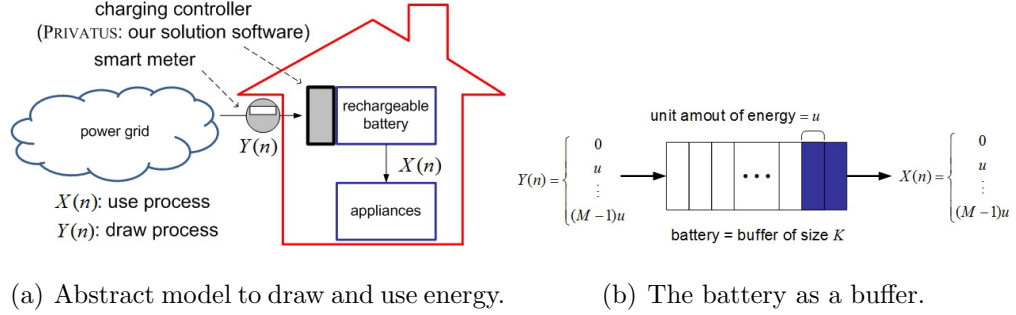


Fig. 5.1. System model.

could provide an extra and significant incentive for users to invest in our solution in addition to privacy protection. The interested reader is referred to Section 5.6 for further discussion about this incentive.

## 5.1 System Model

Suppose that the smart meter measures the energy consumption once in every fixed interval (*e.g.*, 15 minutes), which we call the *measurement interval*. We denote by  $X(n)$  the amount of energy consumed in the  $n$ -th measurement interval. We call  $X(n)$  the *use process*. Denote the amount of energy that we draw from the power grid in the  $n$ -th measurement interval by  $Y(n)$ , which we call the *draw process*. The smart meter measures  $Y(n)$  and reports it to the utility. Without any special technique, *i.e.*, as it happens today, the draw process  $Y(n)$  is the same as the use process  $X(n)$ . What we want to achieve in this paper is to de-correlate  $X(n)$  and  $Y(n)$  so that even if an adversary can observe  $Y(n)$ , no information is leaked about the use process  $X(n)$ . Toward this end, we put a rechargeable battery at the user-end as shown in Figure 5.1(a). The rechargeable battery acts as a buffer between  $X(n)$  and  $Y(n)$ : instead of directly feeding  $X(n)$  by  $Y(n)$ , we charge the battery by  $Y(n)$ , and use the saved energy in the battery to supply  $X(n)$ . We will design an algorithm in the charging controller, which will choose the value of  $Y(n)$  carefully to ensure that the

battery always has the appropriate level of energy (*i.e.*, no shortage to feed  $X(n)$  or no overflow), and that  $X(n)$  looks independent of  $Y(n)$ .

We assume that the values of  $X(n)$  and  $Y(n)$  may take any of the  $M$  different levels  $\{0, u, 2u, \dots, (M-1)u\}$ , where  $u$  represents a unit amount of energy. A given value of  $X(n)$  is assumed to be consumed at a constant speed during the  $n$ -th measurement interval. A given value of  $Y(n)$  is also assumed to be drawn from the power grid at a constant speed. We denote by  $B(n)$  the energy level remaining in the battery at the *end* of the  $n$ -th measurement interval. Assuming for simplicity that there is no energy loss when charging and discharging the battery (for extension to the case with energy loss, see Section 5.6), the value of  $B(n)$  can be expressed as

$$B(n) = B(0) + \sum_{m=1}^n D(m), \quad (5.1)$$

where  $D(m) = Y(m) - X(m)$  and  $B(0)$  is the initial energy level of the battery that is also a multiple of  $u$ . In practice,  $B(0)$  is the remaining energy in the battery at the end of the previous day. Note that  $D(n)$  also takes its value as a multiple of  $u$ , which is over the range  $[-(M-1)u, (M-1)u]$ . We model the battery as a buffer of size  $K$  as illustrated in Figure 5.1(b), which implies that the battery capacity is  $Ku$ , *i.e.*, the range of  $B(n)$  is  $0 \leq B(n) \leq Ku$ .

The probability distributions of  $X(n)$  and  $Y(n)$  are described by  $p_X(i; n)$  and  $p_Y(i; n)$ , respectively, where  $p_X(i; n) = P(X(n) = iu)$  and  $p_Y(i; n) = P(Y(n) = iu)$ . Define the distribution vectors of  $X(n)$  and  $Y(n)$  as

$$P_X(n) = [p_X(0; n), p_X(1; n), \dots, p_X(M-1; n)] \quad (5.2)$$

and

$$P_Y(n) = [p_Y(0; n), p_Y(1; n), \dots, p_Y(M-1; n)], \quad (5.3)$$

respectively. We assume that  $P_X(n)$  is known to the user (*i.e.*, the home owner). We also assume that  $X(n)$  is independent, but does not need to be identically distributed across the measurement interval index  $n$ . This means that for instance,  $X(5)$  is independent of  $X(11)$ , but  $P_X(5)$  can be different from  $P_X(11)$  (if the family leaves

home for work/school at 8 a.m., then clearly the usage before 8 a.m. and after 8 a.m. will be different). As we will see later,  $P_Y(n)$  is our control parameter.

We are interested in the case where the electricity price per unit amount of energy varies from time to time. More specifically, we first focus on the case where there exist two time zones within a day, one of which has a low rate  $R_L$  (dollars/ $u$ ) and the other has a high rate  $R_H$  (dollars/ $u$ ). The zone with a low rate is called the *low-price zone* and the other is called the *high-price zone*. For ease of exposition, we assume that the measurement intervals from  $n = 1$  to  $n = n_L$  fall into the low-price zone, and the measurement intervals from  $n = n_L + 1$  to  $n = n_H$  correspond to the high-price zone. We treat the initial point  $n = 0$  as the beginning of a day and the end of the measurement interval of  $n = n_H$  as the end of the day. In Section 5.6, we will discuss how we can generalize the solution to handle the case with more than two price zones in a day, and the case when the low-price and high-price zones are interleaved.

Because of the page limit, this paper assumes that the total amount of energy usage per day is the same over days on average. Section 5.6 introduces a way to release this assumption and generalize our solution.

## 5.2 Solution Approach I: Basic Formulation

### 5.2.1 Mapping between $X(n)$ and $Y(n)$

In order to hide  $X(n)$  from an external adversary (*i.e.*, an adversary outside the home), we make  $Y(n)$  be independent of  $X(n)$ . This implies that observing  $Y(n)$  gives no meaningful information about  $X(n)$ . This is achieved when we map  $X(n)$  to  $Y(n)$  in such a way that  $p_Y(i; n) \equiv P(Y(n) = iu) = P(Y(n) = iu | X(n) = ju)$  for any possible  $i$  and  $j$ . Practically, we achieve this by probabilistically choosing the value of  $Y(n)$  according to  $P_Y(n)$ , which is decided before the  $n$ -th measurement interval starts, *without considering what the value of  $X(n)$  will be*.

However, selecting  $Y(n)$  randomly without being aware of  $X(n)$  may cause energy shortage or overflow in the battery. For example, when  $B(n - 1) = 0$  (*i.e.*, there is

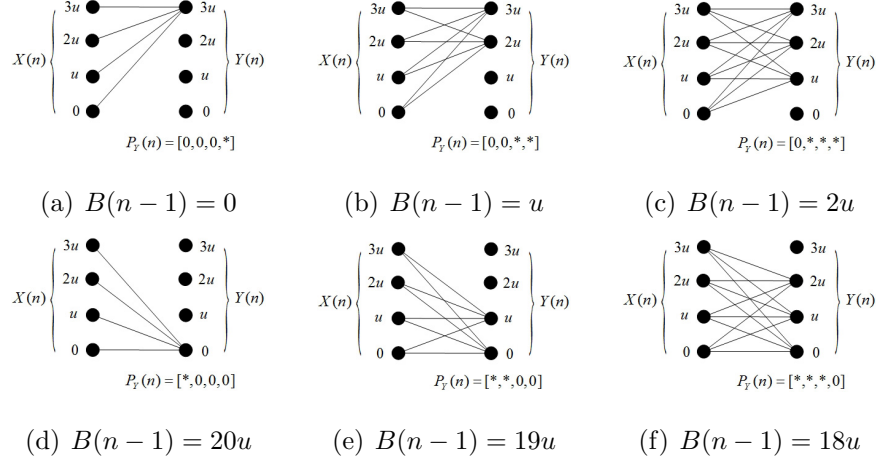


Fig. 5.2. An example of the probabilistic symbol mapping between  $X(n)$  and  $Y(n)$  in the corner cases when  $K = 20$  and  $M = 4$  ((a)-(c): empty or near-empty battery; (d)-(f): full or near-full battery). The symbol ‘\*’ in  $P_Y(n)$  represents the element that can be non-zero.

no energy remaining in the battery before the  $n$ -th measurement interval starts), if  $Y(n)$  is chosen to be zero, we cannot feed any non-zero value of  $X(n)$ . This means that sometimes we cannot use the appliances when we want. Similarly, when  $B(n-1) = Ku$  (*i.e.*, the battery is full), a non-zero value of  $Y(n)$  does not make sense if  $X(n) = 0$ , since we cannot draw the energy from the power grid unless we throw it away.

To handle this issue, we put a restriction on  $P_Y(n)$  when the energy left in the battery is smaller than  $(M-1)u$  (near-empty) or larger than  $(K-(M-1))u$  (near-full), which we call the *corner cases*. More specifically, when  $B(n-1) = ju$  for  $j < (M-1)$ , we choose  $P_Y(n)$  such that  $p_Y(i; n) = 0$  for  $i < (M-1) - j$ . Similarly, when  $B(n-1) = (K-j)u$  for  $j < (M-1)$ , we choose  $P_Y(n)$  such that  $p_Y(i; n) = 0$  for  $i > j$ . Consider the example in Figure 5.2 where  $K = 20$  and  $M = 4$ . The possible realizations of  $Y(n)$  at the near-empty case are:  $3u$  when  $B(n-1) = 0$ ;  $3u$  or  $2u$  when  $B(n-1) = u$ ;  $3u$ ,  $2u$  or  $u$  when  $B(n-1) = 2u$ . At the near-full case, the possible realizations of  $Y(n)$  are:  $0$  when  $B(n-1) = 20u$ ;  $0$  or  $u$  when  $B(n-1) = 19u$ ;  $0$ ,

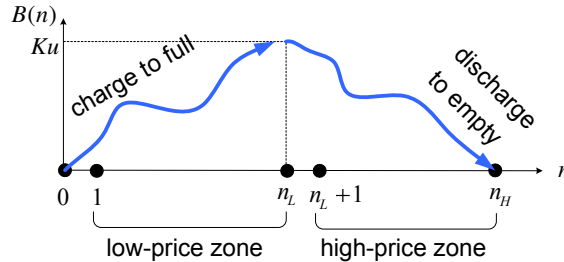


Fig. 5.3. Desired battery state profile.

$u$  or  $2u$  when  $B(n-1) = 18u$ . The rationale behind this restriction on  $P_Y(n)$  is that the battery must always have enough amount of energy to feed  $X(n)$  even at the near-empty case, and that we never charge the battery more than its capacity whatever  $X(n)$  is.

### 5.2.2 Strategy for charging/discharging the battery

The only way to achieve cost saving by exploiting the time-of-use pricing policy is to charge the battery in the low-price zone and use the stored energy in the high-price zone. If we charge  $iu$  amount of energy in the low-price zone and use it in the high price zone, we can save  $(R_H - R_L)i$  (dollars). For this reason, the maximum possible cost saving is  $(R_H - R_L)K$  (dollars) per day, which is obtained when we charge the battery from empty to full in the low-price zone and discharge the battery to zero by feeding  $X(n)$  in the high-price zone. Note that the maximum cost saving is proportional to the battery capacity  $Ku$ .

Therefore, our strategy to achieve the saving in the energy bill is to force the battery state to follow the trend shown in Figure 5.3. We achieve this by changing  $P_Y(n)$  for every  $n$ , which is discussed in detail in the following subsection.

