Purdue University
Purdue e-Pubs

Open Access Dissertations | Theses and Dissertations

Fall 2014

Digital provenance - models, systems, and applications

Salmin Sultana
Purdue University

Follow this and additional works at: https://docs.lib.purdue.edu/open_access_dissertations

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Sultana, Salmin, "Digital provenance - models, systems, and applications" (2014). *Open Access Dissertations*. 370.
https://docs.lib.purdue.edu/open_access_dissertations/370

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Sultana, Salmin

Entitled

Digital Provenance - Models, Systems, and Applications

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

ARIF GHAFOOR
Chair

XIAOJUN LIN

ELISA BERTINO

NINGHUI LI

SONIA A. FAHMY

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Elisa Bertino

Elisa Bertino

Approved by: Michael R. Melloch 09-22-2014
Head of the Graduate Program Date

DIGITAL PROVENANCE - MODELS, SYSTEMS, AND APPLICATIONS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Salmin Sultana

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2014

Purdue University

West Lafayette, Indiana

To my parents, Md. Abu Siddique and Sayeda Parvin, and my late aunt, Rahima Khatun.

ACKNOWLEDGMENTS

For me, the PhD life was not all about pursuing a degree, it was a journey that helped me grow up in all the academic, professional, personal, and social aspects of life. I would like to take this opportunity to convey my gratitude to all the people who positively influenced me throughout this journey. At first, I would like to thank my advisor, Professor Elisa Bertino, for her support, encouragement, and guidance over the years. She extended her help and thoughtful advice to me, whatever it was about an academic or personal matter. Nonetheless, she always inspires me through her wisdom, intelligence, passion, accomplishments, leadership, and contributions to the scientific community. I also thank my committee members: Dr. Arif Ghafoor, Dr. Sonia Fahmy, Dr. Ninghui Li, and Dr. Xiaojun Lin for their valuable comments and feedback on my research. My labmates, who I supervised, collaborated with, or had small talk with every now and often, have been an important part of my academic development. Special thanks to Aditi Gupta, who I shared with the same struggle of the PhD life. We used to exchange views on various issues and cheered each other up with hope and aspiration from time to time. I am also thankful to the ECE graduate office staffs, Matt Golden and Michelle Wagner, who were readily available for any help with the paperworks and necessary resources.

My summer internships at Intel labs played a significant role in my graduate study as the experience, on one hand, boosted my confidence and on the other hand, gave me exposure to the real-world security problems and industry research. I came to realize the immense scope and necessity of security research that widened my research views. I want to thank the group manager David Durham for offering me two summer internships, my mentors Ken Grewal and Ravi Sahita, my colleagues Michael Lemay, Prashant Dewan, and Rekha Bachwani for their cooperation towards making the

summers more productive and enjoyable. I would especially thank Scott Robinson for all those hour-long conversations on the various aspects of security research.

I must thank my friends for being there whenever I needed and listening to me. Special thanks go to Rezwana Karim, Sadia Afroz, and Samira Khan for all the LOL conversations and intellectual discussions we had, which helped me de-stress and get rid of boredom. I was inspired by the positive attitude and passion for work of my friend Susmita Dash. My deepest gratitude goes to her for sharing some wonderful moments during a summer internship and making me a delicious cup of tea every afternoon. I am also grateful to the vibrant and extremely helpful Bangladeshi students (too many to list here) at Purdue, who will make a newcomer feel like home in this foreign land. I will cherish the beautiful moments we spent together in the parties, hangouts, rehearsals of various cultural events we organized, and Bangla music concerts. I particularly recognize the graciousness of Enamul vai and Leena apu, who picked me up from the airport when I first came here, welcomed with warm heart, and hosted me for the next few days despite their very critical situation which I was not aware of. And their affectionate support to me never phased out. I also acknowledge the love and support from Aizaz Bhuiyan and Samina Shams Khan, who cooked us delicious meals every time we were coming back from an internship or Bangladesh. Not to mention, they opened their door to us anytime we were in need.

I would not have made it this far without the continuous love, support, and encouragement from my family. I feel privileged to be born to the parents who valued me as a human being, not just as a girl, raised me as an independent person with self-esteem, believed in me, and inspired me in all of my pursuits. My mother is my greatest inspiration in life, who always regarded education highly and often walked the extra mile for our better development. My father is the coolest person who supported me through thick and thin and encouraged me to work hard. My younger sister has been my best friend who always believed in me and took great care of me. My younger brother has grown up as a reliable person who I can share and discuss with any problem. I always want to get back the good times, laughter, and

joy I had with my siblings. My maternal grandparents, uncles, and aunts have always been a source of motivation for pursuing my dreams and achieving success. Almost every day since my youngest aunt, Rahima Khatun, died in 2000, I remember her. Her morale, compassion, attitude to life, and struggles encourage me to get focused and attain the best. I am also grateful to my mother- and father-in-law who always consider my academic success a top priority and wish all the best for my health and career.

Finally, I cannot thank enough my husband and best friend over the last few years, S M Iftekharul Alam. All the everyday details, research problems, books/movies, recent incidents, or any random questions that we shared, discussed on or fought over have helped me see things from different perspectives, and think more critically. I am also deeply moved by his confidence, focus, intellectual ability, and achievements that inject confidence and positive energy in me. He never fails to uplift me and make me smile when I go through a rough patch. I truly thank my husband for sticking to my side, even when I was irritable and frustrated. Last but most importantly, I am grateful to the almighty Allah for all the favors He bestowed upon me.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

WSN	Wireless Sensor Network
BS	Base Station
IPD	Inter-packet Delay
BF	Bloom Filter
iBF	In-packet Bloom Filter
IDS	Intrusion Detection System
IRPS	Incident Response and Prevention System
OS	Operating System
VFS	Virtual File System
FiPS	File Provenance System
SAO	Security Aware Object
SEE	Secure Execution Environment

ABSTRACT

Sultana, Salmin Ph.D., Purdue University, December 2014. Digital Provenance - Models, Systems, and Applications. Major Professors: Elisa Bertino and Arif Ghafoor.

Data provenance refers to the history of creation and manipulation of a data object and is being widely used in various application domains including scientific experiments, grid computing, file and storage system, streaming data etc. However, existing provenance systems operate at a single layer of abstraction (workflow/process/OS) at which they record and store provenance whereas the provenance captured from different layers provide the highest benefit when integrated through a unified provenance framework. To build such a framework, a comprehensive provenance model able to represent the provenance of data objects with various semantics and granularity is the first step. In this thesis, we propose a such a comprehensive provenance model and present an abstract schema of the model.

We further explore the secure provenance solutions for distributed systems, namely streaming data, wireless sensor networks (WSNs) and virtualized environments. We design a customizable file provenance system with an application to the provenance infrastructure for virtualized environments. The system supports automatic collection and management of file provenance metadata, characterized by our provenance model. Based on the proposed provenance framework, we devise a mechanism for detecting data exfiltration attack in a file system. We then move to the direction of secure provenance communication in streaming environment and propose two secure provenance schemes focusing on WSNs. The basic provenance scheme is extended in order to detect packet dropping adversaries on the data flow path over a period of

time. We also consider the issue of attack recovery and present an extensive incident response and prevention system specifically designed for WSNs.

1. INTRODUCTION

Data provenance refers to the genesis and evolution of a data object as it is processed in and across systems. The most common forms of provenance describe relationships between the input and output data in the form of *why*, *where*, and *how* provenance. *Where* provenance identifies what pieces of input data contributed to the output data, *Why* provenance tracks the processes and inputs that created or transformed the data whereas *How* provenance describes in detail how the output data was produced.

The concept of provenance has been extensively studied for a long time, and widely used in the archival community to denote the chain of ownership and the manipulation history of a document which support document viability, authenticity, and identity in preservation contexts [1]. In digital systems, provenance has had its widest adoption in the scientific and grid computing applications in order to document workflows, data generation and processing which are sufficient to enable reproduction and validation of results [2]. Recording provenance for GIS data helps the user to decide if the data meets the requirements of their applications [18]. The database community has recently explored how to support provenance collection in database records and streams with a view to assuring the accuracy and currency of data. Others have examined the usage of provenance for social networks, general information retrieval processes, operating and storage systems. A more recent area of focus is cloud computing systems where provenance can be used for resource usage and cost optimization, data reliability, fault detection, etc.

In addition to facilitating data reproducibility, rights protection, regulatory compliance, and authentication of information, provenance can be considered as a useful tool in data and systems security. Recent research works manifest the key contribution of provenance in evaluating the trustworthiness of data streams, for example, sensor data [3], location data [4], and multi-hop network [5], etc. Systems security

can be benefited by the forensics facility of provenance which helps expose the dependencies between various components (e.g. processes, pipes, files etc.) in a system or cross-correlate events on different machines and gives system administrators a more complete picture of component interactions thus easing the troubleshooting. Understanding the provenance of polymorphic malware strains can lead to new techniques for detecting and classifying unknown attacks [6]. Possibilities of other security functionalities include performing intrusion detection [7], identifying data exfiltration [8], etc.

While data provenance has been gaining interest as a desideratum in various domains, supporting provenance in current information systems poses several technical challenges. The contents of provenance verily depend on the context and the goals of its usage, and thus has implications on the provenance collection and management model. Existing provenance systems mostly operate at a single level of abstraction at which they record and store provenance. Provenance techniques for experimental systems, such as Chimera [9], ESSW [10], record provenance at the semantic level of application. Other application level provenance systems capture provenance at the level of business objects, lines of source code or other units with semantic meaning to the context. Service oriented workflow systems record provenance at workflow stages and data/message exchange points. System-call based systems such as ES3 [11], PASS [12] operate at the level of system processes and files. While the provenance collected at each level of abstraction is useful in its own right, the integration of provenance captured from different layers achieves the highest benefit of the data provenance. Moreover, most of these systems have their own proprietary protocols for managing provenance, leading to the lack of interoperability at the syntactic level between provenance records generated by heterogeneous systems. Without a unified provenance infrastructure, provenance generated by individual components cannot relate to each other across different layers and systems. To build a unified provenance infrastructure, an expressive provenance model able to represent the provenance of data objects with various semantics and granularity is the first crucial step. In this

thesis, we propose a comprehensive provenance model that can encapsulate the data provenance captured at different stages of a physical/computational process. The model captures the characteristics of standard provenance models (e.g. OPM) and ensures the inter-operability of provenance across different systems.

We then explore the secure provenance solutions for various distributed systems and investigate the utility of a unified provenance model in addressing provenance issues. Our focus is on secure and flexible yet scalable provenance collection, management, communication, and usage in streaming data systems, wireless sensor networks and virtualized environments. Provenance records can grow significantly over time, adding data storage, transmission, and processing costs. Hence, such provenance systems must be vigilant in managing data. Security issues, such as integrity, access control, reliable dissemination, etc. also add significant challenges to making provenance service available and trustworthy. Although provenance modeling, collection, and querying have been investigated extensively for workflows and curated databases, provenance for the considered applications has not been properly addressed. Moreover, existing provenance collection mechanisms fail to achieve sufficient breadth or fidelity to address the challenges mentioned above. In this context, the contributions of this thesis can be summarized as follows:

1.1 Provenance Model

We propose a comprehensive provenance model that is (i) generic enough to record the provenance of any data object, (ii) unified to capture and integrate both the application and system level metadata, and (iii) tailored to fine grained access control and originator preferences on provenance. The expressive nature of the model enables a wide range of provenance queries. We also illustrate the utility of our model in real world data processing systems.

1.2 Provenance Capture and Storage

With the emergence of distributed services (cloud computing, virtualization techniques) and electronic exchanges, the need to identify data origin and its lifecycle history becomes more intense with respect to ensuring full transparency and accountability. In this context, we pursue the design and implementation of a provenance collection system for virtualized environments. Fig. 1.1 shows the provenance framework we consider for this work. As the initial step, we focus on local file provenance system and present the design of such a system, named as FiPS. It is a file system that not only manages files but also transparently captures, stores and manages file provenances. FiPS autonomically collects sufficient metadata, conforming to our provenance model discussed earlier, in order to recreate a file.

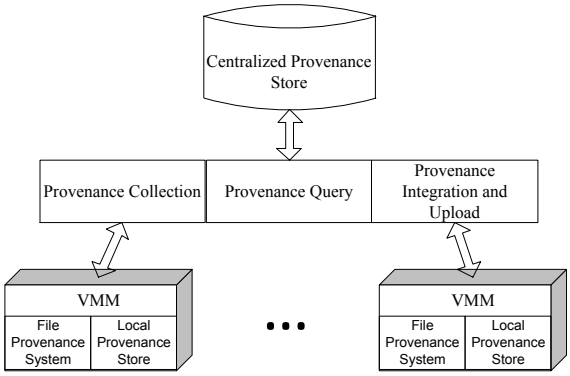


Fig. 1.1.: Provenance Framework for Virtualized Environment

We design FiPS as a thin layer operating between the Virtual File System (VFS) and the underlying file system. In contrast to a system-call based provenance approach [7], we intercept file system calls passed through the VFS layer and then generate provenance records. System call level approaches often fail to see how a system call activity is translated into multiple actions in the lower layers of the OS. Memory-mapped I/O can only be traced at the file system level. In addition, server-side operations of network file system (NFS) are performed directly in the kernel, not

through system calls. The system allows provenance output to a local or networked storage.

1.3 Provenance Communication

Data outsourcing and sharing in distributed services introduce the problem of provenance communication amongst disparate systems. We direct our research on secure and efficient provenance communication to an emerging class of distributed computing applications, namely the stream processing systems (focusing on sensor networks). Here, important challenges arise due to the ephemeral data, and tight storage, energy and bandwidth constraints of the sensor nodes. Therefore, it is necessary to devise a light-weight provenance solution which does not introduce significant overhead. Furthermore, sensors often operate in an untrusted environment, where they may be subject to attacks. Hence, it is necessary to address security requirements such as *confidentiality*, *integrity* and *freshness* of provenance. Our goal is to design a *provenance encoding and decoding mechanism* that satisfies such security and performance needs.

We propose a framework that transmits provenance along with the sensor data by hiding it over the inter-packet delays (IPD) (i.e. the delay between sensor data packets). The provenance of a data packet includes the identities of nodes in the data flow path. Each node in the path encodes one bit of provenance information over each IPD. Hence, the provenance can be decoded by processing the IPDs required to encode all the provenance bits. The embedding of provenance within a host medium makes our technique reminiscent of watermarking [13]. However, since the IPDs are used as watermark carrier, there is no data degradation due to watermarking. We ensure the scalability of the scheme by adopting a *spread spectrum* based technique which supports multi-user communication over the same medium [14].

An underlying assumption for this solution is the provenance remains same for at least a flow of packets. The assumption is reasonable since a routing path does not

change too often. Once the path is constructed, it is stable for a good amount of time until there is any link or node failure. However, some sensor network applications need to last a long time with more stringent energy requirements requiring the network operate at a very low duty cycle. Such low-duty-cycle operation significantly reduces node communication, sensing duty cycles, and hence data transmission. For such low data rate WSNs, we propose a fine-grained per-packet provenance encoding strategy whereby each node on the path securely embeds provenance information within a *Bloom filter*(BF), that is transmitted along with the data. Upon receiving the data, the Base Station extracts and verifies the provenance. While traditional provenance security solutions use intensively cryptography and digital signatures [15], and employ append-based data structures to store provenance, leading to prohibitive costs, our solutions make efficient usage of bandwidth and yield very low error rates in practice.

1.4 Provenance Usage

We further investigate how to utilize provenance information for security enhancement of various systems. For a file system, we design a provenance based mechanism for the detection of a data exfiltration attack. The provenance graph of a file represents data flow that shows which processes wrote to the file, which files were used to modify the file, etc. and provides us with the knowledge of legitimate access patterns to a file. Thus, the file provenance graphs built during a training period help us later on, during regular operation time, to detect an anomalous access pattern to a file if the access pattern cannot be found in the stored provenance graph of the file.

In the domain of WSN, we focus on the intrusion detection capabilities of the WSNs and propose a provenance based mechanism to address a critical security attack named *packet dropping attack*. We extend the BF based provenance solution to detect the attack and to identify the source of the attack i.e. the malicious node. A packet loss is detected at the BS based on the provenance extracted from the BF. The presence of the attack is determined by comparing the empirical average packet loss

rate with the natural packet loss rate of the data ow path. To isolate the malicious link, we check the consistency among the information encoded by the nodes in the path.

However when dealing with attacks and failures it is not sufficient to detect them, one has to react as soon as possible. Today, the intrusion detection systems (IDSes) are not equipped with response tool that would enable automatic responses and recovery actions. Hence, we move a step ahead and present a systematic design of an incident response and prevention system (IRPS) for WSNs. The system reacts not only after an attack has occurred but also on anomalous events so that the WSN is still functional while the attack progresses. The system is dynamic as it selects the actions in a response policy based on the severity of the event. It is distributed since it does not require any central management for triggering the actions. The fine grained analysis and response set optimization facilities help reducing the processing overhead of the system. The simple yet flexible design of the response policies make the system easily extensible to handle newer attacks.

The remainder of the thesis describes our work in detail. The outline can be summarized as follows. We begin with a state-of-the-art summary of related work in Chapter 2. Chapter 3 presents the proposed provenance model. Chapter 4 describes the detailed architecture of the FiPS file provenance system. We elaborate the secure provenance transmission mechanisms for WSNs in chapter 5, followed by chapter 6 explaining the provenance based attack detection schemes. In chapter 7, we present an extensive incident response and prevention system for WSNs which helps attack recovery while maintaining the WSN services. We conclude the dissertation in Chapter 9.

2. RELATED WORKS

In this chapter, we review the existing works in the areas related to our research.

2.1 Provenance Model

Existing provenance systems can be discussed under three categories depending on their underlying data systems: (i) Workflow-based, (ii) Process-based, and (iii) OS-based systems. In this section, we review a selection of these provenance systems and their models and then discuss their lacking in providing a generic, unified framework.

Workflow based provenance systems [9] [16] [17] collect provenance for data-centric workflows in a service oriented architecture. Chimera [9] defines a Virtual Data Language (VDL) to explicitly represent the workflows. VDL conforms to a logical data schema that represents data as abstract typed *datasets* and describes provenance as relationships among *datasets*, *procedures*, *calls to procedures*, and zero or more physical *invocations* of a specific call. Upon execution, workflows automatically create *invocation* objects for each derivation and collects provenance as annotations about the runtime process information. In myGrid, the information model of the provenance logs contain the services invoked, their parameters, the start and end times, the data products used and derived, and ontology descriptions. Karma collects provenance at 11 activities transpired at 3 different levels, namely {*Workflow, Service, Application*} × {*-Started, -Finished, -Failed*}, *Data -Produced*, and *-Consumed*. However, all workflow based provenance models are tightly coupled to a specific system and capture provenance only at a file granularity. Cohen et al. [18] provide a generic and expressive formal model of provenance for scientific workflows.

Process based provenance systems [19] rely on individual services to record their own provenance in the form of assertions that reflect the relationships between

represented services and data. In PreServ [19], a service invocation generates three types of assertions: *interaction* that records the source and sink of the service; *Actor State* with the list of input and output data of the interaction; and two *Relationship* assertions that associate the *Interaction* assertion with the produced and consumed data in the *Actor State* assertion.

PASS [12] and ES3 [11] are examples of the **OS-based provenance** approach. PASS operates at the level of shared storage system and records information about which programs are executed, their inputs, and any new files created as output. ES3 captures provenance metadata including data object identifier, domain name, input and output files. However, none of these systems provides a formal structure for provenance metadata.

From the above discussion, it is obvious that existing provenance models apply only to a particular application/domain and do not support security. Open Provenance Model (OPM), designed by a community effort to address interoperability, meets the following objectives: (i) to define provenance in a precise technology-agnostic manner; (ii) to allow provenance exchange across systems; (iii) to allow developers to build and share tools that operate on such provenance model; (iv) to allow multiple levels of description to coexist. However, OPM is a high level representation of provenance entities and does not consider security and granularity requirements.

Perhaps the provenance model by Ni et al. [20] is the most comprehensive model. However, this model documents provenance data at a granularity of operation which basically indicates functions. This fact makes it difficult to fit the model in workflow systems - composed of services with many underlying processes or in a large organization where there are multiple computing domains. Since the model does not support user specified granularity policies, the execution of a workflow will always generate a large volume of provenance records. In addition, the model does not help generating separate data dependency and process dependency graphs at a fast speed.

2.2 File Provenance System

The Lineage File System (LinFS) [7] is a file system that automatically tracks provenance at the file system level, focusing on the executable, command line arguments and input files of a process as the source of provenance. Apparently, LinFS cannot capture the complete system level provenance as it ignores the hardware and software environment in which the process runs. LinFS modifies the Linux Kernel to log all process creation and file related system calls in the *printk* buffer. A user level daemon wakes up periodically to read the buffer and write provenance records to an external database. This user level writing delays provenance collection and also threatens the provenance security at LinFS.

The Provenance-aware Storage System (PASS) [12] is a file system oriented approach that automatically collects, stores, manages, and provides search capabilities for provenance. The system collects provenance for every process and maintains provenance information in both memory and disk. PASS intercepts system calls, translating them into in-memory provenance records, which are then attached to key kernel data structures. It also maintains the ancestry graph for in-memory objects and finally maps the in-memory graph to the on-disk provenance that is then passed to the storage layer. The storage layer PASTA, composed of a stackable file system, uses in-kernel Berkeley DB (KBDB) [21] to store and index provenance. However, the PASS architecture has a few limitations. Since PASS operates above the Virtual File System (VFS) layer in Linux, it does not know what events pertain to the PASS file systems. Hence, it ends up collecting data for all files on all volumes and then discarding it. Besides, PASS does not provide security and access control for provenance.

Story Book [22] is a file system that implements provenance file system in user space and treats provenance events as a generic event log. A *provenance source* intercepts user interaction events with application data and sends these events to *application specific extensions* which interpret them and generate provenance inserts

into one of *storage backends*. Queries are handled by *Story Book API*. Story Book relies on external libraries to implement each of its modules. FUSE [23] and MySQL intercept file system and database events. Extensions to Story Book's FUSE file system, such as the .txt and .docx modules, annotate events with application-specific provenance. These provenance records are then inserted into either Stasis [24] or Berkeley DB [25]. Although the user space design of Story Book simplifies its implementation, it incurs significant delay overhead to the file system operations. Also the user level components can tamper with the provenance information.

2.3 Secure Provenance Communication

In this section, we review the state of the art related to our work on secure provenance transmission for WSNs. The related works fall into three classes: secure provenance for networking environment, time based flow watermarking, and applications of in-packet Bloom filter (iBF).

2.3.1 Secure Network Provenance

Pedigree [26] captures provenance for network packets in the form of per packet tags that store a history of all nodes and processes that manipulated the packet. However, the scheme assumes a trusted environment which is not realistic in sensor networks. ExSPAN [27] describes the history and derivations of network state that result from the execution of a distributed protocol. This system also does not address security concerns and is specific to some network use cases. SNP [28] extends network provenance to adversarial environments. Since all of these systems are general purpose network provenance systems, they are not optimized for resource constrained WSNs.

Hasan et al. [15] propose a chain model of provenance and ensure integrity and confidentiality through encryption, checksum and incremental chained signature mechanism. Syalim et al. [29] extend this method by applying digital signatures to a DAG model of provenance. However, these generic solutions are not aware of the sensor net-

work specific assumptions, constraints etc. Since provenance tends to grow very fast, transmission of a large amount of provenance information along with data will incur significant bandwidth overhead, hence low efficiency and scalability. Vijaykumar et al. [30] propose an application specific system for near-real time provenance collection in data streams. Nevertheless, this system traces the source of a stream long after the process has completed. Close to our work, Chong et al. [31] propose a scheme for embedding the provenance of data source within the dataset. While it reflects the importance of issues we addressed, it is not intended as a security mechanism, hence, does not deal with malicious attacks. Besides, practical issues like scalability, data degradation, etc. have not been well addressed.

2.3.2 Time based Flow Watermarking

There exists a lot of work regarding active-timing based watermarking for network flow [32–35]. Our watermarking scheme significantly differs from these approaches in various aspects. (i) All of these schemes embed a single watermark message over the IPDs of a flow. On the contrary, we allow multiple nodes to watermark provenance over the same set of IPDs. (ii) Our decoding process is completely different since it does not retrieve the embedded provenance by inferring bits from each IPD. Instead, we use a unique approach based on a cross-correlation and threshold based mechanism (iii) Several mechanisms (e.g. [32]) watermark a bit by controlling the data throughput for a certain amount of time whereas we prolong the IPD by a small amount of time. Though Wang et al. [34], Kiyavash et al. [36] insert a watermark by delaying the transmission of some packets, the first scheme is subject to detection and recovery attack [37]. As described earlier, our scheme is resilient to this attack. While Kiyavash et al. use spread-spectrum technique to make watermark delays much smaller, their decoding process is non-blind and requires the unwatermarked IPDs to be stored in a database.

2.3.3 Applications of iBF

While BFs are commonly used in networking applications, *iBF*s have only recently gained more attention being utilized in applications such as credential based data path security [38], IP traceback [39], source routing and multicast [40, 41] etc. The basic idea in these works is to encode the link identifiers constituent to the packet routing path into an iBF. However, the encoding of the whole path is performed by the data source, whereas the intermediate routers check their membership in the iBF and forward the packet further based on this decision. This approach is infeasible for sensor networks where the paths may change due to several reasons. Moreover, an intermediate router only checks it own membership which may leave several integrity attacks such as *all-one attack*, *random bit flips* etc., undetected. Our approach resolves these issues by encoding the provenance in a distributed fashion.

2.4 Detection of Packet Dropping Attack

The mechanisms to detect packet dropping attacks in WSNs can be classified into following categories: multipath routing protocols, acknowledgement based mechanisms, protocols using specialized hardware.

The multipath routing protocols [42, 43] first discover multiple paths for data forwarding and then uses these paths to provide redundancy in the data transmission from a source. The data is encoded and divided into multiple shares and then sent to the BS via different routes. However, these methods can not identify the malicious node. They increase the network ow significantly, hence are not suitable for the resource constrained sensor networks. Additionally, these mechanisms could be vulnerable to route discovery attacks that prevent the discovery of non-adversarial paths. Examples of protection mechanisms that require specialized hardware include [44], and [45]. The authors in [44] introduce a scheme called packet leashes that uses either tight time synchronization or location awareness through GPS hardware.

The work in [19] relies on hardware threshold signature implementations to prevent one node from propagating errors or attacks in the whole network.

The acknowledgement based protocols [46, 47] expect the authenticated acknowledgement from the intermediate nodes and the BS within a certain time. This method would render malicious packet dropping detectable at the end points (data source or the BS). However, the method incurs high communication overhead and in some cases has to be augmented with other techniques for diagnosis and isolation of the attackers.

2.5 Incident Response and Prevention System

We discuss the work related to Kinesis in following categories: intrusion detection and/or response system for wireless networks, policy specification, daemon selection.

Intrusion detection and response system: A number of IDSes have been proposed for wireless and mobile ad-hoc networks (MANET) and WSNs. The majority of these IDSes just raise an alarm or take simple response actions without following any systematic approach. In a pioneering work, Zhang et al. propose a distributed and cooperative IDS for MANET [48]. Each mobile node runs a local IDS agent that monitors local activities, detects intrusions, and may trigger responses. Neighboring IDS agents cooperate in global intrusion detection when there is inconclusive evidence. The architecture is similar to Kinesis but is more focused on intrusion detection and does not provide a well-designed response framework. Marti et al. propose a mechanism to improve throughput in MANETs in the presence of compromised nodes [49]. They use a watchdog to identify misbehaving nodes and a trust based routing path rating scheme to help routing protocols avoid these nodes. The CONFIDANT protocol aims at detecting and isolating misbehaving nodes, thus making it unattractive to deny cooperation [50]. Trust relationships and routing decision are based on experienced, observed, or reported routing and forwarding behavior of other nodes. The responses in these systems, however, are limited to rerouting data or isolating the misbehaving node.

Ma et al. [51] propose a self adaptive IDS (SAID) for WSN, where three agents, namely monitor, decision, and defense agents, cooperate to defend from intruders in networks. However, the response system in SAID does not follow a systematic approach and the responses are only limited to revoking or suspending a node. Also, the agents need to update a central knowledge base continuously to update the node reputations and to choose response agents accordingly. Hsieh et al. [52] propose an adaptive security design to secure cluster communication via neighbor node authentication, secure link establishment, and send alarms to the BS upon an intrusion. The mechanism proposed by Younis et al. [53] adapts the security provision to the need of the application and the trust of the nodes in the routing path. These mechanisms heavily depend on cryptographic operations and the counterattack is limited to routing path rotation or raising alarms. Taddeo et al. [54] propose a self-adaptation method of security mechanisms. They always start with the highest security level, which may be unnecessary and costly for sensor nodes.

Asim et al. [55] propose an architecture that organizes the WSN nodes in a virtual grid of cells. Each cell has a manager responsible for anomaly detection and recovery. Their approach is not fully distributed and focuses on network failures and energy related issues, rather than on malicious behaviors or attacks. MALADY is a machine learning-based system that enables nodes to use gathered data to make real-time decisions [56]. However, MALADY aims at the detection and learning process rather than response to attacks. Mamun et al. [57] propose a policy based intrusion detection and response system with a four level hierarchy architecture. Their intrusion response system has a general scope based on customizable policies. However, their only responses are suspend or revocation of the suspect node, and are only applicable to the hierarchical architecture they consider. To the best of our knowledge, Kinesis is the first complete system able to manage automated responses not only to attacks, but also to anomalies with an aim to minimize disruption to WSN services while natural error or an attack progresses.

Policy specification: A number of policy languages have been proposed for the specification of policies for quality-of-service management within a network [58], privacy management for web users [59], access control in database systems [60], etc. However, these languages serve specific purposes and do not consider the context of WSNs or IRPSes, required to optimally express the response policies. Hence, the resource constrained nature of sensor devices makes it challenging to utilize the typical policy languages used in general purpose networks, database systems, and other domains. We propose a simple and lightweight policy language considering the IRPS specific requirements for WSNs.

Daemon selection: *Leader* election is a fundamental and well studied problem in fault-tolerant distributed computing. Garcia-Molina [61] first proposed leader election protocols for distributed systems in order to elect a coordinator node which reorganizes the active nodes after a crash failure and helps them continue the desired tasks. In the context of wired and wireless networks, leader election has a variety of applications such as key distribution, routing coordination, general control, etc. and a considerable number of leader election protocols [62] has been proposed over the years. In a similar context, many clustering algorithms [63] have been proposed for WSNs to group sensor nodes into network clusters and to elect a leader for each cluster for cluster management and data aggregation. However, these leader election protocols require multiple rounds of group communication and often time synchronization among the participants. In contrast, we propose a *daemon selection* mechanism that selects a node for executing response action in a neighborhood via a self-organized competition among the neighbors. Each node in a neighborhood competes independently using a locally managed *action timer*. We **do not need any time synchronization or message exchanges** among the neighbors.

3. A COMPREHENSIVE PROVENANCE MODEL

In this section, we present our proposed provenance model that is well comprehensive to represent the provenance of data objects with various semantics and granularity. We start by discussing the goals that shaped our design and also present an abstract schema of the model. We also compare the capabilities of our provenance model with other major models from various aspects.

