CERIAS Tech Report 2013-12

Kinesis: A Security Incident Response and Prevention System for Wireless Sensor Networks

by Salmin Sultana, Daniele Midi, Elisa Bertino
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Kinesis: A Security Incident Response and Prevention System for Wireless Sensor Networks

Salmin Sultana#, Daniele Midi†, Elisa Bertino†
School of ECE, † *Dept. of Computer Science, Purdue University*
{ssultana, dmidi, bertino}@purdue.edu

***Abstract*—Due to resource constraints, unattended operating environment, and communication phenomena, Wireless Sensor Networks (WSNs) are susceptible to operational failures and security attacks. However, WSNs must be able to continuously provide their services despite anomalies or attacks and to effectively recover from attacks. In this paper, we propose Kinesis - the first systematic approach to a security incident response and prevention system for WSNs. We take a declarative approach to support the specification of the response policies, based on which Kinesis selects the response actions. The system is distributed in nature, dynamic in actions depending on the context, quick and effective in response, and secure. We implement Kinesis in TinyOS. Testbed experiments and extensive TOSSIM simulations show that the system successfully counteracts anomalies/attacks and behaves consistently under various attack scenarios and rates.**

I. Introduction

Large-scale WSNs are being deployed to enable economically viable solutions for numerous application domains, such as cyber-physical infrastructures, power grids, wireless health, environmental monitoring, etc. WSNs for healthcare have emerged in recent years to provide continuous and unobtrusive services to diverse applications, ranging from emergency response [5], real-time patient monitoring [3], in-hospital communication, elderly care [17], etc. With the evolution of the Internet-of-Things (IoT), the recent trend is to augment physical devices with sensing, computing and communication capabilities, integrate them into the ecosystems, and make use of the collective effect of networked smart things to create smart environments.

However, the WSN applications impose stringent requirements on end-to-end system reliability, trustworthy data delivery, and service availability. The problem is further exacerbated by security attacks, where an attacker may exploit the resource constraints of the sensor devices, the unreliable nature of low power wireless communications, and also the communication medium. By exploiting these vulnerabilities, it is possible for an attacker to falsify context, modify access rights, mount denial-of-service attacks, and, in general, disrupt the system operation [8]. This can result in a wide area blackout, a patient receiving the wrong treatment, or worse, facing a life risk. Critical life devices, like insulin pump and pacemaker, have already been hacked remotely by exploiting their insecure wireless communications [1], which demonstrate the possibility of catastrophic attacks on healthcare WSNs with multi-hop wireless communication infrastructure. Hence, WSNs must be able to continuously provide their services despite failures or attacks and to effectively recover from these attacks.

Over the recent years, a number of Intrusion Detection Systems (IDSes) [10], [15], [13] have been proposed specifically for WSNs, which cooperatively detect intrusions and report possible attacks to a central authority. However, when dealing with attacks and failures, it is not sufficient to detect them; one has to react as soon as possible. Today, however, IDSes 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, distributed systems, etc., cannot be directly used in WSNs due to their significant differences in terms of operation, resources, and communication. In this paper, we focus on a systematic approach to design an Incident Response and Prevention System (IRPS) with particular concerns to WSN specifics. The unique nature of sensor environment imposes a set of challenges to the response system solution:

(i) The IRPS must be able to keep the WSN functional over time and be able to recover from attacks without significant interruption.

(ii) Instead of heavy interactions with a centralized system, the IRPS should use local and **cooperative** strategies. However, in the context of IRPS, distributed schemes may raise issues related to 1) triggering the action executions and optimizing redundant actions, 2) proper load distributions among the nodes in a neighborhood.

(iii) The IRPS should respond in **real-time**, yet apply the most effective action for each incident. The response policies should be specified in a way that it does not incur too much overhead while selecting the appropriate response actions.

(iv) The IRPS system should be **lightweight** in terms of computational cost, and resource usage.