### 5.2.3 Basic approach

We first define the distribution vector space  $\mathcal{P}$  as follows.

$$\mathcal{P} = \left\{ [p_0, p_1, \dots, p_{(M-1)}] : \sum_{i=0}^{M-1} p_i = 1, 0 \leq p_i \leq 1 \right\}, \quad (5.4)$$

where we limit the value of  $p_i$  to be a multiple of a constant  $c$  ( $0 < c < 1$ ), in order to make  $\mathcal{P}$  be a finite set. For example, when  $c = 0.1$  and  $M = 4$ , the distribution vector space  $\mathcal{P}$  contains  $[0.1, 0.2, 0.3, 0.4]$  and  $[0.5, 0.5, 0, 0]$  as two of its elements. Then,  $P_Y(n)$  is assigned one element in  $\mathcal{P}$  in the  $n$ -th measurement interval. Recall that we force some elements of  $P_Y(n)$  to be zero, depending on the battery level (Section 5.2.1). Therefore, the possible choice set in the  $n$ -th measurement interval is dependent on  $B(n-1)$  and we denote it by  $\mathcal{P}_{B(n-1)}$ . Now, the key question for us is “*what would be the best choice for  $P_Y(n) \in \mathcal{P}_{B(n-1)}$  for each  $n$  to maximize the cost saving?*” This question is answered by solving the following stochastic optimal control problems:

$$\max_{\substack{P_Y(n) \in \mathcal{P}_{B(n-1)} \\ 0 < n \leq n_L}} E(B(n_L) | B(0), P_Y(1), P_Y(2), \dots, P_Y(n_L)) \quad (5.5)$$

in the low-price zone, and

$$\min_{\substack{P_Y(n) \in \mathcal{P}_{B(n-1)} \\ n_L < n \leq n_H}} E(B(n_H) | B(n_L), P_Y(n_L+1), P_Y(n_L+2), \dots, P_Y(n_H)) \quad (5.6)$$

in the high-price zone. Namely, we maximize (or minimize) the expected amount of the energy in the battery when each zone ends, given the battery level at the beginning of the zone and the distribution vectors  $P_Y(1)$  through  $P_Y(n_L)$  (or  $P_Y(n_L+1)$  through  $P_Y(n_H)$ ). We solve these optimization problems using dynamic programming [26].

To see how we use dynamic programming, let us first consider the following simple example in the low-price zone, where  $n_L = 3$ . Then, the optimization objective is to maximize  $E(B(3) | B(0), P_Y(1), P_Y(2), P_Y(3))$ , which is equal to  $B(0) +$



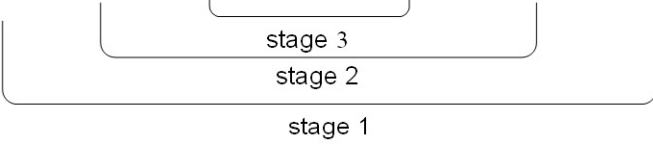
$$\begin{aligned}
& E\left(\sum_{n=1}^3 D(n)|B(0), P_Y(1), P_Y(2), P_Y(3)\right) \\
&= E(D(1)|B(0), P_Y(1)) + E(D(2)|B(1), P_Y(2)) + E(D(3)|B(2), P_Y(3)) \\
&= E(D(1)|B(0), P_Y(1)) + E(D(2) + E(D(3)|B(2), P_Y(3))|B(1), P_Y(2)) \\
&= E(D(1) + E(D(2) + E(D(3)|B(2), P_Y(3))|B(1), P_Y(2))|B(0), P_Y(1))
\end{aligned}$$


Fig. 5.4. An example to derive the dynamic programming framework.

$E(\sum_{n=1}^3 D(n)|B(0), P_Y(1), P_Y(2), P_Y(3))$ , where  $D(n) = Y(n) - X(n)$  as introduced earlier. Since  $B(0)$  is given, we only need to focus on maximizing

$$E\left(\sum_{n=1}^3 D(n)|B(0), P_Y(1), P_Y(2), P_Y(3)\right), \quad (5.7)$$

which can be re-written as shown in Figure 5.4. Note in the figure that the calculations can be done recursively. Stage 2 calculations are based on stage 3, stage 1 only on stage 2. Thus, the optimal solution can be performed by maximizing the stage 3, stage 2, and stage 1 in this order. In this manner, we first compute the optimal value of  $P_Y(3)$  given  $B(2)$ , then we compute the optimal value of  $P_Y(2)$  given  $B(1)$  until we reach and compute the optimal value of  $P_Y(1)$ . In the general case,  $P_Y(n_L)$  is computed first and then other  $P_Y(n)$ 's are computed in a backward direction (time-wise) till  $P_Y(1)$  is computed.

Namely, the optimal solution for (5.5) is obtained by a backward-directional computation procedure. In general, this procedure can be described by the following recursive equation, called the Bellman equation:

$$\begin{aligned}
& J(n_L + 1, B(n_L)) = 0, \\
& J(n, B(n-1)) = \max_{P_Y(n) \in \mathcal{P}_{B(n-1)}} E(D(n) + J(n+1, B(n))|B(n-1), P_Y(n)), \quad (5.8)
\end{aligned}$$

for  $n = n_L, (n_L - 1), \dots, 1$ . Note that  $J(n, B(n - 1))$  in (5.8) represents the maximum possible value of  $E(\sum_{m=n}^{n_L} D(m))$  when  $B(n - 1)$  is given and the optimal decision is made at time  $n, (n + 1), \dots, n_L$ . In (5.8), the expectation is with respect to the conditional distribution of  $D(n)$  given  $B(n - 1)$  and  $P_Y(n)$ , and thus it can be written as

$$\begin{aligned} & E(D(n) + J(n + 1, B(n)) | B(n - 1), P_Y(n)) \\ &= \sum_{\substack{-(M-1) \leq j \leq (M-1), \\ 0 \leq i+j \leq K}} p_{i,(i+j)}(n) (j + J(n + 1, (i + j)u)) \end{aligned} \quad (5.9)$$

with  $i = B(n - 1)/u$ . Here,  $p_{i,(i+j)}(n)$  denotes the probability of transition from  $B(n - 1) = iu$  to  $B(n) = (i + j)u$ , resulting from  $D(n) = ju$ . Take  $M = 4$  as an example. The transition from  $B(n - 1) = u$  to  $B(n) = 3u$  can happen if  $X(n) = 0$  and  $Y(n) = 2u$ , or  $X(n) = u$  and  $Y(n) = 3u$ . Thus, it is easy to see that the transition probability  $p_{i,(i+j)}(n)$  is in general given as follows: for  $j = 0$ ,

$$p_{i,(i+j)}(n) = \sum_{m=0}^{M-1} p_X(m; n) p_Y(m; n), \quad (5.10)$$

for  $j > 0$  such that  $j \leq (M - 1)$  and  $i + j \leq K$ ,

$$p_{i,(i+j)}(n) = \sum_{m=0}^{M-1-j} p_X(m; n) p_Y(m + j; n), \quad (5.11)$$

and for  $j < 0$  such that  $j \geq -(M - 1)$  and  $i + j \geq 0$ ,

$$p_{i,(i+j)}(n) = \sum_{m=0}^{M-1-j} p_X(m + j; n) p_Y(m; n), \quad (5.12)$$

and  $p_{i,(i+j)}(n) = 0$ , otherwise. Solving (5.8) results in the optimal decision for  $P_Y(n)$  when the value of  $B(n - 1)$  is given, in the sense that  $P_Y(n)$  will maximize  $E(B(n_L))$ . The optimal solution for (5.6) can also be obtained in a similar way.

In summary, what we have done is to calculate a decision table. Each entry in the decision table maps the given values of  $n$  and  $B(n - 1)$  to the optimal vector  $P_Y(n)$  at the state. Note that the decision table can be pre-calculated before the run-time. During the run-time, we just look up the decision table for a given state, *i.e.*,

$n$  and  $B(n - 1)$ , and probabilistically choose the value of  $Y(n)$  via the distribution specified by the decision table entry. The size of this table can be large in practice if  $K$  and  $n_H$  are large. Thus, calculating the decision table can be computationally expensive. However, note that the table can be reused from one day to another till the distribution of the use process  $X(n)$  changes significantly. Section 5.6 provides discussion about the table complexity.

#### 5.2.4 Simulation study for the basic approach

We now present simulation results for our basic solution approach. By this simulation study, we will identify the issues with the basic approach, which will motivate us to improve our solution in Section 5.3.1 and propose PRIVATUS.

In the simulation, we choose  $M = 4$ ,  $K = 20$ , and  $c = 0.1$ . We fix each measurement interval to be 15 minutes and thus we have 96 measurement intervals a day. Thus, the value of  $n_H$  becomes  $n_H = 96$  and we set  $n_L = 32$ . In order to see more clearly what  $Y(n)$  looks like compared to  $X(n)$ , we make  $X(n)$  as a known repeated pattern, instead of generating it randomly (Figure 5.5).

A sample result of the simulation is shown in Figure 5.5, where “ $P_Y(n)$  (index)” in the bottom graph means the index number of the element in  $\mathcal{P}$  selected as  $P_Y(n)$ . We can see that at each measurement interval, the values of  $X(n)$  and  $Y(n)$  are mapped to each other in a random fashion. Further, the battery level indeed moves according to the trend that it is charged to the full level in the low-price zone and fully discharged in the high-price zone. However, we also observe that there exist similar patterns for the sequences of  $X(n)$  and  $Y(n)$  for the measurement intervals of  $16 \leq n \leq 32$  and  $70 \leq n \leq 96$ . More precisely, we see that the value of  $X(n)$  highly likely reappears as the value of  $Y(n + 1)$  when the battery is at the corner cases. This is an undesirable behavior because if the adversary learns this characteristic, he or she may infer the original values of  $X(n)$  with high accuracy by observing the values

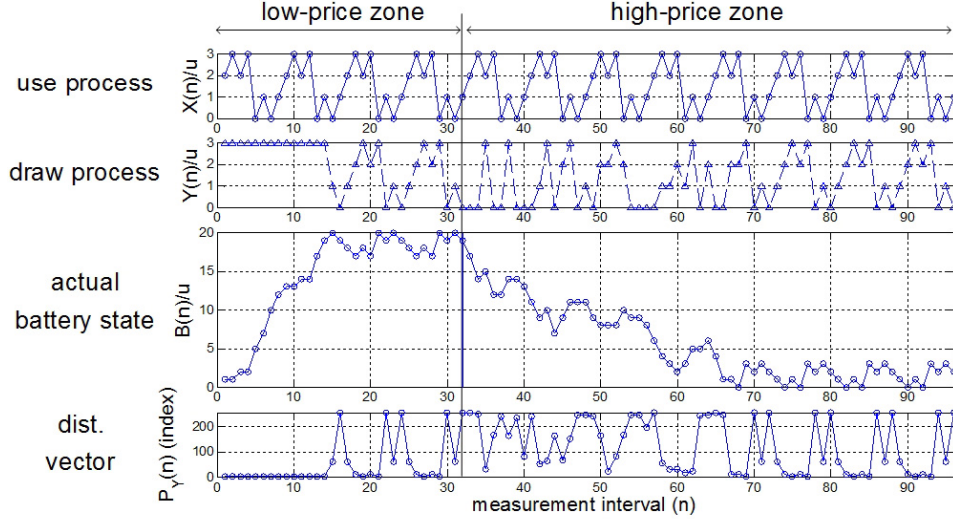


Fig. 5.5. Simulation results for the basic approach.

of  $Y(n+1)$ . Through this, we realize that our point-by-point de-correlation between  $X(n)$  and  $Y(n)$  leaves an obvious vulnerability in practice.