3.1 Design Goals

In order to provide a generic provenance structure for all kinds of data objects, the provenance model must meet the following requirements:

Unified Framework: The model must be able to represent metadata provided by the various provenance systems. Although a number of system-call based provenance architectures [11] [12] have been proposed to capture file provenance, there is no well defined model to represent and organize such low level metadata. One important goal for any comprehensive provenance model is to bridge this gap and provide a unified model able to represent provenance for any kind of data at any abstraction layer. To this end, it is crucial to identify a comprehensive set of features that can characterize the existing provenance systems and systemize provenance management.

Provenance Granularity: Provenance may be fine-grained, e.g. provenance of data tuples in a database [64], or coarse-grained, such as for a file in a provenance-aware file system [12] or for collections of files generated by an ensemble experiment run [17]. The usefulness of provenance in a certain domain is highly related to the granularity at which it is recorded [65]. Thus, the provenance model should be flexible enough to encapsulate various *subjects and details of provenance* based on user specifications.

Security: The model must support provenance security. Access control and privacy protection are primary issues in provenance security [66]. The problem of access control for provenance is complicated by the fact that different access control policies, possibly from different sources, may have to be enforced. Moreover, the data originators may specify personal preferences on the disclosure of particular provenance information. To meet these requirements, the provenance model must support the specification of privacy-aware fine grained access control policies and user preferences.

Interoperability: A data object can be modified by and shared among multiple computing systems and so is the provenance. To support provenance exchange, the model must support interoperability among provenance models and integration of provenance across different systems. Thus the model must conform to the Open Provenance Model (OPM) which provides a high level representation of provenance focusing on interoperability.

Provenance Queries and Views: The model should support various types of provenance queries. Historical dependencies as well as subsequent usages of a data object should be tracked easily. If a data is processed in multiple system domains, an administrator might want to see a high level machine, system or domain view of the provenance graph. In addition, to find relevant information from large provenance graphs, one should be able to filter, group or summarize all/portions of provenance graphs and to generate tailored provenance views. Thus, the model should be able to distinguish the provenance generated from different systems and facilitate queries for constructing **specialized views** of provenance graphs.

3.2 The Model

Fig. 4.1(a) shows the proposed provenance model consisting of entities and the interactions among them. To characterize our model, we define the provenance as:

Definition (Provenance). *The provenance of a data object is the documented history of the actors, process, operations, inter-process/operation communications, envi-*

ronment, access control and other user preferences related to the creation and modification of the object. The relationships between provenance entities form a provenance graph (DAG) for the data object.

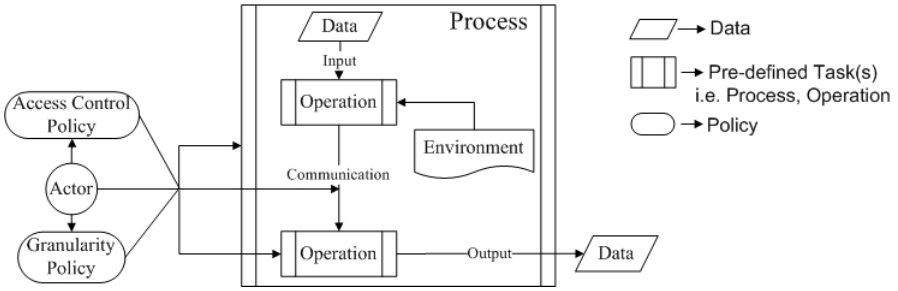


Fig. 3.1.: Proposed Provenance Model

Data creation or manipulation is performed by a sequence of *operation*s initiated by a *process*. A *process*, consisting of a sequence of operations, may be a service/activity in a workflow, a user application, or an OS-level (e.g. UNIX) process. An *operation* executes specific task(s) and causes manipulation to some system or user data. Thus, the operations do not only generate/modify persistent data but also generate intermediate results or modify system configurations. *Communication* represents the interaction (e.g. data flow) between two processes or two operations in a process. Communication between two operations in a process means the completion of an operation following the start of another operation. When the preceding operation results in data, the communication may involve data passing between the operations. The communication may also contain triggers, specific messages, etc. However, in most of the cases there might be no explicit message (i.e. communication record) exchange between two operations. Web service, user application, and UNIX process are examples of *process*es; statements within an executable, function, command line, etc. exemplify the *operation*s; while data flow, copy-paste, inter-process communication in UNIX, etc. represent the *communication* between operations or processes.

An operation may take data as input and output some data. Each data object is associated with a *lineage* record which specifies the immediate data objects that

have been used to generate this data. *Lineage* is particularly helpful for producing the data dependency graph of a data object.

Processes, operations, and communications are operated by *actor*s that can be human users, workflow templates, etc. Where data provenance is used to detect intrusion or system changes, the knowledge of a user role or the workflow template may be helpful. *Environment* refers to the operational state, parameters, system configurations that also affect the execution of an operation and thus output data. This additional provenance information is crucial for understanding the performance of the operation and the nature of the output [66].

Security and privacy of provenance are crucial since data or provenance may contain sensitive or commercially valuable information. The nature of this confidential information is specific to the applications and hence the protection policies and the access control can be handled by the involved actors. To address these requirements, *access control policies* by actors are included in the provenance model. These access control policies specify whether and how other actors may utilize process, operation, communication and lineage records.

Since our provenance model can capture the very details of an operation, it might by preferable to allow users to specify the desired level of provenance details. For example, in a scientific workflow, it may suffice to capture the provenance information in a service/activity whereas in a command line (e.g. *sort*), it may be required to record the OS level operations, system configuration etc. The *granularity policies* allow the users to specify how detailed provenance data they want to be captured and stored.

Table 3.1: Mapping between the entities in OPM and our model

Property	OPM Entity	Entity in our Model
Physical or digital data object	Artifact	Data Object
Action(s) performed on or by artifacts	Process	Process consisting of Operations and Communications
Contextual entity controlling process execution	Agent	Actor, Environment

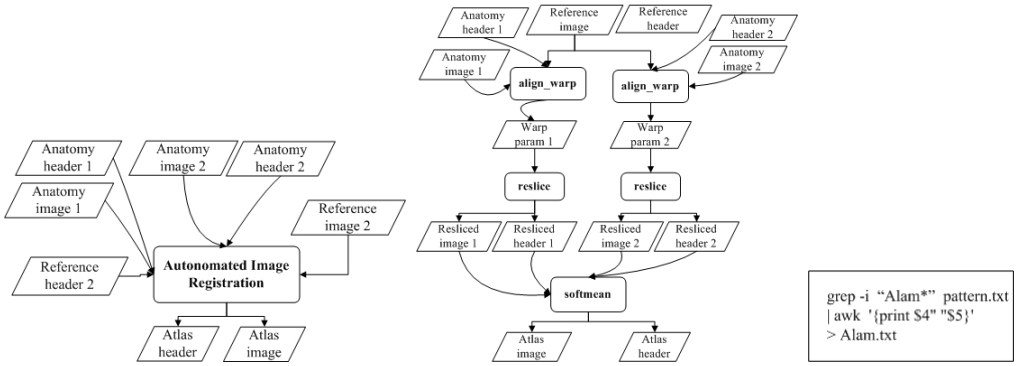
Our model conforms to the OPM representation. Provenance in OPM is described using a directed graph consisting of entities with connecting edges [67]. OPM entities are of three types, namely *artifact*, *process*, *agent*. There are five types of edges which represent the causal dependencies amongst entities. Table 3.1 shows how our provenance model complies with the OPM by listing the OPM entities and their counterparts in our model. Table 3.2 shows a comparison of our provenance model with other major models from various design aspects.

Table 3.2: Comparison between our model and existing provenance models.

	Our Model	Qun Ni Model	Chimera	myGrid	Karma	PReServ	ES3	PASS
Target System	Any	Any	Workflow	Workflow	Workflow	Service	Workflow	File System
Data Granularity	Any data object	Any data object	Abstract dataset	Abstract resources	Data in a workflow	Process	File	File
Inter-operability	Yes	No	No	No	No	No	No	Yes
Security	Yes	Yes	No	No	No	No	No	No
Level of Granularity	Flexible	Rigid	Rigid	Rigid	Rigid	Rigid	Rigid	Rigid
Representation Scheme	Any	Any	VDL	XML /RDF	XML	XML	XML	Berkeley DB
Abstraction	Yes	No	Yes	No	Yes	No	No	No
Query Language	Any	Any	VDL	XML	XQuery	Custom query tool	XML	Custom query tool

3.3 Use Case

To illustrate the utility of our provenance model, we consider some use cases and identify the provenance entities in these contexts. Figure 3.2(a) shows a workflow example from the field of functional MRI research [68], where brain images of some subjects are spatially aligned and then averaged to produce a single image. The workflow contains the automated image registration (AIR) process that operates on a collection of anatomy images and produces an averaged brain image. An *actor* (e.g. an administrator of the experiment system) specifies a *granularity policy* for automated provenance collection to capture provenance at the process granularity. In this context, the provenance for 'Atlas image' and 'Atlas header' contains the



(a) Workflow for 'Automated Image Registration' (AIR) process

(b) Break down of AIR process into operations and data flows between them

(c) A shell script representing a user program and corresponding OS-level process

Fig. 3.2.: Usecases for Provenance Modeling

AIR *process* with anatomy and references images & headers as the input *lineage* data. Since no details about the AIR process are captured, we assume the process consists of a single *operation* named as AIR. Figure 3.2(b) presents the breakdown of the AIR process into operations and interactions between them. If a user defined policy requires to capture operation level provenance, the provenance graph for 'Atlas image' will contain the AIR *process* with operation hierarchy **align_warp** -> **reslice** -> **softmean**. The data flow between operations represents their communication; for example the transfer of **Warp param 1** indicates the communication between **align_warp** and **reslice** operations. However, the data dependency graph of 'Atlas image' contains the input images as well as all the intermediate results.

Finally, we consider a UNIX shell script - 'pattern.sh', shown in Figure 3.2(c) to show the applicability of our model to provenance aware file/storage systems, operating systems, etc. The script uses the 'grep' command to extract all the patterns starting with 'Alam' from the 'data.txt' file and sends the output to the 'awk' command through a pipe. The 'awk' command then extracts particular information from the input data and writes the information in the output file 'Alam.txt'. The execution of the script (namely 'pattern' *process*) may be assigned a unique process ID

by the system. The process consists of two operations, 'grep' and 'awk'. Thus, the provenance of 'Alam.txt' contains the operation dependency **grep -> awk** and the data dependency on 'data.txt' and the intermediate pipe (identified by an ID).

3.4 Provenance Records

Data provenance is stored as a set of provenance records in a *provenance repository* [20]. Provenance storage, manipulation and query can be implemented using data management systems characterized by different data models such as the relation model, XML, and RDF. Since our provenance model is generic, we do not specify implementation details here. We represent our model as the relationships among the following provenance records (see Fig 3.3): (i) *Process* (ii) *Operation* (iii) *Communication* (iv) *Actor* (v) *Environment* (vi) *Lineage* (vii) *Access Control Policy* (viii) *Granularity Policy*.

Each record consists of several attributes some of which are optional based on the provenance capturing granularity. Each data object and provenance record is uniquely identified by an ID attribute. Since provenance information may be exchanged across different systems, we use *domain* to specify the scope of the provenance records i.e. the system where the executions and data manipulations occur. The *domain* value may include a particular application, a workflow, a machine, a system domain, or any combination of these. This attribute is extremely useful when customizing the provenance graph to efficiently generate an **abstract domain view**. Some records contain a timestamp attribute for supporting time-sensitive provenance queries and access control policies.

We describe a process with the base class *process* and differentiate between the high level and the system process by creating two inherited classes of *process*. Each *process* is executed by an *actor* in a certain computational *environment* and may generate *output* data. If the *process* is part of a scientific workflow, web service, etc., it is distinguished by the subclass *Application Process* which also contains the

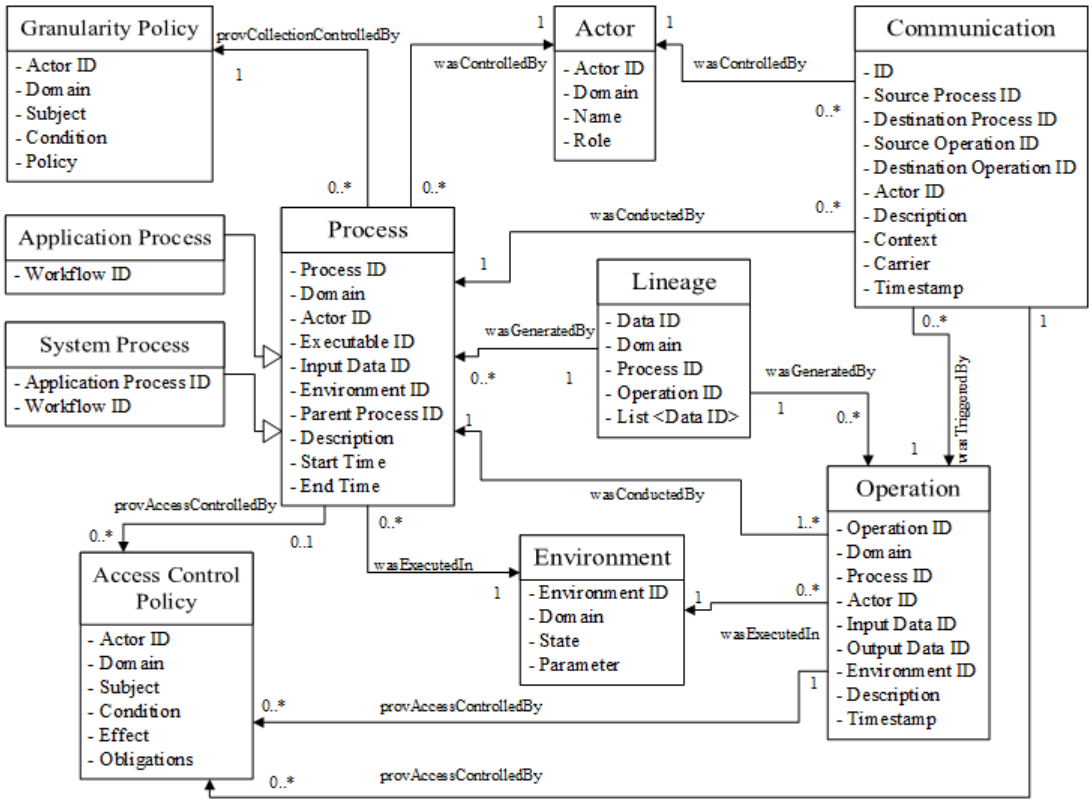


Fig. 3.3.: Class Diagram of the Provenance Model

workflow ID. The *System Process* class describes the OS level processes and possesses workflow ID as well as the host application process ID.

Operation record attributes include ID, process ID, actor ID, environment ID, description, input and output ID. Depending on the applications, the description attribute may contain a statement or a block of statements, a function defined by pseudo-code or source code, but it can also be only a function name. The output of an operation may be not only the persistent data but also intermediate results for which the user might not be interested to store the provenance.

A *communication* record is the provenance of a real or virtual message implementing the interaction between two operations in a process or two different processes. Specific details of its *description* attribute depend on applications. The carrier attribute includes the message transferring channel, e.g. email, which may be sensitive and useful in some cases, e.g. digital forensics.

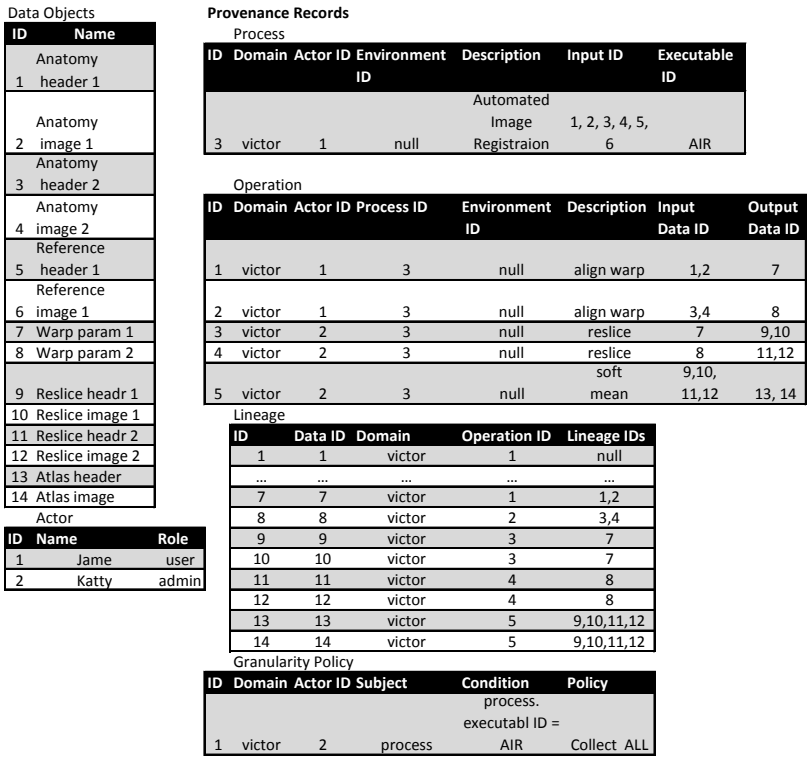
Lineage record attributes include lineage ID, corresponding data ID, the process & operation ID that produced this data operating on a list of data objects (described by List <Lineage ID>). The attributes of *Actor* record include actor ID, name, and role. Actors usually have names and roles. A role is a job function of the actor. Someone may argue why not use the actor information from human resource databases. A human being may have different roles during the career time. Thus he/she may have different versions of actor records with different roles for different process/operation/preference records. This is the reason why we cannot rely only on the information from the actor records stored in human resource databases. Such a record usually only stores the latest actor information, but an actor record in a provenance store needs to record complete historical actor information.

The content of the *Environment* record heavily depends on the application domain. Usually each operation, sometimes a process, has at most one environment record. We choose to separate this record from the process/operation record because of two reasons. First, the schema and size of environment records vary with respect to different process/operation records. Some process records do not have an environment record; however others may have a complex environment. Second, it is possible that two different process records may share the same environment record. Environment records do not need timestamps because their timestamps are determined from the parent process/operation records.

Access Control Policy record attributes include policy ID, actor ID, subject, condition, effect, obligations. This record is used to specify the access preferences of the actor. Sometimes it is also useful to record the preferences expressed by the subject of the operation/communication, for example a patient in the case of health care applications. The actor ID logs the author of the record. The subject attribute is used to specify the record(s) at which the access control aims. The subject of an access control policy record only refers to a process, operation or a communication record.

Granularity Policy record comprises of policy ID, actor ID, subject, condition and policy attributes. This record allows the users to specify the level of details desired for

provenance metadata and is applicable when capturing provenance. For example, an actor may define policies to capture provenance only at the process level or to exclude the lineage information for a particular application. The subject attribute states the targeted record at which the granularity policy applies based on the condition value.



Data Objects

ID	Name
1	Anatomy header 1
2	Anatomy image 1
3	Anatomy header 2
4	Anatomy image 2
5	Reference header 1
6	Reference image 1
7	Warp param 1
8	Warp param 2
9	Reslice headr 1
10	Reslice image 1
11	Reslice headr 2
12	Reslice image 2
13	Atlas header
14	Atlas image

Actor

ID	Name	Role
1	Jame	user
2	Katty	admin

Provenance Records

Process

ID	Domain	Actor ID	Environment ID	Description	Input ID	Executable ID
3	victor	1	null	Automated Image Registraion	1, 2, 3, 4, 5, 6	AIR

Operation

ID	Domain	Actor ID	Process ID	Environment ID	Description	Input Data ID	Output Data ID
1	victor	1	3	null	align warp	1,2	7
2	victor	1	3	null	align warp	3,4	8
3	victor	2	3	null	reslice	7	9,10
4	victor	2	3	null	reslice	8	11,12
5	victor	2	3	null	soft mean	9,10, 11,12	13, 14

Lineage

ID	Data ID	Domain	Operation ID	Lineage IDs
1	1	victor	1	null
...	...	...	...	...
7	7	victor	1	1,2
8	8	victor	2	3,4
9	9	victor	3	7
10	10	victor	3	7
11	11	victor	4	8
12	12	victor	4	8
13	13	victor	5	9,10,11,12
14	14	victor	5	9,10,11,12

Granularity Policy

ID	Domain	Actor ID	Subject	Condition	Policy
1	victor	2	process	process. executabl ID = AIR	Collect ALL

Fig. 3.4.: Provenance Records for workflow in Fig 3.2(b).

It is important to mention that a provenance graph is maintained as a DAG. In case of write operation to an existing file, a cycle may be introduced. To preserve the DAG property, we allow the versioning of data objects where manipulating an existing data object takes the existing object as an input and outputs a latest version of the data object. Another relevant issue is to allow actors to rationally change their access control/granularity policies on their own records. Since a provenance store is immutable, in order to support such selective updates, we use policy versioning

and time stamping. Given a query, if multiple access control policies from the same actor apply, the most recent one will take precedence. Such an approach preserves the previous policies and the associated data provenance records. To illustrate the application of provenance records to the use cases from section 3.3, we consider a RDBMS implementation of the provenance storage. Figure 3.4 shows the data objects and related provenance records generated from the workflows in 3.2(b). For simplicity, we do not show some attributes e.g. timestamp.

3.5 Supported Queries

Having defined a comprehensive provenance model, we can use any standard query language to query the entities in the model. The wide range of queries supported by our model can facilitate the users in many different respects, such as scientific reproducibility, script generation, anomaly detection, etc. We discuss below the various queries supported by our provenance model:

Fundamental Queries on Entity Attributes: These queries retrieve information about the fundamental entities of the provenance model. Examples of such queries are: find processes by namespace, find all the operations belonging to a process, generate the sequence of processes/operations in a workflow. These queries can help in detecting anomalies by comparing the expected output of an operation in the recorded environment with the actual result. Users that have executed anomalous operations can also be identified by finding out the actors that invoked the operations.

Queries on Invocations: These queries retrieve the set of commands involved in the manipulation of a selected data object. Users can set various filters while retrieving the provenance, such as remove commands that occurred before or after a given point of time. We also support lookup queries that allow users to search for data objects based on arguments to the processes that modified them. These queries facilitate users in using provenance for reproducing a data object, detecting system

changes or intrusions, finding out the system configuration during process invocation, understanding system dependencies, etc.

Queries on Lineage: The historical dependencies of a data object can be determined by traversing the provenance graph backward whereas data usage can be traced by forward traversal of the graph. A simple query is of the form: *find the ancestor data objects to data d.* More complex queries may refer to patterns within the derivation graph. The basic approach is to match specific patterns of processes consisting of operations and communications and enabling the composition of *flowpattern* objects. The flowpattern graphs can match either a fixed or varying number of nodes of their corresponding types in any workflow defined in the database. Possible queries include:*find the data objects that are result of a specific flowpattern*, and *find all operations in a workflow whose inputs have been processed by a specific flow pattern.*

Provenance View: Since provenance grows fast, it might be convenient (often required) to compress or summarize the provenance graph for efficient querying and navigation. For example, instead of keeping track of how different processes and people modified a document five years ago, we can replace the part of the provenance with the end result of the modification. Since our provenance model has a modularized structure, we support queries to generate any abstraction of a provenance graph. The *domain* attribute in the provenance records greatly helps in writing a quick and effective abstraction function for an intended purpose.

4. A FILE PROVENANCE SYSTEM

In this chapter, we discuss the design objectives for an efficient file provenance system and then present the modular design of a low-overhead file provenance system, called FiPS. The system supports the automatic collection and management of file provenance metadata, characterized by our provenance model proposed in chapter 3.

4.1 Design Goals

Based on the features of the existing file system provenance solutions and their limitations, we outline the following design goals required to build a robust file provenance system:

Portability: The file provenance system should capture provenance for any file system, without modifying the OS or the provenanced file system. FiPS is designed as a stackable filesystem and thus can be layered on top of any conventional file system. In addition, FiPS is to be implemented as a kernel module which requires no kernel modification in order to collect provenance.

Efficiency: It is essential that provenance capture and management do not add too much overhead to the file system operations with respect to space and time. The provenance system should provide fast, high-throughput provenance operations in order to avoid impacting operating system and application performance. The system must record enough provenance metadata to serve the desired purpose but not any unintended information. Hence, it should distinguish between data objects that are required to be provenanced and data objects that are not. On the other hand, capturing provenance information by intercepting system calls often misses information about how a system call activity is translated into multiple actions in the lower layers of the OS. Also NFS servers cannot work with system call level logging

since they operate directly in kernel, not through system calls. Finally, it is more natural to manage file system provenance in terms of file system instead of system calls. We design FiPS as a thin layer between the Virtual File System (VFS) and any other file system which results in space and time efficiency.

Security: The system must capture and store provenance in a way so that the information is kept secure against attacks and subversion. Besides, the provenance information may require access control to be protected from unauthorized user access. Our in-kernel system design provides stronger security. Moreover, the provenance processor can be implemented in a way to apply appropriate security mechanisms (e.g. encryption, signature) while sending provenance to persistent storage.

Queries on Provenance: Collecting data provenance is not useful unless the provenance can be accessed and utilized easily. Hence, the file provenance system must provide support for a structured storage of provenance which in turn will facilitate provenance queries. The management system should also respond quickly to the relationship queries leading to the generation of ancestry or descendancy graphs.

4.2 FiPS - The Proposed Provenance Framework

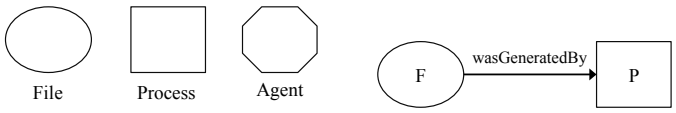
In this section, we present the provenance framework we propose. Our provenance framework consists of the following components: (i) Provenance Collector, (ii) Provenance Log, (iii) Provenance Processor, and (iv) Provenance Storage. Below, we provide more details on the various provenance components.

4.2.1 Provenance Model

FiPS is designed to collect and store provenance for the data objects at a file granularity. We define the *provenance of a data object (file) as the documented history of the actors, process, operations, inter-process/operation communications, input/output data, and OS environment related to the creation and modification of the object.* The input data may contain references to data objects which are also provenanced. Thus,

the complete provenance of a data object is the transitive closure over all such references which form a directed acyclic graph (DAG), referred to as the *provenance graph*.

To represent provenance graph, we use a subset of entities from our provenance model, which includes controlling *Actor*, executing *Process*, and data *File* as node types. The edges between nodes are characterized by the relationships that relate which process *wasControlledBy* which actor, which file *wasGeneratedBy* which process, which process *used* which file, which process *wasTriggeredBy* which other process, and which file *wasDerivedFrom* which other file. Figure 4.1(a) illustrates the graph representation of various types of nodes, whereas the edges are annotated with the relationship type. Figure 5.1 shows an example provenance graph where process P writes to a file F.



(a) Graph representation of various types of nodes in a provenance graph

(b) Example of a provenance graph

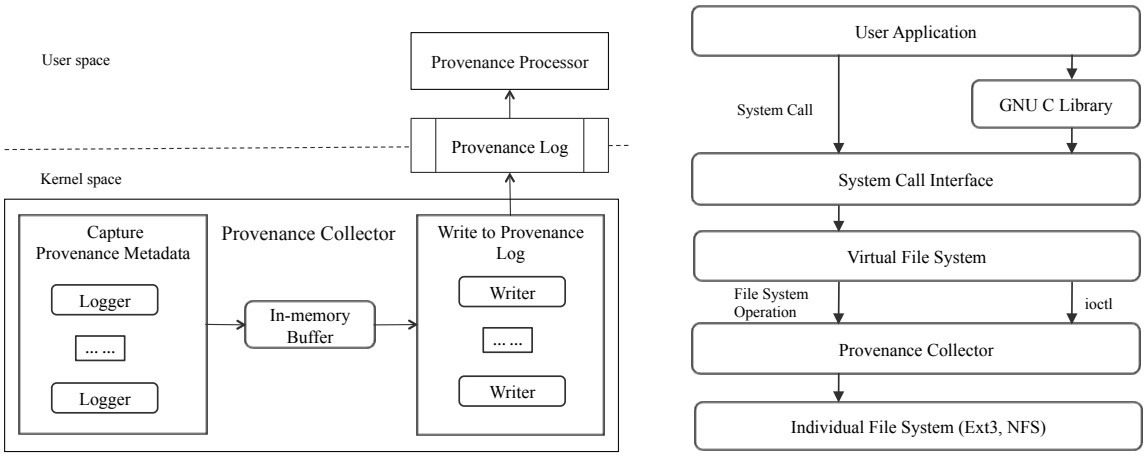
Fig. 4.1.: Provenance graph representation and example

More formally, a *provenance graph* for a *file* can be represented as $g = (V, E)$, where V is a set of vertices and E is a set of directed edges connecting the vertices, and we use $g.root$ to represent the root node of graph g (i.e., the file). A vertex can either represent a process, user agent or file object. An edge between two vertices v_1 and v_2 is introduced when a dependency (of any of the above mentioned types) is created from the entity that corresponds to v_1 to the entity that corresponds to v_2.

4.2.2 Provenance Collector

We design the provenance collector as a stackable file system [69] that can work on top of any underlying file system. Figure 4.2(b) shows how the collector is placed

between the *Virtual File System* (VFS) and any other file system. In a traditional file system, the system calls related to file operations invoke VFS calls which in turn invoke underlying file system procedures. When integrated, the provenance collector intercepts the VFS calls, extract arguments and other necessary information from kernel data structures, constructs a log entry and writes it to an in-memory buffer. At the end, the collector sends the in-memory log entries to the userspace as a log file.



(a) Architecture of the provenance management system

(b) Provenance Collector as a stackable file system

Fig. 4.2.: Proposed Provenance Framework

Figure 4.2(a) shows the detailed architecture of the provenance collector. The key components are: the provenance *logger* which captures the provenance metadata and translates them into in-memory log entries, and the provenance *writer* that stores in-memory entries into an userspace log file. Below, we briefly discuss the components of the provenance collection infrastructure:

Logger: The role of a logger is to observe provenance generating events, to capture relevant metadata from kernel for each event, and then to write one or more entries to an intermediate storage. Our design supports multiple *logger* threads where the intercepted VFS calls pertaining to an application or process will be handled by

one *logger*. Such a design will increase the speed of the provenance tracking for simultaneous processes and make it easier to deal with the granularity policies.

All files and processes in our system are considered provenanced kernel objects. To generate provenance records, the logger intercepts the following process and file related system calls: *fork(), clone(), exit(), read(), write(), rename(), truncate, symlink(), readlink(), unlink()*. By intercepting the system calls, the logger can capture a wide variety of events:

- Reads and writes to file descriptors, including regular files, device files, and pipes.
- File operations: renaming, changing permissions, etc.
- Inter-process communication, such as shared memory, message queues, and UNIX domain sockets.
- Network communication between provenanced hosts.
- Program execution with full arguments and environment.

On each event, a logger thread collects provenance metadata, creates a log entry and stores the entry in an in-memory *First In, First Out* (FIFO) buffer. The typical information included in a log entry is shown in Table 4.1.