Addressing the requirements discussed above, we design and propose *Kinesis* - a rule based distributed incident response and prevention system 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 possibly prevent the attacks. According to the design, every sensor in a WSN is a watchdog monitor [13] 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. Being notified an incident, Kinesis matches the appropriate response policy from the set of response policies specified by the base station (BS)

according to the policy language we define for WSNs. A response policy is defined on an incident and specifies different actions corresponding to different severity levels. The severity of an incident is estimated based on (i) the incident detection confidence, (ii) the security status of the suspect nodes, and (iii) the attack impact and 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. 1) and a rigorous set of response actions (Table II).

Kinesis is truly distributed in terms of triggering action executions since the node that will take the action in a neighborhood is selected via a self-organizing competition by an *action timer*. Thus, Kinesis does not require any message exchanges due to response action synchronization and has no communication overhead. The action timer value is locally estimated based on: (i) neighborhood size, (ii) link quality, (iii) time since last action. It reflects the effectiveness of a node and ensures load distribution among the neighbors. The distributed nature of Kinesis also adds security value to it. Even if a node is compromised, other legitimate nodes in the neighborhood can continue with the Kinesis functionalities. Kinesis is also secure in terms of response 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 [7].

Contributions: Our contributions include:

- The first systematically designed incident response and prevention system for WSNs.
- A declarative approach to define the response policies in a simple and extensible way, considering the resource constraints of sensors.
- A framework for selecting the most appropriate response action depending on the impact of the anomaly/attack and history of the suspect node.
- A simple yet robust mechanism to synchronize action executions in a neighborhood without any communication overhead. A local per-node *action timer* based design to manage the actions by a node while minimizing redundant actions and ensuring load distributions.
- A fine-grained analysis scheme to precisely detect the type of attack in order to enhance the execution performance of the response engine in case of more than one possible attacks.
- An implementation of Kinesis in TinyOS. Testbed experiments and extensive simulations that demonstrate the effectiveness of Kinesis in counteracting various attacks and making the WSN operate like in any attack-free environment. The system shows consistent behaviors under various attack scenarios and rates.

The rest of the paper is organized as follows: Sec. II briefly discusses the WSN attacks and IDSes. Sec. III presents the design overview of Kinesis and we discuss all the design details in Sec. IV. The simulation results are reported in Sec. V. Sec. VI presents the performance of Kinesis in a real testbed. Sec. VII discusses the state of the art. Sec. VIII discusses future works and concludes.

II. Background and System Model

A. 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 BS is assumed to be secure and to have 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.

The BS assigns each node u a unique nodeID and a cryptographic key K_u for message encryption in order to protect confidentiality and privacy. The sensor node also shares a pairwise key $K_{u,k}$ with each neighbor k and a group key K_g with all the neighboring nodes. A node can monitor the activities of its neighbors and locally detect a misbehavior, anomaly or intrusion in the neighborhood. The neighboring nodes can also cooperate for more accurate intrusion detection or critical decision making.

B. Threat Model and Security Objectives

We assume that the BS is trusted, but any other arbitrary node may be malicious. WSNs maintain the standard layered architecture of protocol stack which enables typical as well as WSN specific attacks to these layers. These attacks can be directed to exploit or impair the following resources: (i) Communication network, (ii) Control and data messages, (iii) Sensor device resources such as, memory, power, etc. Below, we categorize and discuss the attacks from the perspective of the target resources.

Communication Network: WSNs maintain the standard layered architecture of protocol stack which enables typical as well as WSN specific attacks to each of these layers. *Jamming* can disrupt a portion of the network or even the entire network. Attacks at the link layer include purposely introduced collisions, resource exhaustion, and unfairness in case of medium access. The attacker may attempt to transmit data simultaneously with a legitimate node, leading towards a collision and data loss at the receiver. Repeated collisions can be introduced by the attacker to cause resource exhaustion.