After more careful study, we find that this issue occurs because of two reasons: **(R1)** The first reason is that we charge/discharge the battery too fast. In the low-price zone, the battery reaches the full state much earlier than the end of the zone. Once at the full state, the battery stays close to the near-full states, since there is no benefit to bring the energy level down to a lower one according to our optimization objective in (5.5). The near-constant energy level of the battery implies that whatever the value of  $X(n)$  is, the draw process  $Y(n)$  should somehow compensate for it. Since the value of  $Y(n)$  is chosen before the value of  $X(n)$ , we see this compensation effect in  $Y(n+1)$ . Similar logic applies to the high-price zone; **(R2)** The second reason is that we have too much freedom when choosing  $P_Y(n)$ . As a result, the draw process can take a specific symbol with a very high probability to compensate the use process. For example, if  $X(n) = 3u$  and the draw process needs to compensate it (due to the first reason), the basic approach will likely choose  $P_Y(n+1) = [0, 0, 0, 1]$ . This implies that we will charge with the current value of  $3u$  with probability 1 at the  $(n+1)$ -th

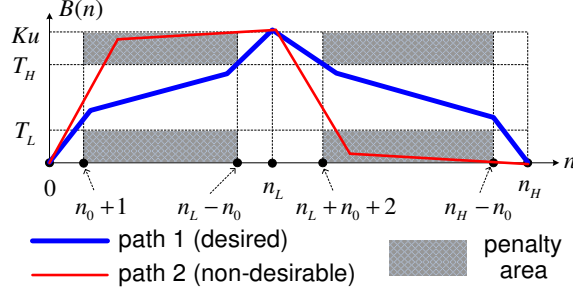


Fig. 5.6. Penalty areas.

measurement interval. In other words, due to the high degree of freedom to choose  $P_Y(n)$ ,  $Y(n)$  is chosen to be very similar to  $X(n-1)$  in the corner cases.

In the next section, we will propose PRIVATUS that suppresses these undesirable effects **(R1)** and **(R2)**.

### 5.3 Solution Approach II: Advanced Formulation

#### 5.3.1 Advanced approach: PRIVATUS

In order to fix **(R1)**, we introduce penalty areas for when the battery level gets too close to empty or too close to full as shown in Figure 5.6. The penalty areas correspond to the battery states higher than the upper threshold  $T_H$  or lower than the lower threshold  $T_L$ . In each zone (low-priced or high-priced), the penalty areas begin after  $n_0$  measurement intervals, and end  $n_0$  measurement intervals before the end of the zone. We modify our optimization objective in such a way that we incur some penalty, whenever the battery state  $B(n)$  falls into the penalty areas. Hence, the optimal decision for  $P_Y(n)$  would be changed to the one that still charges or discharges the battery according to the trend in Figure 5.3, but does not hit the penalty areas in the middle of the zones. In this sense, the modified optimization objective would result in “path 1”-like battery profile rather than “path 2”-like one

in Figure 5.6. The “path 2”-like battery profile is what we have seen in the basic approach.

We consider the *effective battery state*  $B_e(n)$  in the optimization objective function, instead of the actual battery state  $B(n)$ . The effective battery state  $B_e(n)$  is designed to increase as the actual battery state  $B(n)$  increases in the low-price zone (or  $B(n)$  decreases in the high-price zone). However, every time  $B(n)$  goes into a penalty area,  $B_e(n)$  is deducted by some penalty amount. Denote by  $[x]^+$  the projection of  $x$  to non-negative values, *i.e.*,  $[x]^+ = x$  if  $x > 0$ , and  $[x]^+ = 0$  if  $x \leq 0$ . Then, the effective battery state  $B_e(n)$  in the low-price zone is defined as  $B_e(n) = B_e(0) + \sum_{m=1}^n D_e(m)$ . Here,  $B_e(0) = \alpha B(0)$  and  $D_e(m)$  is given as, if  $m \leq n_0$  or  $m > n_L - n_0$  (*i.e.*, in near-beginning or near-end of the low-price zone),  $D_e(m) = \alpha D(m)$ , and if  $m > n_0$  and  $m \leq n_L - n_0$ ,

$$D_e(m) = \alpha D(m) - \beta ([B(m) - T_H]^+ + [T_L - B(m)]^+), \quad (5.13)$$

where  $\alpha$  and  $\beta$  are positive integers,  $T_L = (M - 1)u$ , and  $T_H = (K - (M - 1))u$ . In the high-price zone, we define  $B_e(n)$  as  $B_e(n) = B_e(n_L) + \sum_{m=n_L+1}^n D_e(m)$ , where  $B_e(n_L) = \alpha(Ku - B(n_L))$ , and further, if  $m \leq n_L + n_0$  or  $m > n_H - n_0$ ,  $D_e(m) = -\alpha D(m)$ , and if  $m > n_L + n_0$  and  $m \leq n_H - n_0$ ,

$$D_e(m) = -\alpha D(m) - \beta ([B(m) - T_H]^+ + [T_L - B(m)]^+). \quad (5.14)$$

Note that if we ignore the second terms in (5.13) and (5.14), we simply have  $B_e(n) = \alpha B(n)$  in the low-price zone, and  $B_e(n) = \alpha(Ku - B(n))$  in the high-price zone. That is,  $B_e(n)$  increases from zero to the maximum  $\alpha Ku$  in both zones as  $B(n)$  moves like in Figure 5.3. Thus, our optimization objective for achieving the maximal cost saving is to maximize  $E(B_e(n_L))$  in the low-price zone and  $E(B_e(n_H))$  in the high-price zone, given initial conditions. On the other hands, the terms leading by  $\beta$  in (5.13) and (5.14) take into account the penalty. Whenever  $D(n)$  causes  $B(n)$  to fall into a penalty area, we subtract  $\beta[B(n) - T_H]^+$  or  $\beta[T_L - B(n)]^+$  from  $B_e(n)$ . Hence, we will expect that in the optimal decision for  $P_Y(n)$ ,  $B(n)$  would avoid hitting the

penalty area, or  $B(n)$  would attempt to get out of a penalty area if  $B(n - 1)$  was already in the penalty area. The relative magnitudes of  $\alpha$  and  $\beta$  determines how sensitive we are to the penalty. If  $\beta$  is very large compared to  $\alpha$ ,  $B(n)$  may not even go close to the penalty area to avoid any chance of incurring a high penalty score.

On the other hand, to address **(R2)**, we adopt two strategies. *First*, we put the restriction on  $\mathcal{P}_{B(n-1)}$  that it only contains the vectors  $v \in \mathcal{P}$  such that  $\|v - V_k\| < T_k$ . Here,  $T_k$  is a threshold at  $B(n - 1) = ku$ , and  $V_k$  is the distribution vector of  $Y(n)$  for which the possible values of  $Y(n)$  at  $B(n - 1) = ku$  are selected equi-probably. For instance, when  $M = 4$  and  $K = 10$ , we have  $V_5 = [0.25, 0.25, 0.25, 0.25]$  when  $B(n - 1) = 5u$ , and  $V_1 = [0.5, 0.5, 0, 0]$  when  $B(n - 1) = u$ . With this strategy, we are forcing the different elements of  $P_Y(n)$  to be more or less equal, thus eliminating the possibility that  $Y(n)$  is chosen deterministically (or with a high probability). By controlling the threshold  $T_k$ , we can control how close to equal probability we want. If  $T_k$  is low, then the choices are close to equally probable, but we also lose controllability in forcing  $B(n)$  to the desired state according to the trend in Figure 5.3.

*Second*, we add one more restriction on  $P_Y(n)$  in non-corner cases (*i.e.*, battery neither empty nor full) such that it does not differ significantly from  $P_Y(n - 1)$ . If the two differ significantly, then  $Y(n)$  may try compensating for the use value in the previous measurement interval and will hence track  $X(n - 1)$ . Therefore, our strategy is that  $\|P_Y(n) - P_Y(n - 1)\| < T_D$ , where  $T_D$  is called the distance threshold. We enforce this restriction to be applied only when the actual battery state stays in non-corner cases for two consecutive measurement intervals, *i.e.*,  $T_L \leq B(n - 2) \leq T_H$  and  $T_L \leq B(n - 1) \leq T_H$ . Our intention behind this is to quickly get out of the corner cases (which hits the penalty areas). In the extreme case, with this strategy,  $P_Y(n - 1) = P_Y(n)$  implying that  $Y(n)$  is independent of  $X(n - 1)$ .

Reflecting all the changes, the optimal choice for  $P_Y(n)$  in the low-price zone is obtained by solving the following Bellman equation.

$$J(S(n_L + 1)) = 0,$$

$$J(S(n)) = \max_{P_Y(n) \in \mathcal{P}_{B(n-1)}^*} E(D_e(n) + J(S(n+1)) | S(n)), \quad (5.15)$$

for  $n = n_L, (n_L - 1), \dots, 1$ . Here,  $S(n)$  represents the state vector defined as  $S(n) = [n, B(n-1), B_e(n-1), P_Y(n-1)]$ .  $\mathcal{P}_{B(n-1)}^*$  is defined as a subset of  $\mathcal{P}$  whose element  $v$  is such that the two restrictions described above are satisfied, *i.e.*,  $v \in \mathcal{P}_{B(n-1)}$ , and if  $T_L \leq B(n-2) \leq T_H$  and  $T_L \leq B(n-1) \leq T_H$ ,  $\|v - P_Y(n-1)\| < T_D$ . The optimal choice for  $P_Y(n)$  in the high-price zone can also be decided in a similar way.

### 5.3.2 Simulation study for PRIVATUS

Now, we conduct a simulation test for PRIVATUS. In order to see the difference from the basic approach, we use the same simulation environment as in Section 5.2.4. We choose  $T_k = 0.3$  for  $k = 3, 4, \dots, 17$ ;  $T_k = 0.25$  for  $k = 2, 18$ ;  $T_k = 0.2$  for  $k = 1, 19$ ;  $T_k = 0.1$  for  $k = 0, 20$ . With these threshold values,  $\mathcal{P}_k$  only contains  $[0, 0, 0.4, 0.6]$ ,  $[0, 0, 0.5, 0.5]$ , and  $[0, 0, 0.6, 0.4]$  for  $k = 1, 19$ , for instance. For the remaining parameters, we set  $\alpha = 2$ ,  $\beta = 1$ ,  $n_0 = 3$ , and  $T_D = 0.2$ .

Figure 5.7 shows a sample result for the simulation, where the solid red lines in the “ $B(n)/u$ ” graph indicate the energy levels corresponding to the penalty area thresholds  $T_H$  and  $T_L$ . First, we can see that  $B(n)$  follows the trend in Figure 5.3, and it seldom hits the penalty area as we desired. Although  $B(n)$  enters the penalty area at around  $n = 39, 76, 92$ , we can also see that  $B(n)$  tries to get out of penalty area quickly. As a result, the battery neither goes to the full-state too quickly in the low-price zone, nor goes to the empty-state too quickly in the high-price zone. Second, in the “ $P_Y(n)(\text{index})$ ” graph, we observe that for many times, the decision for  $P_Y(n)$  remains the same, or the speed of changing a decision becomes much slower (compared to the result in Figure 5.5). By these two fixes, we see that the correlation between the use process and the draw process is significantly reduced. We can no



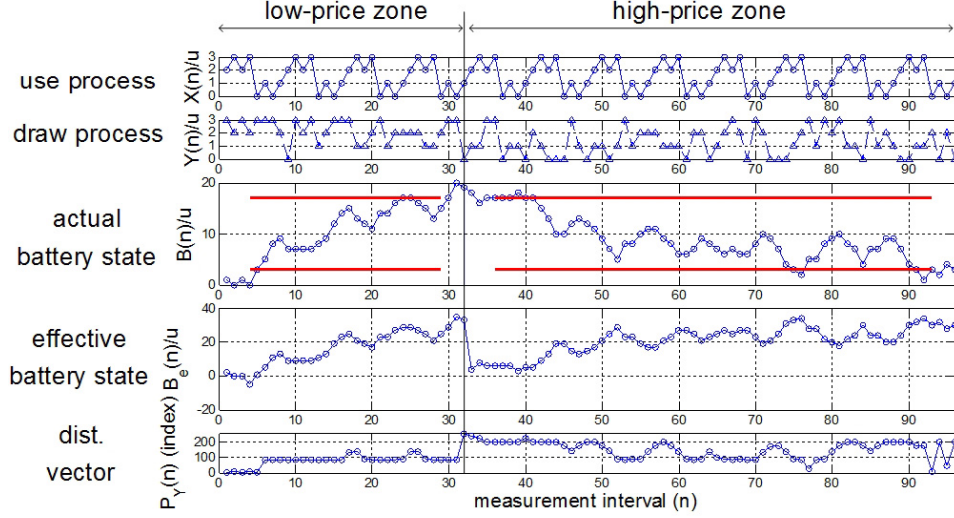


Fig. 5.7. Simulation results for PRIVATUS.

longer find similar patterns between the two. The point-by-point comparison of  $X(n)$  and  $Y(n)$  still gives no meaningful clue from  $Y(n)$  to  $X(n)$ , as this is by design that is maintained in the basic approach and PRIVATUS. Of course, this might be seen as a subjective interpretation of the result. Thus, in the experiment section, we will consider a metric to quantitatively measure how well we are protecting the privacy and re-visit these results.