Table 4.1: Typical Provenance Metadata

Field	Explanation
provid	Provenance identifier
name	Name of the file/process executable
pid	OS assigned process identifier
ppid	Parent process identifier
uid	OS assigned user id
argv	The command line
env	The process environment, i.e., OS version, etc.
input	Name or inode number of the input files

For the purposes of recording provenance, each log entry is assigned an identifier, referred to as *provid*. All the log entries related to the activity of a process in a single

session are assigned the same and an unique *provid*. The *provid* is generated as a small unique integer.

Writer: A writer dequeues a log record from the in-memory FIFO and writes out the record in a userspace log file, referred to as *Provenance Log*. The system also activates multiple writers, like loggers, in order to fasten the performance.

4.2.3 Provenance Log

The provenance log is a medium of communicating provenance from kernel to user space. Since provenance collection generates a large volume of data [70], we need an efficient and reliable mechanism for making large quantities of kernel data available to userspace. Existing systems have accomplished this by using an expanded *printk* buffer [7], writing directly to on-disk log files [71], using FUSE [23] or *relayfs* [72], a specially designed mechanism to efficiently transfer data from kernel to user space. None of these methods but *relayfs* is appropriate for our system design. Hence, we follow [72] to use relayfs for our purpose.

A relay [73] is a kernel ring buffer made up of a set of preallocated sub-buffers. Once the relay has been initialized, the collector writes provenance data to it using the relay_write function. This data then appears in userspace as a regular file, which can be read by the provenance processor. Since the relay is backed by a buffer, it retains provenance data even when the processor is not running, as is the case when the processor crashes and must be restarted. Since the number and size of the sub-buffers in the relay are specified when it is created, the relay has a fixed size. Although the collector can act accordingly if it is about to overwrite provenance which has not yet been processed by the processor, it is better to avoid this situation altogether. To this end, we allow the relays size parameters to be specified when the provenance system is started.

4.2.4 Provenance Processor

The responsibility of the provenance processor is to interpret, process, and store the provenance data after it is collected. In our design, we decouple the provenance processor from the collection process in order to allow the system administrator to implement the processor to support the need of the system. For example, a system may want to process the provenance information in a specific way, aggregate/truncate provenance information before sending to persistent storage or even may use different storage mechanism.

Such a modular design also keeps complex algorithms out of the collector. Existing systems have devoted considerable effort to dealing with problems in provenance representation, such as compact storage or graph cycles [74]. Our design simply allows the processor to address these problems in whatever way is most appropriate.

4.2.5 Provenance Storage

To facilitate collecting large volumes of provenance metadata and primarily initiating graph queries, we use graph database as the provenance storage. A graph database stores data in a graph, the most generic of data structures, capable of elegantly representing any kind of data in a highly accessible way. The fundamental units that form a graph are *nodes* and *relationships*. Both nodes and relationships can contain properties, a record that has named values. *Relationships* organize *Nodes* into arbitrary structures, allowing a *Graph* to resemble a List, a Tree, a Map, or a compound Entity any of which can be combined into yet more complex, richly inter-connected structures. Apart from properties and relationships, nodes can also be labeled with zero or more labels. *Labels* are a means of grouping the nodes in the graph. They can be used to restrict queries to subsets of the graph, as well as enabling optional model constraints and indexing rules.

After reading and processing a log entry from the provenance log, the provenance processor creates appropriate provenance entities and relationships. The module then

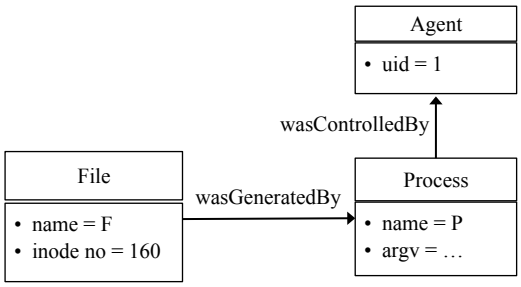


Fig. 4.3.: An example provenance graph stored in provenance database

stores these entities and relationships in a graph database, referred to as provenance database. An example provenance graph, for the event when process P writes to a file F, stored in the provenance database is shown in Fig. 4.3.

4.3 Prototype Implementation

We implement our system in Ubuntu 12.04 LTS. To layer our provenance collector on top of any conventional file system, we implement our functionalities on the stackable wrapper file system *Wrapfs* [69]. As we discussed in Sec., we use kernel ring buffer, relay, to present provenance records as userspace Provenance Log file. We implement provenance processor as a Java application which processes *Provenance Log* file and also interacts with *Provenance Database*. The graph database Neo4j provides us with the facility to store provenance information as a graph and to perform graph queries. It guarantees us efficiency while searching for a given access graph in the provenance database, and thus ensures the scalability of the system.

5. LIGHTWEIGHT SECURE PROVENANCE SCHEMES FOR WIRELESS SENSOR NETWORKS

In this chapter, we embark on an exploration into provenance management for streaming data focusing on WSNs. Specifically, we examine a secure mechanism to form and transmit the provenance graph of a data packet in a distributed setting where a source node generates the data and the intermediate node(s) towards the BS may process the in-transit data. A possible approach to the problem could be based on traditional security solutions like encryption, digital signature, and message authentication code (MAC). In a digital signature (or MAC) based mechanism, each party involved in the data processing would append its information to data and sign it (or compute and attach the MAC) to ensure authenticity. In addition, encryption and an incremental chained signature based approach for secure document provenance [75] could be adapted for use in sensor networks. However, such approaches are not applicable in resource constrained WSNs, because provenance information tends to grow very fast, often becoming several magnitudes in size larger than the original data [75]. Such a characteristic thus would force the transmission of a vast amount of provenance information along with data. Encryption/signature/MAC based mechanisms cannot help in reducing such size even after compaction. Hence, traditional security means incur significant bandwidth overhead and impact efficiency and scalability. Addressing the above challenges, we propose two techniques - (i) a watermarking scheme for per-flow provenance encoding and decoding over the inter-packet delays (IPD), (ii) a per-packet provenance scheme using iBF. Different WSN applications may prefer one solution over the other depending on the network data rates.

5.1 Background and System Model

In this section, we introduce the network, data and provenance models used. We also present the threat model and security requirements. Finally, we provide brief primers on digital watermarking, spread-spectrum watermarking and fundamental fundamental properties and operations of Bloom filters.

Network Model. We consider a multihop wireless sensor network, consisting of a number of sensor nodes and a base station (BS) that collects data from the network. The network is modeled as a graph $G(N, L)$, where $N = \{n_i|, 1 \leq i \leq |N|\}$ is the set of nodes, and L is the set of links, containing an element $l_{i,j}$ for each pair of nodes n_i and n_j that are communicating directly with each other. Sensor nodes are stationary after deployment, but routing paths may change over time, e.g., due to node failure. Each node reports its neighboring (i.e. one hop) node information to the BS after deployment. The BS assigns each node a unique identifier $nodeID$, a symmetric cryptographic key K_i, and a unique pseudo noise (PN) sequence, denoted as $\mathbf{pn}_i = pn_i[1]\, pn_i[2] \ldots pn_i[L_p]$, where L_p, an integer greater than 0, is the length of the PN sequence. In addition, a set of hash functions $H = \{h_1, h_2, ..., h_k\}$ are broadcast to the nodes.

Data Model. We assume a multiple-round process of data collection. Each sensor node generates data periodically, and individual values are routed and aggregated towards the BS using any existing hierarchical (i.e., tree-based) dissemination scheme, e.g., [76]. A data path of p hops is represented as $< n_l, n_1, n_2, ..., n_p >$, where n_l is a leaf node representing the data source, and node n_i is i hops away from n_l. Each non-leaf node in the path aggregates the received data and provenance with its own locally-generated data and provenance.

Each data packet contains (i) a unique packet sequence number (ii) a data value, (iii) timestamp, and (iv) provenance. The sequence number is attached to the packet by the data source, and all nodes use the same sequence number for a given round [77].

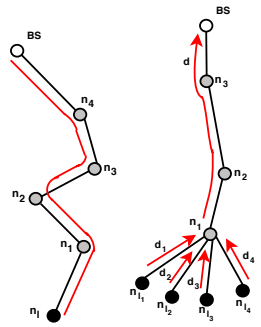


Fig. 5.1.: Provenance graph for a sensor network.

Depending on the solution approach considered, the timestamp/sequence number integrity is ensured through message authentication codes (MAC).

Provenance Model. We consider *node-level* provenance, which encodes the nodes that are involved at each step of data processing. This representation has been used in previous research for trust management [78] and for detecting selective forwarding attacks [79].

Given a data packet d, its provenance is modeled as a directed acyclic graph $G(V, E)$ where each vertex $v \in V$ is attributed to a specific node $HOST(v) = n$ and represents the provenance record (i.e. nodeID) for that node. Each vertex in the provenance graph is uniquely identified by a vertex ID (VID) which is generated by the host node using cryptographic hash functions. The edge set E consists of directed edges that connect sensor nodes.

Definition 5.1.1 (Provenance) *Given a data packet d, the provenance p_d is a directed acyclic graph G(V,E) satisfying the following properties: (1) p_d is a subgraph of the sensor network $G(N, L)$; (2) for $v_i, v_j \in V$, v_i is a child of v_j if and only if HOST $(v_i) = n_i$ participated in the distributed calculation of d and/or forwarded the data to HOST $(v_j) = n_j$; (3) for a set $U = \{v_i\} \subset V$ and $v_j \in V$, U is a set of children of v_j if and only if HOST (v_j) collects processed/forwarded data from each HOST$(v_i \in U)$ to generate the aggregated result.*

Figure 5.1 shows two provenance examples in sensor networks. In Figure 5.1(a), the leaf node n_l generates a data packet d and each intermediate node aggregates its own sensory data with d then forwards it towards the BS. Hence, the provenance corresponding to d is $< v_l, v_1, v_2, v_3 >$, which can be represented as a simple path. In Figure 5.1(b), the internal node n_1 generates the data d by aggregating data d_1 , ..., d_4 from n_{l_1} , ..., n_{l_4} and then passes d towards the BS. Here, n_1 is an aggregator and the aggregated provenance $< \{v_{l_1}, v_{l_2}, v_{l_3}, v_{l_4}\}, v_1, v_2, v_3 >$ is represented as a tree.

Threat Model and Security Objectives. We assume that the BS is trusted, but any other arbitrary node may be malicious. An adversary can eavesdrop and perform traffic analysis anywhere on the path. In addition, the adversary is able to deploy a few malicious nodes, as well as compromise a few legitimate nodes by capturing them and physically overwriting their memory. These malicious nodes might collude to attack the system. If an adversary compromises a node, it can extract all key materials, data, and codes stored on that node. The adversary may drop, inject or alter packets on the links that are under its control. We do not consider denial of service attacks such as the complete removal of provenance, since a data packet with no provenance records will make the data highly suspicious [15] and hence generate an alarm at the BS. Instead, the primary concern is that an attacker attempts to misrepresent the data provenance. Our objective is to achieve the following security properties:

- *Confidentiality*: An adversary cannot gain any knowledge about data provenance by analyzing the IPDs or the contents of a packet. Only authorized parties (e.g., the BS) can process and check the integrity of provenance.

- *Integrity*: An adversary, acting alone or colluding with others, cannot add or remove non-colluding nodes from the provenance of benign data (i.e. data generated by benign nodes) without being detected.

- *Freshness*: An adversary cannot replay captured data and provenance without being detected by the BS.

However, an adversary may increase network jitter in a way that the recorded IPD at the BS is much larger than the desired value. Such an attack is intended to destroy the embedded provenance. As we discuss later, our scheme can recover provenance if the IPD is altered within a certain limit. In any case, the BS can detect such malicious activity and may utilize some auxiliary mechanism to identify the attacker and take necessary actions. Moreover, the attacker can inject or drop data packets which also alters the IPDs and interfere with the embedded provenance. We successfully recover provenance against the *insertion attack* but survive the *deletion attack* to a certain extent.

Digital Watermarking. The key idea of digital watermarking is to hide a secret information (watermark) related to a digital content within the content itself thereby ensuring the movement of the watermark along with the content. Thus, digital watermarking involves the selection of a watermark carrier domain and the design of two complementary processes:

(1) An *embedding process* E that utilizes the watermark carrier A, the watermark message w, and, possibly, a key K to generate the watermarked data AW as $E(A, w, K) = AW$

(2) A *detector process* that determines the existence of a watermark within the received signal (with the key, if applicable) and extracts it.

Spread Spectrum Watermarking. Spread spectrum is a transmission technique by which a narrowband data signal is *spread* over a much larger bandwidth so that the signal energy present in any single frequency is undetectable [14]. In our context, the IPD is the *communication channel* and the *provenance* is the *signal* transmitted through it. Provenance is spread over many IPDs such that the information present in one IPD (i.e. container of information) is small. Consequently, an attacker needs to add high amplitude noise to all of the containers in order to destroy the provenance. Thus, the use of the spread spectrum technique for watermarking provides strong security against different attacks. We have adopted the *direct sequence spread spectrum* (DSSS) technique which is widely used for enabling multiple

users to transmit simultaneously on the same frequency range by utilizing distinct pseudo-noise (PN) sequences [14]. The intended receiver can extract the desired user's signal by regarding the other signals as noise-like interferences. The components of a DSSS system are:

Input:

- The original data signal *d(t)*, as a series of +1, -1.
- A PN sequence *px(t)*, encoded like the data signal. N_c is the number of bits per symbol and is called PN length.

Spreading: The transmitter multiplies the data with the PN code to produce spreaded signal as *s(t) = d(t) px(t)*

Despreading: The received signal *r(t)* is a combination of the transmitted signal and noise in the communication channel. Thus *r(t) = s(t) + n(t)*, where *n(t)* is a white Gaussian noise. To retrieve the original signal, the correlation between *r(t)* and the PN sequence $pr(t)$ at the receiver is computed as $R(\tau) = \frac{1}{N_c} \sum_{t=T}^{T+N_c} r(t)\, pr(t+\tau)$. If $px(t) = pr(t)$ and $\tau = 0$ i.e. $px(t)$ is synchronized with $pr(t)$, then the original signal can be retrieved. Otherwise, the data signal cannot be recovered. So, a receiver without having the PN sequence of the transmitter cannot reproduce the originally transmitted data. This fact is the basis for allowing multiple transmitters to share a channel. In this paper, we refer to $R(0)$ as *cross-correlation*.

In case of multiuser communication in DSSS, spreaded signals produced by multiple users are added and transmitted over the channel. To retrieve the signal for j-th user, the cross-correlation between $r(t)$ and $px_j(t)$ is computed. Multi-user communication introduces noise to the signal of interest and interfere with the desired signal in proportion to the number of users. The condition for error free communication in DSSS can be derived from Shannon's channel-capacity theorem

$$C = B \log_2 (1 + \frac{S}{N}) \tag{5.1}$$

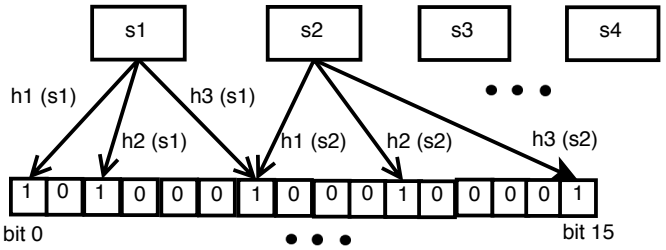


Fig. 5.2.: A Bloom filter with $n = 4$, $m = 16$ and $k = 3$.

where C is the amount of information allowed by the communication channel, B is the channel bandwidth, and S/N is the signal-to-noise ratio. As S/N is usually $\ll 1$ for spread-spectrum applications, the expression becomes $\frac{C}{B} \approx \frac{S}{N}$. Thus to propagate error-free information for a given noise-to-signal ratio in the channel, the bandwidth should be increased to an appropriate level.

Bloom Filters (BF). A Bloom filter is a space-efficient data structure for probabilistic representation of a set of items $S = \{s_1, s_2, ..., s_n\}$ using an array of m bits with k independent hash functions $h_1, h_2, ..., h_k$. The output of each hash function h_i maps an item s uniformly to the range [0, m-1] and is interpreted as an index pointing to a bit in a m-bit array. Hence, the BF can be represented as $\{b_0, \ldots, b_{m-1}\}$. Initially each of the m bits is set to 0.

To insert an element $s \in S$ into a BF, s is hashed with all the k hash functions producing the values $h_i(s)(1 \leq i \leq k)$. The bits corresponding to these values are then set to 1 in the bit array. Figure 5.2 illustrates an example of BF insertion. To query the membership of an item s' within S, the bits at indices $h_i(s')(1 \leq i \leq k)$ are checked. If any of them is 0, then certainly $s' \notin S$. Otherwise, if all of the bits are set to 1, $s' \in S$ with high probability. There exists a possibility of error which arises due to *hashing collision* that makes the elements in S collectively causing indices $h_i(s')$ being set to 1 even if $s' \notin S$. This is called a *false positive*. Note that, there is no *false negative* in the BF membership verification.

The cumulative nature of BF construction inherently supports the *aggregation* of BFs of a same kind, by performing bitwise-OR between the bitmaps.

5.2 Watermarking based Provenance Scheme

Fig. 5.3 shows an overview of the distributed approach we propose to watermark provenance over the delay between consecutive data packets. For a data packet, *provenance encoding* refers to generating the vertices in the provenance graph and watermarking them over IPDs. Each vertex originates at a node in the data path and represents the provenance record of the host node. A vertex is uniquely identified by the vertex ID (VID). The VID is represented by the PN sequence (of L_p bits) of a node and requires a number of $(L_p + 1)$ packets for encoding. Due to the adoption of DSSS based watermarking, all nodes in the provenance use the same medium for transmitting their PN sequences. Hence, only L_p bits of digital information are required for watermarking the provenance. Since we utilize the IPDs, L_p IPDs (in other words, a sequence of L_p+1 packets) are required for embedding and transmitting the provenance of a data packet. We assume that, at least for such number of packets, the provenance (i.e. data flow path) of the packets generated by a source node would be the same. Below we discuss the provenance encoding mechanism by the sensor nodes and decoding at the BS:

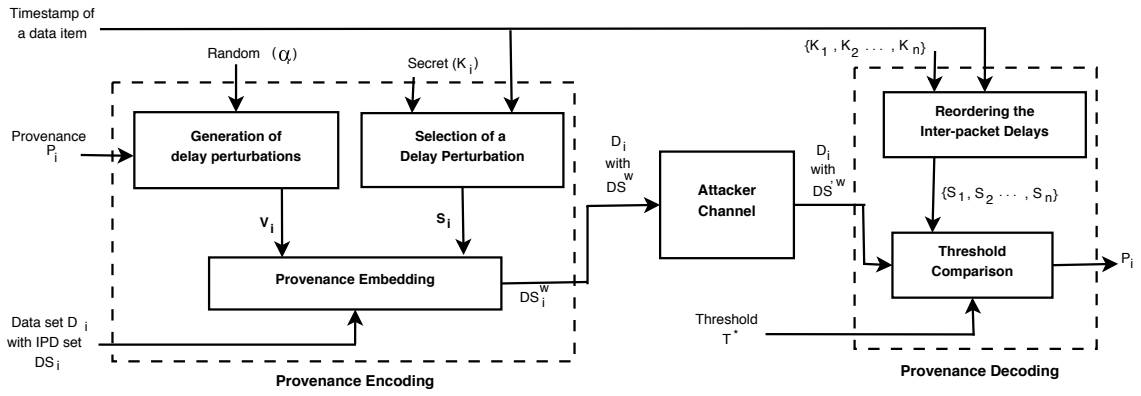


Fig. 5.3.: Stages of Provenance Encoding at a Sensor Node and Decoding at the Base Station

5.2.1 Provenance Encoding

After generating a data packet, the source node marks it with the generation time and ensures the integrity of the timestamp with a MAC. The MAC is computed using the node specific secret key K_i. The next L_p data packets generated by the node, more specifically, the sequence of L_p IPDs is the medium where we hide the provenance of the packets. We denote the set of IPDs by $\mathcal{DS} = \{ \Delta[1], \Delta[2], ..., \Delta[L_p] \}$, where $\Delta[j]$ represents the IPD between *j-th* and *(j+1)-th* data packet. The data source encodes a bit of its PN sequence over each IPD. Throughout the transmission of a packet towards the BS, each intermediate node also encodes 1-bit of provenance information over the associated IPD. Hence, an IPD recorded at the BS carries the sum of 1-bit information from each node in the path. The process also uses the secret K_i and a locally generated random number α_i (known as impact factor). The BS only knows the distribution of the α_i's. The process a node n_i follows to encode a bit of PN sequence over an IPD is summarized below:

1. Generation of Delay Perturbations: n_i generates a set of delay perturbations by using the PN sequence $\mathbf{pn_i}$ and impact factor α_i. α_i is a random (real) number generated according to a normal distribution $N(\mu, \sigma)$. μ and σ are predetermined and known to the BS and all the nodes. Thus, the BS only knows the distribution of α_i's, but not their exact values. However, n_i generates the set of delay perturbations $\mathcal{V}_i$ as a sequence of real numbers as follows

$$\begin{aligned} \mathcal{V}_i &= \alpha_i \times \mathbf{pn_i} \\ &= \alpha_i \times \{\ pn_i[1],\ pn_i[2],\ ...,\ pn_i[L_p]\ \} \\ &= \{\ (\alpha_i \times pn_i[1]),\ ...,\ (\alpha_i \times pn_i[L_p])\ \} \\ &= \{v_i[1],\ v_i[2],\ ...,\ v_i[L_p]\} \end{aligned} \tag{5.2}$$

Note that, $v_i[j]$ corresponds to the provenance bit $pn_i[j]$. However, the node may perform the computation offline since it is independent of any packet specific information.

2. Selection of a Delay Perturbation: On the arrival of any $(j+1)-th$ data packet, n_i records the IPD $\Delta[j]$ and assigns a delay perturbation $v_i[k_j] \in \mathcal{V}$ to it. To ensure the robustness of the scheme, the delay perturbations are not assigned sequentially to the IPDs i.e. $v_i[j]$ is not assigned to $\Delta[j]$. Instead, a delay perturbation $v_i[k_j]$ is selected using the secret K_i and the packet timestamp. The algorithm uses the following formula

$$selection(\Delta[j]) = H(ts[j+1] \,\|\, K_i) \; mod \; L_p \tag{5.3}$$

Here, H is a lightweight, secure hash function, $\|$ is the concatenation operator, and $ts[j+1]$ represents the packet timestamp. Since secure hash functions generate uniformly distributed message digests, each execution of the selection mechanism will result in a unique integer in the range $[0, L_p - 1]$. The resulting integer can be used to index a distinct element in $\mathcal{V}_i$.

As part of the provenance encoding process, each node executes the algorithm once for each of the L_p IPDs returning a set of indices as the permutation of integers from 0 to $L_p - 1$. The indices are used to point the elements in $\mathcal{V}_i$. Thus, the order according to which each node embeds the delays from $\mathcal{V}_i$ over the IPDs forms a permutation of the elements different from the sequential order. This sequence is denoted as $S_i = \{s_i[1],\, s_i[2],\, ...\, s_i[L_p]\} = \{v_i[k_1], v_i[k_2], ..., v_i[k_{L_p}]\}$. Note that, given an IPD, the algorithm will select differently indexed delays for different nodes based on the key K_i. Thus, an attacker cannot predict the *IPD-to-Delay Perturbation* assignment without the knowledge of secrets K_i and L_p. Keeping the provenance length secret is not a requirement but keeping it secret makes it harder for an attacker to regenerate the selections.

3. Provenance Embedding: In this step, n_i delays the packet transmission by $v_i[k_j]$ time unit. As $v_i[k_j]$ corresponds to the provenance bit $pn_i[k_j]$, through this step a provenance bit is embedded over an IPD. This notion makes our scheme reminiscent of watermarking. we present the provenance embedding algorithm into two steps:

(i) Simple Provenance Embedding: As shown in Fig. 5.1(a), the simple provenance is represented as a simple path. Each node in the path watermarks its PN sequence over a set of L_p IPDs i.e. $(L_p + 1)$ packets are utilized. Intuitively, the first packet in a data flow does not experience any delay due to provenance embedding. For any other $(j+1)-th$ data packet (sent/forwarded), each node in the path hides a provenance bit over the associated IPD $\Delta[j]$. Interchangeably, a node n_i uses the IPD $\Delta[j]$ to accommodate a delay perturbation $v_i[k_j](= s_i[j])$. Using $s_i[j]$, the delay to be added to $\Delta[j]$ is computed as:

$$\lambda_i[j] = s_i[j] \times T \tag{5.4}$$

where T is the value of a time unit. If $s_i[j] > 0$, the resulting $\lambda_i[j] > 0$ and then we can perform watermarking by simply adding $\lambda_i[j]$ to $\Delta[j]$. But if $s_i[j] < 0$, the delay to be added to an IPD is negative. To avoid this situation, we introduce a constant offset when calculating $\lambda_i[j]$, which ensures that $\lambda_i[j]$ is always positive. The offset may be any constant leading to $\lambda_i[j] > 0$. We use $(\mu + const * \sigma)$ in our scheme, where *const* is any constant that makes $\lambda_i[j]$ greater than 0 i.e. $const > \dfrac{-(s_i[j]+\mu)}{\sigma}$. Thus, the final equation is

$$\lambda_i[j] = (s_i[j] + (\mu + const * \sigma)) \times T \tag{5.5}$$

n_i then performs watermarking by adding $\lambda_i[j]$ to $\Delta[j]$ i.e. delaying the packet transmission by $\lambda_i[j]$ time. Thus, n_i formulates the watermarked IPD $\Delta^w[j]$ and transmission time of the (j+1)-th packet $t'_i[j+1]$ as follows

$$\Delta^w[j] = \Delta[j] + \lambda_i[j] \tag{5.6}$$

$$t'_i[j+1] = t_i[j+1] + \lambda_i[j] + c \tag{5.7}$$

where c is a constant > 0 corresponding to the delay added by a sensor node, including processing and any other delay. After watermarking, n_i sends the $(j+1)-th$ packet towards the BS at instant $t'_i[j+1]$. Throughout the transmission, all other nodes in the provenance embed one bit of provenance information over the IPD following the same procedure.

(ii) Aggregate Provenance Embedding: Figure 5.1(b) shows the aggregate provenance, represented as a tree. Assume that in an aggregate provenance tree, n_a is the aggregator possessing U children $n_{l_1}, n_{l_2}, ..., n_{l_U}$. At any $(j+1)-th$ sensing interval $(1 \leq j \leq L_p)$, the child nodes send data to n_a embedding their provenance information over the locally managed IPDs. Watermark delays of the children are denoted by $\lambda_{l_1}[j], ..., \lambda_{l_U}[j]$ respectively. n_a computes the aggregated data, attaches authenticated timestamp from one of its children, and also maintains the corresponding IPD in such a way that this delay represents the provenance embedding for the aggregator and its children. Intuitively, we could accomplish this by adding a delay of $\lambda_A[j] = \sum_{i=1}^{U} \lambda_{l_i}[j] + \lambda_a[j]$ to the unwatermarked IPD, where $\lambda_a[j]$ represents the watermark delay computed by the aggregator. The $\lambda_{l_i}[j]$'s can be approximated by the aggregator from the IPD observations while data is being received from the corresponding child. However, this scheme would impose a major delay to the aggregated data which would abruptly reduce data throughput. To address this problem, we propose a different solution based on some mathematical tricks.

Let the watermark delays for a child node n_{l_i} average to Λ_{l_i}. Utilizing the Λ_{l_i}'s of child nodes, n_a computes the watermark delay (denoted as $\lambda_A[j]$) for aggregated provenance as follows:

$$\lambda_A[j] = \lambda_a[j] + \sum_i (\lambda_{l_i}[j] - \Lambda_{l_i}) \tag{5.8}$$

$\lambda_A[j]$ in Eq. (5.8) may also be negative. So, we also add the constant offset to make $\lambda_A[j]$ always positive. The reason why this solution works is explained in sec. 5.2.2.

By following the above procedure, each node in the flow path encodes its 1-bit information. Consequently, the provenance bits are watermarked over the L_p IPDs by manipulating them with corresponding delay perturbations, termed as *watermark delay*. This way, $\mathcal{DS}$ is transformed into the watermarked version $\mathcal{DS}^w$. However, data packets may also experience different propagation delays or attacks aimed at destroying the provenance information. At the end, the BS receives the dataset along with watermarked IPDs $\mathcal{DS}^w$, which can be interpreted as the sum of delays imposed by the intermediate nodes, the attackers, and the difference between consecutive propagation delays along the data path. Thus, $\mathcal{DS}^w$ represents the DSSS encoded signal in our context.

5.2.2 Provenance Decoding

The provenance retrieval algorithm recovers provenance using the secret parameters including the keys $\{K_1, K_2, ..., K_n\}$, the PN length L_p, and the optimal threshold T^*. The threshold, corresponding to the network diameter and PN length, is calculated once after the deployment of the network. The way how to calculate this decoding threshold is described below in section 5.2.3.

The BS records the watermarked IPDs and executes the retrieval process whenever it collects a number of L_p IPDs denoted by the set $\mathcal{DS}^w$. Since the BS does not know which nodes embedded their identities in the provenance, it executes the process for all of the nodes in the network and tries to identify the desired nodes. For each

node, the BS generates a node specific sequence of real numbers by reordering the IPDs in $\mathcal{DS}^w$ according to the bit selection algorithm. We denote such a sequence by $\mathbf{CS_i} = \{cs_i[1],\ cs_i[2],\ ...,\ cs[L_p]\}$. Any element (i.e. IPD) in this sequence can be interpreted as the sum of delays added by the nodes in provenance, the difference of propagation delay between two consecutive data packets, and possibly any delay added due to malicious attacks. Thus,

$$cs_i[j] = \sum_{k,m} \lambda_k[m] \ + \ \sum_{k,m} \Delta tr_{(k\,,\,k+1)}[m] + D[m] \tag{5.9}$$

where $\Delta tr_{k\,,\,k+1}$ is the difference between the propagation delays of two consecutive data packets from *k-th* intermediate node to *(k+1)-th* node and $D[m]$ is any delay added due to attacks. We can expand the equation as

$$\begin{aligned} cs_i[j] &= \sum_{k,m} s_k[m] \times T \ + \ \sum_{k} (\mu \ + const * \ \sigma) \times T \\ &\quad + \sum_{k} \Delta tr_{(k\,,\,k+1)}[m] + D[m] \end{aligned} \tag{5.10}$$

As $(\mu \ + const * \ \sigma) \times T$ is a constant, the sum over this constant can be denoted as another constant Tc. To determine whether a node contributes to a data flow, the cross-correlation between $\mathbf{CS_i}$ and provenance information $\mathbf{pn_i}$ is computed as follows

$$\begin{aligned} R_i &= \mathbf{CS_i} \,.\, \mathbf{pn_i} = \sum_{j} cs_i[j] \times pn_i[j] \\ &= \sum_{j} \sum_{k,m} (s_k[m] \times T) \times pn_i[j] + \sum_{j} Tc \times pn_i[j] \\ &\quad + \sum_{j} \sum_{k,m} \Delta tr_{(k\,,\,k+1)}[m] \times pn_i[j] + \sum_{j} D[m] \times pn_i[j] \end{aligned} \tag{5.11}$$

As $\mathbf{pn_i}$ has an equal number of 1's and -1's, $\sum_j pn_i[j]$ becomes 0 resulting in $\sum_j Tc \times pn_i[j] = 0$. Due to this special property of $\mathbf{pn_i}$, any constant delay added during watermarking will contribute a 0 to the cross-correlation. For the same reason, adding

a delay of $(\lambda_{l_i}[j] - \Lambda_{l_i})$ during aggregation instead of $\lambda_{l_i}[j]$ has the same effect. The constant Λ_{l_i}, if added, would have been eliminated from the cross-correlation.