Messages: In a sensor network, all the nodes act as routers. Hence, an attacker may spoof, alter, or replay routing messages in order to disrupt network traffic through creating routing loops, modifying source routes, attracting or repelling traffic from selected nodes, increasing end-to-end delay, etc. For example, in a *sinkhole* attack a compromised node forges routing messages to attract traffic from all the neighboring nodes to pass through. A malicious insider may also *selectively forward* certain messages and drop others. A specific form of this attack is the *black hole* attack where a node drops all of its received messages instead of forwarding them. Even without compromising a node, an attacker can tunnel the messages to another part of the network through a low-latency link and then replay them. This kind of attack is referred to as *wormhole* attack. Integrity attacks, spoofing, replay, selective forwarding attacks can be also performed on data packets. Besides, there may be false data injection, delayed forwarding, etc., which are directed to degrade data quality and utility.

Sensor Device: Sensor devices come without any tamper-resistant packaging, hence add the risk of physical attacks, e.g., physical capture, tampering, etc. An adversary can easily extract all the secrets stored on captured sensors' chip and cause substantial damage by exploiting software vulnerabilities. The adversary can also produce a large number of replicas of the captured sensor with its keys and place them into network at chosen locations. This attack is named as *replication* attack. Once these replicas gain the trust of others, they can launch a variety of insider attacks described above. ID spoofing such as, *Sybil* attack also 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. Hence, to keep the WSN functional no matter what, there should be effective mechanisms to detect failures/attacks on these resources and to safeguard them through proper response actions. In this context, our objective is to achieve the following security properties:

- Once an anomaly or attack is detected, appropriate response actions should be executed in order to continue the WSN services as well as to effectively recover from the attacks. Since the severity of a failure/attack depends on how the incident is affecting the infrastructure, the importance of the asset under attack, impact analyses, and speculations, the response system should incorporate these key features into decision making while issuing response actions. In other words, our objective is to prevent data loss and communication failure despite failures and attacks.
- The dissemination, update, storage, and execution of response policies should be secure.

C. Intrusion Detection Systems

A number of IDSes have been proposed specifically for WSNs that cooperatively detect intrusions. Marti et al. [13] introduce the *watchdog* mechanism where a node identifies a misbehaving neighbor node by observing the neighbor behaviors. Such a node is termed *watchdog monitor* (a.k.a monitor) in the literature. Since sensor nodes are characterized by resource constraints, short transmission range, vulnerabilities, and frequent failures, watchdog based *node cooperation* has been adopted in IDSes for sensor systems. 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 monitor nodes based on their analyses, and a set of pre-defined or adaptive inference rules. The feature space, i.e. the attributes monitored to detect anomalies may include packet arrival rate, transmission ratio, data integrity, etc. The relationships between the features used by these IDSes and the various attacks are shown in Figure 1.

However, the IDSes mostly generate alerts on attacks to a centralized authority, which leaves the most important concern of recovering the incident still unsolved.

III. Design Overview of Kinesis

To bridge the gap, we propose *Kinesis* - a security solution for incident response and prevention for WSNs. According

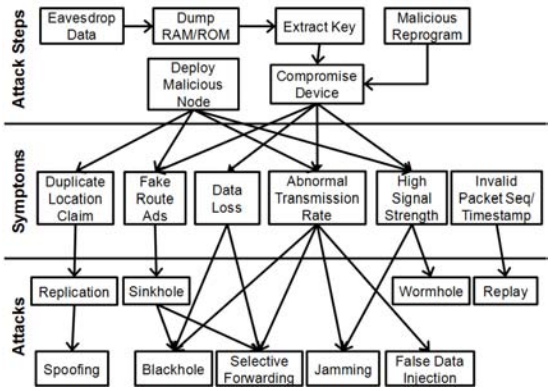


Fig. 1. Attack Graph

to the design of Kinesis, each monitor hosts a distributed IDS and the Kinesis system. Through the IDS, the monitor observes neighbor behaviors, detects suspicious events (anomaly/attack) in the neighborhood, and notifies Kinesis for automated response action. However, as we see in section IV-C, Kinesis depends on 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.