## 5.4 Per-day Energy Usage Flattening

### 5.4.1 Total energy usage different across days

So far, we have seen that PRIVATUS hides the energy consumption pattern within a day. However, the average total usage per day, *i.e.*,  $E(\sum_{n=1}^{n_H} X(n))$  may be different across days, and this information can still be revealed to the adversary (by which the adversary may know whether you are home or not for a given day). Thus, if there is a significant difference in the average total usage between days, PRIVATUS hides this information by flattening the energy use across days on the average.

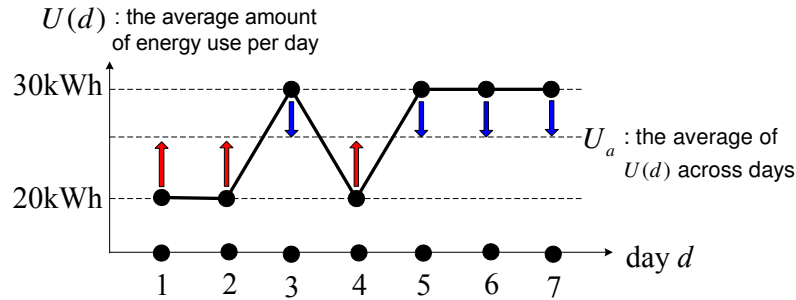


Fig. 5.8. Example of average daily usage across days ( $P = 7$ ). In order to flatten the the daily consumption, the days 1,2, and 4 need to use more energy by the amount indicated by the red arrow, and the days 3,5,6, and 7 are required to lessen the consumption by the amount indicated by the blue arrow.

#### 5.4.2 Solution summary

We assume that the average energy consumption per day varies in a cycle of  $P$  days. For example,  $P = 7$  implies that a regular pattern of living is repeated every week. We denote by day  $d$  the  $d$ -th day of the period, and define the index set for the days as  $I_d = \{1, 2, \dots, P\}$ . Define  $U(d)$  as the average amount of energy consumption for day  $d$ . Namely,  $U(d)$  is  $E(\sum_{n=1}^{n_H} X(n))$  for day  $d$ . The average of  $U(d)$  across days is denoted by  $U_a$ , which is expressed as follows:

$$U_a = \frac{1}{P} \sum_{d \in I_d} U(d). \quad (5.16)$$

We categorize the days into two types, depending on whether  $U(d) < U_a$ . Type 1 day is the day with  $U(d) < U_a$ . Type 2 day is the day with  $U(d) \geq U_a$ . The index sets of type 1 days and type 2 days are denoted by  $I_d^1 = \{d \in I_d : U(d) < U_a\}$  and  $I_d^2 = \{d \in I_d : U(d) \geq U_a\}$ , respectively. Figure 5.16 shows an example of average energy consumption with the period of a week, where the days 1, 2, and 4 are of type 1, and the days 3, 5, 6, and 7 are of type 2. In this example, to achieve the flat use  $U_a$  across days, the type 1 days need to use more energy by the red-arrow amount, and the type 2 days need to use less energy by the blue-arrow amount.

Then, the question is how we can change  $U(d)$  to  $U_a$ . In type 1 days, PRIVATUS consumes more energy than  $U(d)$  by charging more energy in the low-price zone than used in the high-price zone, and by keeping the unused energy in the battery. This kept energy is used up in type 2 days to bring  $U(d)$  down to  $U_a$ . Specifically, in type 2 days, PRIVATUS charges less energy than used in the high-price zone: the gap is supplied by the energy kept in type 1 days.

### 5.4.3 Virtual battery state

In order to flatten the energy usage across days, we apply the virtual battery state  $B_v(n)$  to (5.15) in the place of the actual battery state  $B(n)$ . The virtual battery state  $B_v(n)$  is defined as follows:

$$B_v(n) = B(n) - E_k, \quad (5.17)$$

where  $E_k$  denotes the amount of energy that is kept for future use. Namely, the virtual battery state is the battery state that is adjusted to be lower by the amount of the kept energy. The value of  $E_k$  increases whenever we keep some amount of energy in type 1 days, and decreases whenever we use the kept energy in type 2 days. We use a decision table that is obtained in (5.15) when the actual battery capacity is  $K_v u$ , and just replace  $B(n)$  in (5.15) with  $B_v(n)$ . This implies  $0 \leq B_v(n) \leq K_v u$ . We call the maximum virtual battery state the *virtual battery capacity*. That is, the virtual battery capacity is  $K_v u$ . We will see later that the actual battery capacity  $K u$  required for flattening may be larger than the virtual battery capacity  $K_v u$ .

Figure 5.9 illustrates how PRIVATUS keeps energy in type 1 days and uses the kept energy in type 2 days, and how the virtual battery state changes accordingly. To keep  $U_k$  amount of energy in a day of type 1, we update  $E_k$  as  $E_k \leftarrow (E_k + U_k)$  before the  $(n_L + 1)$ -th measurement interval starts, *i.e.*, before the high-price zone begins (see Figure 5.9(a)). This results in the sudden drop in  $B_v(n)$  in the boundary between the low-price and high-price zones. Effectively, this causes the  $U_k$  amount out of what is charged in the low-price zone to be kept for future use, and only the

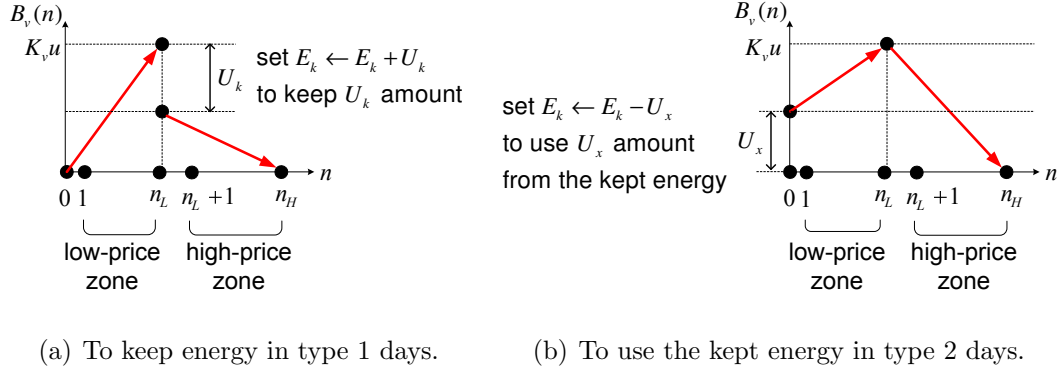
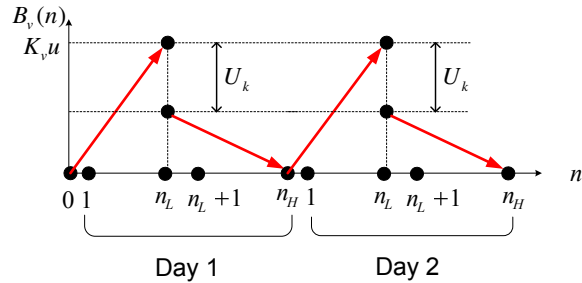


Fig. 5.9. The methods to keep energy and use the kept energy.

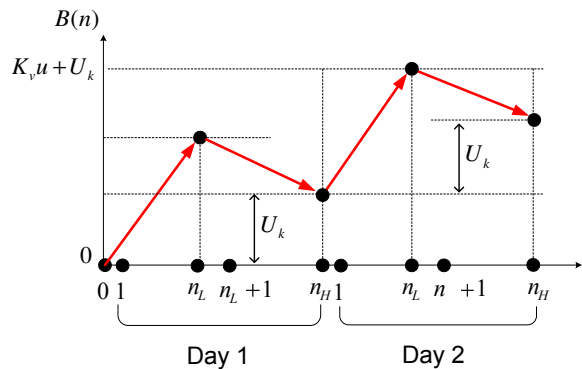
rest of the charged energy is used in the high-price zone. On the other hand, to use  $U_x$  amount of energy from the kept energy in a day of type 2, we update  $E_k$  as  $E_k \leftarrow (E_k - U_x)$  before the first measurement interval starts, *i.e.*, before the beginning of the low-price zone of the day (see Figure 5.9(b)). This change in  $E_k$  leads to the sudden jump in  $B_v(n)$  in the boundary between days. Since the battery looks already charged by  $U_x$  amount, we can only charge up to  $(Ku - U_x)$  amount in the low-price zone. Thus, we can reduce the energy consumption by  $U_x$  amount in a day of type 2. Note that this approach does not affect PRIVATUS's privacy protection mechanism within a day, because what we do is only to modify the initial value of the battery state at the beginning of the price zones. It only changes the amount of energy that is used or charged per day.

#### 5.4.4 Battery capacity

To see that the actual capacity  $Ku$  may be larger than the virtual battery capacity  $K_v u$ , consider two consecutive days of type 1 in Figure 5.10(a). In day 1, we keep  $U_k$  amount of energy. Thus, when day 2 begins, the actual battery already contains  $U_k$  amount of energy as shown in Figure 5.10(b), although the virtual battery state starts from zero. In the low-price zone of day 2, up to  $K_v u$  amount of energy can be



(a) Two consecutive days of type 1.



(b) Actual battery profile according to (a).

Fig. 5.10. Virtual battery state and corresponding actual battery state.

charged, which results in the actual battery state that reaches  $K_v u + U_k$ . Thus, in this example, we need  $K_v u + U_k$  capacity for the actual battery. In general, when we have  $m$  days of type 1 in a period, the actual battery capacity that we need is at most  $Ku = K_v u + (m - 1)U_k^{max}$ , where  $U_k^{max}$  is the maximum amount of energy that we keep for a day. Note that although the actual battery capacity is  $K_v u + (m - 1)U_k^{max}$ , we pretend to have a battery of capacity  $K_v u$  (*i.e.*, the virtual battery capacity) in each day. The extra capacity over the virtual capacity  $K_v u$  is solely for keeping energy for future use.

#### 5.4.5 Per-day usage flattening algorithm

---

**Algorithm 1** Energy Usage Flattening Across Days
 

---

```

1: for  $d = 1$  to  $P$  do
2:   if  $d \in I_d^1$  then
3:      $K_L = K_v - (U_a - U(d))/u$ 
4:      $U_k = \max\{B(n_L) - K_L u, 0\}$ 
5:     set  $E_k \leftarrow (E_k + U_k)$  before the  $(n + 1)$ -th measurement interval starts
6:   else
7:      $U_x = \min\{U(d) - U_a, E_k\}$ 
8:     set  $E_k \leftarrow (E_k - U_x)$  before the first measurement interval starts
9:   end if
10: end for

```

---

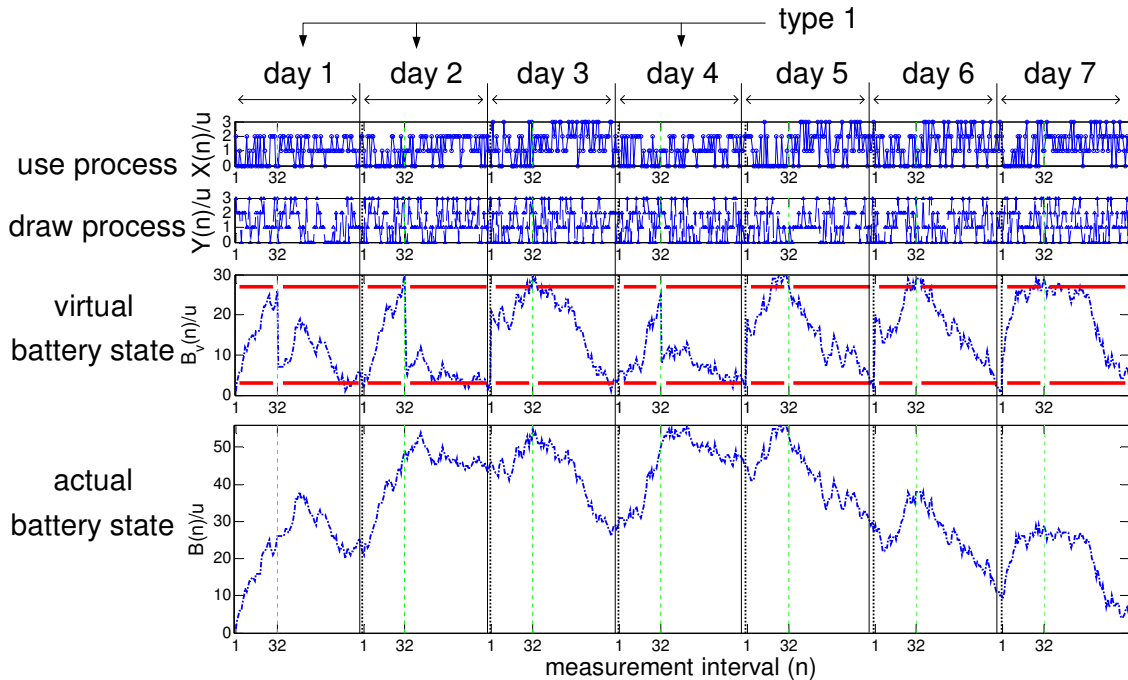


Fig. 5.11. Simulation results for PRIVATUS's per-day usage flattening.