Note that, the last two terms, representing difference in propagation delays and attacker induced delays, are negligible compared to the first term i.e. ideal cross-correlation value. So, the inclusion of the node in provenance can be decided correctly by a comparison of R_i with T^*. If $R_i \geq T^*$, the identity of this node was embedded i.e. the node contributed to data flow. Otherwise, the node did not participate. After successfully retrieving the provenance information, the BS resets $\mathcal{DS}^w$ and starts collecting IPDs for future provenance retrievals.

The decoding error can be reduced further by embedding the provenance, i.e. each $v[j] \in \mathcal{V}$, multiple times. The number of repetitions is called *redundancy factor*. At the BS, the provenance is extracted multiple times and the decision about the presence of a node in the provenance is taken based on a *majority voting technique*. Thus, the effect of any unusual propagation delay or malicious attacks is mitigated. Besides, the knowledge of diameter H of the sensor network can be used to determine the nodes in the data flow path more accurately by selecting H nodes with the highest cross-correlation values.

5.2.3 Decoding Threshold Evaluation

This section presents the evaluation of an optimal threshold T^* that minimizes the *probability of decoding error* which is defined as the probability of retrieving provenance incorrectly. Let P_{err}, P_1, and P_0 represent the probability of decoding error, probability that a node embeds its identity (i.e. PN sequence) in the provenance and probability of not embedding, respectively. Variables p_e and p_r denote the probability of embedding and retrieval of a node's PN sequence, respectively ($p_e = 1$ implies that the PN sequence of a node was embedded, $p_r = 1$ implies the PN sequence

was retrieved). $f(r)$ is the probability density function of cross-correlation. P_{err} is calculated as:

$$\begin{aligned} P_{err} &= P(p_r = 0, p_e = 1) + P(p_r = 1, p_e = 0) \\ &= P(p_r = 0|p_e = 1)P_1 + P(p_r = 1|p_e = 0)P_0 \\ &= P(r < T|p_e = 1)P_1 + P(r > T|p_e = 0)P_0 \\ &= P_1 \int_{-\infty}^{T} f(r|p_e = 1)dr + P_0 \int_{T}^{\infty} f(r|p_e = 0)dr \end{aligned} \tag{5.12}$$

To minimize the probability of decoding errors (P_{err}), we take the first order derivative of P_{err} with respect to T to locate the optimal threshold T^* as follows:

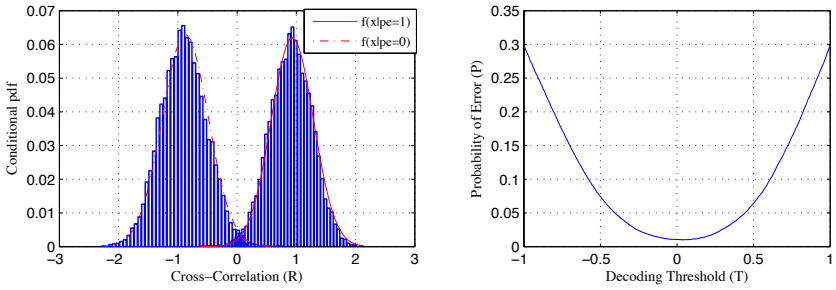
$$\begin{aligned} \frac{\partial P_{err}}{\partial T} &= P_1 \frac{\partial}{\partial T} \int_{-\infty}^{T} f(r|p_e = 1)dr + P_0 \frac{\partial}{\partial T} \int_{T}^{\infty} f(r|p_e = 0)dr \\ &= P_1 f(T|p_e = 1) - P_0 f(T|p_e = 0) \end{aligned} \tag{5.13}$$

The distributions $f(r|p_e = 0)$ and $f(r|p_e = 1)$ are estimated from the statistics of sets R'_e and R_e, respectively. The experimental observations of cross-correlation for the nodes present in the provenance are stored in a set R_e and for those that are not present are stored in R'_e. The values of R'_e and R_e show that the distributions $f(r|p_e = 0)$ and $f(r|p_e = 1)$ can be estimated as Gaussian distributions $N(\mu_0, \sigma_0)$ and $N(\mu_1, \sigma_1)$ respectively. However, the following analysis can still be performed with other types of distributions. P_0 could be estimated by $\frac{|R_e|}{|R_e| + |R_{e'}|}$ and $P_1 = 1 - P_0$. Substituting the Gaussian expressions for $f(r|p_e = 0)$ and $f(r|p_e = 1)$ in Eq. 5.13 and equating it to zero we get the following quadratic equation

$$\frac{\sigma_0^2 - \sigma_1^2}{2\sigma_0^2\sigma_1^2} T^{*2} + \frac{\mu_0\sigma_1^2 - \mu_1\sigma_0^2}{\sigma_0^2\sigma_1^2} T^* + ln\left(\frac{P_0\sigma_1}{P_1\sigma_0}\right) + \frac{\mu_1^2\sigma_0^2 - \mu_0^2\sigma_1^2}{2\sigma_0^2\sigma_1^2} = 0 \tag{5.14}$$

The roots of this equation give the optimal threshold T^* that minimizes P_{err}. The second order derivative of P_{err} is evaluated at T^* to ensure that the second order necessary condition ($\frac{\partial^2 P_{err}(T^*)}{\partial T^2} > 0$) is met. To show the high dependency of the

probability of decoding errors on the choice of decoding threshold T^*, we conducted experiments with a sensor network of diameter 12 and PN Length = 240 bits. The histograms and the Gaussian estimates of R_e and R'_e obtained from the experiment are reported in Fig. 5.4(a). The optimal computed threshold T^* is indicated by the dotted vertical line. As we can see from Fig. 5.4(a), the two distributions are far apart which is a direct result of using the competing objects for b_i equal to 1 and 0. Fig. 5.4(b) shows the probability of decoding error for different values of the threshold, which in turn shows the presence of an optimal threshold that minimizes the probability of decoding error.



(a) $f(x|p_e = 0)$, $f(x|p_e = 1)$ and the optimal threshold $T^* = 0.0303$

(b) Experimental values of P_{err} for different decoding threshold (T) values

Fig. 5.4.: Decoding Threshold Evaluation for Provenance Watermarking Scheme

5.2.4 Security Analysis

In this section, we discuss the security and resiliency of our provenance scheme against various outside and inside attackers.

Outside Attacker

With the capability of capturing data packets and inter-packet timing characteristics, an outside attacker may try to disrupt provenance security in different ways.

Provenance detection and retrieval: An attacker might want to identify and extract the provenance embedded by a node. Several attacks have been devised to detect and corrupt the active timing-based watermark in network flows. Cabuk et al. implement a covert network timing channel which transmits one packet in a time interval to encode the bit '1' and stays silent for a '-1' bit [32]. Such a static encoding of messages leads to a highly regular behavior in the inter-packet delays, whereas overt traffic arrives anytime, resulting in an irregular pattern. Cabuk et al. show how to detect the covert channel by identifying a regular pattern in the IPDs. However, in our scheme, the watermarked IPDs do not follow any regular pattern, rather the IPDs appear random in nature and it is hard to distinguish the patterns generated by the watermarking from natural variation in traffic rates. Hence, our scheme can evade detection based on regularities in data traffic [32].

Peng et al. develop an attack technique [37] to detect, recover, duplicate or remove a message, watermarked in a flow according to the scheme proposed in [34]. The attacker tries to infer important watermarking parameters (such as quantization step used to compute watermark delay, proportion of watermarked IPDs etc.) using packet timestamps at each intermediate host and achieves the attack goals utilizing these parameters. Since our watermarking recovery process uses every IPD for watermarking purpose, this attack process does not help an attacker in extracting the provenance, embedded according to our scheme. Moreover, the lack of clock synchronization between nodes will weaken this attack.

Luo et al. propose an approach to detect and autonomously remove spread spectrum flow watermarks (SSFW) [80]. Since the encoder needs to throttle the flow's throughput to a low value for a given period T_c for embedding a '-1' and spreading the watermark using PN codes increases the number of such low-throughput periods significantly, the SSFW causes an abnormal sequence of low-throughput periods (large delays) in the flow. Hence, the attacker can detect the SSFW by identifying the presence of anomalous sequences of low-throughput periods. Kiyavash et al. have devised a multi-flow attack to detect the SSFW [36] based on the observation of long

low-throughput period on several flows compared to a trained model. Compared to existing SSFW techniques, our scheme uses low amplitude watermarks i.e. much smaller delays (on the order of few milliseconds) that appear close to natural network jitter. It makes the provenance invisible to attackers and thus prevents the attackers from detecting and removing the provenance. In our system, each node possesses unique provenance information that is watermarked in the flow and also the embedding position of the provenance bits is changed continuously. Thus, a multi-flow attack cannot defeat our scheme.

It is important to notice that these attacks mainly focus on detecting whether a data flow has a secretely embedded watermark and, if present, then on recovering/removing it. On the contrary, the attacker in our context might have prior knowledge about the fact that a timing-based provenance watermarking scheme is applied in the sensor network. Also we are not considering the complete removal of provenance as well. Therefore, attacks conducted to only detect the existence of provenance will not help the attacker anyway, unless the attacker can retrieve the provenance information of a node. In addition, most of these attacks are addressed to specific watermarking techniques and hence cannot be generalized to disrupt any watermarking scheme. However, the following claim shows that our scheme can evade such detection and retrieval attacks

However, a statistical test, based on the assumption that IPDs of covert traffic center on limited numbers of distinct values instead of being randomly distributed [81], can detect the presence of provenance in the time domain. The reason is that the mean of watermark delays for '1' and '-1' bits converges to two separate values in our scheme. Still, an attacker cannot retrieve the provenance information of a node by observing the IPDs of flows from/to that node. The embedding positions of provenance bits are changed in every round of embedding based on the packet timestamp and they also differ from node to node. Hence, given a sequence of L_p IPDs, the attacker has to try all combinations of these numbers to get the order of bits in the provenance information. For example, given 120 delays for a 120 bit provenance information

(with equal number of 1's and -1's), the attacker has to try $\binom{120}{60}$ combinations to get the original sequence of provenance bits.

Replay Attack: An adversary may replay previously heard data packets (transmitted by legitimate nodes) to give a false idea about the sensed environment [82]. For an IPD based provenance transmission system (like ours), the attacker also observes the timing characteristics in order to maintain them during packet replay. To make the replayed data appear as fresh, the attacker will update the packet timestamp to a recent value. Nevertheless in our scheme, the selection of provenance bit for any j-th IPD depends on the timestamp of $(j+1)$-th packet and thus changes with the varying timestamp. So, sustaining the old time observations while marking the packet with a new timestamp does not allow the BS to extract provenance successfully. Consequently, the provenance integrity check fails and the data is discarded.

Inside Attacker

An inside attacker may want to generate fake data and construct the provenance including some innocent nodes $\{n_{i_1}, n_{i_2}, ..., n_{i_U}\}$ in order to mark them as untrustworthy by making them responsible for false data. However, this attack will fail since the provenance embedding process requires node-specific secrets, like the PN code, the secret key, and the impact factor, and the attacker does not know these for the uncompromised nodes.

Provenance Modification Attack: An attacker, acting alone or colluding with others, may want to add or remove nodes from the provenance of data generated by benign nodes. Assume that n_e and n_m are compromised nodes and collude to execute the attack. A benign data item d, with provenance $p_d = \{n_{i_1}, n_{i_2}, ..., n_{i_U}\}$, is routed through n_e which wants to remove n_{i_2} from p_d and replace it with n_m. To remove n_{i_2} from provenance, n_e has to remove the delays added by n_{i_2} from IPDs. Since negative delays cannot be added, n_e will adjust the $j\text{-}th$ IPD by delaying the j-th packet which decreases the delay introduced to the $(j+1)-th$ packet for provenance embedding.

The amount of delay to be added can be found by observing the timing characteristics of packets to and from n_{i_2}. Note that n_e has to adjust the IPDs in reverse order, from $j = L_p$ to 1. To achieve this, n_e has to accumulate all the $(L_p + 1)$ packets, adjust their IPDs, and then transmit these packets towards the BS maintaining the adjusted timings. Such an attack scheme will add too much delay to the packets, which will definitely be detected at the BS. Regarding the addition of a node, n_e can easily add n_m in the provenance if they collude. However, the provenance integrity check at the BS will fail and detect an attack.

Forgery: Our scheme can also detect provenance forgery i.e. given the valid provenance for a data packet, the attacker cannot associate this provenance with a data packet with a difference provenance. A malicious routing node n_e can perform two types of attack:

Forgery attack 1. Suppose that the data packet d belongs to a data flow generated by a benign node n_s. The provenance of d is $p_d = \{n_s, n_{i_1}, ..., n_{i_U}\}$. n_e might want to associate p_d with a fake packet d_e. To achieve this, n_e tries to insert d_e in the flow while maintaining the observed timing characteristics. However, to certify that d_e is a part of the flow generated by n_s, n_e must generate the MAC of the data value and timestamp by using the secret key of n_s. Being unaware of the secrets of n_s, n_e cannot generate the MAC.

Forgery attack 2. d_1, d_2 belong to two different data flows generated by n_{s_1} and n_{s_2}, respectively. n_e swaps the data value of these packets. Hence, the BS will now identify n_{s_2} as a part of the provenance for d_1 whereas d_1 contains the MAC generated by n_{s_1}.

Hence, the data integrity check will detect the provenance forgery in both cases.

Unauthorized Access: Only authorized parties can access and check the integrity of provenance. This follows from the provenance decoding process which requires the PN sequences and secret keys of all nodes in the network (at least for nodes in the provenance). Only the authorized party (the BS in our case) that has access to these information can retrieve the provenance and thereafter check the integrity.

From the above security analysis it follows that an adversary cannot access or modify the provenance without being detected. However, modifications may destroy the provenance and impact the robustness of the scheme. The strength of our scheme is that, with some redundancy and detection mechanism, it can recover the provenance upto a great extent. We consider the following attacks by compromised nodes:

Deletion Attack: A compromised node can destroy the information carried out by the IPDs by dropping data packets routed through it. Dropping any j-th data packet consumes the (j-1)th and j-th IPD. However, we can mitigate this attack by embedding the provenance multiple times and employing the *majority voting technique* when retrieving the provenance, as discussed in sec. 5.2.2. The impact and effectiveness of the *redundancy factor* (i.e. how many times the provenance is embedded) on provenance recovery is evaluated and reported in Fig. 5.5(c)

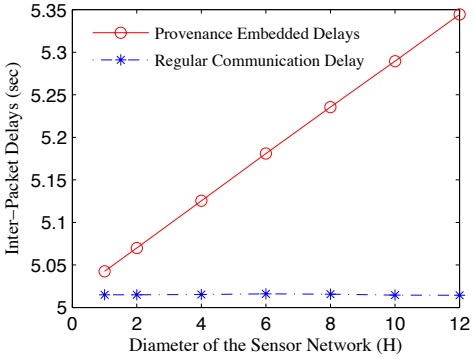
Alteration Attack: This attack perturbs the IPDs with the goal of moving the cross-correlation values from above the threshold T^* to below the threshold T^* and vice versa, leading the erroneous retrieval of provenance. As in the deletion attack, embedding provenance multiple times will reduce the impact of this attack. However, the attacker may try to change the IPDs within a safe range, since an alteration to an IPD beyond a certain limit would be recognized by the BS as an attack. Such a modification, however, would affect the cross-correlation value negligibly, thus leaving the provenance decoding process undisturbed.

Insertion Attack: A malicious routing node may insert fake data in the data flow generated by a legitimate node. Through the MAC verification or using some standard detection mechanism, the BS can detect such false data packets and discard them. Thus, an insertion attack will have almost no effect on the provenance decoding.

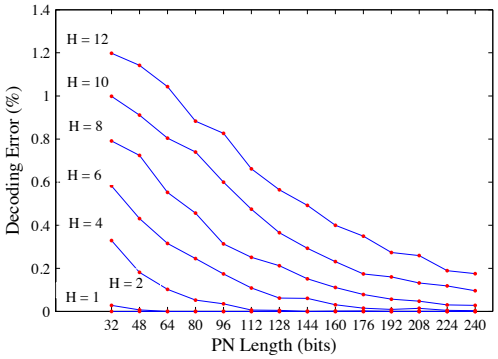
5.2.5 Experimental Evaluation

We evaluate the scalability and robustness of our scheme. For the experiments, we simulate the sensor network as a tree with diameter H. The network consists

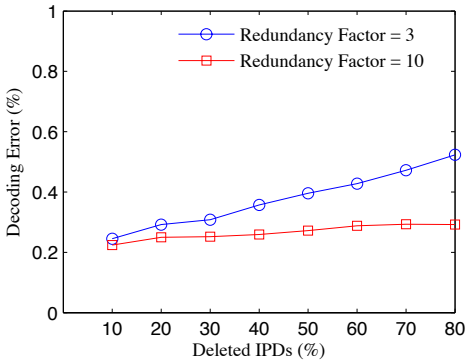
of 1000 nodes with default values of $H = 8$, and $L_p = 160$ bits. Other parameters include $\mu = 5, \sigma = 0.005, const = 100$, time unit = 5 ms, and redundancy factor = 1. Sensor data is generated every 5 seconds. For each experiment, the simulations were run 100 times.



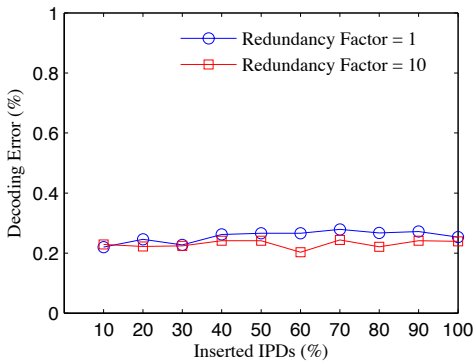
(a) Delays incurred due to provenance embedding



(b) Decoding error



(c) Resilience to Deletion attack



(d) Resilience to Insertion attack

Fig. 5.5.: Performance of provenance watermarking scheme

Scalability

The scalability of our solution is evaluated by quantizing the impact of H on the overall delay due to provenance embedding. The reason why we investigate the relationship of delay to the network diameter instead of the number of nodes is that the provenance length increases linearly with the diameter. In comparison, the effect

of the total number of nodes is much lower. Fig. 5.5(a) shows a comparison of natural IPDs with the watermarked IPDs. The watermarked IPD increases from natural IPD by a maximum of 6%.The graph also shows that the watermarked IPD linearly increases with H though the increasing rate is not high.

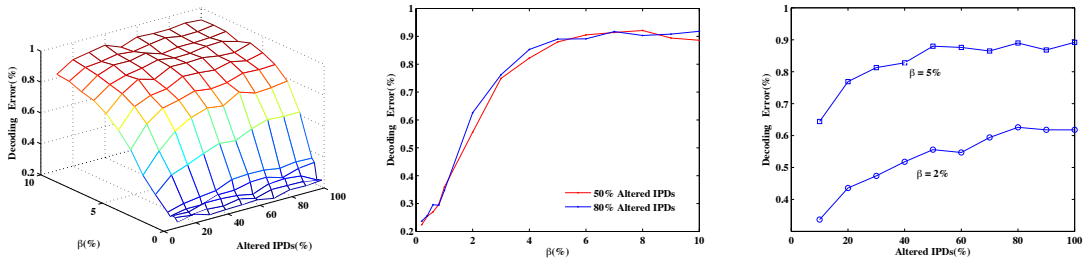
As the diameter of sensor network has a direct influence on the PN length, one has to determine the optimal PN length for a particular H that ensures a low decoding error. Fig. 5.5(b) reports the percentage of the provenance decoding error for different PN lengths with varying network diameters. Predictably, an increase in the PN length results in a decrease of the decoding error for a particular diameter as well as the increase in diameter imposes a higher error rate for a particular PN length.

Provenance Recovery

These experimental results show how well the decoding process can recover the provenance against various attacks discussed in sec. 7.1.2.

Deletion Attack: The adversary randomly drops α data packets (of a data flow) routed through it. The provenance is then decoded and the decoding error is measured for different α values as reported in Fig. 5.5(c). We evaluated the performance of our scheme for various redundancy factors. The decoding error decreases with increasing values of the redundancy factor.

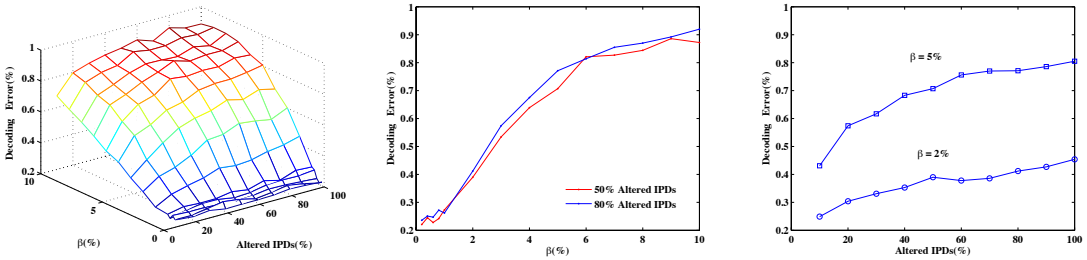
Alteration Attack: We evaluated the performance of our decoding technique against two types of alteration attacks namely, fixed and random (α, β) alteration attacks. In the fixed-(α, β) alteration attack, the attacker randomly selects $\frac{\alpha}{2}$ data packets and delays them by multiplying the corresponding IPDs by $(1 + \beta)$. Consequently, each following IPD (total $\frac{\alpha}{2}$) is decreased by $(1 - \beta)$. Here, β is a fixed value. In the random-(α, β) attack, the IPDs are multiplied by $(1 + x)$, where x is a uniform random variable $\in [0, \beta]$. Figures 5.6(a)(b)(c) show the behavior of our scheme against the fixed-(α, β) alteration attack. In Fig. 5.6(a), as the percentage of IPDs altered and the alteration factor increases, so does the decoding error. However, our solution



(a) Fixed-(α, β) alter attack: Decoding errors cors. to various α, β

(b) Fixed-(α, β) alter attack: Decoding errors against β

(c) Fixed-(α, β) alter attack: Decoding errors cors. to the rate of IPD alterations



(d) Random-(α, β) alter attack: Decoding errors cors. to various α, β

(e) Random-(α, β) alter attack: Decoding errors against β

(f) Random-(α, β) alter attack: Decoding errors cors. to the rate of IPD alterations

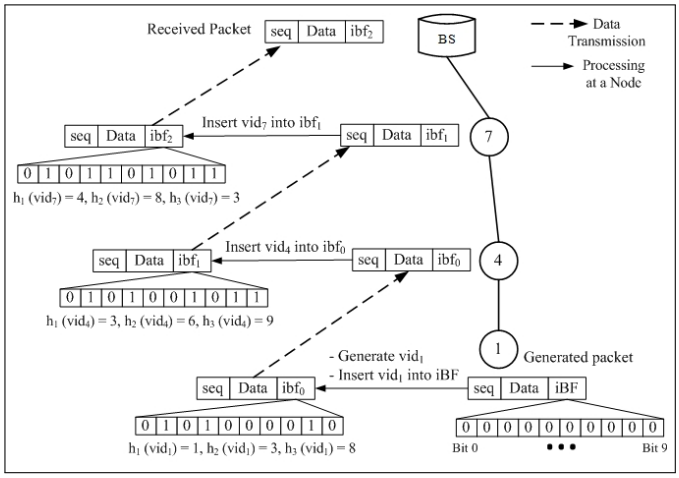
Fig. 5.6.: Resilience to fixed-(α, β) and random-(α, β) alter attacks

seems surprisingly resilient. The provenance decoding error shows low increases for increasing percentages of altered IPDs (b) or the alteration factor (c). Similar results were experienced for the random-(α, β) attack as shown in Figures 5.6(d)(e)(f).

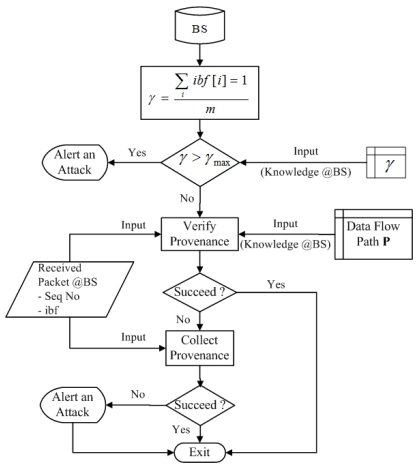
Insertion Attack: In this experiment, we insert α data packets in the flow i.e. add α IPDs. Utilizing our detection mechanism, we can achieve an almost constant decoding error while varying the percentage of inserted IPDs. Fig. 5.5(d) shows the robustness of our solution against the insertion attack.

5.3 Bloom Filter based Provenance Scheme

To enable per-packet provenance transmission for sensor data, we propose a light-weight distributed scheme relying on in-packet Bloom filters [83] to encode provenance. In addition, we introduce efficient mechanisms for provenance verification and



(a) Provenance encoding scheme



(b) Provenance processing workflow at the BS upon receiving a packet

Fig. 5.7.: iBF based provenance scheme

reconstruction at the BS. Each packet consists of a unique sequence number, data value, and an iBF which holds the provenance. We focus on transmitting provenance graph vertices over an iBF.

We emphasize that our focus is on securely transmitting provenance to the BS. In an aggregation infrastructure, securing the data values is also an important aspect, but that has been already addressed in previous work (e.g., [84]). Our secure provenance technique can be used in conjunction with existing work to obtain a complete solution that provides security for data, provenance and data-provenance binding.

5.3.1 Provenance Encoding

For a data packet, *provenance encoding* refers to generating the vertices in the provenance graph and inserting them into the iBF. Each vertex originates at a node in the data path and represents the provenance record of the host node. A vertex is uniquely identified by the vertex ID (VID). The VID is generated per-packet based on the packet sequence number (seq) and the secret key K_i of the host node. We use a *block cipher function* to produce this ID in a secure manner. Thus for a given data

packet, the VID of a vertex representing the node n_i is computed with a secure block cipher (such as, AES, etc) E as follows:

$$vid_i = generateVID(n_i, seq) = E_{K_i}(seq) \tag{5.15}$$

Whenever a source node generates a data packet, it also creates a BF (referred to as ibf_0), initialized to all 0's. The source then generates a vertex according to Eq. 5.15, inserts the VID into ibf_0 and transmits the BF as a part of the packet.

Upon receiving the packet, each intermediate node n_j performs data as well as provenance aggregation. If n_j receives data from a single child n_{j-1}, it aggregates the partial provenance contained in the packet with its own provenance record. In this case, the iBF ibf_{j-1} belonging to the received packet represents a partial provenance i.e. the provenance graph of the sub-path from the source upto n_{j-1}. On the other hand, if n_j has more than one child, it generates an aggregated provenance from its own provenance record and the partial provenance received from its child nodes. At first, n_j computes a BF $ibf_{(j-1)}$ by bitwise-ORing the iBFs received from the children. ibf_{j-1} represents a partial but aggregated provenance from all of the child nodes. In either case, the ultimate aggregated provenance is generated by encoding the provenance record of n_j into $ibf_{(j-1)}$. To this end, n_j creates a vertex using Eq. 5.15, inserts the VID into $ibf_{(j-1)}$ which is then referred to as ibf_j.

When the packet reaches the BS, the iBF contains the provenance records of all the nodes in the path i.e. the full provenance. We denote this final record by ibf.

Example: We illustrate the encoding mechanism by using the example network in Fig. 5.7(a). The data path considered is $< 1, 4, 7 >$, where node 1 is the data source. We use a 10-bit BF and a set of 3 hash functions $H = \{h_1, h_2, h_3\}$ for BF operations. When node 1 generates a data packet with sequence number seq, it creates the BF ibf_0 which is set to all 0's. The node then creates a vertex corresponding to its provenance record and computes the VID as $vid_1 = E_{K_1}(seq)$. To insert vid_1 into ibf_0, node 1 generates three indices as $h_1(vid_1) = 1$, $h_2(vid_1) = 3$, $h_3(vid_1) = 8$. The

VID is then inserted by setting $ibf_0[1]$, $ibf_0[3]$, and $ibf_0[8]$ to 1. The updated ibf_0 along with the packet is then sent towards the BS.

Upon receiving the packet, node 4 performs data and provenance aggregation. Since the node has one child, it only aggregates its own provenance record with ibf_0. For this purpose, the node generates a VID vid_4; computes 3 indices as $h_1(vid_4) = 3$, $h_2(vid_4) = 6$, $h_3(vid_4) = 9$; and inserts vid_4 into ibf_0 by setting bits 3, 6, 9 of the iBF to 1. This updated iBF is referred to as ibf_1. The data packet with ibf_1 is then forwarded to node 7 which repeats the provenance aggregation steps. At the end, the BS receives the packet with the final iBF (ibf_2 from node 7) and stores this iBF for further processing.

5.3.2 Provenance Decoding

When the BS receives a data packet, it executes the *provenance verification* process, which assumes that the BS knows what the data path should be, and checks the iBF to see whether the correct path has been followed. However, right after network deployment, as well as when the topology changes (e.g., due to node failure), the path of a packet sent by a source may not be known to the BS. In this case, the provenance collection process is executed, which retrieves provenance from the received iBF and thus the BS learns the data path from a source node. Afterwards, upon receiving a packet, it is sufficient for the BS to verify its knowledge of provenance with that in the packet. Below we discuss these two processes in more details:

Provenance Verification: The BS conducts the verification process not only to verify its knowledge of provenance but also to check the integrity of the transmitted provenance. Algorithm 1 shows the steps to verify provenance for a given packet. We assume that the knowledge of the BS about this packets path is P. At first, the BS initializes a Bloom filter BF_c with all 0's. The BF is then updated by generating the VID for each node in the path P and inserting this ID into the BF. BF_c now reflects the perception of BS about the encoded provenance. To validate its perception, the

BS then compares BF_c to the received iBF ibf. The provenance verification succeeds only if BF_c is equal to ibf . Otherwise, if BF_c differs from the received iBF, it indicates either a change in the data flow path or a BF modification attack. The verification failure triggers the provenance collection process which attempts to retrieve the nodes from the encoded provenance and also to distinguish between the events of a path change and an attack.

Algorithm 1 ProvenanceVerification

Input: Received packet with sequence *seq* and iBF *ibf*.
Set of hash functions H, Data path $P' = < n'_{l_1}, ..., n'_1, ..., n'_p >$
$BF_c \leftarrow 0$ // Initialize Bloom Filter
for each $n'_i \in P$ **do**
    vid'_i = generateVID $(n'_i, Seq\,No)$
    insert vid'_i into BF_c using hash functions in H
endfor
if $(BF_c = ibf)$ **then**
    return true // Provenance is verified
endif
return false

Provenance Collection: As illustrated in Algorithm 2, the *provenance collection* scheme makes a list of potential vertices in the provenance graph through the ibf membership testing over all the nodes. For each node n_i in the network, the BS creates the corresponding vertex (i.e. v_i with VID vid_i) using Eq. 5.15. The BS then performs the membership query of vid_i within ibf. If the algorithm returns true, the vertex is very likely present in provenance, i.e., the host node n_i in the data path. Such an inference might introduce errors because of false positives (a node not on the route is inferred to be on the route). However, as shown in Section 5.3.3, the false positive probability obtained is very low.

Once the BS finalizes the set of potential candidate nodes S $= < n'_{l_1}, ..., n'_1, n'_2, ..., n'_p >$, it executes the provenance verification algorithm on this set. This step is required to distinguish between the cases of a legitimate route change and that of malicious activity. If the verification succeeds, we decide that there was a natural change in

the data path and we have been able to determine the path correctly. Otherwise, an attack has occurred.

Algorithm 2 ProvenanceCollection

Input: Received packet with sequence *seq* and iBF *ibf*.
Set of nodes (N) in the network, Set of hash functions H

1. Initialize
   Set of Possible Nodes $S \leftarrow <>$
   Bloom Filter $BF_c \leftarrow 0$ // To represent S
2. Determine possible nodes in the path and build the representative BF
   for each node $n_i \in N$ **do**
    vid_i = generateVID (n_i, seq)
    if (vid_i is in ibf) **then**
     $S \leftarrow S \cup n_i$
     insert vid_i into BF_c using hash functions in H
    endif
   endfor
3. Verify BF_c with the received iBF
   if ($BF_c = ibf$) **then**
    return S // Provenance has been determined correctly
   else
    return NULL // Indicates an in-transit attack
   endif

A possible attack is the *all-one* attack where all bits in the provenance are set to 1, which implies the presence of all nodes in the provenance. To address the issue, we use a *density metric* γ introduced in [38]. γ reflects the number of 1's in the provenance (i.e. the iBF) as a fraction of the total size. To consider the provenance valid, we require that the density is equal or below a certain threshold: $\gamma \leq \gamma_{max}$. Such a requirement is reasonable since in a BF with n elements and k hash functions, there may be at most kn bits marked as '1'. Hence, we can always find an upper bound for the number of 1's in a BF. Thus, the maximum number of allowable 1's is $m\gamma_{max}$. Within this bound, an attacker may also *randomly flip some bits* to add or delete a legitimate node. The chance of being successful in this attack is very small since the attacker has to identify k bit positions corresponding to the node, which again change for each packet. If each bit is guessed randomly, the probability

that the attacker guesses all of them correctly is given by $\frac{1}{2^m}$. Moreover, an attempt of blindly altering some bits is detected since the verification process at the end of the *provenance collection* phase does not succeed. A successful attack occurs when the bits set by the attacker (limited by γ_{max}) make all the k bits corresponding to a legitimate node turn out to be '1'. If the data provenance includes n nodes, the kn hash results may map to at least one and at most $m\gamma_{max}$ bits. Thus a smart attacker marks upto $(m\gamma_{max} - 1)$ bits. The total number of bit patterns by $(m\gamma_{max} - 1)$ hash computations is

$$B = \sum_{i=1}^{(m\gamma_{max}-1)} \binom{m}{i} \tag{5.16}$$

Randomly guessing one of them has $\frac{1}{B}$ chance of success. Hence, the success in manipulation attack has a very small probability. The workflow shown in Fig. 5.7(b) summarizes the provenance decoding process.

5.3.3 Performance Analysis

We present an analysis of the space and energy overhead of the proposed scheme. To the best of our knowledge, no secure provenance scheme has been proposed for sensor networks. Hence, we use the following two benchmarks:

(i) We adapt the generic secure provenance framework *SProv* [15] to sensor networks. In this lightweight version of the scheme, referred to as SSP, we simplify the provenance record at a node n_i as $P_i =< n_i, hash(D_i), C_i >$, where $hash(D_i)$ is a cryptographic hash of the updated data, and C_i contains an integrity checksum as $Sign(hash(n_i, hash(D_i)|C_{i-1}))$.

(ii) We also consider a MAC-based provenance scheme, referred to as MP, where a node transmits the nodeID and a MAC computed on it as the provenance record.

Space Complexity

To implement SSP, we use SHA-1 (160 bit) for cryptographic hash operations and the TinyECC library [85] to generate 160-bit digital signatures (ECDSA). The nodeID has length 2 bytes, thus the length of each provenance record is 42 bytes. For MP, we use TinySec library [86] to compute a 4-byte CBC-MAC. Hence, a provenance record has 6 bytes in this case. As each node in the path encodes its own provenance record, the provenance size increases linearly with the number of hops. For a D-hop path, the provenance is $42D$ bytes in SSP and $6D$ bytes in MP.

Since our approach is based on BF, the provenance length depends on parameter selections for the BF. The false positive probability for a BF is defined as [87]

$$P_{fp} = \frac{n_a - n}{n_t - n} \tag{5.17}$$

where n_t is the total number of distinct elements in the element space, n is the number of elements actually encoded in the BF and n_a is the number of elements retrieved by querying the BF. Let m be the BF size, k the number of hash functions and D the maximum number of nodes in any path. The false positive probability is equal to that of getting 1 in all the k array positions computed by the hash functions while querying the membership of an element that was not inserted in the BF. This probability is

$$P_{fp} = (1 - (1 - \frac{1}{m})^{kD})^k \approx (1 - e^{-\frac{kD}{m}})^k \tag{5.18}$$

For a given m and D, the number of hash functions that minimizes the false positives can be computed as

$$k_{opt} = \frac{m}{D} ln2 \tag{5.19}$$

Given D and a desired false positive probability P_{fp}, the required number of bits m can be computed by substituting the optimal value of k in Eq. (5.18) and then simplifying it to

$$ln(P_{fp}) = -\frac{m}{D} * (ln2)^2 \ \Rightarrow m = \frac{-D * ln(P_{fp})}{(ln2)^2} \tag{5.20}$$

This means that in order to maintain a fixed false positive probability, the length of a BF should grow with the number of elements to be inserted. If we consider $P_{fp} = 0.02$ and a 14-hop path, the BF size m is computed as 114 bits and $k_{opt} = 6$. Thus, a 120-bit (15 byte) BF is sufficient to encode provenance while maintaining low false positives. In practice, we bound P_{fp} by a small constant $\delta\,(> 0)$ such that $P_{fp} < \delta$. To find the appropriate value of m we have

$$ln(P_{fp}) > ln\delta \ \Rightarrow -\frac{m}{D} * (ln2)^2 > ln\delta \ \Rightarrow m < \frac{Dln\frac{1}{\delta}}{(ln2)^2} \tag{5.21}$$

Energy Consumption

For a D-hop path, SSP has to transmit $42*D$ bytes ($= 336*D$ bits), MP transmits $6 * D$ bytes ($= 48 * D$ bits) whereas our scheme requires m bits transmitted. SSP, MP and our scheme consume a radio energy proportional to $(336 * D)$, $(48 * D)$ and $\frac{ln\frac{1}{\delta}}{(ln2)^2} * D$, respectively. Although all of the terms are proportional to D, the constant coefficient in the first two terms are much larger than the last one. For example, if we set $\delta = 10^{-4}$ then the coefficient in our scheme is 19.17 which is much smaller than the coefficients in SSP and MP. Another part of overhead comes from the signature, MAC and hash computations. However, in sensor networks, usually computation overhead is much smaller than that of communication and adds only marginal energy consumption [88].

5.3.4 Security Discussion

Confidentiality. Provenance is encoded using BF hashing functions, and the hashed value takes into account the secret key K_i of each node as part of the vertex VIDs (Eq. 5.15), as well as a unique sequence number. Hence, even if an attacker collects a large sample of iBFs, it cannot perform a dictionary attack without knowing the node secret key.

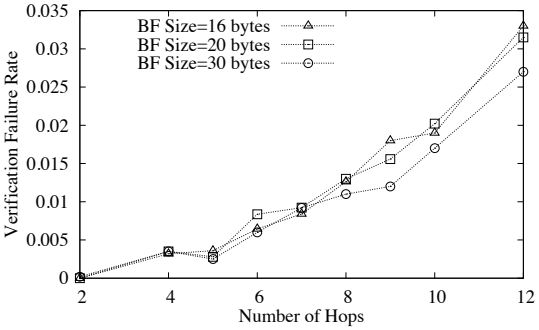
Integrity. First, an attacker cannot add legitimate nodes to the provenance of data generated by the compromised nodes. Assume the attacker attempts to frame some uncompromised nodes $< n_l, n_1, n_2, \ldots, n_p >$ to make them responsible for false data. Provenance embedding requires the node secret key K_i to compute the VID_i, which the attacker does not have. Hence, the attack is not successful.

Second, an attacker cannot selectively add or remove nodes from the provenance of data generated by uncompromised nodes. Assume that nodes n_e and n_m collude to execute an attack. A benign packet with provenance $< n_l, ..., n_1, n_2, ..., n_p >$ is routed through n_e, and n_e attempts to remove n_2 from the provenance and to replace it with another legitimate node n_2. When the packet reaches n_e , it contains the partial provenance $< n_l, ..., n_1, ..., n_e >$ encoded in the iBF ibf_{pp}. To remove n_2 from provenance, at first n_e has to construct the Bloom filter BF_2 containing the provenance record of n_2. The bitwise-AND of the negated value of BF_2 with ibf_{pp} removes the information of n_2 from the provenance. Assume the modified iBF is ibf_{pp}. To add n_2' to the provenance after the removal of n_2, the BF corresponding to n_2 should be built and then OR-ed with ibf_{pp}. In both cases, the attackers are unable to construct a BF representing uncompromised nodes, due to the absence of the secret keys of legitimate nodes.

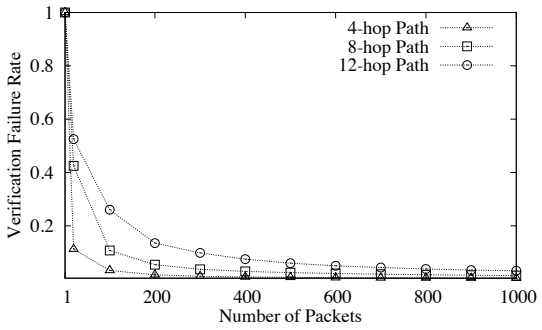
Freshness. Provenance replay attacks are detected by our proposed scheme, since provenance is derived using a unique packet sequence number and the secret key of the node. An attempt to change the sequence number of a packet without having the key will be detected at the BS, according to the integrity property discussed above.

5.3.5 Simulation Results

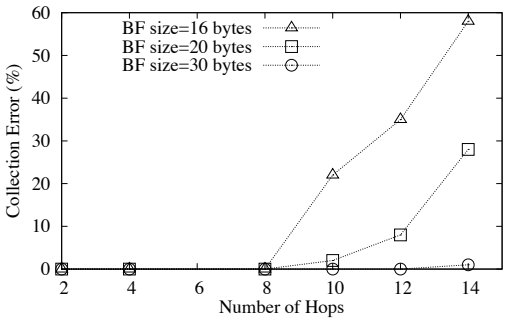
We implemented and tested the proposed technique using the TinyOS simulator (TOSSIM) [89], and we have used the *micaz* energy model. We consider a network of 100 nodes and vary the network diameter from 2 to 14. All results are averaged over 100 runs with different random seeds.



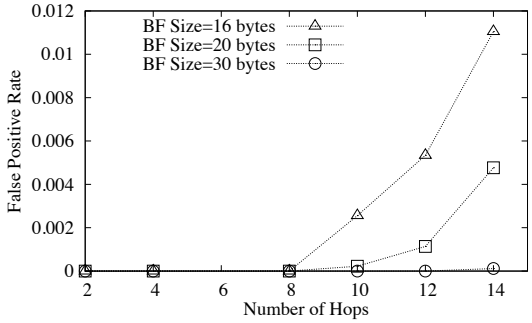
(a) Provenance VFR vs path length



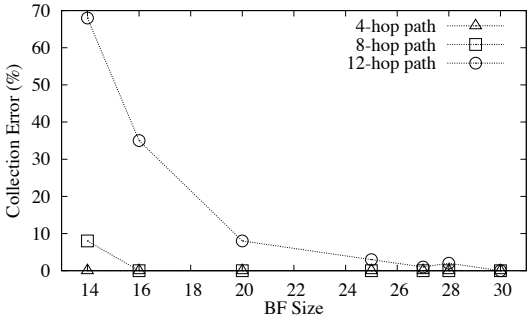
(b) VFR variation with time as network stabilizes



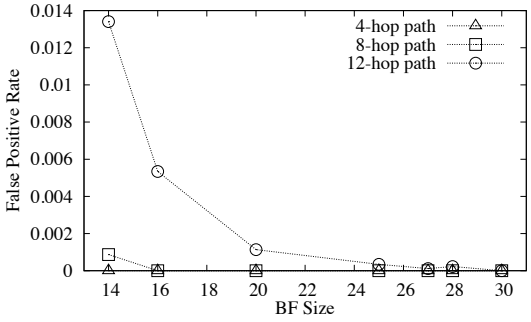
(c) Collection Error for various path lengths



(d) False Positive Rate for various path lengths



(e) Collection Error for various BF sizes



(f) False Positive Rate for various BF sizes

Fig. 5.8.: Performance of the iBF based provenance scheme

Provenance Decoding Error

The provenance decoding process retrieves the provenance from the in-packet Bloom filter, and consists of the *verification* and *collection* phases. To quantify the accuracy and efficiency of our provenance scheme, we measure decoding error in both the above phases, i.e., *verification* and *collection error*.

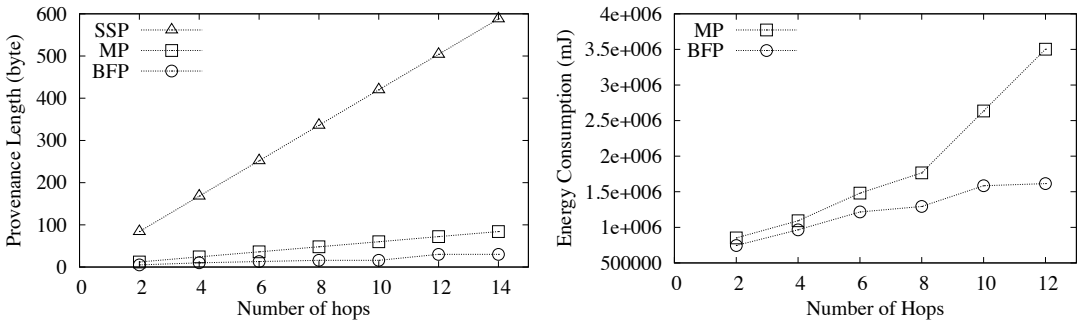
Algorithm 1 shows that the verification fails when the provenance graph in the packet does not match with the local knowledge at the BS. This may happen when there is a data flow path change or upon a BF modification attack. Provenance *verification failure rate (VFR)* measures the ratio of packets for which verification fails. Fig. 5.8(a) shows the VFR for paths of 2 to 12 hops with various BF sizes. For each path length, the VFR is averaged over 1000 distinct paths. The results show that the provenance verification process fails only for a very small fraction of packets. Thus, for most packets the *lightweight verification* process is sufficient to retrieve the provenance. The more costly provenance collection process is executed only for a very few packets when verification fails. As expected, VFR increases linearly with the increase of the path length. On the other hand, VFR is not significantly influenced by BF size, proving that even small BF sizes provide good protection. Fig 5.8(b) shows the variation of VFR over time, as the number of packet transmissions increases. As the network gets stable with time, the data paths do not change often, and hence the VFR approaches 0.

Fig. 5.8(c) and 5.8(d) plot the percentage of *provenance collection error* for different number of hops and the corresponding false positive rates, respectively. Recall that, the collection phase is executed when provenance verification fails. Fig. 5.8(e) and 5.8(f) show the collection error corresponding to various BF sizes and the related false positives, respectively. The number of hash functions used are determined using Eq. (5.19). The resulting false positive rates vary from $0 \sim 0.013$ and it is observed that the collection error becomes negligible when the false positive rate drops at or below 10^{-4}. It is also seen that a BF size of 16 bytes is enough to ensure no decoding

error for up to 8-hop paths. The empirical BF size required is much less than the theoretical one($\sim$ 20 bytes for a 8-hop path).

Space Complexity and Energy Consumption

Fig. 5.9(a) shows a comparison among SSP, MP and our provenance mechanism in terms of bytes required to transmit provenance. The provenance length in SSP and MP increases linearly with the path length. For our scheme, we empirically determine the BF size which ensures no decoding error. Although the BF size increases with the expected number of elements to be inserted, the increasing rate is not linear. We see that even for a 14-hop path, a 30 byte BF is sufficient for provenance decoding without any error.



(a) Provenance length (b) Aggregate energy consumption

Fig. 5.9.: Space complexity and energy consumption of the provenance scheme

We also measure the energy consumption due to provenance construction and transmission for various hop counts. Note that, modern sensors use *ZigBee* specification for high level communication protocols which allows upto 104 bytes as data payload. Hence, **SSP and MP can be used to embed provenance (in data packet) for maximum 2 and 14 nodes, respectively**. Figure 5.9(b) compares the aggregate energy consumption of MP with that of our scheme over 100 packet transmissions. The results confirm the energy efficiency of our solution.

6. PROVENANCE USAGE

In this chapter, we demonstrate how the provenance systems designed by us can be utilized to provide security solutions in various domains. First, we propose a mechanism for detecting potential data exfiltration attack, based on the file provenance information collected by FiPS. Second, we present a provenance based mechanism for detecting packet dropping adversary in WSNs, utilizing our iBF based provenance solution. We extend the basic provenance scheme in order to detect packet dropping activity on the data flow path over a period of time and then to identify malicious link(s). It is assumed that the links on the path exhibit natural packet loss and several (colluding) adversarial nodes may exist on the path.

6.1 Detection of Data Exfiltration

The key idea to the detection mechanism is to characterize data exfiltration by detecting anomalous file access patterns. To distinguish cyber-insider mission actions from legitimate file activities, we record file provenance, i.e. the history of which processes wrote to the file, which files were used to modify the file, etc. Based on the provenance graph built during a training period, we can find the legitimate access patterns to a file, which help us later on to detect an anomalous access to the file. Figure 7.2 shows the architecture of the proposed exfiltration detection mechanism.

6.1.1 Training Phase

Since we propose a data-centric approach towards the detection of data exfiltration, we establish a baseline for expected access patterns to a file during the training

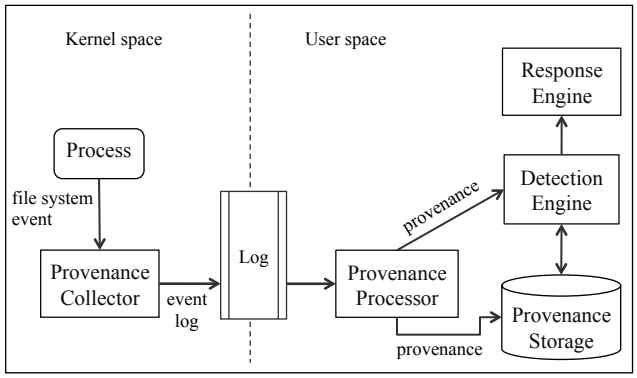


Fig. 6.1.: Architecture of Data Exfiltration Detection System using File Provenance

phase. In this phase, we assume there is no anomaly in the system and the files are accessed in order to accomplish regular tasks.

As described earlier, the provenance collector gathers provenance metadata on a provenance related event and sends off the provenance records to the provenance processor through a log file. During the training phase, the provenance processor translates the log data to provenance records, generates appropriate provenance entities and relationships and adds them to the provenance graph of the corresponding file. The provenance graphs are stored in the provenance database. After the training period, we expect a robust knowledge of legitimate accesses to files through the file provenance graphs, which will help us taking a pragmatic approach to detect anomalous activities later on.

6.1.2 Detection Engine

During normal operation of the system, the provenance processor not only processes and stores the provenance information, but also interacts with the detection engine to detect any anomalous access. When a file is about to be modified, deleted, etc., the provenance processor generates the *access graph* of the file in the current session and reports it to the detection engine. The detection engine then searches for the given access pattern within the provenance graph of that file, stored in the prove-

nance database. If the module cannot match the pattern, it indicates a suspicious access path to the file, which might have been changed or created by an anomaly. Thus the detection engine raises a flag in this case.

It is to be mentioned that, sensitive data leakage may happen in various ways, such as, information transfer via USB media, ftp, telnet, email, etc. To be successful in the mission, an insider attacker may make unauthorized write to OS files, scripts, and executables, send suspicious outbound traffic, make unauthorized modification of data in databases, etc. Thus, we have to take care of suspicious writes to not only regular files, but also many other medias like, USB, network, system file, etc. The Linux system, however, treats all the peripherals as files. Thus when any write is about to happen to any of these special purpose files, the detection engine, using the detection mechanism, tracks the files that are being read and sent off and checks whether this is legitimate through access pattern search.

6.2 Packet Dropping Adversary Identification in WSNs

In this scheme, we use a packet-acknowledgement based approach which requires the sensors to transmit more meta-data in the provenance record. For a data packet, the provenance record generated by a node now consists of the node ID and the sequence number of the lastly seen (processed/forwarded) packet belonging to that data flow. If there is an intermediate packet drop, some nodes in the path do not receive the packet. Hence during the next round of packet transmission, there will be a mismatch between the acknowledgements (for the lastly seen packet) generated by the nodes. We utilize this fact to detect the packet dropping attack and then to localize the malicious link. The detection of a packet loss requires the data source to securely transmit the sequence number of the packet it generated in the previous round. The mechanism for detecting packet dropping attack is explained below. For discussion, we consider the data flow path $P =< n_l, n_1, ..., n_i, ..., BS\}$, where n_l is the

only data source and $n_d = BS$ is the base station. For simplicity, we denote the link between nodes n_i and $n_{(i+1)}$ as l_i.

1. Provenance Encoding: Fig. 6.2 depicts the provenance encoding scheme, which is a simple extension to the basic scheme. A provenance record here includes node ID and an acknowledgement to the lastly observed packet of the flow. The acknowledgement can be generated in various ways to serve the purpose. To keep the solution simpler, we transmit packet sequence number to acknowledge a packet. For any *j-th* packet, a node n_i creates a vertex v_i and the vertex ID as follows

$$\begin{aligned} vid_i &= generateVID\,(\,n_i,\, seq,\, pSeq_i\,) \\ &= E_{K_i}(\,seq\,||\,pSeq_i\,) \end{aligned} \tag{6.1}$$

where seq is the sequence number attached to the current packet and $pSeq_i$ is the stored information at n_i about the sequence number of the $(j-1)$-th packet. To update the provenance graph of the packet, n_i then inserts vid_i into the associated iBF. To

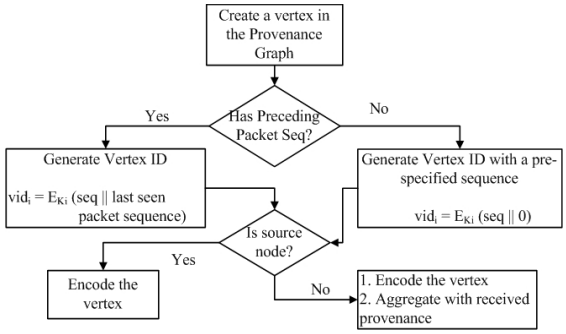


Fig. 6.2.: Extended provenance framework to detect packet dropping attack and identify the malicious link

be noted that, a node must maintain a per-flow record to store the previous packet sequence for each data flow passed through it. Whenever a node processes/forwards a packet, it updates the previous packet record of the appropriate data flow with the recently process packet sequence. If a node receives packet from a data flow for which it has no packet sequence information, then it may use a *pre-specified special purpose*

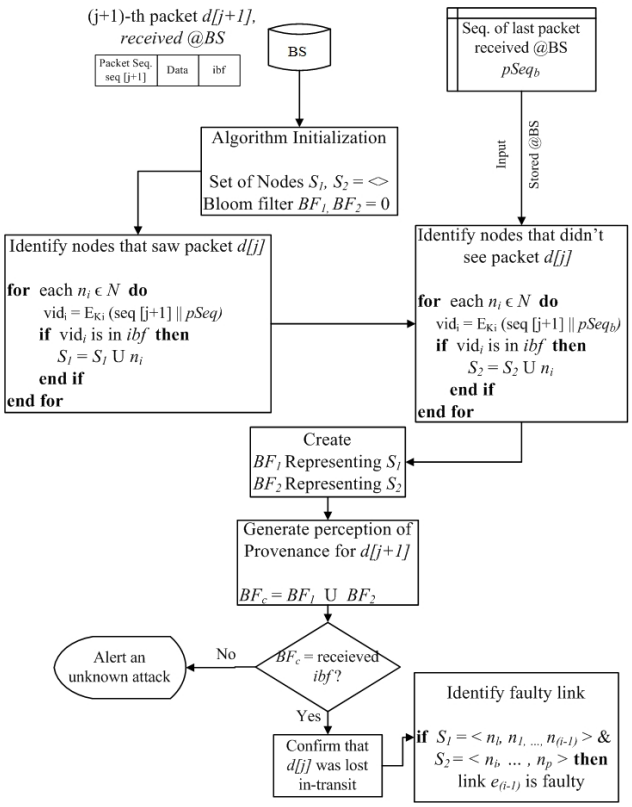


Fig. 6.3.: Packet loss detection and faulty link identification using provenance.

identifier, such as 0. It addresses the case of routing path change where a new node in the path can use this special identifier for encoding provenance. Moreover, if a node does not receive packets from a data flow for a long time, it can erase the previous packet information for that flow to reduce space overhead. Anyway, the node can get updated and maintain this record when it receives packets from that flow more frequently.

2. Provenance Decoding at the BS: The BS also stores the sequence number of the most recent packet processed for each data flow. Upon receiving a packet, the BS retrieves the sequence of the last packet transmitted by the source node from the packet header, fetches the previous packet sequence for the flow from its local storage and then compares these two sequence numbers. If there is no packet dropping attack, each node in the path as well as the BS receives all packets in the flow and

thus possesses the same previous packet sequence. Otherwise, if the BS observes a difference between these two sequence numbers, it infers about a possible packet loss and then takes necessary actions to confirm the event and to localize the faulty link. However, the provenance verification and/or collection are performed according to the algorithms 1 and 2, respectively. The only difference is that, the BS now creates the vertex ID corresponding to a node according to the Eq. 6.1.

3. Faulty Link Identification using Provenance: Assume, a data packet $d[j]$ has been dropped at an intermediate node n_i. Thus, the nodes $n_l, n_1, ..., n_i$ received $d[j]$ and updated their lastly seen packet sequences to $seq[j]$. On the contrary, nodes $n_{i+1}, ..., n_p$ as well as the BS did not observe $d[j]$, They have no way to update the preceding packet sequence but to retain the same old identifier $seq[j-1]$. Upon receiving the next packet in the flow, $n_l, n_1, ..., n_{i-1}$ certainly include $seq[j]$ in the provenance metadata whereas $n_{i+1}, ..., n_p$ use $seq[j-1]$ for this purpose. However, the malicious node n_i may either (i) use $seq[j]$, which leads the BS to detect l_i as faulty (ii) use $seq[j-1]$, in which case the link $l_{(i-1)}$ is identified as faulty. In any case, an adjacent link to the malicious node is identified and held responsible for. Without the loss of generality, we assume that the malicious node encodes $seq[j-1]$.

Figure 6.3 shows the algorithm to identify an earlier packet loss, to localize the faulty link, and also to ensure that no other attack has been encountered on the current packet. It uses the received iBF to identify the contributing nodes and to collect their encoded provenance records. For this purpose, it checks the membership of all nodes in the network within the iBF using a two step process. The first query is performed with the previous packet identifier ($pSeq$) contained in the packet header and the next one with the sequence number ($pSeq_b$) recorded at the BS. Let, the set of nodes found in the first and second step are respectively $S_1 = < n'_l, n'_1, ..., n'_{(i-1)} >$ and $S_2 = < n'_i, ..., n'_p >$. The BF constructed with S_1 and S_2 are BF_1 and BF_2, respectively. The final Bloom filter BF_c is constructed as a bitwise-OR of BF_1 and BF_2. If BF_c and the received iBF ibf completely matches, the event of a packet loss is confirmed. In this case, the path constructed on the set of nodes S = S_1 U S_2 is

equivalent to the path P as well as $S_1 = < n_l, n_1, ..., n_{(i-1)} >$ and $S_2 = < n_i, ..., n_p >$. Thus, we can conclude that the link $l_{(i-1)}$ is faulty and causes the packet loss.

4. Certification of Attack: To confirm that the faulty link $l_{(i-1)}$ is actually malicious (i.e. causes packet dropping attack), the BS observes more packets. Whenever the BS identifies a packet loss and the responsible link $l_{(i-1)}$, it updates the empirical loss rate $el_{(i-1)}$ for the link. Assume, the *drop rate threshold* for a link is α, where α is greater than the natural loss rate of any link. If after a number of packet transmissions, $el_{(i-1)} > \alpha$, then the BS convicts $l_{(i-1)}$ as a malicious link.

6.2.1 Performance Analysis

Provenance is used to detect a packet loss and to identify the faulty link. Thus the faulty link detection error depends on the BF parameters and can be analyzed in a similar way as in sec. 5.3.3. The only difference is that the element space is larger now due to considering an element as the concatenation of nodeID and a packet sequence. Hence a larger BF is required to keep the false positive rate small.

Since packet dropping attack directly reduces the amount of legitimate data throughput, we also analyze our scheme to provide the theoretical bounds for guaranteed end-to-end throughput under attacker's control and also attack detection rate. Let ρ_i be the natural packet loss rate of link l_i, and ρ_i's are i.i.d. random variables with maximum value ρ. Let α denotes the per-link drop rate threshold. The theoretical bounds are computed under the converged condition when the empirical loss rate converges to its true value within a small uncertainty interval. The detection rate of the proposed scheme i.e. the number of data packets transmitted by the source before reaching the converged condition is computed in the following theorem

Theorem 1: *Given the threshold $\alpha = \rho + \epsilon$ and the allowed false positive σ, the scheme requires the $\frac{\ln \frac{2}{\sigma}}{8\epsilon^2.(1-\rho)^D}$ packets transmitted by the source to converge.*

The following theorem provides a bound on the damage that an adversary can inflict to the networks end-to-end throughput by the time of detection.

Theorem 2: *Given a path of length D, an adversary in control of z intermediate links can cause (at most) the $z\alpha$ end-to-end packet loss rates without being detected.*

6.2.2 Simulation Results

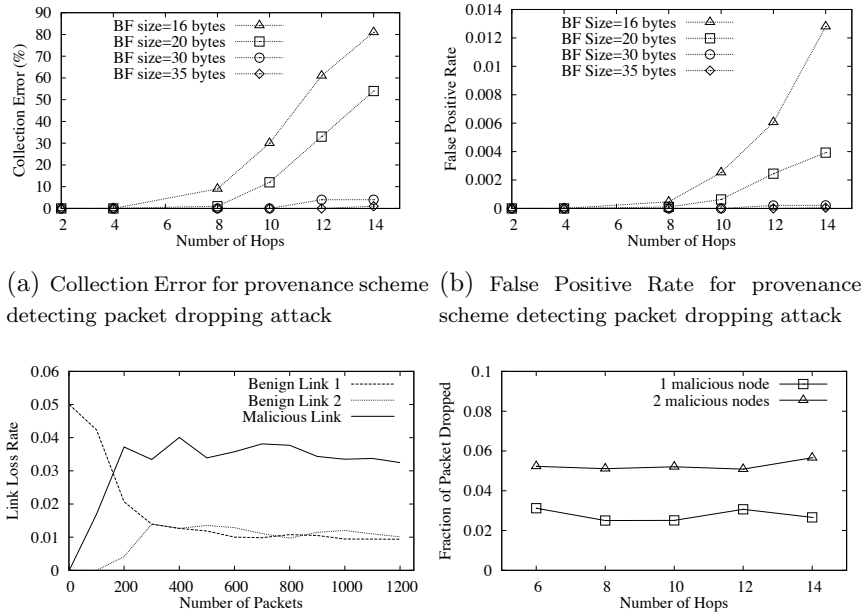
We use simulations to further verify our analysis and to show the effectiveness of the proposed schemes. The scheme is implemented in TinyOS and the simulations are conducted using the TinyOS simulator - TOSSIM. We consider a network of 100 nodes where the maximum number of hops vary from 2 to 14. For energy analysis, PowerTOSSIM z [90] is used which utilizes the *micaz* energy model. Unless otherwise stated, the simulation results are averaged over 100 runs.