The resulting algorithm for flattening the total usage per day is given in Algorithm 1. The initial value of  $E_k$  at the beginning of a cycle is the final value of  $E_k$  in the previous cycle. Note that in type 1 days, we may keep smaller amount of energy than  $(U_a - U(d))$ . We first calculate the battery level  $K_L u$  that is  $(U_a - U(d))$  smaller than  $Ku$ . Then, we only keep the energy beyond  $K_L u$  for future use (line 4). Similarly, in type 2 days, the amount of energy that we take from the kept energy may be smaller than  $(U(d) - U_a)$  if the kept energy  $E_k$  is not enough (line 7).

#### 5.4.6 Simulation study for per-day usage flattening

Figure 5.11 shows a simulation run for PRIVATUS's per-day usage flattening. In this simulation, we choose  $P = 7$ ,  $\alpha = 1$ ,  $\beta = 1$ , and  $K = 30$ . The other parameters are the same as in Section 5.3.2. Type 1 days consist of days 1, 2, and 4, where we randomly generate  $X(n)$  through  $P_X(n) = [0.6, 0.2, 0.2, 0.0]$  in the low-price zone

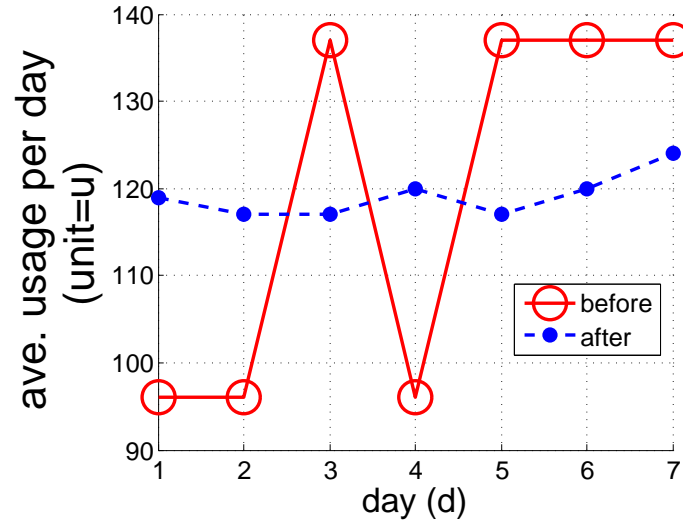


Fig. 5.12. Per-day average usage changes. In the figure, ‘before’ represents the per-day average use before flattening (*i.e.*,  $U(d)$ ), and ‘after’ means the per-day average use after flattening.

and  $P_X(n) = [0.2, 0.4, 0.4, 0.0]$  in the high-price zone. With this distribution, the average usage in a day of type 1 is  $96u$ . In type 2 days, we also generate  $X(n)$  randomly, but  $P_X(n)$  is changed to  $P_X(n) = [0.5, 0.2, 0.2, 0.1]$  in the low-price zone and  $P_X(n) = [0.1, 0.3, 0.4, 0.2]$  in the high-price zone, respectively. This distribution causes the average usage per day to be  $137u$  in a day of type 2. Thus, we have  $U_a = 120u$ .

We can see from the figure that there is a clear difference in the use process  $X(n)$  between type 1 and type 2 days. However, in the draw process  $Y(n)$ , it is hard to say if there is any difference across days. As explained in Section 5.4.3, the virtual battery state has a sudden drop before the start of the  $(n_L + 1)$ -th measurement intervals in a day of type 1, to keep at most  $24u$  ( $=120u-96u$ ) of energy. In a day of type 2, the virtual battery state has a sudden jump before the start of the first measurement interval, to use at most  $17u$  ( $=137u-120u$ ) of energy from the kept energy.

Figure 5.12 shows how the average usage changes with PRIVATUS. We can see the per-day average usage stays around  $120u$  across days after flattening. In Section 5.5,



we will see in more detail the influences of the per-day usage flattening on privacy protection.

#### 5.4.7 Effects on cost saving

In Section 5.2.2, we have discussed that the maximum possible saving for a day is determined by the amount of energy that is charged in the low-price zone and then used in the high-price zone. Denote this amount of energy by  $U_s$ . Without the per-day usage flattening, we know that  $U_s$  is equal to the actual battery capacity across days, *i.e.*,  $U_s = Ku$  for every day.

However, when the per-day usage flattening is applied,  $U_s$  becomes smaller than  $Ku$ , and varies according to the type of a day. Specifically, in a type 1 day, we charge  $K_vu$  amount of energy in the low-price zone, but use only  $K_vu - U_k$  amount of energy in the high-price zone. Thus, in a type 1 day, we have  $U_s = K_vu - U_k$ . On the other hand, in a type 2 day, although we charge  $K_vu - U_x$  amount of energy in the low-price zone, we use  $K_vu$  amount of energy, because the  $U_x$  amount of energy shortage is supplemented from the kept energy in type 1 days. The  $U_x$  amount of energy is the one that is charged in the low-price zone in days of type 1. Thus, we can say that  $U_s = K_vu$  in a type 2 day. When there are  $m$  days of type 1 out of  $P$  days of the period, the per-day average of  $U_s$  is  $\frac{1}{P}(m(K_vu - U_k^{max}) + (P - m)K_vu) = K_vu - \frac{m}{P}U_k^{max}$ , assuming  $U_k = U_k^{max}$  for each day of type 1.

Therefore, we have that the worst-case average for the maximum possible cost saving per day is  $(R_H - R_L)(K_vu - \frac{m}{P}U_k^{max})/u$ . Recall that without the per-day usage flattening, the maximum possible cost saving per day is  $(R_H - R_L)(K_vu + (m - 1)U_k^{max})/u$ . When we define the ratio  $r_s$  of the maximum possible cost savings with and without the per-day usage flattening as

$$r_s = \frac{K_vu - \frac{m}{P}U_k^{max}}{K_vu + (m - 1)U_k^{max}}, \quad (5.18)$$

we can see  $r_s \leq 1$ . This means that with the per-day usage flattening, we have a loss in terms of the cost saving. Figure 5.13 shows how much the loss is depending on

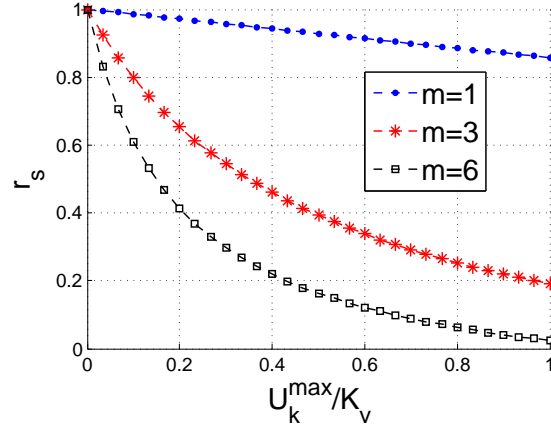


Fig. 5.13. Cost saving reduction due to the per-day usage flattening when  $P = 7$ . The ratio  $r_s$  is defined as  $r_s = \frac{K_v u - \frac{m}{P} U_k^{\max}}{K_v u + (m-1) U_k^{\max}}$ , which implies the ratio of the maximum possible cost savings with and without the per-day usage flattening.

conditions. First, we can see that as  $m$  increases with a fixed value of  $U_k^{\max}$ , the cost saving reduction also increases. This is because we keep more amount of energy in type 1 days for future use, by which we loss the chance to achieve the cost saving. For a similar reason, the increment in  $U_k^{\max}$  with a fixed value of  $m$  also results in the decrement in the cost saving. Note that when  $U_a$  remains constant,  $m$  and  $U_k^{\max}$  are inversely proportional to each other: the increment in  $m$  means the decrement in  $U_k^{\max}$ , and vice versa.

## 5.5 Experiment

### 5.5.1 Metrics and simulation parameters

First, we define the metric of information leakage from the use process to the draw process as follows: for a positive integer  $m$ ,

$$L_{(n,m)}^s = I(\bar{X}_{(n,m)}; \bar{Y}_{(n,m)}^s) / H(\bar{X}_{(n,m)}), \quad (5.19)$$

where

$$\bar{X}_{(n,m)}=[X(n-m+1),X(n-m),\dots,X(n)], \quad (5.20)$$

$$\bar{Y}_{(n,m)}^s=[Y(n-m+1+s),Y(n-m+s),\dots,Y(n+s)], \quad (5.21)$$

and  $s$  is a non-negative integer called the timeshift offset. Here,  $H(\mathcal{X})$  denotes the *uncertainty* of  $\mathcal{X}$ , and  $I(\mathcal{X};\mathcal{Y})$  is the mutual information between  $\mathcal{X}$  and  $\mathcal{Y}$ . Namely,  $H(\mathcal{X}) = -\sum_i P(\mathcal{X} = i) \log P(\mathcal{X} = i)$  and

$$I(\mathcal{X};\mathcal{Y})=\sum_i \sum_j P(\mathcal{X}=i,\mathcal{Y}=j) \log \frac{P(\mathcal{X}=i,\mathcal{Y}=j)}{P(\mathcal{X}=i)P(\mathcal{Y}=j)} \quad (5.22)$$

Note that  $\bar{X}_{(n,m)}$  and  $\bar{Y}_{(n,m)}^s$  represent sequences of length  $m$  in the use process and the draw process, respectively, with the draw process being time delayed by  $s$  measurement intervals. Since  $I(\bar{X}_{(n,m)}; \bar{Y}_{(n,m)}^s) = H(\bar{X}_{(n,m)}) - H(\bar{X}_{(n,m)} | \bar{Y}_{(n,m)}^s)$ , the metric  $L_{(n,m)}^s$  can be interpreted as a measure of the *uncertainty reduction* in  $\bar{X}_{(n,m)}$  by observing  $\bar{Y}_{(n,m)}^s$ , normalized to the uncertainty of  $\bar{X}_{(n,m)}$ . Thus, by this metric, we can quantify *how uncertain the adversary is when he attempts to guess the sequence  $\bar{X}_{(n,m)}$  of the use process, based on the observed sequence  $\bar{Y}_{(n,m)}^s$  of the draw process.* For example, the adversary knows that  $\bar{X}_{(n,m)}$  is surely the same as  $\bar{Y}_{(n,m)}^s$ , when  $L_{(n,m)}^s = 1$ . In contrast,  $L_{(n,m)}^s = 0$  means that  $\bar{Y}_{(n,m)}^s$  gives no clue about  $\bar{X}_{(n,m)}$  at all.