Fig. 6.4(a) and 6.4(b) show the percentages of provenance collection error and corresponding false positive rates for the provenance scheme to detect packet dropping attack. Since the element space for the received iBF here increases compared to the basic scheme, the error rates increase than earlier for the same BF sizes. Since a very little fraction of packets change the data flow path, provenance verification process succeeds most of the time and the collection process is executed so infrequently. Hence, the collection error does not affect much the accuracy of the *faulty link identification* process. The impact of the collection error can further be minimized by getting the order of the nodes in the path using topology knowledge. We have found that encoding edges greatly helps in this regard by providing the exact topology information.

Fig. 6.4(c) shows the accuracy of the *faulty link identification* process over time and how it leads to the detection of packet dropping attack. The figure plots the link loss rates over packet transmissions in order to show the convergence of link statistics to their actual values. For an uncompromised node, the link loss rate should converge to the natural loss rate whereas for a malicious node the link statistics should tend towards a significantly higher loss rate which would confirm the packet dropping attack. For this experiment, we consider an arbitrary 6 hop path where n_3 is the

malicious node and controls the link l_3. Natural link loss rate $\rho = 0.01$, malicious link loss rate α is 0.03, and allowed false positive $\sigma = 0.003$. The dynamics prove that eventually the packet dropping attack is detected successfully. However, there is probability of error since in earlier stage the loss rate of malicious link seems to be much less than 0.03 while the loss rates of benign links seem high.

Fig. 6.4(d) presents the degradation of data throughput by the time the attack is detected in case of 1 and 2 malicious nodes deployed. The fraction of dropped traffic are ~ 0.03 and ~ 0.055 for 1 and 2 intermediate malicious nodes, respectively. The empirical results are bounded by the theoretical estimations.



(a) Collection Error for provenance scheme detecting packet dropping attack

(b) False Positive Rate for provenance scheme detecting packet dropping attack

(c) Accuracy of *faulty link identification* and the detection of attack over time

(d) End-to-end packet drop rate

Fig. 6.4.: Performance of Provenance Scheme Detecting Packet Dropping Attack

7. A SECURITY INCIDENT RESPONSE AND PREVENTION SYSTEM FOR WIRELESS SENSOR NETWORKS

Wireless Sensor Networks (WSNs) are susceptible to operational failures and security attacks due to resource constraints, unattended operating environment, and communication phenomena. However, WSN applications impose stringent requirements on reliable and trustworthy data delivery, and service availability. It becomes more challenging to satisfy these requirements when attackers exploit the insecure and vulnerable nature of sensor environments to falsify context, modify access rights, and, in general, disrupt the system operation [91]. This can result in a wide area blackout, a patient receiving the wrong treatment, or worse, facing a life risk [92]. Thus, WSNs must be able to continuously provide their services despite anomalies or attacks and to effectively recover from attacks without significant interruption.

Over the recent years, Intrusion Detection Systems (IDSes) [49, 93, 94] have been proposed specifically for WSNs, which cooperatively detect intrusions and report possible attacks to a central authority. However, these systems are not equipped with response tools that would enable automatic responses and recovery actions. The intrusion response systems developed for other domains, such as database systems, distributed systems, cannot be directly used in WSNs due to significant differences in their operations, resources, and communication. In the context of WSNs, we need an intrusion response system that is **lightweight** in terms of computational cost, and resource usage. To fulfill this objective, the system should use local and **cooperative** strategies instead of heavy interactions with a central authority. Also the response policies should be specified so not incur much overhead when selecting the appropriate response actions. Nonetheless, the response system should respond in **real-time**, yet execute the most effective action for each anomaly or attack in a secure fashion.

In this section, we describe *Kinesis* - the first systematic approach to a security incident response and prevention system (IRPS) for WSNs. We extend the concept of traditional intrusion response systems to an extensive response framework that not only recovers from attacks, but also reacts to anomalies in order to prevent service disruptions and attacks. The system is lightweight, cooperative, and distributed in design. According to our design, each sensor in the WSN is a watchdog monitor [49] and hosts both an IDS, and the Kinesis system. Through the IDS, the monitor observes neighbor behaviors, detects suspicious incidents (anomaly/attack) in the neighborhood, and notifies Kinesis. However, Kinesis depends on the IDS only for the notifications on good/bad neighbor behaviors which is the basic functionality of an IDS. Upon being notified of an incident, Kinesis matches the appropriate response policy from the set of response policies specified by the base station (BS).

To support the specification of response policies in Kinesis, we propose a WSN specific lightweight policy language based on the Event-Condition-Action (ECA) paradigm [95]. A response policy is defined on an incident and specifies different actions based on security assessments of the suspect node. A monitor estimates the security level of the suspect node based on the (i) incident detection confidence, (ii) suspect's behavior history, and (iii) incident impact on the WSN. This strategy helps selecting the most effective response action at any instant. We have surveyed the various attacks in WSNs and created a taxonomy of attacks (Fig. 7.1) and a comprehensive set of response actions (Table 7.3). However, Kinesis can generate responses against an unknown attack based on the anomalous behavior the attack manifests.

To trigger the response execution corresponding to an incident, Kinesis selects a *daemon* node in a neighborhood via a self-organized competition among the neighbors. The competition is controlled in a distributed fashion by a per-node *action timer*. The node whose timer fires first wins the competition and executes the action. Most of the actions involve a transmission which is overheard by the neighbors and then allows the neighbors to stop their action timers and to refrain from taking redundant actions. Thus, Kinesis does not require any message exchanges for the response action

synchronization and has no communication overhead. A node's action timer value is locally estimated based on: (i) neighborhood size, (ii) neighbor link qualities, (iii) time since its last action. It reflects the effectiveness of a node in executing the action and ensures load distribution among the neighbors.

The distributed nature of Kinesis also enhances security. When a node is compromised, other legitimate nodes in the neighborhood can continue with the Kinesis functionalities. Kinesis is secure in terms of policy dissemination and storage since the BS specifies the policies, converts them to a binary code and disseminates the binary throughout the network with a secure dissemination protocol [96].

7.1 Background and System Model

7.1.1 Network Model

We consider a multi-hop wireless sensor network, consisting of a number of sensor nodes and a base station (BS) that collects data from the network. A node is assumed to have more than one neighbor node which can monitor its behaviors. The BS is secure and has a secure mechanism to broadcast authentic messages and to disseminate code updates in the network. Sensor nodes are stationary after deployment, but routing paths may change over time, e.g., due to node failure. Once after the deployment, the BS assigns each node u a unique nodeID and a cryptographic key K_u. Each node also shares a pairwise key $K_{u,k}$ with each neighbor k and a group key K_g with all the neighbors.

7.1.2 Threat Model and Security Objectives

We consider the BS as trusted, but any other node may be malicious. We assume a majority of honest nodes in a neighborhood. The WSN maintains the standard layered architecture of protocol stack which enables typical as well as WSN specific attacks to these layers. The attacks are directed to impair the following resources: (i)

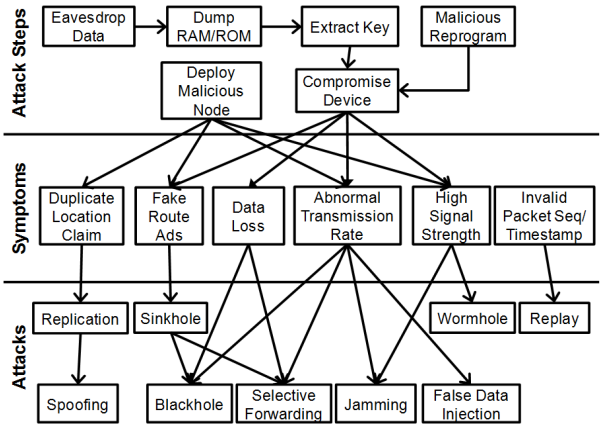


Fig. 7.1.: Attack Graph

Network, (ii) Control and data message, (iii) Sensor device resources, e.g. memory, power, etc. Below, we discuss these attacks with respect to the target resources.

Communication Network: *Jamming* disrupts a sub-network or even the entire network. Attacks at the link layer include purposely introduced collisions, resource exhaustion, and unfairness in medium access.

Messages: In a WSN, all the nodes act as routers. Hence, an attacker may spoof, alter, or replay routing messages to disrupt network traffic through creating routing loops, changing routes, attracting or repelling traffic from selected nodes, increasing latency, etc. Examples include *sinkhole*, *selective forwarding*, *blackhole*, *wormhole* attack. While these attacks can also be performed on data packets, additional attacks like false data injection, and delayed forwarding may be conducted to degrade data quality and utility.

Sensor Devices: Sensor devices come without tamper-resistant packaging, which adds the risk of physical attacks, e.g., physical capture, tampering, etc. An adversary can extract the secrets stored on captured sensors' chip and cause substantial damage by exploiting software vulnerabilities. The adversary can also replicate the captured sensors and place them into network at chosen locations (*replication* attack). Once these replicas gain the trust of others, they can launch a variety of insider attacks

described above. ID spoofing, e.g. *Sybil* attack, poses threat by enabling a malicious node to present multiple false identities to the network.

To summarize, attacks may take place in many forms but they disrupt the WSN by affecting one or more of the above resources. Thus when an anomaly or attack is detected, our objective is to issue and execute the most effective response actions in a secure manner so that the WSN suffers minimum impact on its resources and recovers from the attack.

7.1.3 Intrusion Detection System (IDS)

A number of IDSes [49, 93, 94] have been proposed specifically for WSNs that cooperatively detect intrusions. Due to the broadcast nature of wireless channels, overhearing is a natural phenomenon in WSNs. Neighboring nodes overhear transmissions from each other, even if they are not the intended recipients [97]. Utilizing this fact, Marti et al. [49] introduce the *watchdog* mechanism by which a node identifies a misbehaving neighbor node by observing the neighbor behaviors. Such a node is termed *watchdog monitor* (a.k.a monitor). Each monitor observes its neighbors, collects audit data, and then performs behavioral analysis for each of them to detect any suspicious activity. The intrusions are cooperatively detected by the monitors based on their analyses, and a set of pre-defined or adaptive inference rules. The relationships between the symptoms used by the IDSes and the various attacks are shown in Figure 7.1.

7.2 Design Overview of Kinesis

In Kinesis, each monitor hosts a distributed IDS and the Kinesis system. Through the IDS, a monitor observes neighbor behaviors, detects suspicious incidents in the neighborhood, and notifies Kinesis for automated response action. However, as we see in section 7.3.3, Kinesis depends on the IDS only for the notifications on good/bad

behaviors which is the basic functionality of an IDS. Hence, the design or any concern specific to IDS are **out of the scope** of our work.

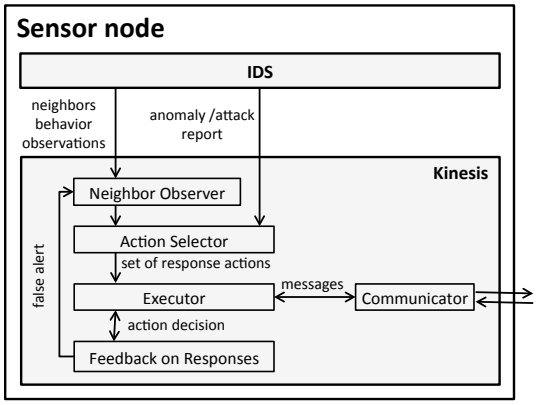


Fig. 7.2.: Overview of the Kinesis Architecture

Figure 7.2 shows the architecture of Kinesis. The background process *Neighbor Observer*, with the help of IDS observations, records recent behaviors for each monitored neighbor and periodically updates the neighbor's security status based on this history. Upon detecting an incident, the IDS reports to Kinesis the possible anomaly/attacks, suspect node(s), and alert confidence for each reported anomaly/attack. The *Action Selector* then performs the *security assessment* of the suspect node based on the *alert confidence*, the *suspect behavior history*, and the *incident impact*. Based on the security assessment, the action(s) to be executed are selected dynamically from the response policy matched on the incident. Due to the incident based approach, Kinesis can handle unknown attacks based on the anomalous behaviors they manifest.

Given a set of response action(s), the *Executor* triggers and executes the actions. A monitor competes to be the next *daemon* (i.e. one to take the response action) by setting an action timer inversely proportional to its *action effectiveness* and takes the action when the timer fires. Note that some actions, such as *log, analyze*, etc., are executed by each node independently whereas for actions, like *retransmit_data*, redundant actions by the neighbors should be minimized. In the latter case, upon hearing an action taken by a monitor, other monitors in the neighborhood stop their

action timers to refrain themselves from taking any further action for that incident. Any communication related to response actions or with the BS is handled by the *Communicator* module.

7.3 Kinesis System Details

7.3.1 State Information

Each node u maintains a list of its neighbors, $N(u)$, and link quality, $L(u, k)$, with each neighbor $k \in N(u)$. Also, u retains: **(i)** Per-neighbor *sliding window* w_k of size W to record the neighbor behavior observations. Using the behavior history, u updates the security estimation and state of the neighbors. **(ii)** An *action timer* value to indicate how long u waits before triggering the next action, if it wins the competition.

7.3.2 Response Policy Specification

For modeling response policies in Kinesis, we propose a WSN specific lightweight policy language based on the Event-Condition-Action (ECA) paradigm [95]. A response policy is defined on an **incident** (equivalent to *event* in ECA) and specifies **action**s for different **security** estimations, which combine the various *condition*s on the incident and the suspect. We adopt the ECA paradigm since it is inline with our incident-centric approach for response management and simplifies policy specification.

The response policies are specified as a set of rules, expressed with the grammar in Table 7.1. The words within quotes ' ' are static tokens and the *italics* represent functions. The main construct of the language is <rule> which defines the response policy corresponding to an attack or anomaly.

Table 7.1: Response Policy Language

```
<rules> ::= 'Begin' <rule-list> 'End'
<rule-list> ::= <rule> <rule-list> | <rule>
<rule> ::= 'on' <incident> (<condition> <action-list>)+
<incident> ::= <anomaly> | <attack>
<anomaly> ::= data_loss | data_alteration | data_replay | ...
<attack> ::= unknown | selective_forwarding | jamming | ...
<condition> ::= <condition>*|'if' <incident> 'then'
              |'if' severity(<suspect>,<incident>) <op> (<value>|<range>) 'then'
<op> ::= '<' | '>' | '<=' | '>=' | '==' | '!=' | 'IN'
<action-list> ::= <action>, <action-list> | <action>
<action> ::= <conservative-action> (<suspect>)*
              |<moderate-action> (<suspect>)*
              |<aggressive-action> (<suspect>)*
<aggressive-action> :: = revoke | reauthenticate | rekey | ...
<moderate-action> ::= retransmit_data | trigger_data_authentication | ...
<conservative-action> ::= nop | analyze | alert | ...
<suspect> ::= <digit>+ | <literal> (<literal>*<digit>*)*
<range> ::= ('['|'(') <value>–<value> (')'|']')
<value> ::= <digit> | <digit>+. <digit>+
<digit> ::= ['0'-'9']
<literal> ::= ['A'-'Z"a'-'z']
```

Through a detailed analysis of the various attacks in WSNs and corresponding remedies, we have identified a comprehensive set of response actions, listed in Table 7.3. The actions are categorized into three classes based on the severity:

(i) *Conservative*: Low severity actions that may help a monitor in more precise attack detection or in not executing erroneous responses, but cannot prevent or recover from attacks.

(ii) *Moderate*: Actions intended to preserve the WSN services under failures or attacks.

(iii) *Aggressive*: High severity responses which are executed to recover from an attack and to prevent further malicious attempts. These actions may be executed at local sensors or may require help from the BS to execute them.

An example policy for *data_alteration* incident is shown in Table 7.2. Here, $nodeID$ refers to the suspect node identifier.

Table 7.2: Response Policy Example

on 'data_alteration' if *severity*(data_alteration, nodeID) <= 0.3 then *retransmit_data* if *severity*(data_alteration, nodeID) IN (0.3,0.6] then *retransmit_data*, *trigger_route_change* if *severity*(data_alteration, nodeID) > 0.6 then *revoke* nodeID

Table 7.3: Taxonomy of Response Actions

Actions	Descriptions
CONSERVATIVE: Low Severity	
nop	No actions to take
log, *analyze*	Record auxiliary information and analyze
alert	Notify the suspicious node(s) or other neighbors/the BS about the misbehavior
MODERATE: Medium Severity	
discard_data	Prevent forwarding false data
retransmit_data	Retransmits cached data
trigger_reauthentication	Re-authenticate the suspicious node
trigger_route_change	Change route and notify others
trigger_multipath_routing	Route data through multiple paths
suspend	Temporarily block the suspect node
AGGRESSIVE: High Severity	
revoke	Black list/block the convicted node
re-program	Re-program the malicious node
re-key	Re-key the (sub) network
flood_alerts	Flood alert messages in the network

7.3.3 Policy Matching and Response Selection

Since response policies are defined specific to incidents, it is straightforward to match the policy for an incident in Kinesis. However, the action to execute is selected dynamically from the action set specified by the matched policy, based on the *security assessment* of the suspect. This strategy ensures that Kinesis takes the most effective action at any incident.

The security assessment of a node is quantified by a *Security Index (SI)*. In Kinesis, a monitor continuously updates per-neighbor security state records based on its observations of the neighbor behaviors. The SI of a neighbor is also updated on each observation. If a neighbor shows legitimate behavior, its SI is updated based on the

behavior observations only. Otherwise, if an incident is reported (i.e. a misbehavior is observed), SI is updated based on three factors:

(i) Incident Confidence: The confidence with which an incident is detected, denoted by a *Confidence Index (CI)*.

(ii) Incident Impact: A numeric value of the impact of the incident on the WSN, denoted by an *Impact Index (II)*.

(iii) Neighbor behavior history: The continuous behavior observations and security state of the neighbor, reflecting how much the monitor believes the suspect node.

In what follows, we discuss how Kinesis computes these indices and then selects the response action based on *SI*.

Confidence Index (CI)

The IDS associates a confidence value with each incident reported to indicate the likelihood of its occurrence. We utilize it to select a response action since it measures how effective the IDS is in detecting an incident and how severe the response should be. However, if the IDS does not provide an in-built confidence value, Kinesis computes CI as follows:

(i) For *Anomalies*, we consider $CI = 1$. This is reasonable since watchdog monitors can correctly identify a failure or misbehaving event [49].

(ii) For *Attacks*, CI is computed as a *false alarm rate* based on the past performance of the IDS about successfully detecting attacks. Thus, CI is computed as:

$$CI = \frac{\#\ of\ true\ attacks}{\#\ of\ attacks\ reported} \tag{7.1}$$

The details of how Kinesis gets feedback about false alerts are discussed in section 7.3.6.

Impact Index (II)

The *II* estimates the overall impact of an anomaly/attack and implies the urgency and extremity of the response action Despite extensive work on vulnerability scoring in enterprise networks [98], little attention is paid to WSNs. A few mathematical risk models for WSNs have been proposed [99], but they do not provide a complete framework considering the WSN specific practical concerns. In this work, we propose a simple mechanism to estimate the impact of an incident.

Table 7.4 lists the consequences of incidents to the WSN services. Based on the priority of the WSN, the BS assigns static scores to the impacts and configures the nodes with the incident-impact mapping and impact scores. On receiving a report of incident x, Kinesis computes the incident impact as:

$$I^k(x) = \frac{\sum_{j=0}^{n} impact_x^k[j] \times r^k[j]}{\sum_{j=0}^{n} r^k[j]} \tag{7.2}$$

where k is the type of impact, n is the total number of *k-type* impacts, $impact_x^k$ is an *n-length* array of *k-type* impacts for incident x where $impact_x^k[j] = 1$ means that the incident has *j-th* impact, and r^k is an array of impact scores associated with the *k-type* impacts. Using Eq. 7.2, Kinesis computes the *Data Impact (I^d), Network Impact (I^n), Node Impact (I^s)* of the incident and then the II as follows:

$$II(x) = \beta_d \times I^d(x) + \beta_n \times I^n(x) + \beta_s \times I^s(x) \tag{7.3}$$

where, the coefficients $\beta_d, \beta_n, \beta_s \geq 0$ are real numbers such that $\beta_d + \beta_n + \beta_s = 1$. Note that if the network administrator does not change the WSN priorities, the *Impact Index*es are **static** and need to be calculated only **once** after deployment.

Table 7.4: Possible impacts of WSN anomalies and attacks

Data Impact	Data delay, unavailability, alteration, falsification
Network Impact	Network unavailability, disruption; Path unavailability
Node Impact	Node unavailability, misbehavior, malfunction

Neighbor Behavior Observations

The neighbor behaviors help a monitor assess how vulnerable the neighbor is and how likely that it is going to make an attack. Hence, we consider the behavior observations of the suspect node while determining the severity of the response action. Usually IDSes maintain the behavior history and trust scores [100]. However to conform with IDSes without such facilities, we provide a design to record the neighbor behaviors and to utilize them in computing security score and state.

To justify the accuracy of the response action, we utilize the history of neighbor behaviors rather than the latest single behavior. Kinesis maintains a per-neighbor *sliding window* w_k of size W to keep track of the neighbor's most recent W behaviors. When the IDS notifies about a behavior of neighbor k, Kinesis pushes out the oldest behavior from w_k and stores the recent one. We consider two types of behaviors:

(i) *Service Behavior*: How trustworthy a neighbor node is in providing WSN services, e.g., in-time packet forwarding.

(ii) *IPRS Behavior*: How efficient and honest the neighbor is in taking required and desired actions.

Security Index and State Update

A monitor u computes SI for each neighbor $k \in N(u)$ on each behavior observation for k and updates the security state accordingly. A node is estimated to be in five possible states: (i) Fresh, (ii) Suspicious, (iii) Secure, (iv) Malicious, and (v) Revoked. Figure 7.3 shows the security state transition diagram. After the network deployment,

a monitor assigns to all its neighbors the *Fresh* state with $SI = 0$. For a pre-specified amount of time t_f, a neighbor is considered to be in *Fresh* state while its SI is updated on behavior observations according to Eq. 7.6. The significance of *Fresh* state is that a neighbor is given the *benefit-of-doubt* while being in this state. Although the SI of a suspect node in *Fresh* state affects the response selection, no aggressive action is taken against the node, i.e., the node is not revoked, reprogrammed, etc. After a time of t_f, the neighbor moves to *Suspicious* or *Secure* state based on its SI. A node in the *Suspicious* state moves to the *Secure* state if its SI decreases due to legitimate behaviors. On the contrary, if a node in the *Suspicious* state continues its anomalous behavior, its SI goes above a pre-defined threshold σ_2 and the node moves to the *Malicious* state. When a neighbor goes to the *Malicious* state, the monitor initiates an aggressive action against the node. A neighbor node can also be revoked anytime due to the monitor's own decision or action initiated by neighboring monitors. In this case, the monitor enlists the suspect node as *Revoked* and discards further request/data from the node.

We formulate the computation of SI of a neighbor k with two auxiliary functions $f(x)$ and $g(SI)$, where $f(x)$ computes the *severity* of an incident x and $g(SI)$ returns a coefficient based on the current SI and security state of k.

$$g(SI) = \begin{cases} 1 & ; SI \leq \sigma_1 \text{ i.e } k \text{ is } \textit{Fresh/Secure} \\ 1.5 & ; \sigma_1 \leq SI \leq \sigma_2 \text{ i.e } k \text{ is } \textit{Suspicious} \\ 2 & ; SI > \sigma_2 \text{ i.e. } k \text{ is } \textit{Malicious} \end{cases} \tag{7.4}$$

$$f(x) = \begin{cases} 0 & ; x \text{ is good behavior} \\ min(CI \times II(x) \times g(SI), 1) & ; \text{otherwise} \end{cases} \tag{7.5}$$

On each i-th behavior observation for neighbor k, its SI is computed at a monitor as

$$SI = \begin{cases} \frac{\sum_{j=1}^{i} f(w_k[j])}{i} & , \text{if } i \leq W \\ \frac{\sum_{j=1}^{W} f(w_k[j]) - f(w_k[0]))}{W} & , \text{if } i > W \end{cases} \tag{7.6}$$

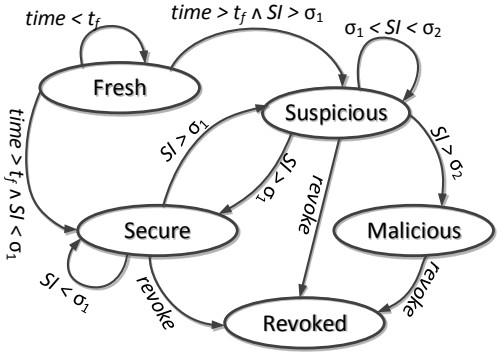


Fig. 7.3.: Security State Diagram of a Monitored Node

7.3.4 Response Computation and Optimization

If the IDS reports a single anomaly/attack corresponding to an incident, Kinesis computes the *SI*, matches the response policy and selects the *SI* based action(s) from the matched policy. When multiple possible incidents are reported, we may follow the same procedure to select the action(s) for each reported incident and compute the final action set as a union of these actions. However, each individual action set may be inclusive, overlapping, inconsistent with respect to the other sets. Moreover, before considering new action(s) for execution, we should check the on-line actions to find out the same relationships. To resolve this issue for a limited resource system, we introduce the action precedence graph.

The **Action precedence graph** (APG) is a directed graph which describes the precedence relationship between actions in terms of their effectiveness. Here, (i) each node a_i is an action, (ii) an edge $a_i \rightarrow a_j$ denotes that the parent action a_i invalidates the child action a_j, and (iii) a *black edge* $a_i \Rightarrow a_j$ denotes that a_i and a_j are contradictory actions and on conflict, a_i is executed. Thus the execution of an action a_i invalidates all of its successors, and a_j *not reachable by* a_i means that they are independent actions. Two actions a_i, a_j conflict if one can reach the other only through a path of black edges. An example APG is shown in Figure 7.4 where the *reprogram* action overrules all of its successors, {*log, analyze, alert*} are independent

of each other, and {*retransmit_data, reauthenticate_data, discard_data*} conflict. We assume that the **BS pre-configures** the nodes with all possible response actions and the precedence relationships between them.

Algorithm 3 : cors() - Computation of Optimized Response Set

Input: Response sets A = $\{a_i\}$, B = $\{b_i\}$
Output: Optimized response set O

if A = B **then**
  O ← A // A is *equivalent* to B
else if $\forall a_i, \exists b_j$, $b_j \rightarrow a_i$ **then**
  O ← B // B *covers* A
else if $\forall a_i, \forall b_i$, $a_i \Rightarrow b_j$ or Vice-versa **then**
  O ← A (or B) // A *contradicts* B
else if $\exists a_i, \exists b_j$, $a_i \rightarrow b_j$ **then**
  O ← A ∪ (A\B) // A *intersects* B
else
  O ← A ∪ B // A is *independent* to B
end if

By utilizing the APG, Algo. 3 computes the equivalence, independence, intersection, and coverage relationships between two action sets. To compute the optimized action set from n different action sets $\{A_1, A_2, \ldots, A_n\}$ (each specific to an individual incident), Kinesis runs a recursive algorithm initialized with $O_1 = A_1$ and then computing $O_i = cors(O_{i-1}, A_i)$ for $i = 2, 3, \ldots, n$.

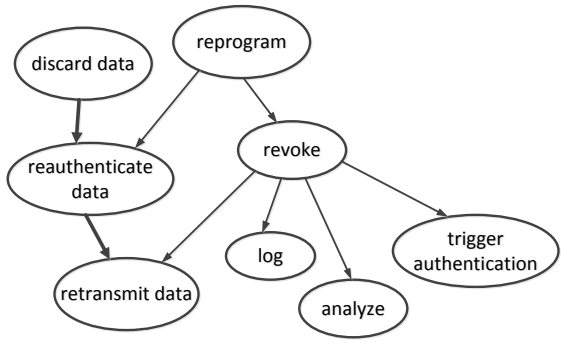


Fig. 7.4.: Example of an Attack Precedence Graph

7.3.5 Execution of a Response Action

The response action executions are fully distributed in Kinesis. The low/medium severity actions are executed by the monitors solely based on their own decisions. The high severity actions against convicted nodes require consensus among the monitors in the neighborhood. In the latter case, a selected monitor node (*daemon*) broadcasts a message asking the decisions of other monitors, performs a *majority voting* on the collected replies, and then executes the agreed upon action. Some aggressive actions, such as *reprogram, rekey*, etc. cannot be completed at the sensors. In such a scenario, the *daemon* node notifies the BS with an authenticated report and the BS then performs the action. In addition, even though some actions like *retransmit_data, alert_others*, etc. can be executed upon a monitor's own decision, they require interactions with other nodes. In all these cases, **a monitor has to initiate** the action and take over all the related responsibilities. Kinesis dynamically selects the most competent node as the *daemon* to ensure the action effectiveness and to avoid the same node doing all the job all the time.

Selection of the Daemon

A node is selected as the *daemon* via a self-organized competition among neighboring monitors. The novelty of our scheme is that we do not need any message exchange or special time synchronization among neighbors to manage the action executions. Each node in a neighborhood competes independently through a locally managed back-off timer, called *action timer*. The timer value of a node u depends on the *action effectiveness*, $AE(u)$, of the node, which is estimated locally based on: (i) neighborhood size, (ii) one-hop link qualities, and (iii) time since last action. Intuitively, if a node has more neighbors with good link qualities, it can interact with more monitors and help minimize redundant actions. Again, if the node is idle for a

long time, it should take the action to ensure load distribution in the neighborhood. Thus, the $AE(u)$ is computed as follows:

$$AE(u) \propto c_1 \times t_l + c_2 \times \sum_{\substack{k \in N(u) \\ k \in N(s)}} L(u,k) \tag{7.7}$$

Here, c_1, c_2 are real numbers, $N(u)$, $N(s)$ denote the neighbors of u and the suspect node, respectively, $L(u,k)$ is the link quality between u and the monitor k, and t_l is the *time since last action* by u. The higher the $AE(u)$, the more effective u's action is. u joins the competition to be next daemon by setting the timer value inversely proportional to $AE(u)$.

$$ActionTimer(u) \propto \frac{1}{AE(u)} \tag{7.8}$$

Thus, a node with better AE has lower back-off period and wins the *daemon* selection competition. When the action involves a transmission and a neighbor k overhears it, the node stops its running timer to avoid any redundant action for the same incident and updates its t_l and AE value. Kinesis could allow redundant actions with a goal to enhance the system reliability. For example, suppose that node u drops data packets and one of its monitors retransmits the dropped packets. If another attacker v in the data flow path drops retransmitted packets, then redundant transmissions by multiple neighbors of u may help mitigate data loss. However when v drops data, its neighboring monitors retransmit the data, which invalidates the necessity of redundant actions by u's neighbors. Our experimental results also support the design of minimizing redundant actions as we see very low data loss rate in the presence of multiple attackers.

Consensus among the monitors

To execute high severity actions, the monitors consult with each other and decide an action based on **majority voting**. After selecting a response action, the *daemon*

node broadcasts an authenticated *status_req_msg* in the neighborhood. The message contains the (i) detected attack, (ii) the suspect node, (iii) the response decision, and (iv) a Message Authentication Code (MAC) computed on the data using the group key K_g.

Upon receiving the message, each neighboring monitor replies with an authenticated *status_reply_msg*, containing the local response decision. The *daemon* node computes and broadcasts again the majority voting result. Based on the voting decision, the *daemon* may execute the agreed upon action or notify the BS with an authenticated report to trigger the action. The neighboring monitors also observe each other to check whether they abide by the voting decision and otherwise records a *bad* behavior for the misbehaving node.

7.3.6 Response Feedback

The majority voting decision gives a feedback to the monitors about their accuracy in terms of detecting an incident and selecting the actions. If the severity of the agreed upon action is lower than the locally determined action at a node, it implies a false alarm and decreases the confidence of the monitor. Every monitor node keeps the records of its false alarms and updates its *CI*. Note that we do not consider false negatives here. Response feedback may also help assess the effectiveness of the taken action for an incident. However, we do not investigate the direction in this work.

7.3.7 Secure Policy Storage and Dissemination

A naive approach to store the policies is to use a file or database, which Kinesis would read to select the response policy for an incident. Despite simplicity and incremental update facility for policies, this approach has significant drawbacks. Most of the operating systems for sensors do not support file or memory protection. So, malicious modules can manipulate the policy file/database. Also reading the

file/database on each incident results in a large number of expensive operations for resource constrained sensors.

To overcome these difficulties, Kinesis allows the BS to generate a binary from the input policy file and to disseminate the binary throughout the WSN as standard code dissemination. To implement new response actions or update response policies, the BS generates a binary of the updated Kinesis implementation and disseminates the updated binary. The binary dissemination is likely to be more expensive than an incremental policy update as in the naive approach. However, we assume that **action or policy changes are infrequent** and thus do not become a serious concern. It also eliminates the need for expensive read of flash memory at run-time. To maintain the integrity of policies, we utilize secure code dissemination protocols for WSNs [96]. Since policy dissemination is secure and a node only installs an authenticated binary at any time, the policy integrity is ensured.

7.3.8 Implementation and Configuration

We implement Kinesis in TinyOS 2.x. We adapt the Skipjack encryption based CBC-MAC implementation in TinySec [86] for TinyOS 2.x to compute a 4-byte MAC while majority voting. The implementation is lightweight since it takes 0.38 ms for Mica2 motes [86] and would take less time in TelosB platform since TelosB has higher processing capability than Mica2 mote.

The modular design and implementation of Kinesis add flexibility to the system. According to the policy language we define in Sec. 7.3, policy rules are implemented as *switch-case* based on incident. This strategy optimizes the implementation. Security state thresholds (σ_1, σ_2) are used to specify the severities in policies. To compute σ_1, σ_2, we average over all the incident impacts, measure SI with this average impact for various attack rates, and select the values based on the tolerance to attack rates. The particular set of response actions used to specify the policies are determined based on the security goals of the WSN application.

As Kinesis configuration, the WSN administrator configures the sensors with the incident-impact mappings, impact scores, and the real coefficients. The *data*, *network*, or *node* impacts of an incident do not vary across different WSNs, hence the incident-impact mappings are static. On the contrary, how severely an incident affects the WSN services may well depend on the network application. Thus, the impact scores and β coefficients, used to compute the II, should be set by the administrator according to the application requirements. However, we assume that these configuration parameters are changed infrequently over the network lifetime.

7.4 Simulation Results

In this section, we present simulation results to show the performance of Kinesis under various network settings.

7.4.1 Simulation Setup

For simulation, we use the TinyOS simulator TOSSIM. The network topologies are generated with symmetric links. As a routing protocol, we use the standard *Collection Tree Protocol* (CTP). In the experiments, we consider the following application and network layer anomalies and attacks: (i) data loss, (ii) data alteration, (iii) selective forwarding, and (iv) sinkhole attack. The policies considered for these incidents are shown in Table 7.5. To detect incidents, we implement a simple watchdog monitor based IDS in TinyOS 2.x.

To configure Kinesis, we assign equal weight to the real coefficients in Eq. 7.3, i.e., $\beta_d = 0.34$, $\beta_n = 0.33$, $\beta_s = 0.33$. The size of the per-neighbor *sliding window*, W, is set to 100. In Sec. 7.3.5, we have stated how we determine the values of σ_1, σ_2. A data source periodically sends out data every 2 seconds. In each run of simulation, the results are averaged over $3,000$ data transmissions. Unless otherwise stated, we use the above default values in simulation.

Table 7.5: Considered Response Policies

on 'data_alteration' if *severity*(data_alteration, nodeID) IN (0,0.2] then *retransmit_data* if *severity*(data_alteration, nodeID) IN (0.2,0.4] then *trigger_route_change*, *retransmit_data* if *severity*(data_alteration, nodeID) > 0.4 then *retransmit_data*, *revoke* nodeID
on 'data_loss' if *severity*(data_loss, nodeID) IN (0,0.2] then *retransmit_data* if *severity*(data_loss, nodeID) IN (0.2,0.4] then *trigger_route_change*, *retransmit_data* if *severity*(data_loss, nodeID) > 0.4 then *retransmit_data*, *revoke* nodeID
on 'selective forwarding' *retransmit_data*, *revoke* nodeID
on 'inconsistent_etx' if *severity*(inconsistent_etx, nodeID) IN (0,0.4] then *NOP* if *severity*(inconsistent_etx, nodeID) > 0.4 then *revoke* nodeID
on 'sinkhole' *revoke* nodeID

7.4.2 Performance Metrics

The metrics considered to evaluate Kinesis are:

(1) **Effectiveness:** Since our goal is to minimize the impact of data, network, etc. failure, we show the effectiveness of Kinesis from two aspects:

(i) *Data Loss Rate at the BS*: The frequency with which the BS experiences the effect of an incident i.e. the rate of reception failures at the BS. In this context, we compare the performance of our system with (i) **an attack free typical sensor** environment, and (ii) an **under-attack network** to show that Kinesis can get back the WSN into a normally operating environment, even under anomalies or attacks.

(ii) *Average Data Transmission Delay*: On average, the amount of time a packet

takes to reach the BS since its transmission by the source. Here, we compare the performance of Kinesis with an **attack free** scenario.

(2) **Optimization of Redundant Actions:** The *average number of actions taken per incident* by the monitors in a neighborhood. We also measure the *rate of redundant actions per incident* as a ratio of the number of monitors taking response actions to the number of monitors that detected the incident. They justify our *action timer* design based distributed scheme to trigger the response actions.

(3) **Load Balance:** How evenly the response action executions are distributed in the neighborhood. This is indicated by the standard deviation among the number of actions taken by the monitors in a neighborhood.

(4) **Energy Consumption:** The sum of energy usage by all the nodes when Kinesis along with an IDS is in operation.

7.4.3 Grid Network Experiments

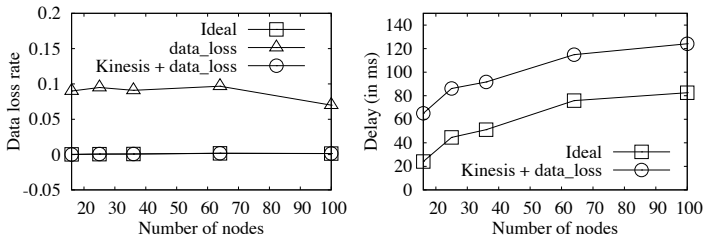
We place 16 to 100 nodes in grid topologies of dimensions from 4×4 to 10×10, respectively. The nodes are spaced 1.5 meter apart. For each network, a data source and an attacker are randomly selected and the results are averaged over 10 runs. The attack rate is set to 0.1. For concurrent attacks, we place a second attacker both in the same and different neighborhood than the first one. Both attackers are equally likely to make an attack.

Single Attack

First, we show the performance of Kinesis in case of a single incident (anomaly/attack) in the network.

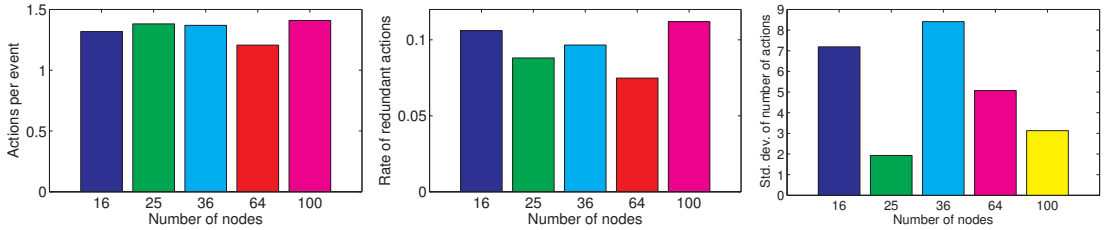
***data_loss* incident**: In this case, a node may be faulty or malicious and drops data packets intermittently instead of forwarding them to the BS. Fig. 7.5 shows the performance of Kinesis under *data_loss* incidents in WSNs of sizes from 16 to 100. As

shown in Fig. 7.5(a), Kinesis reduces the data loss rate of a network under attack from [0.073, 0.103] to $\sim$ 0.002, which is similar to the natural data loss rate ($\sim$ 0.0018) in a network without attack. It proves the effectiveness of Kinesis both in small and large networks.



(a) Data reception failure rate at the BS

(b) Average data transmission delay



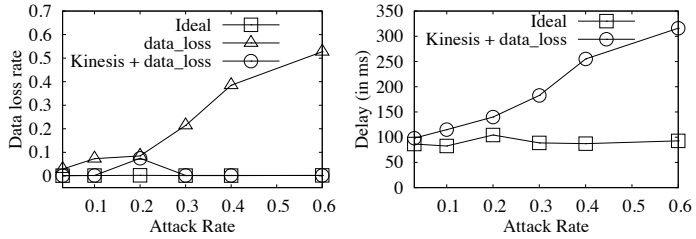
(c) Average number of actions per *data_loss* incident

(d) Rate of redundant actions per *data_loss* incident

(e) Load balance among neighbor monitors

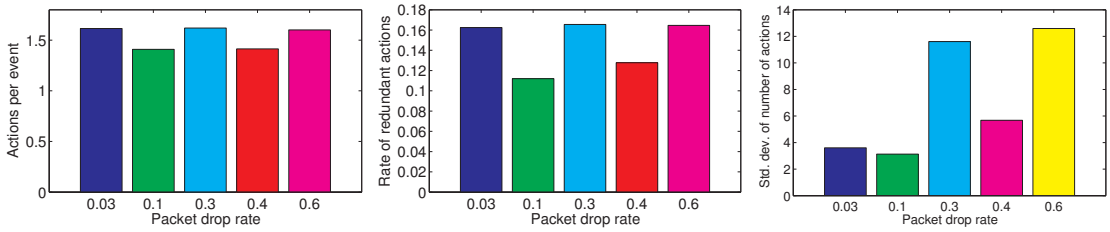
Fig. 7.5.: Kinesis Performance for *data_loss* incidents with rate 0.1 in grid networks of various sizes

Fig. 7.5(b) reveals the linearly increasing trend in average transmission latencies with network sizes. However, the average latency Kinesis adds due to action execution is almost invariant ([39.03, 41.607] ms) in different networks. The delay incurred by Kinesis is mostly due to the *action timer*. According to Eq. (7.7) and (7.8), the *action timer* value depends on the number of neighbors and the link qualities with them. In the experiments, neighborhood sizes vary from 3 to 5 in different networks and the link quality values lie in [0.8, 0.976]. The combined effect of neighborhood size and link qualities made the *action timer* values almost invariant in different networks. Thus, the increasing trend in transmission delays is mainly due to the increase in routing path length with network sizes.



(a) Data reception failure rate at the BS

(b) Average data transmission delay



(c) Average number of actions per *data_loss* incident

(d) Rate of redundant actions per *data_loss* incident

(e) Load balance among neighbor monitors

Fig. 7.6.: Kinesis Performance for *data_loss* incidents of various rates in a 10×10 grid network

Fig. 7.5(c) shows that Kinesis is not always able to take a single action per incident as in ideal case. Occasionally, it triggered as high as 1.4 actions per incident on average. However the rate of redundant actions, as shown in Fig. 7.5(d), is bounded by 0.11. The phenomena of redundant actions may occur due to two reasons:

A. Hidden node problem: The problem occurs when the monitors of the source and the attacker are not connected or weakly connected. We explain the scenario with Fig. 7.7 - a segment of the attacker's neighborhood found from a simulation topology. Node 8 is the source, 18 is the attacker and others are the watchdog monitors.

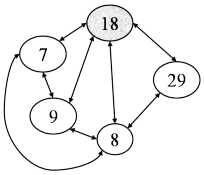


Fig. 7.7.: A segment of the attacker's neighborhood in the simulation topology

When node 18 drops a packet, all the monitors 7, 9, 29 start their action timers. When the timer in one of the nodes 7 and 9 fires and for example 7 wins, it retransmits the dropped data and 9 stops its timer upon overhearing the action. Since 29 does not possess link to either 7 or 9, it cannot overhear whoever takes the action. Being unaware of other actions on the same incident, 29 will execute the action when its timer fires. This kind of **redundancy is not a sole problem of Kinesis**, but will be a problem for any **overhearing based solutions**.

B. Action Timer Value: Action timer values at two monitors may be close when the *load balancing factor* (i.e. time since last action) is same in both of them and *link qualities* with the neighbors cannot make a big difference, and vice versa. Thus, a monitor may take redundant action if it does not get enough time to hear others' actions by the time its action timer fires.

The small standard deviation ($[1.93, 8.41]$) in the number of actions by neighboring monitors, as shown in Fig. 7.5(e), indicates the high success of Kinesis in load balancing.

To further analyze the scalability of Kinesis, we measure its performance under various attack rates in a 100-node network and show in Fig. 7.6 how well Kinesis survives, even for very high attack rates. As expected and consistent to earlier results, Kinesis counteracts the data loss attacks and gets the network back to normal operating condition. Fig. 7.6(a) shows that Kinesis reduces the data loss rate of a network under attack from [0.02, 0.52] to $\sim$0.0001, which proves its effectiveness and scalability, even under higher attack rates. Fig. 7.6(b) reveals the linearly increasing trend in average transmission latencies with higher rate attacks. Even average latencies introduced by Kinesis with varying attack rates are negligible ([12,223] ms).

Fig. 7.6(c) shows that the average number of actions per incident is $\sim$1.5. The action redundancy per incident is bounded by 0.16. However, both numbers are almost invariant with respect to attack rates. This is because the number of actions depends on the link quality among the neighbors and the differences in their action timer values. Fig. 7.6(e) shows a small standard deviation in the number of actions

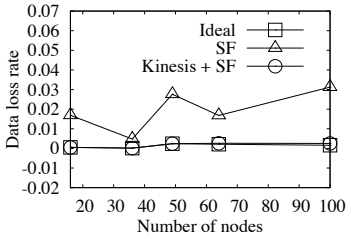
taken by the neighbors, which indicates the effectiveness of the distributed scheme of Kinesis in triggering action executions.

***data_alteration* attack**: In this attack, a malicious node selectively modifies the data value in a data packet before forwarding it to the BS. We run simulations for *data_alteration* attacks and find similar trends in the results as in *data_loss* incidents. Later on, we show the performance of Kinesis for concurrent incidents of *data_loss* + *data_alteration*, hence we do not report the graphs here.

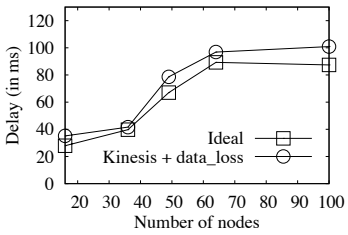
***selective_forwarding* attack**: In a selective forwarding attack, the monitor nodes initially observe data loss by the attacker and hence retransmit the dropped data. Once they detect a selective forwarding attack, the *daemon* issues a *state_req_msg* to the neighborhood. The neighboring monitors reply with their own action decision about the suspect in a *status_reply_msg*. Based on the majority voting decision from the replies, the daemon possibly issues a revocation request to the BS. The BS then disseminates a *revoke* command to the network, upon receiving which all the nodes exclude the attacker from the routing path.

Fig. 7.8 reports the performance of Kinesis under selective forwarding attacks in networks of various sizes. In a selective forwarding attack, no matter whether the attacker is revoked from the network or not, Kinesis retransmits the packet dropped by the attacker. Hence, Kinesis reduces the data loss rate of a network under attack to that of a network without attack. Fig. 7.8(a) supports the claim by showing that the natural data loss rate and the loss rate of a network under attack with Kinesis enabled are almost equal.

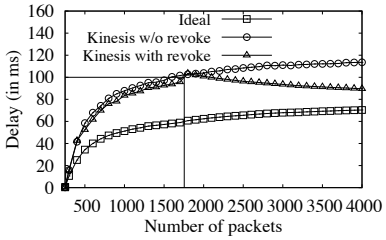
Fig. 7.8(b) shows an interesting and significantly different trend in transmission delays with Kinesis under *selective_forwarding* attack. In this case, the average transmission delays are much lower compared to that of *data_loss* incidents and quite close to the natural data transmission delays. To analyze the performance better, we show the average transmission delays over time in Fig. 7.8(c). Initially when the monitors do not detect the selective forwarding attack yet but only observe data losses, they retransmit dropped packets and thus add latencies to data transmissions. After the



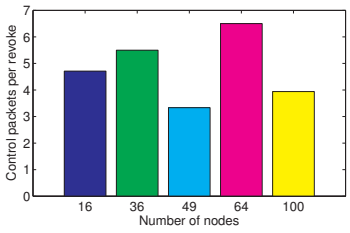
(a) Data reception failure rate at the BS



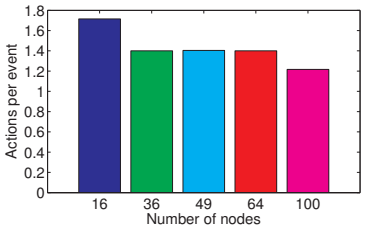
(b) Average data transmission delay



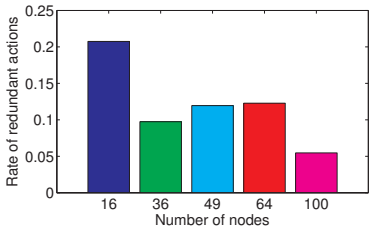
(c) Avg. transmission delays over packets



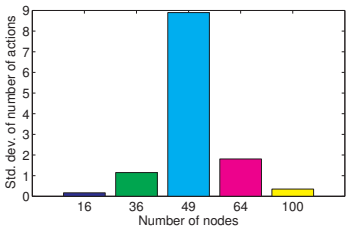
(d) Average number of control message exchanges in a neighborhood on *revoke*



(e) Average number of actions per incident



(f) Rate of redundant actions per incident



(g) Load balance between neighbor monitors

Fig. 7.8.: Kinesis Performance for *selective_forwarding* (SF) attacks in grid networks of various sizes

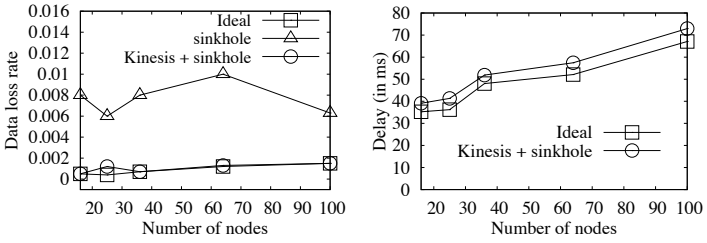
revocation of the attacker at packet 1755, there is no attack and hence no delay is incurred due to response execution.

Fig. 7.8(d) shows the average number of control messages (*state_req_msg* + *status_reply_msg*) exchanged in a neighborhood for majority voting. The *state_req_msg* is of 27 bytes and *state_reply_msg* is of 35 bytes. The number of control messages per majority voting is $\leq$6.2 packets. However, it is proportional to the neighborhood size and thus does not vary with network sizes unless the number of neighbors varies.

Fig. 7.8(e) - 7.8(g) show that the average number of actions, action redundancy and load distribution measurements are consistent with the earlier experiments and can be explained in a similar way.

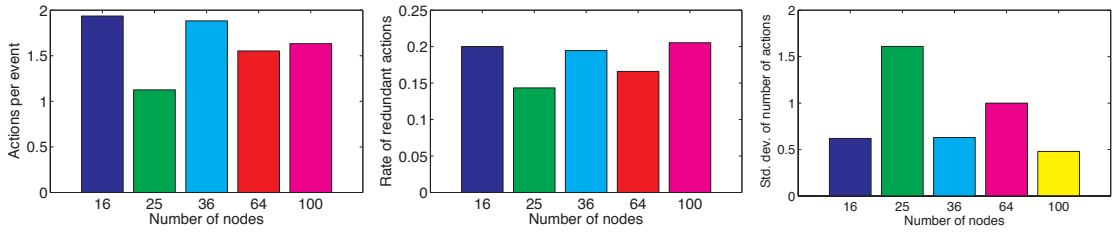
For the *selective_forwarding* attacks, the monitors always agreed on the decision to *revoke* the suspect node. The average time to perform the majority voting and execute the decided action is $\sim$ 96.4 ms, most of which is contributed by the action timer value.

***sinkhole* attack**: For *sinkhole* attack, we modify the CTP protocol to enable the attacker advertising low cost routing path through it. Once the attacker attracts all the data in the neighborhood, it drops data at a rate of 0.2.



(a) Data reception failure rate at the BS

(b) Average data transmission delay



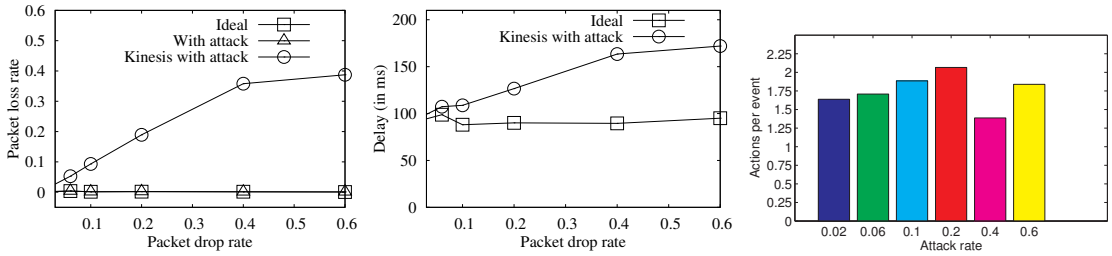
(c) Average number of actions per incident

(d) Rate of redundant actions per incident

(e) Load balance between neighbor monitors

Fig. 7.9.: Kinesis performance for *sinkhole* attack

In Kinesis, a monitor suspects a potential *sinkhole* attack upon hearing an inconsistent path cost advertisement, which results in an update in SI for the suspect but NOP as a response. During the subsequent packet drop observations, the monitors retransmit the dropped data and eventually revoke the malicious node when



(a) Data reception failure rate at the BS (b) Average data transmission delay (c) Average number of actions per incident

Fig. 7.10.: Kinesis performance for *data_loss* + *data_alteration* incidents with various rates in a 10×10 grid network

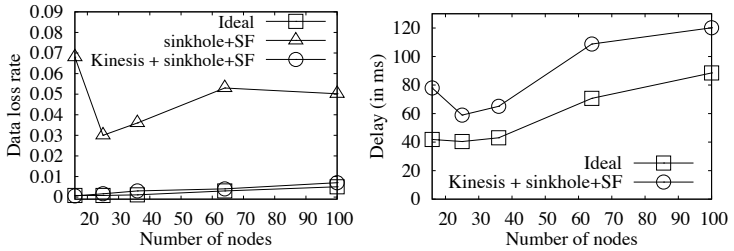
the attack is confirmed. Fig. 7.9(a) shows that Kinesis reduces the data loss rate to ~ 0.0015. At the same time, it keeps the transmission delays closer to natural latency, as shown in Fig. 7.9(b), due to the quick revocation of the attacker node. Note that the *sinkhole* attack often created routing loop causing as high as 3.5% data loss. By revoking the attacker, Kinesis made the WSN stable again.

Concurrent Attacks

In concurrent attack experiments, we consider two cases, (i) two simultaneous but independent attackers, and (ii) two colluding attackers.

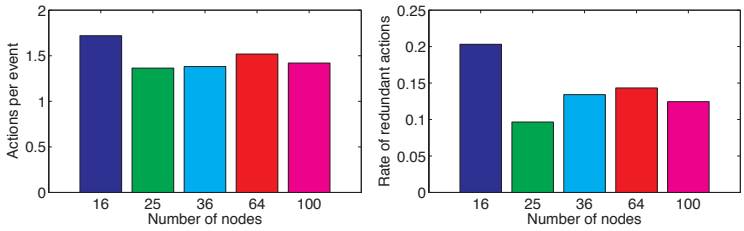
(i) In case of two concurrent but independent attackers, we consider an attacker causing *data_loss* and the other conducting *data_alteration* at various rates in a 10×10 grid WSN. As we see in Fig. 7.10, Kinesis shows behaviors consistent with the single attack scenario, in all the aspects. Thus, Kinesis is effective under concurrent and high rate attackers.

(ii) Next, we consider two colluding attackers performing *sinkhole* and *selective_forwarding* (SF) attack. When the *sinkhole* attacker is revoked, routing path changes enable data routing through the *SF* attacker which then drops data at a rate of 0.5, and vice versa. Fig. 7.11 shows how Kinesis performs in such scenario.



(a) Data reception failure rate at the BS

(b) Average data transmission delay



(c) Average number of actions per incident

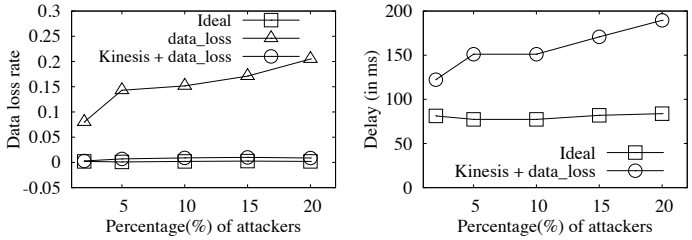
(d) Rate of redundant actions per incident

Fig. 7.11.: Kinesis performance for *sinkhole + SF* attacks in grid networks of various sizes

The irregularity occurring when the number of nodes is equal to 16 is due to the temporary routing instability after revocations.

Varying the Number of Attackers

To further show the scalability of Kinesis, we present its performance in a multi-attacker environment. Here, we consider *data_loss* incident and vary the number of attackers from 2% to as high as 20% of the total nodes in a 100-node network. Fig. 7.12(a) shows that Kinesis still keeps the data loss rate lower than 0.009. Due to Kinesis, the average transmission latencies vary within [122.33,189.46] ms, as shown in Fig. 7.12(b). The results are consistent to earlier results.



(a) Data loss rate at the BS (b) Average data transmission delay

Fig. 7.12.: Kinesis Performance for *data_loss* for various % of attackers (with rate 0.1) in a 10×10 grid network.

Energy Consumption

We measure and compare the aggregated energy consumption of the WSN under various incidents while Kinesis (as well as the IDS) is in operation and in an attack-free scenario where Kinesis is not deployed (baseline). For energy measurement, we consider *MICAz* platform and use PowerTOSSIM z [101] plugin. Due to the scalability limit of PowerTOSSIM z, we consider a 6×6 grid WSN with one source and one attacker.

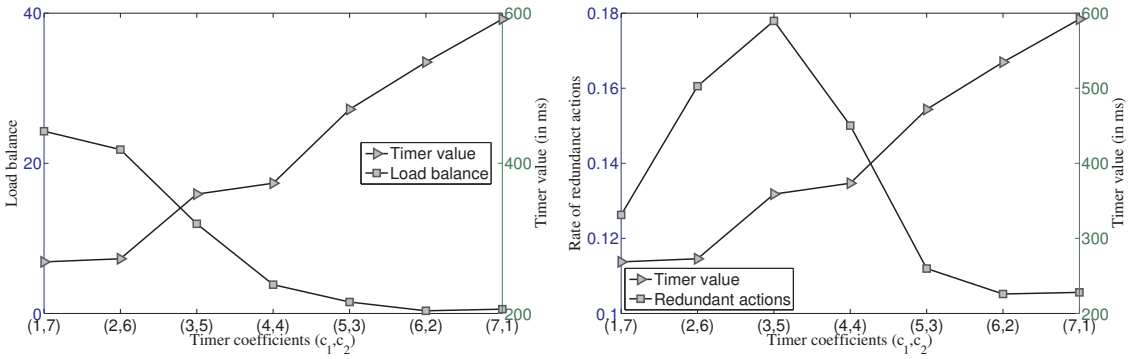
In a Kinesis enabled system, overhearing does not incur overhead since it is inherent in WSNs. In TinyOS, the radio stack requires a node to receive and process all the packets transmitted in a neighborhood to understand whether the packet is destined to it or not. TinyOS also exposes the Receiver [102] interface that allows one to perform actions upon overhearing a message in transit. Thus, the only energy overhead imposed to the nodes is due to the IDS and Kinesis operations. The results reported in Table 7.6 show that Kinesis system incurs only a maximum of 0.06% energy overhead.

Table 7.6: Aggregated energy cost of the WSN without and with Kinesis + IDS

—	Baseline	Kinesis		
		data_loss	SF	sinkhole
Energy usage ($\times 10^7$ mJ)	1.320488	1.320482	1.321356	1.320480

Action Timer Configuration

Action timer design is crucial in Kinesis and its configuration impacts the performance with respect to redundant actions and load balance. Hence, we vary the coefficient factors (c_1, c_2) in Eq. 7.7 and see the impact of timer values on Kinesis performance. Since c_1, c_2 are weight coefficients, $c_1 + c_2$ should be bounded to optimize the timer value. If $c_1 + c_2$ is too small, the action timer fires frequently which increases the number of actions. If $c_1 + c_2$ is too big, the latency increases. In our experiment, we fixed $c_1 + c_2$ to 8. Fig. 7.13(a) shows that the optimum values of (c_1, c_2) in terms of load balance are near (3,5). In Fig. 7.13(b) the optimum value is after (4.5, 3.5). To optimize both the action redundancy and load balance, (c_1, c_2) should be selected onwards (4.5, 3.5).



(a) Load balance vs Action timer

(b) Action redundancy vs Action timer

Fig. 7.13.: Coefficient configuration for Action Timer

7.5 Testbed Evaluation

We ported the Kinesis implementation to the TelosB platform and placed battery-powered TelosB motes in an indoor environment. Our motes have a 8 MHz TI MSP430 microcontroller, 2.4 GHz radio, 10 KB RAM, and 48 KB ROM. We ran experiments for *data_loss*, *selective_forwarding*, and *sinkhole* attacks and use the same metrics as

in simulation for performance evaluation. The results of the experiments are averaged over 1500 packets.