Second, given that the battery capacity is  $Ku$ , we define the metric for the cost saving for a day as

$$S_{(r,K)} = E \left( -\sum_{m=1}^{n_L} r R_H D(m) - \sum_{m=n_L+1}^{n_H} R_H D(m) \right), \quad (5.23)$$

where  $r$  denotes the ratio of  $R_L$  to  $R_H$ . The term  $S_{(r,K)}$  is the expected difference between the original cost for what the user actually consumes,

$$\sum_{m=1}^{n_L} r R_H X(m) + \sum_{m=n_L+1}^{n_H} R_H X(m), \quad (5.24)$$

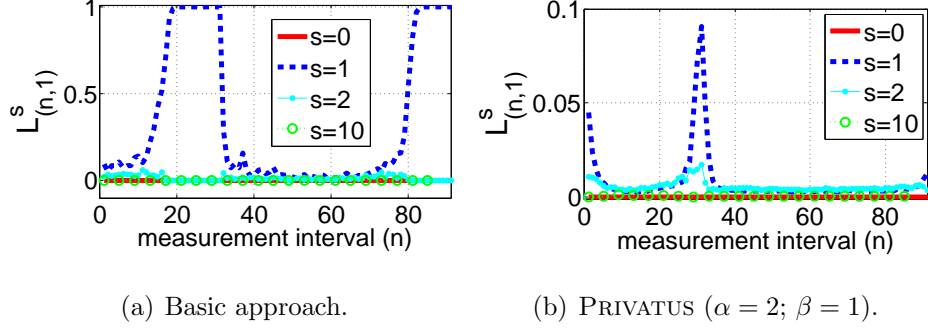


Fig. 5.14. Information leakage when  $K = 20$  and  $m = 1$ .

and the money that a user pays to the utility company,

$$\sum_{m=1}^{n_L} r R_H Y(m) + \sum_{m=n_L+1}^{n_H} R_H Y(m). \quad (5.25)$$

A positive value of  $S_{(r,K)}$  means that we achieve cost saving. If  $S_{(r,K)}$  is negative, it means that we have to pay more compared to the baseline no-privacy-protection scheme.

To be consistent with the previous simulations (in Figures 5.5 and 5.7), we use the same parameters as before (*i.e.*,  $M = 4$ ;  $K = 20$ ;  $n_L = 32$ ;  $n_H = 96$ ;  $\alpha = 2$ ;  $\beta = 1$ ;  $n_0 = 3$ ;  $c = 0.1$ ) throughout the whole experiments, unless otherwise stated. However, we randomly generate  $X(n)$  through  $P_X(n) = [0.5, 0.2, 0.2, 0.1]$  in the low-price zone and  $P_X(n) = [0.1, 0.3, 0.4, 0.2]$  in the high-price zone. This setting results in about  $137u$  for the expected daily usage  $E(\sum_{n=1}^{n_H} X(n))$ . To get the results, we run 100,000 days in such a way that the remaining energy in the battery at the end of a day becomes the initial energy level of the battery in the next day.

### 5.5.2 Information leakage and cost saving

**General performance trend:** Figure 5.14 shows the general performance trend of our solution approaches (for  $m = 1$ ). We can see that when  $s = 0$ ,  $X(n)$  and  $Y(n)$  are indeed independent in both the basic approach and PRIVATUS. We can

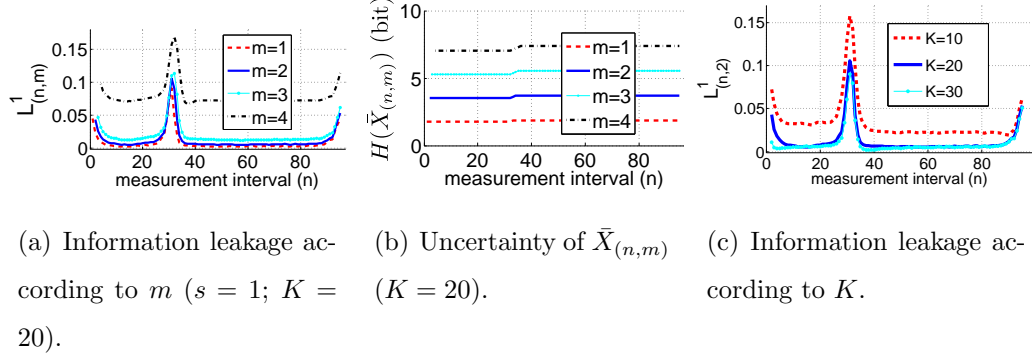
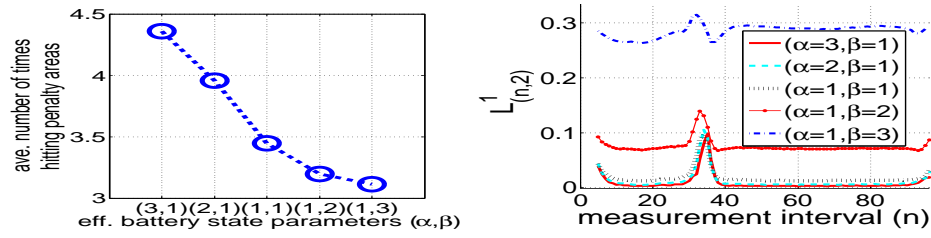


Fig. 5.15. Effects of sequence length  $m$  and capacity  $K$  in PRIVATUS ( $\alpha = 2$ ;  $\beta = 1$ ).

also see that information leakage is the highest when  $s = 1$ , *i.e.*,  $X(n)$  and  $Y(n + 1)$  has the highest dependency in our solution approaches. This is due to our solution's inherent nature that  $Y(n)$  is chosen to change the current battery state resulting from  $X(n - 1)$  and the previous battery state. Figure 5.14(a) confirms again that in the basic approach, this issue can be quite significant because  $Y(n)$  perfectly compensates  $X(n - 1)$  and reveals all information about  $X(n - 1)$  (*i.e.*,  $L^1_{(n,1)} = 1$ ) when the battery is in the corner cases. However, we see in Figure 5.14(b) that this compensation effect is greatly reduced. That is, in PRIVATUS,  $Y(n)$  results in mostly near-zero uncertainty reduction about  $X(n - 1)$ . In even the worst case (for some measurement intervals, with delay of 1 measurement interval), the uncertainty reduction is less than 10%. We see that the worst-case information leakage in the advanced approach occurs around the price zone boundaries. We suspect that this is because around the price zone boundaries, there is no penalty defined and thus the battery state has a relatively higher chance to remain constant, which again makes it more likely that  $Y(n)$  tries to compensate for  $X(n - 1)$ . On the other hand, we can see from the case when  $s = 10$  that, with higher delays (*i.e.*, larger values of  $s$ ), the sequences of the use process and the draw process become independent.

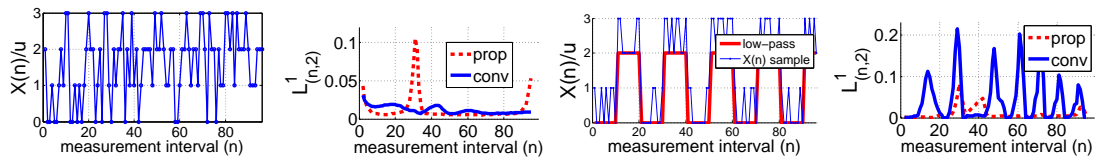
**Effect of sequence length:** In Figure 5.15(a), we see that in PRIVATUS, the information leakage increases as the sequence length  $m$  increases. This seems to imply that the adversary gains more information when he observes longer sequences. However, note from Figure 5.15(b) that the uncertainty of the use-process sequence  $H(\bar{X}_{(n,m)})$  also grows as  $m$  increases. In Figure 5.15(b),  $x$ -bit uncertainty can be understood in such a way that approximately the use-process sequence has  $2^x$  possible realizations with equal probability  $1/2^x$ . Since  $M = 4$ , the uncertainty of the use-process sequence becomes larger by a factor close to  $\log_2 4$  (more precisely,  $\log_2 2^{1.7}$  in our simulation setting) as  $m$  increases by 1. Thus, the minor increment in percentage-wise uncertainty reduction does not make it easier for the adversary to make guesses about the use-process sequence. For example, when  $m = 3$  and  $n = 32$ , the uncertainty of the use-process sequence is 5.3 bits and uncertainty reduction is 11%. This implies that the remaining uncertainty of the use-process sequence after observing the draw-process sequence is  $5.3(1 - 0.11) = 4.72$  bits, *i.e.*, the adversary faces the uncertainty to pick one out of  $2^{5.3(1-0.11)} = 26.3$  possible sequences, in order to make a guess about the use-process sequence. On the other hand, when  $m = 4$  and  $n = 32$ , the uncertainty is 7.0 bits and the uncertainty reduction is 17%. This results in  $2^{7.0(1-0.17)} = 56.1$  possible sequences as candidates for the use-process sequence. Therefore, we conclude that the adversary has no advantage in observing a longer sequence in the draw process.

**Effects of battery capacity:** Figure 5.15(c) shows how PRIVATUS acts when the battery capacity varies. We can infer from the figure that when the battery capacity is too small, information leakage may be significant. This can be explained again by the compensation effect of our solution. If the battery capacity is too small, there is not much room for the battery state to fluctuate between the two penalty area thresholds  $T_L$  and  $T_H$  (see Figure 5.7). This means that the battery state remains relatively constant, which makes the compensation effect prominent. On the other hand, once the battery capacity is above a threshold, further increasing the battery capacity leads to little benefit in terms of further reducing the information leakage.



(a) The number of times hitting the penalty areas in a day according to  $\alpha$  and  $\beta$ . (b) Information leakage according to  $\alpha$  and  $\beta$ .

Fig. 5.16. Effects of  $\alpha$  and  $\beta$  in PRIVATUS ( $K = 20$ ).



(a)  $X(n)$  w/o a significant low-pass component. (b) Information leakage for (a). (c)  $X(n)$  w/ a significant low-pass component. (d) Information leakage for (c).

Fig. 5.17. Information leakage comparison between PRIVATUS with  $\alpha = 2$  and  $\beta = 1$  (legend: ‘prop’) and an existing scheme [23] (legend: ‘conv’), when  $K = 20$  and  $m = 2$ . The higher is  $L^1_{(n,2)}$ , the worse is the information leakage.

**Effects of different values of  $\alpha$  and  $\beta$ :** Figure 5.16(a) shows the average number of times that PRIVATUS hits the penalty areas, given  $\alpha$  and  $\beta$ . As explained before, when the ratio of  $\alpha$  to  $\beta$  goes down, the frequency to hit the penalty areas also decreases. However, from Figure 5.16(b), we see a negative effect in terms of information leakage, when the ratio of  $\alpha$  to  $\beta$  is too low. In that case, the actual battery state wants to stay in the middle of the two penalty area thresholds  $T_H$  and  $T_L$  to avoid getting a penalty score. This makes the compensation effect larger.

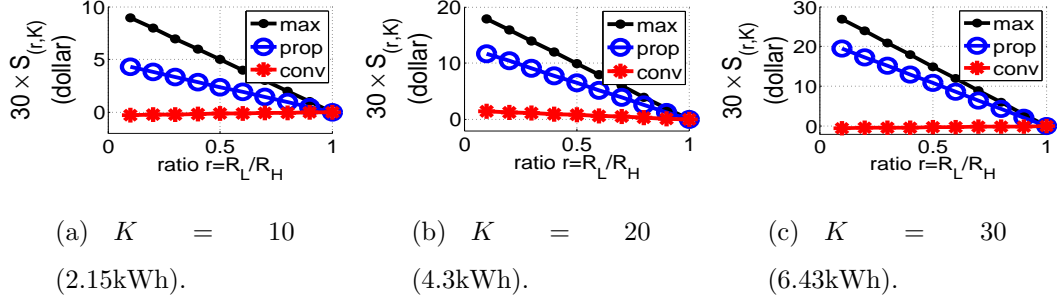
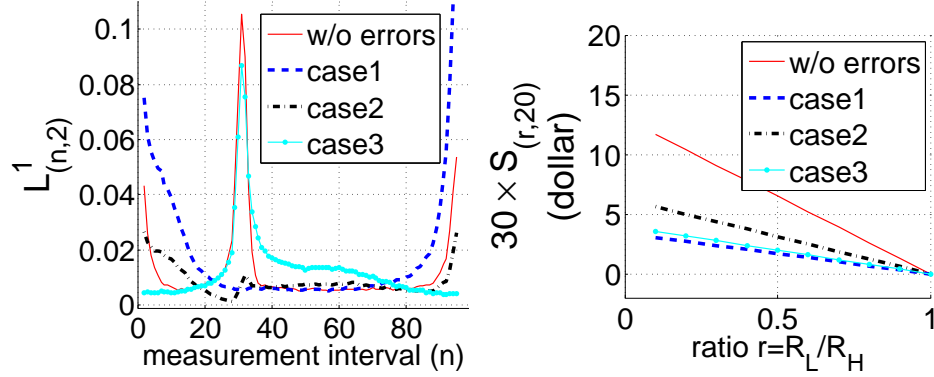


Fig. 5.18. Cost saving comparison between PRIVATUS and an existing scheme [23]. Here, we set  $u = 0.2143\text{kWh}$  and  $R_H = \$0.033/u = \$0.155/\text{kWh}$ . This results in the average daily usage (*i.e.*,  $E(\sum_{n=1}^{n_H} X(n))$ ) equal to 30kWh.

**Comparison to prior work:** In Figures 5.17 and 5.18, we compare PRIVATUS (‘prop’ in the figures) with an existing scheme (‘conv’ in the figures) proposed by Kalogridis *et. al.* [23]. Kalogridis’ scheme performs a simple low-pass filtering over the use process in a best-effort manner without considering the energy cost factor. Thus, it reduces the high frequency variations in the resulting draw process. Kalogridis’ scheme needs to estimate the value of  $X(n)$  beforehand (refer to [23] for detail). We assume in the simulation that the estimation is perfect (*i.e.*, without errors). Figure 5.17(a) shows a sample realization of  $X(n)$ , obtained from  $P_X(n)$  given in Section 5.5.1. Note that since  $X(n)$  is randomly chosen among  $M$  possible values from  $P_X(n)$ , which is the same within each price-zone, there is not a significant low-frequency component in  $X(n)$ . In this case, we can see from Figure 5.17(b) that PRIVATUS performs slightly better than Kalogridis’ to keep the privacy information, except at the price zone boundaries. However, if there is a significant low-pass component in  $X(n)$ , PRIVATUS will provide much better privacy protection than Kalogridis’. This is because Kalogridis’ scheme still allows the low-pass component of load profile to be revealed. To see this, we generate  $X(n)$  by adding a random value 0 or  $u$  to a rectangular pulse whose period is 20 measurement intervals, as shown in Figure 5.17(c). Comparison result in such a case is given in Figure 5.17(d). Indeed, PRIVATUS results





(a) Information leakage when  $m = 2$ , and  $s = 1$ . (b) Cost saving when  $u = 0.2143\text{kWh}$ , and  $R_H = \$0.03/u$ .

Fig. 5.19. Effects of the estimation error for the distribution of  $X(n)$  when  $K = 20$ .

in better lower information leakage than Kalogridis' when there exists a considerable low-frequency component in  $X(n)$ . Meanwhile, Figure 5.18 shows that from the cost saving point of view, PRIVATUS has a huge advantage against Kalogridis'. In all of the cases studied, Kalogridis' scheme does not achieve a significant cost saving. On the other hand, compared to the maximum possible cost saving, computed according to Section 5.2.2 ('max' in the figures), PRIVATUS achieves the saving of 48% of the maximum when  $K = 10$ , 66% of the maximum when  $K = 20$ , and 72% of the maximum when  $K = 30$ . Thus, PRIVATUS strikes a desirable balance between privacy and cost saving. Considering that the average electricity consumption for a U.S. residential customer was 30kWh per day [46], Figure 5.18(c) shows that a typical home can achieve about \$16 saving for a month with a 6.43kWh battery, based on the following tariff example:  $R_L = 0.04/\text{kWh}$  and  $R_H = 0.15/\text{kWh}$  [45].

**Effects of estimation error in  $P_X(n)$ :** In Figure 5.19, we study the effect of the estimation error in  $P_X(n)$ . For this, we consider the situation where our estimation is  $P_X(n) = [0.5, 0.2, 0.2, 0.1]$  in the low-price zone and  $P_X(n) = [0.1, 0.3, 0.4, 0.2]$  in the high-price zone (*i.e.*, the decision table is calculated based on these distributions),

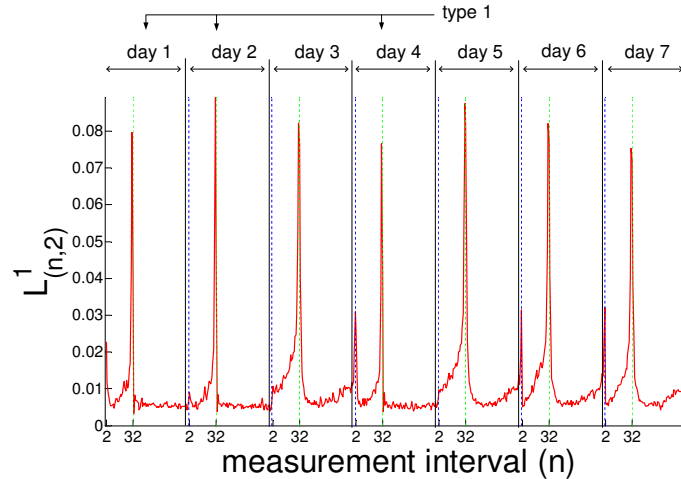


Fig. 5.20. Information leakage ( $m = 2$ , and  $s = 1$ ) with the per-day usage flattening, where  $K = 30$ ,  $\alpha = 1$ ,  $\beta = 1$ , and  $P = 7$ . Days 1, 2, and 4 are of type 1.

but  $X(n)$  is generated by different distributions:  $P_X(n) = [0.1, 0.2, 0.3, 0.4]$  ('case1'),  $P_X(n) = [0.25, 0.25, 0.25, 0.25]$  ('case2'), and  $P_X(n) = [0.4, 0.3, 0.2, 0.1]$  ('case3') for all  $n$ . Figure 5.19(a) shows that although there exists an estimation error in  $P_X(n)$ , it does not affect the information leakage much. However, we see some changes around the day boundary. Note that 'case1' may represent the situation where the use process requires more energy on average than we have estimated. In this case, since the battery has less chance to hold the energy, the battery state may stay flat around the start and the end of a day, which increases the information leakage there. The opposite situation is represented by 'case3', where the battery has more chance to hold some energy around the start and end of a day, since the use process requires less energy than estimated. Thus, the information leakage around the day boundary is reduced compared to the case without the estimation error. In the meantime, we see in Figure 5.19(b) that the estimation error influences the cost saving more significantly: the magnitude of the saving goes down with the estimation error.

#### Effects of the per-day usage flattening:

Figure 5.20 shows the information leakage after per-day usage flattening in the same environment as Section 5.4.6. Although days 1, 2, and 4 are of type 1, and others are of type 2, we see from Figure 5.20 that there is no significant difference in the information leakage across days. Compared to Figure 5.15(c), we can see that the per-day usage flattening of PRIVATUS does not change the privacy protection performance significantly at a similar condition.

## 5.6 Discussion

**Battery cost:** In Section 5.5, we showed that a 6.43kWh battery can achieve \$16 saving per month, assuming 30kWh use in a day. People may argue that this is the relatively small savings compared to the high battery cost. Indeed, initial costs for residential batteries range from \$80 to \$200 per kWh [47], and thus the battery cost of 6.43kWh may range from \$514 to \$1,280. However, note that people buy a hybrid car to save the fuel-cost and the environment, although it requires a considerable initial cost due to the battery. Even though the fuel saving of the hybrid cars does not completely offset its high cost, the fuel saving serves as a significant incentive for consumers (who may only be mildly environment-conscious) to buy hybrid cars. Similarly, in our case, the cost savings will encourage privacy-conscious customers to buy our solution. In addition, given a 6.43kWh battery and \$16 saving per month, the battery cost may be balanced out by the saving in 2.6 to 6.6 years. We think that this is similar to the period to recover the additional cost of a hybrid car compared to a normal car.

**Energy loss in a battery:** By multiplying coefficients ( $< 1$ ) by  $X(n)$  and  $Y(n)$  in (5.1), our model can be easily extended to include the energy loss in the battery that occurs when charging and discharging.

**More than two price zones:** Once we know the rates of energy usage and the boundaries of each price zone, we can calculate the desired pattern of battery charge and discharge—akin to that in Figure 5.3. Namely, what we need to do is to

calculate to what level the battery can be charged or discharged in each zone. Then, the solution approach outlined earlier applies directly to the case with more than two price zones.

**Interleaved low-price and high-price zones:** This situation is equivalent to the case where there are multiple price zones, one group of which have a low price, and the other group have a high price. Thus, this case can be treated in the same way as the above.

**The amount of energy usage per day varying over days:** This paper focuses on hiding the energy consumption pattern within a day. Across days, the total usage per day can still be revealed to the adversary (by which the adversary may know whether you are home or not for a given day). The other part of PRIVATUS, which is not presented in this paper due to the page limit, handles this issue. At the high level, the solution is to flatten the energy use across days, by charging more in days with less usage and by using the saved energy in days with more usage. The solution does not affect the current randomization framework within each day; it only modifies the total use in each day, and requires larger capacity battery.

**Complexity:** We saw that PRIVATUS may result in better privacy protection than our basic approach. However, a tradeoff is that its computation complexity is increased, mainly because of the increment in the size of the decision table for  $P_Y(n)$ . Let us compare the number of the decision table entries in the low-price zone. Basically, the number of entries in the decision table is the number of possible state vectors (*i.e.*,  $[n, B(n-1)]$  for the basic approach and  $[n, B(n-1), B_e(n-1), P_Y(n-1)]$  for PRIVATUS). Thus, the decision table in the basic approach has  $n_L(K+1)$  entries. Regarding the effective battery state, it can have its minimum value when  $B(n) = 0$  for all the times in the zone. In this case, the penalty score is  $-\beta(M-1)(n_L - 2n_0)u$ . The maximum value of the effective battery state becomes  $\alpha Ku$ . On the other hand, the possible choices for  $P_Y(n)$  in PRIVATUS are the elements in the set  $\cup_{k=0}^K \mathcal{P}_k$ . Denoting the number of elements in this set by  $Q$ , PRIVATUS has  $n_L(K+1)(\alpha K + \beta(M-1)(n_L - 2n_0) + 1)Q$  entries in the decision table. Note that the number of the entries

is proportional to  $O(n_L K)$  in the basic approach, but  $O(n_L^2 K + n_L K^2)$  in PRIVATUS. For example, with the parameters used in Sections 5.2.4 and 5.3.2, the decision tables in the basic approach and PRIVATUS have 672 and 3,358,656 entries, respectively. Although PRIVATUS's decision table gets much larger, we think that, compared to the cost saving, the memory cost to accommodate such a table is reasonable (see the memory price trend in [48]: the per-Mbyte cost in 2010 is only \$0.0122). Further, the computation overhead to obtain the decision table is proportional to its size. However, the table calculation needs to be carried out infrequently, *i.e.*, only when  $P_X(n)$  changes significantly. The computation overhead of PRIVATUS in normal operations (*i.e.*, looking up the decision table) is very low.

## 6. CONCLUSION

We have presented a synchronization protocol called HARMONIA targeted to low duty cycle multi-hop wireless networks. The requirements for the synchronization protocol come from a wastewater monitoring and actuation application called CSOnet, deployed city-wide in South Bend, Indiana. CSOnet has been operational for over a year now. CSOnet used to have a synchronization protocol which exchanges synchronization messages once every day, but needs manual resynchronization at an average rate of once every 30 days due to its coarse synchronization accuracy and lack of failure handling mechanism. Based on experiments done by EmNet, LLC, in small segments of the network, HARMONIA is expected to eliminate the inconvenience of manual synchronization and reduce the frequency of synchronization from once a day to once every 5 days (recall the frequency of once every 13 hours calculated in Section 4.5.3 is the worst-case estimate).