7.5.1 Experimental Setup

We build a 6×6 grid WSN, consisting of 36 TelosB motes, in a $160 \times 200\,cm^2$ indoor environment. Fig. 7.14 shows the coordinates of the network nodes, labeled from 2 to 37. Node 10 is the source, which sends data every 1 second. We controlled

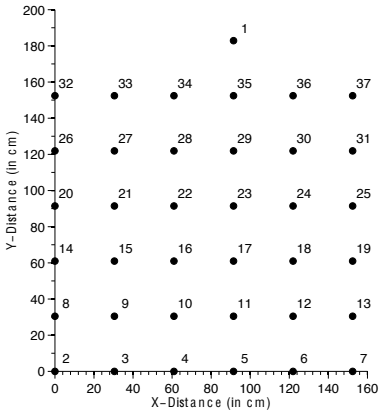


Fig. 7.14.: Node placement in an indoor 6×6 grid WSN

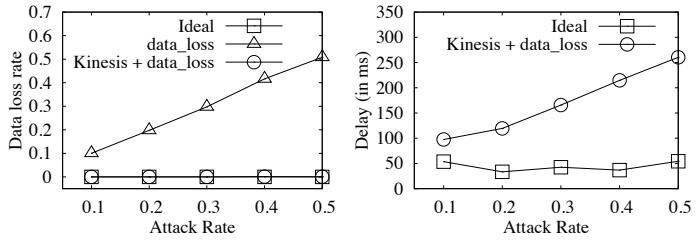
the transmission power of motes to ensure multi-hop communication. A special mote, labeled as node 1, is set to the root node. For performance analysis, the root collects statistics on the number of data and action packets transmitted, number of actions per incident, transmission delays and passes these data to a laptop through serial forwarder.

7.5.2 Kinesis Performance

Below, we present the testbed performance of Kinesis for *data_loss*, *selective_forwarding*, and *sinkhole* incidents. For the first two incidents, we set node 16 as the attacker.

***data_loss* incident**: We evaluate the performance of Kinesis under various rates of *data_loss* incidents. Fig. 7.15(a) shows that Kinesis reduces the data loss rate of

the WSN under attack from [0.1, 0.51] to ≤ 0.0015, similar to the natural data loss rate. The average transmission delays when Kinesis is in operation vary within [97.5, 260.4] ms, as shown in Fig. 7.15(b). Kinesis triggers on average [1.28, 1.97] actions per incident. Thus the testbed performance of Kinesis is consistent to that in simulations and justifies the simulation results.



(a) Data loss rate at the BS (b) Average data transmission delay

Fig. 7.15.: Testbed performance of Kinesis for *data_loss* incidents of various rates in a 6×6 grid WSN.

***selective_forwarding* (SF) attack**: Table 7.7 summarizes the performance of Kinesis under SF attack, where the attacker drops packets at a rate of 0.4. The attacker is revoked at packet 604. Hence there is no attack and Kinesis actions afterwards, which keeps the average transmission delays much lower compared to that of *data_loss* incidents.

Table 7.7: Testbed performance of Kinesis on SF attack

	Ideal	SF	Kinesis + SF
Data loss rate	0.0008	0.064	0.0008
Avg. transmission delay (ms)	32.89	N/A	61.11
Avg. actions per incident	N/A	N/A	1.6875

***sinkhole* attack**: We conduct two sets of *sinkhole* attack experiments, setting two different nodes 21 and 22 as attackers. Once an attacker is able to attract surrounding data packets, it drops data at a rate of 0.2. The performance results of Kinesis are presented in Table 7.8.

Table 7.8: Testbed performance of Kinesis on *sinkhole*

	Ideal		sinkhole		Kinesis + sinkhole	
	Exp 1	Exp 2	Exp 1	Exp 2	Exp 1	Exp 2
Data loss rate	0.011	0.086	0.015	0.20	0.011	0.086
Avg. transmission delay (ms)	71.17	113.03	N/A	N/A	75.27	177.36
Avg. actions per incident	N/A	N/A	N/A	N/A	1	1.604

The results of experiment 1 are quite similar to simulation results. The attacker is revoked at packet 158. It is to be noted that the sinkhole attack in this case created routing loops due to low cost path advertisements by the attacker and thus resulted in data loss. However, Kinesis took a quick response action to revoke the attacker, which brought back the routing stability and helped keep the data loss rate minimal.

In experiment 2, we see comparatively higher data reception failure and transmission delay at the BS. This is due to the routing instability created when Kinesis revoked the attacker at packet 376. Consequently, some packets were lost while a few others needed unusually longer time to reach the BS until a stable routing path was re-established.

Energy consumption: Due to the difficulty of measuring energy directly on the sensor hardware [103], we adopt the energy model proposed by Polastre et al. [104] to estimate the energy cost in testbed. The energy cost of a node is estimated as a sum of energy usage due to sensing, transmission, and reception. The energy for a type of operation is computed by multiplying the battery voltage with the current draw and time spent (according to the TelosB datasheet) for the operation. The aggregated energy cost of the WSN in case of baseline, *Kinesis+data_loss*, *Kinesis+SF* are 4232.86, 4447.44, 4467.46 mJ, respectively. Thus, Kinesis (along with the IDS) incurs a maximum of 5.5% energy overhead.

7.6 Discussion

In this section, we analyze the characteristics of Kinesis from various aspects and discuss possible improvements.

False Positive: A false positive occurs if an attack is detected when there is none. In Kinesis, when a monitor observes an anomalous activity by a neighbor node, it does not immediately conclude that this is an attack. It continues to observe the node while taking appropriate response action(s) (conservative or moderate) to mitigate disruption to WSN services. Thus, as long as Kinesis can keep the WSN functional (e.g., send data successfully to the BS) and minimize the disruption, our security goal is achieved.

When the security estimation for the monitored node exceeds a threshold, specified in the matched response policy, the monitor may go for an aggressive action. The decision of an aggressive action, however, requires consensus among a minimum number of neighboring monitors. It is highly unlikely that all of these monitors will detect a false attack.

Hidden Node Problem: As discussed in Sec. 5.3.1, two disconnected monitors monitoring the same node may take the same but redundant actions. Since our security goal is to minimize WSN disruptions (e.g., data, network failure), occasional redundant actions will not cause problem with respect to this goal.

If these two monitors, however, have different link qualities with respect to the monitored node, i.e., one of the monitors has good connectivity, and the other does not have so good connectivity and misses some activities of the node under consideration, then the monitors may have different security evaluations for the monitored node at a time instant and may decide to execute different actions. This may raise consistency issues. If both actions are non-conflicting and are conservative or moderate (e.g., *log* and *retransmit_data*), they will not cause any inconsistency. The only side effect would be some additional resource (computation, transmission, etc.) usage. The same holds true when the two actions are conflicting but conservative or moderate,

for example, *retransmit_data* and *trigger_route_change*. However, if at least one of the actions is aggressive, the monitor node first sends a message to the BS, consisting of the decision and authenticated agreements from at least a minimum numbers of neighboring monitors. After verifying the agreements, the BS also checks whether the action causes any redundancy or inconsistency. Only then, the BS executes the aggressive action. Thus, the network nodes need not do anything to handle the consistency issues, if there is any.

From the discussion, it is clear that even if nodes take different actions, Kinesis still achieves its security goals. It is to be mentioned that in our experiments, we observe redundant yet same actions, but no inconsistent actions. To address the hidden node problem in future, we will investigate a scheme utilizing 2-hop topology knowledge. We will also explore existing solutions in the context of distributed systems.

Majority Voting: A set of colluding attackers may mislead the majority voting to decide on a wrong response action. If the attacker(s), replying with a low severity action, can affect the voting decision to be an action of lower severity than those reported by honest monitors, it also makes them detecting a false positive and lowering the monitor confidence. These attacks, however, will not succeed as we assume a majority of honest nodes in a neighborhood.

A solution to deal with such attacks on majority voting is to set higher weights on the local decisions of more trustworthy nodes. An alternative approach is to use complementary methods with Kinesis to detect such attacks. For example, we may use our previous work on lightweight provenance techniques that enables the BS to detect a data dropping attack and identify the misbehaving node, based on the data provenance, i.e., the identities of the source and routing nodes that processed or forwarded the data towards the BS [105]. In case a number of colluding attackers falsely report an honest node as a data dropping attacker and ask the BS to take aggressive action (e.g., *revoke*) against it, the BS will find an inconsistency between the reports of the colluding attackers and the data provenance. The reason is that, based on the provenance information, the BS will not be able to detect any data

dropping attacks by the honest node. The BS may then conclude about a highly probable collusion attack and respond accordingly. As part of future work, we will extend our current implementation by integrating such an approach.

Jamming: An attacker may interrupt Kinesis operation by jamming a part of the network and disabling data communication. We implemented a jamming attack following the method described in [106]. This jamming attack, however, results in no more than 20-30% data loss, which is the same as the data loss in *data_loss* incidents. As part of future work, we will implement stronger jammers able to block the channel completely and will investigate whether Kinesis, in response, can send a top priority message to the BS through the border nodes.

8. FUTURE RESEARCH DIRECTIONS

8.1 Recovery of Local Fault and Security Attacks at Sensor Nodes

While sensor nodes, through neighbor monitoring, can detect anomalous actions by a compromised neighbor node and take recovery actions, it is an interesting question whether a sensor node itself can predict operational failure or security attacks on it and be proactive. A node may monitor its residual energy and be conservative in energy usage by reducing the frequency of sensor operations and may inform the BS or other nodes when it is about to die out of power. The node may also monitor the local packet queue, memory write, etc. node properties and take response actions to prevent failures and attacks, such as network congestion, node compromise through over-the-air reprogramming, etc.

8.2 Secure Provenance Management in Untrusted Systems

With the emergence of distributed services (e.g. cloud computing, web services, etc.), sensitive user data may move around multiple organizations and be processed and stored outside the owner's control. The problem is exacerbated by the security incidents involving sensitive data leakage, unauthorized access, and integrity violations which are a daily occurrence. In this context, recording all the operations performed on a data (i.e. data provenance) is valuable to check the compliance of the data to quality and trust. The questions to investigate include the (1) reliable provenance collection in untrusted systems, and (2) secure provenance communication as the data flows through various systems.

Recently, the notion of secure and self-contained data has been introduced [107] in the context of building a deployable data protection system. This data object,

referred to as *Security Aware Object* (SAO), provides protection to sensitive data by containing five key components: (1) authentication and authorization tools; (2) self-enforcement policy engine; (3) security policies in executable form; (4) secure connection manager; and (5) protected data. Utilizing the SAO capabilities, we provide a conceptual architecture of the provenance framework in untrusted systems in Fig. 8.1.

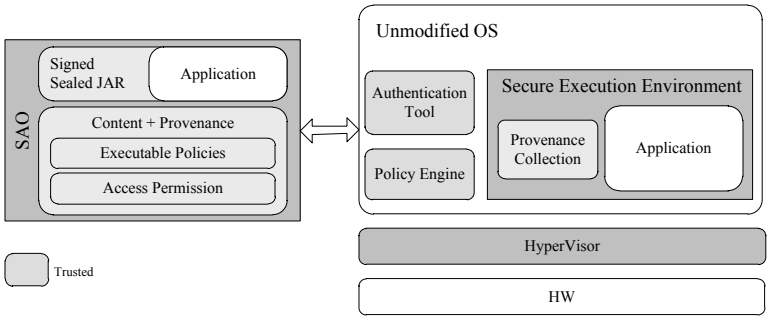


Fig. 8.1.: Conceptual Architecture of Provenance Framework in Untrusted Systems

Every data object is encapsulated by a SAO containing the protected data and provenance, access control policies on them, the trusted/untrusted application to perform operations on the data. A secure execution environment (SEE) in an untrusted environment ensures the secure execution of untrusted applications and secure provenance capture. The authentication tool and policy engine guarantee authorized access to data and provenance. The most significant challenge implied by the conceptual architecture is how to collect provenance at a fine granularity yet with high performance. Design and implementation of the SEE is the core challenge to this work. Regarding the provenance communication, the focus should be on low-overhead provenance update and encapsulation on every data access.

9. CONCLUSION

In this thesis, we propose an extensive and inter-operable provenance model that can encapsulate the provenance of data objects with various semantics and granularity. We then explore the secure provenance collection, communication and usage for various distributed systems and investigate the utility of the proposed provenance model in these systems. We present the preliminary design of a low-overhead file provenance system with an application to the provenance infrastructure for virtualized environments. The system supports the automatic collection and management of file provenance metadata, characterized by our provenance model. We then investigate secure provenance communication in streaming environment and propose two secure provenance schemes focusing on WSNs. The basic provenance scheme is extended in order to detect packet dropping adversaries on the data flow path over a period of time. We also consider the issue of attack recovery and present the design of an extensive incident response and prevention system for WSNs.

LIST OF REFERENCES

LIST OF REFERENCES

[1] "Data dictionary for preservation metadata," May 2005. Online at `http://www.oclc.org/research/projects/pmwg/premis-final.pdf`.

[2] P. Buneman, S. Khanna, and W.-c. Tan, "Why and where: A characterization of data provenance," *ICDT*, vol. 1973, pp. 316–330, 2001.

[3] H. Lim, Y. Moon, and E. Bertino, "Provenance-based trustworthiness assessment in sensor networks," in *Workshop on Data Management for Sensor Networks*, pp. 2–7, 2010.

[4] C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu, "An approach to evaluate data trustworthiness based on data provenance," in *Proc. of SDM*, pp. 82–98, 2008.

[5] X. Wang and P. Mohapatra, "Provenance based information trustworthiness evaluation in multi-hop networks," in *Proc. of IEEE GLOBECOM*, 2010.

[6] T. Dumitras and I. Neamtiu, "Experimental challenges in cyber security: A story of provenance and lineage for malware," in *CSET*, 2011.

[7] C. Sar and P. Cao, "Lineage file system," January 2005. Online at `http://crypto.stanford.edu/cao/lineage.html`.

[8] S. Jones, C. Strong, D. D. E. Long, and E. L. Miller, "Tracking emigrant data via transient provenance," June 2011.

[9] I. Foster, J. Vckler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proc. of the Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 37–46, 2002.

[10] G. Janée, J. Mathena, and J. Frew, "A data model and architecture for long-term preservation," in *Proc. of the conference on Digital libraries*, pp. 134–144, 2008.

[11] J. Frew, D. Metzger, and P. Slaughter, "Automatic capture and reconstruction of computational provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 485–496, April 2008.

[12] K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proc. of the USENIX Annual Technical Conference*, 2006.

[13] I. Cox and M. Miller, "Electronic watermarking: the first 50 years," in *IEEE Workshop on Multimedia Signal Processing*, pp. 225 –230, 2001.

[14] R. C. Dixon, *Spread Spectrum Systems: With Commercial Applications*. New York, USA: John Wiley & Sons, Inc., 3rd ed., 1994.

[15] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," in *Proc. of the Conference on File and Storage Technologies (FAST)*, pp. 1–14, 2009.

[16] J. Zhao, C. Goble, R. Stevens, and S. Bechhofer, "Semantically linking and browsing provenance logs for e-science," *Semantics of a Networked World Semantics For Grid Databases*, pp. 158–176, 2004.

[17] B. Plale, D. Gannon, D. Reed, K. Droegemeier, B. Wilhelmson, and M. Ramamurthy, "Towards dynamically adaptive weather analysis and forecasting," in *Proc. of ICCS workshop on Dynamic Data Driven Applications*, pp. 624–631, 2005.

[18] S. Cohen, S. Cohen-boulakia, and S. Davidson, "Towards a model of provenance and user views in scientific workflows," in *Proc. of Data Integration in the Life Sciences*, pp. 264–279, 2006.

[19] P. Groth, S. Miles, and L. Moreau, "PReServ: Provenance recording for services," *Translator*, 2005.

[20] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han, "An access control language for a general provenance model," in *Proc. of the VLDB Workshop on Secure Data Management*, SDM, pp. 68–88, 2009.

[21] A. Kashyap and A. Kashyap, "File system extensibility and reliability using an in-kernel database," Tech. Rep. FSL-04-06, Master's Thesis, Stony Brook University, 2004.

[22] R. Spillane, R. Sears, C. Yalamanchili, S. Gaikwad, M. Chinni, and E. Zadok, "Story book: An efficient extensible provenance framework," in *Proc. of the Usenix workshop on the theory and practice of provenance (TAPP)*, 2009.

[23] D. Morozhnikov, "Fuse: Filesystem in userspace," 2006. `http://fuse.sourceforge.net/`.

[24] R. Sears and E. Brewer, "Stasis: flexible transactional storage," in *Proc. of the Symposium on Operating Systems Design and Implementation*, pp. 29–44, 2006.

[25] M. Olson, K. Bostic, and M. Seltzer, "Berkeley db," in *Proc. of the USENIX Annual Technical Conference*, pp. 43–53, 1999.

[26] A. Ramachandran, K. Bhandankar, M. Tariq, and N. Feamster, "Packets with provenance," Tech. Rep. GT-CS-08-02, Georgia Tech, 2008.

[27] W. Zhou, M. Sherr, T. Tao, X. Li, B. Loo, and Y. Mao, "Efficient querying and maintenance of network provenance at internet-scale," in *Proc. of ACM SIGMOD*, pp. 615–626, 2010.

[28] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. Loo, and M. Sherr, "Secure network provenance," in *Proc. of ACM SOSP*, pp. 295–310, 2011.

[29] A. Syalim, T. Nishide, and K. Sakurai, "Preserving integrity and confidentiality of a directed acyclic graph model of provenance," in *Proc. of the Working Conf. on Data and Applications Security and Privacy*, pp. 311–318, 2010.

[30] N. Vijayakumar and B. Plale, "Towards low overhead provenance tracking in near real-time stream filtering," in *Proc. of the Intl. Conf. on Provenance and Annotation of Data (IPAW)*, pp. 46–54, 2006.

[31] S. Chong, C. Skalka, and J. A. Vaughan, "Self-identifying sensor data," in *Proc. of IPSN*, pp. 82–93, 2010.

[32] S. Cabuk, "IP covert timing channels: Design and detection," in *Proc. of ACM Conf. on Computer and Communications Security (CCS)*, pp. 178–187, 2004.

[33] A. Houmansadr, N. Kiyavash, and N. Borisov, "Multi-flow attack resistant watermarks for network flows," in *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1497–1500, 2009.

[34] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proc. of ACM Conf. on Computer and Communications Security (CCS)*, pp. 20–29, 2003.

[35] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proc. of the IEEE Symposium on Security and Privacy (SP)*, pp. 116 –130, 2007.

[36] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *Proc. of the USENIX Conference on Security Symposium*, pp. 307–320, 2008.

[37] P. Peng, P. Ning, and D. S. Reeves, "On the secrecy of timing-based active watermarking trace-back techniques," in *Proc. of the IEEE Symposium on Security and Privacy (SP)*, pp. 334–349, 2006.

[38] T. Wolf, "Data path credentials for high-performance capabilities-based networks.," in *Proc. of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems.*, pp. 129–130, 2008.

[39] R. Laufer, P. Velloso, D. Cunha, I. Moraes, M. Bicudo, M. Moreira, and O. Duarte, "Towards stateless single-packet ip traceback," in *Proc. of IEEE LCN*, pp. 548–555, 2007.

[40] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander, "Lipsin: line speed publish/subscribe inter-networking," in *Proc. of ACM SIGCOMM*, pp. 195–206, 2009.

[41] A. Ghani and P. Nikander, "Secure in-packet bloom filter forwarding on the netfpga," in *Proc. of the European NetFPGA Developers Workshop*, 2010.

[42] S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *IEEE International Conference on Communications*, vol. 10, pp. 3201–3205, 2001.

[43] Y. Ming Lu and V. W. S. Wong, "An energy-efficient multipath routing protocol for wireless sensor networks: Research articles," *International Journal of Communication Systems*, vol. 20, pp. 747–766, July 2007.

[44] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *IEEE INFOCOM*, pp. 1976–1986, 2003.

[45] C. Basile, Z. Kalbarczyk, and R. Iyer, "Neutralization of errors and attacks in wireless ad hoc networks," in *International Conference on Dependable Systems and Networks (DSN*, pp. 518–527, 2005.

[46] X. Zhang, A. Jain, and A. Perrig, "Packet-dropping adversary identification for data plane security," in *Proc. of the ACM SIGCOMM Intl. Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, 2008.

[47] B. Carbunar, I. Ioannidis, and C. Nita-Rotaru, "Janus: Towards robust and malicious resilient routing in hybrid wireless networks," in *ACM Workshop on Wireless Security (WiSe)*, pp. 11–20, 2004.

[48] Y. Zhang and W. Lee, "Intrusion Detection in Wireless Ad-hoc Networks," in *International Conference on Mobile Computing and Networking (MobiCom)*, pp. 275–283, 2000.

[49] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," in *International Conference on Mobile Computing and Networking (MobiCom)*, pp. 255–265, 2000.

[50] S. Buchegger and J.-Y. Boudec, "Performance Analysis of the CONFIDANT Protocol," in *ACM International Symposium on Mobile Ad Hoc Networking (MobiHoc)*, pp. 226–236, 2002.

[51] J. Ma, S. Zhang, Y. Zhong, and X. Tong, "SAID: A Self-adaptive Intrusion Detection System in Wireless Sensor Networks," in *International Conference on Information Security Applications*, pp. 60–73, 2007.

[52] M.-Y. Hsieh, Y.-M. Huang, and H.-C. Chao, "Adaptive Security Design with Malicious Node Detection in Cluster-based Sensor Networks," *Computer Communications*, vol. 30, no. 11-12, pp. 2385–2400, 2007.

[53] M. Younis, N. Krajewski, and O. Farrag, "Adaptive Security Provision for Increased Energy Efficiency in Wireless Sensor Networks," in *IEEE Conference on Local Computer Networks*, pp. 999–1005, 2009.

[54] A. Taddeo, L. Micconi, and A. Ferrante, "Gradual Adaptation of Security for Sensor Networks," in *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pp. 1–9, 2010.

[55] M. Asim, H. Mokhtar, and M. Merabti, "A self-managing fault management mechanism for wireless sensor networks," *CoRR*, vol. abs/1011.5072, 2010.

[56] S. Krishnamurthy, G. Thamilarasu, and C. Bauckhage, "MALADY: A Machine Learning-Based Autonomous Decision-Making System for Sensor Networks," in *International Conference on Computational Science and Engineering - Volume 02*, pp. 93–100, 2009.

[57] M. Mamun, A. Kabir, M. Hossen, and R. Khan, "Policy based intrusion detection and response system in hierarchical WSN architecture," *CoRR*, vol. abs/1209.1678, 2012.

[58] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pp. 18–38, 2001.

[59] W3C, "A p3p preference exchange language 1.0 (appel1.0)." http://www.w3.org/TR/P3P-preferences/, 2002.

[60] OASIS, "Oasis extensible access control markup language (xacml)," 2005.

[61] H. Garcia-Molina, "Elections in a Distributed Computing System," *IEEE Transactions on Computers*, vol. C-31, pp. 48–59, Jan 1982.

[62] S. Vasudevan, J. Kurose, and D. Towsley, "Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks," in *IEEE International Conference on Network Protocols (ICNP)*, pp. 350–360, 2004.

[63] A. Abbasi and M. Younis, "A Survey on Clustering Algorithms for Wireless Sensor Networks," *Computer Communications*, vol. 30, no. 14-15, pp. 2826–2841, 2007.

[64] A. Woodruff and M. Stonebraker, "Supporting fine-grained data lineage in a database visualization environment," in *Proc. of the International Conference on Data Engineering (ICDE)*, pp. 91–102, 1997.

[65] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Record*, vol. 34, pp. 31–36, 2005.

[66] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, "An architecture for provenance systems," *Contract*, no. D3.1.1, pp. 1–5, 2006.

[67] M. et al., "The open provenance model core specification (v1.1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743 – 756, 2011.

[68] S. Huettel, A. Song, and G. McCarthy, "Functional magnetic resonance imaging," 2004.

[69] E. Zadok and I. Badulescu, "A stackable file system interface for linux," Tech. Rep. CUCS-021-98, Columbia University, 1998.

[70] U. Braun, S. Garfinkel, D. A. Holland, K.-K. Muniswamy-Reddy, and M. I. Seltzer, "Issues in automatic provenance collection," in *Proc. of the International Conference on Provenance and Annotation of Data (IPAW)*, pp. 171–183, 2006.

[71] K.-K. Muniswamy-Reddy, U. Braun, D. A. Holland, P. Macko, D. Maclean, D. Margo, M. Seltzer, and R. Smogor, "Layering in provenance systems," in *Proc. of the USENIX Annual Technical Conference*, pp. 10–10, 2009.

[72] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-fi: Collecting high-fidelity whole-system provenance," in *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pp. 259–268, 2012.

[73] T. Zanussi, K. Yaghmour, R. Wisniewski, R. Moore, and M. Dagenais, "relayfs: An efficient unified approach for transmitting data from kernel to user space," in *In Proceedings of the Ottawa Linux Symposium 2003*, pp. 494–507, 2003.

[74] K.-K. Muniswamy-Reddy and D. A. Holland, "Causality-based versioning," in *Proc. of the Conference on File and Storage Technologies (FAST)*, pp. 15–28, 2009.

[75] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," in *Proc. of the Conf. on File and Storage Technologies (FAST)*, pp. 1–14, 2009.

[76] S. Madden, J. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, Dec. 2002.

[77] K. Dasgupta, K. Kalpakis, and P. Namjoshi, "An efficient clustering based heuristic for data gathering and aggregation in sensor networks," in *Proc. of Wireless Communications and Networking Conference*, pp. 1948–1953, 2003.

[78] H. Lim, Y. Moon, and E. Bertino, "Provenance-based trustworthiness assessment in sensor networks," in *Proc. of Data Management for Sensor Networks*, pp. 2–7, 2010.

[79] S. Sultana, E. Bertino, and M. Shehab, "A provenance based mechanism to identify malicious packet dropping adversaries in sensor networks," in *Proc. of ICDCS Workshops*, pp. 332–338, 2011.

[80] X. Luo, J. Zhang, R. Perdisci, and W. Lee, "On the secrecy of spread-spectrum flow watermarks," in *Proc. of the European Conference on Research in Computer Security (ESORICS)*, pp. 232–248, 2010.

[81] V. Berk, A. Giani, and G. Cybenko, "Detection of covert channel encoding in network packet delays," tech. rep., Darmouth College, 2005.

[82] T. G. Roosta, *Attacks and Defenses of Ubiquitous Sensor Networks*. PhD thesis, University of California, Berkeley, 2008.

[83] C. Rothenberg, C. Macapuna, M. Magalhaes, F. Verdi, and A. Wiesmaier, "In-packet bloom filters: Design and networking applications," *Computer Networks*, vol. 55, no. 6, pp. 1364 – 1378, 2011.

[84] M. Garofalakis, J. Hellerstein, and P. Maniatis, "Proof sketches: Verifiable in-netwok aggregation," in *Proc. of ICDE*, pp. 84–89, 2007.

[85] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. of IPSN*, pp. 245–256, 2008.

[86] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 162–175, 2004.

[87] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422–426, 1970.

[88] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," in *Proc. of INFOCOM*, pp. 839–850, 2004.

[89] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire tinyos applications," in *Proc. of the Intl. Conf. on Embedded networked sensor systems*, pp. 126–137, 2003.

[90] E. Perla, A. Catháin, R. Carbajo, M. Huggard, and C. M. Goldrick, "PowerTOSSIM z: realistic energy modelling for wireless sensor network environments," in *Proc. of the workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, pp. 35–42, 2008.

[91] J. Ko, C. Lu, M. Srivastava, J. Stankovic, A. Terzis, and M. Welsh, "Wireless Sensor Networks for Healthcare," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1947–1960, 2010.

[92] W. Alexander, "Barnaby Jack Could Hack Your Pacemaker and Make Your Heart Explode." http://www.vice.com/en_ca/read/i-worked-out-how-to-remotely-weaponise-a-pacemaker, June 2013.

[93] I. Krontiris, T. Giannetsos, and T. Dimitriou, "LIDeA: A Distributed Lightweight Intrusion Detection Architecture for Sensor Networks," in *International Conference on Security and Privacy in Communication Networks (SecureComm)*, pp. 20:1–20:10, 2008.

[94] Y. Ponomarchuk and D.-W. Seo, "Intrusion Detection Based on Traffic Analysis in Wireless Sensor Networks," in *Annual Wireless and Optical Communications Conference*, pp. 1–7, 2010.

[95] U. Dayal, *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publishers Inc., 1994.

[96] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks," in *International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 445–456, 2008.

[97] K. Daabaj, M. Dixon, and T. Koziniec, "Traffic Eavesdropping Based Scheme to Deliver Time-Sensitive Data in Sensor Networks," in *IEEE International Performance Computing and Communications Conference (IPCCC)*, pp. 302–308, 2010.

[98] P. Mell, K. Scarfone, and S. Romanosky, *CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0*, 2007.

[99] R. Falcon, A. Nayak, and R. Abielmona, "An Evolving Risk Management Framework for Wireless Sensor Networks," in *Conf. on Computational Intelligence for Measurement Systems and Applications*, 2011.

[100] A. Hasswa, M. Zulkernine, and H. Hassanein, "Routeguard: An Intrusion Detection and Response System for Mobile Ad Hoc Networks," in *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob)*, pp. 336–343, 2005.

[101] E. Perla, A. O. Catháin, R. S. Carbajo, M. Huggard, and C. Mc Goldrick, "PowerTOSSIM Z: Realistic Energy Modelling for Wireless Sensor Network Environments," in *ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, pp. 35–42, 2008.

[102] P. Levis, "Collection." http://www.tinyos.net/tinyos-2.x/doc/html/tep119.html.

[103] V. Sundaram, P. Eugster, and X. Zhang, "Prius: Generic Hybrid Trace Compression for Wireless Sensor Networks," in *nternational Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 183–196, 2012.

[104] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 95–107, 2004.

[105] S. Sultana, G. Ghinita, E. Bertino, and M. Shehab, "A lightweight secure scheme for detecting provenance forgery and packet drop attacks in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2014.

[106] A. Chakraborty and P. Banala, "An Experimental Study of Jamming IEEE 802.15.4 compliant Sensor Networks (Progress Tracking)."

[107] A. Squicciarini, G. Petracca, and E. Bertino, "Adaptive data management for self-protecting objects in distributed systems," in *International Conference on Network and Service Management, 2012*, CNSM 2012, (Las Vegas, NV, USA).

VITA

VITA

Salmin Sultana was born in a small beautiful green country, named Bangladesh. She used to live in the capital of the country, Dhaka, and completed her high school there in 2001. Afterwards, she went to the best engineering university of Bangladesh, namely Bangladesh University of Engineering and Technology (BUET), and received the degree of Bachelor of Science in Computer Science and Engineering in June, 2007. In the following year, she was employed as a member of Research and Development at Commlink Info Tech Ltd., Dhaka. In August, 2008, she started the PhD study in Computer Engineering at Purdue University. She all join the Intel Labs, OR as a security researcher soon after the graduation.