The nodes in CSOnet stay awake for only 6 seconds in every 5-minute interval in current deployment and use an external clock called the RTC, which has a low drift, but a coarse 1 second resolution. The RTC is used for driving the sleep-wake periods on the nodes. The radio used on the nodes does not allow MAC layer time-stamping, a technique commonly used in synchronization protocols. The fundamental innovation in HARMONIA can be simply stated as follows - it uses a fine granularity clock (MCC) with a relatively high drift rate to achieve synchronization of a coarse granularity clock that runs even when the node is asleep. Additionally, this process is done quickly so that in the common case, for reasonably sized networks (say, less than 17 hops in diameter) the process can be accomplished within 2 seconds, one-third of one wake-up interval of the network. By this, the synchronization error is in the microsecond range despite the coarse granularity of the RTC. In case that some parts of the network remain unsynchronized due to the time limit of the awake period, HAR-

MONIA’s fast recovery mechanism attempts to synchronize them in the next slot, not waiting for BS to initiate another synchronization round. The fast recovery can also locally handle transient node- and link-failures, not overburdening the whole network with synchronization-related messages. Experiment results show that HARMONIA’s synchronization error is higher than that of FTSP, but is still acceptable for CSOnet, being in the range of tens of microseconds. However, HARMONIA is significantly faster than FTSP with respect to the network-wide synchronization time, making it a good fit for low-duty cycle networks.

In event monitoring, a number of sensor nodes are deployed over a region where some phenomenon is to be monitored. When an event is detected, the sensor nodes report it to a base station, where a network operator can take appropriate action using the event report. However, such an event reporting process can be easily attacked by compromised nodes in the middle that drop, modify, or delay the report packet. No prior work has been able to provide a security guarantee for timely and reliable collection of event reports at a base station in the presence of Byzantine adversarial nodes that are capable of colluding among themselves. To resolve this issue, we have presented SEM, a secure event monitoring protocol. SEM can work in hostile environments, where Byzantine adversary nodes want to disrupt the event reporting procedures. As long as the compromised nodes do not want to be detected, SEM enables a legitimate sensor node to report an event to the BS within a bounded time; If the compromised nodes prevent the event from being reported to the BS within the bounded time, SEM will be able to identify a small set of nodes that is guaranteed to contain a malicious node.

In order to resolve the privacy issue in smart grid, we proposed PRIVATUS to de-correlate the meter reading information from user behavior. PRIVATUS uses a rechargeable battery to make the meter reading reported to the utilities look independent of the actual usage at any given measurement interval. The correlation between the meter readings and the actual usage pattern over multiple measurement intervals is also reduced by changing the probability distribution of charging the bat-

tery in each interval through careful design. PRIVATUS is also geared to the future of time-of-use pricing of electricity and it ensures that the battery is charged to achieve the maximal savings in the energy cost. We formulate the problem rigorously and use stochastic dynamic programming to devise our solution. The experiment results show that PRIVATUS is successfully able to hide the actual usage from what is drawn from the grid, and achieves considerable amount of saving in the energy cost, subject to the availability of a reasonable-sized battery. Compared to prior work, we achieve much better privacy when there is a conspicuous low-frequency component in load profile, and significantly higher cost savings. PRIVATUS can also flatten the per-day average usage across days at the expense of the decrement of cost saving.



## 7. FUTURE WORK

The research work presented in this thesis provides a foundation to explore several future research avenues in embedded systems. Below, we summarize possible research directions that we can further consider.

1. Tree topology creation in HARMONIA and test in a real network.
2. SEM deployment into a testbed.
3. Handling practical issues in PRIVATUS.

### 7.1 Tree topology creation in HARMONIA

Currently, we are missing an algorithm that makes a tree topology in CSOnet. For this reason, we are hardcoding the topology information at each node. This requirement leads to unnecessary inconvenience to modify the topology information at a node whenever the tree topology changes. To resolve this issue, we would design a protocol that automatically makes the tree topology without human intervention.

### 7.2 SEM deployment into a testbed

We are planning to deploy SEM into a testbed. This work will involve implementing packet fragmentation and reassembly between a sender and a receiver, combining with TinyECC cryptography suite. For neighbor authentication, a mechanism for sharing a symmetric key between neighbors should also be implemented. In case of large-scale deployment, an automatic ECC formation algorithm would be considered.

### 7.3 Handling practical issues in PRIVATUS

Our future work will focus on generalizing PRIVATUS to handle various practical issues, *e.g.*, energy loss in a battery. Currently, we assume there is no loss when charging and discharging the battery. However, this assumption does not hold in practice. Appropriate loss modeling would be needed to figure out a more accurate amount of cost saving. Further, we will incorporate the non-linear characteristic of the battery into our model.

We will also collect real meter reading data from dynamic environment. This would help us study further about user activity patterns, and come up with more reasonable estimates for possible cost saving.

## LIST OF REFERENCES

## LIST OF REFERENCES

- [1] D. Mills, “Internet time synchronization: The network time protocol,” *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.
- [2] F. Sivrikaya and B. Yener, “Time synchronization in sensor networks: A survey,” *IEEE network*, vol. 18, no. 4, pp. 45–50, 2004.
- [3] S. Ganeriwal, R. Kumar, and M. Srivastava, “Timing-sync protocol for sensor networks,” in *Proc. of 1st intl. conf. on Embedded networked sensor systems*, pp. 138–149, 2003.
- [4] M. Maroti, B. Kusy, G. Simon, and Á. Lédeczi, “The flooding time synchronization protocol,” in *Proceedings of the 2nd intl. conf. on Embedded networked sensor systems*, pp. 39–49, 2004.
- [5] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, “Elapsed time on arrival: A simple and versatile primitive for canonical time synchronization services,” *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC)*, vol. 1, no. 4, pp. 239–251, 2006.
- [6] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, “Firefly-inspired sensor network synchronicity with realistic radio effects,” in *Proceedings of the 3rd intl. conf. on Embedded networked sensor systems*, pp. 142–153, 2005.
- [7] T. Schmid, Z. Charbiwala, J. Friedman, Y. H. Cho, and M. B. Srivastava, “Exploiting manufacturing variations for compensating environment-induced clock drift in time synchronization,” *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 97–108, 2008.
- [8] T. Schmid, J. Friedman, Z. Charbiwala, Y. H. Cho, and M. B. Srivastava, “Low-power high-accuracy timing systems for efficient duty cycling,” in *ISLPED ’08: Proceeding of the thirteenth intl. symposium on Low power electronics and design*, pp. 75–80, 2008.
- [9] P. Sommer and R. Wattenhofer, “Gradient Clock Synchronization in Wireless Sensor Networks,” in *Information Processing in Sensor Networks, 2009. IPSN’09. Intl. Conf. on (To Appear)*, pp. 1–12, 2009.
- [10] J. Carle and D. Simplot-Ryl, “Energy-efficient area monitoring for sensor networks,” *Computer*, vol. 37, pp. 40–46, February 2004.
- [11] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, and L. Gu, “Energy-efficient surveillance system using wireless sensor networks,” in *In Mobisys*, pp. 270–283, ACM Press, 2004.

- [12] T. Yan, T. He, and J. A. Stankovic, "Differentiated surveillance for sensor networks," in *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, SenSys'03, 2003.
- [13] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, (Washington, DC, USA), pp. 8020–, IEEE Computer Society, 2000.
- [14] P. Rothenpieler, D. Krger, D. Pfisterer, S. Fischer, D. Dudek, C. Haas, and M. Zitterbart, "FleGSens - secure area monitoring using wireless sensor networks," *World Academy of Science, Engineering and Technology*, August 2009.
- [15] D. E. Robling Denning, *Cryptography and data security*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1982.
- [16] D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms," PODS '01, (New York, NY, USA), pp. 247–255, ACM, 2001.
- [17] L. Sweeney, "k-anonymity: a model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, pp. 557–570, October 2002.
- [18] R. Stallman, "Is digital inclusion a good thing? How can we make sure it is?," *Comm. Mag.*, vol. 48, pp. 112–118, February 2010.
- [19] NIST, "Nist framework and roadmap for smart grid interoperability standards, release 1.0," *Nist Special Publication*, vol. 0, pp. 1–90, 2009.
- [20] H. Khurana, M. Hadley, N. Lu, and D. A. Frincke, "Smart-grid security issues," *IEEE Security and Privacy*, vol. 8, pp. 81–85, 2010.
- [21] E. L. Quinn, "Privacy and the new energy infrastructure," *SSRN*, 2009.
- [22] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, ACM, 2011.
- [23] G. Kalogridis, C. Efthymiou, S. Z. Denic, T. A. Lewis, and R. Cepeda, "Privacy for smart meters: Towards undetectable appliance load signatures," *2010 First IEEE International Conference on Smart Grid Communications*, 2010.
- [24] S. McLaughlin, P. McDaniel, and W. Aiello, "Protecting consumer privacy from electric load monitoring," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, 2011.
- [25] D. P. Varodayan and A. Khisti, "Smart meter privacy using a rechargeable battery: Minimizing the rate of information leakage," in *ICASSP*, 2011.
- [26] D. P. Bertsekas and S. E. Shreve, *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 2007.
- [27] M. Schütze, A. Campisano, H. Colas, W. Schilling, and P. Vanrolleghem, "Real time control of urban wastewater systems - where do we stand today?," *Journal of Hydrology*, vol. 299, no. 3-4, pp. 335–348, 2004.

- [28] Tinyos-help, “FTSP on Tmotes.” <http://www.mail-archive.com/tinyos-help@millennium.berkeley.edu/msg07079.html>.
- [29] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, “Fidelity and yield in a volcano monitoring sensor network,” *7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pp. 381–396, 2006.
- [30] Maxim Inc., “DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal.” [http://www.maxim-ic.com/quick\\_view2.cfm/qv\\_pk/4627](http://www.maxim-ic.com/quick_view2.cfm/qv_pk/4627).
- [31] L. Montestruque and M. Lemmon, “Csonet: a metropolitan scale wireless sensor-actuator network,” in *MODUS '08: International Workshop on Mobile Device and Urban Sensing*, 2008.
- [32] Digi International Inc., “9XTend OEM RF Module.” <http://www.digi.com/products/wireless/long-range-multipoint/xtend-module.jsp>.
- [33] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, “Design of a wireless sensor network platform for detecting rare, random, and ephemeral events,” in *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, IEEE Press, 2005.
- [34] S. Massoud Amin and B. Wollenberg, “Toward a smart grid: power delivery for the 21st century,” *Power and Energy Magazine*, vol. 3, pp. 34–41, September 2005.
- [35] “Perimeter security alarm system.” <http://www.hkvstar.com>.
- [36] S. Marti, T. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in *MOBICOM*, 2000.
- [37] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, “ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks,” *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 4, pp. 1–35, 2008.
- [38] X. Zhang, A. Jain, and A. Perrig, “Packet-dropping adversary identification for data plane security,” in *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, 2008.
- [39] A. Liu and P. Ning, “TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks,” in *Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, IPSN'08, 2008.
- [40] “TOSSIM tutorial.” <http://docs.tinyos.net/index.php/TOSSIM>.
- [41] “TinyOS Documentation Wiki.” <http://docs.tinyos.net/index.php>.
- [42] “Building a network topology for TOSSIM.” <http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/usc-topologies.html>.
- [43] H. Beckman, “Lawsuit filed to stop installaton of smart meters.” <http://napervillesun.suntimes.com/news/9723766-418/lawsuit-filed-to-stop-smart-meter-installation.html>.

- [44] B. Sullivan, “What will talking power meters say about you?.” <http://redtape.msnbc.msn.com>.
- [45] Tucson electric power, “Residential time-of-use pricing plan.” <http://https://www.tep.com/doc/customer/rates/R-21F.pdf>.
- [46] U. E. I. Administration, “Average electricity consumption for a us residential customer.” <http://www.eia.gov/tools/faqs/faq.cfm?id=97&t=3>.
- [47] U.S. Department of Energy, “Battery power for your residential solar electric system.” <http://www.nrel.gov/docs/fy02osti/31689.pdf>.
- [48] J. C. McCallum, “Memory prices (1957-2010).” <http://www.jcmit.com/memoryprice.htm>.

VITA



## VITA

Jin Kyu Koo received his B.E. degree in Electrical Engineering from Korea University, Seoul in August 2001. He obtained his M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST) in February 2004. He joined Purdue University in the fall of 2007 as a Ph.D. student in Electrical and Computer Engineering. He worked as a research assistant for Prof. Saurabh Bagchi and Prof. Xiaojun Lin. His research interests include embedded systems, sensor networks, network security, optimization, and operating systems. He has worked as a research engineer for 4G communication systems at Samsung Electronics for more than 3 years. He spent a summer during his Ph.D. study at Bosch Research and Technology Center in Palo Alto, CA to develop an embedded sensor platform.