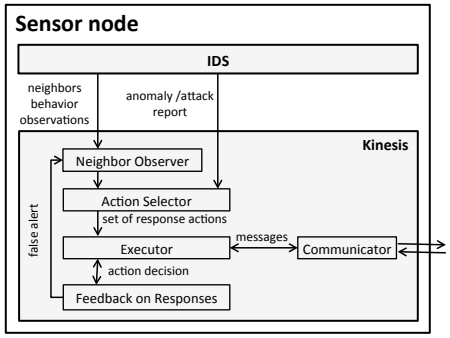


Fig. 2. Design Overview of Kinesis

Figure 2 shows the modular architecture of Kinesis. The *Neighbor Observer* is a background process that, with the help of IDS observations, keeps a record of the recent behaviors for each monitored neighbor and periodically updates the node'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* module then estimates the severity for each anomaly/attack based on the *alert confidence*, the *security state of the suspect*, and the *attack impact*. Depending on the severity measure, the particular set of action(s) to be executed are selected dynamically from the response policy matched on the incident. We propose a high-level language for the specification of response policies particularly for WSNs, which is simple yet robust and extensive and makes it easy to specify the policies for WSNs and to match a policy on each incident.

Given a set of response action(s), the *Executor* component triggers and executes the actions. A monitor competes to be the next *demon* (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 action timer fires. It is to be mentioned that some actions such as, LOG, ANALYZE, etc. are to be executed by each node independently whereas for actions, like RETRANSMIT_DATA. the 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 in order to refrain themselves from taking any further actions for that incident. Any communication related to response actions as well as the communication interface with the BS is taken care of by the *Communicator* module.

IV. KINESIS SYSTEM DETAILS

This section presents the detailed design of *Kinesis*.

A. State Information

Each node u maintains a list of its neighbors, $N(u)$, and link quality, $L(u, k)$, with every neighboring node, $k \in N(u)$. Apart from that, node u maintains:

(i) Per-neighbor *sliding window of size W* to keep the history of neighbor behaviors. Using these behavior observations, the node also maintains security state and level of trust to the neighbors.

(ii) The *action timer* value which indicates how long a node u waits before triggering the next action, if it wins the competition.

B. Response Policy Specification

The resource constrained nature of sensor devices makes it challenging to utilize the typical response policy languages used in general purpose networks, database systems, and other domains. In order to be scalable and deployable, the response policies for WSNs should be simple, lightweight yet comprehensive so that they can successfully serve the purpose. In Kinesis, we design a response policy language specific for use in WSNs. The response policies are specified as a set of rules, which can be expressed with the grammar in Table I. Each policy is specified on an incident and contains different actions applicable to various contexts of the attack and the suspect. The words within quotes ' ' are static tokens and the *italics* represent functions.

<rule>: This construct defines a response policy corresponding to an attack or anomaly and the context. The various constructs in a rule are as follows:

<anomaly>: This clause specifies data and network failures due to natural errors or malicious attempts. Examples include data loss, data alteration, transmission delay, etc.

<attack>: This clause specifies an attack detected by the IDS.

<condition>: This clause specifies the conditions to be used to select the set of responses. When the condition is evaluated, a function *severity* is called to assess the threat of the attack and then conditions are generated.

<action-list>: This clause specifies the response actions to deploy. An action is taken w.r.t a <suspect> node. Based on the severity measure, three classes of actions may be executed:

TABLE I. 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']
```

TABLE II. 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	Send cached data in case data loss or modification at other node
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

(i) *Conservative Actions*: These are low severity actions that may enable logging, fine-grained analysis on incidents, alerting suspicious node(s)/monitor(s)/others, etc. Though these actions can help in identifying attacks more precisely or restraining a watchdog monitor from deploying erroneous responses, they cannot proactively prevent or recover from the intrusions.

(ii) *Moderate Actions*: These actions are intended to maintain the continuity of data and network service under failures or attacks. Examples may include *discard_data* to stop forwarding false data, *retransmit_data* in case of packet dropping or modification attack, etc.

(iii) *Aggressive Actions*: These are high severity responses and are executed to recover from an attack and to prevent further malicious attempts. Such actions may initiate recovery by reprogramming or revoking the malicious node(s), rekeying, re-authenticating a subnetwork, etc., sometimes even before the attack occurs. These actions can be executed at local sensors or may require help from the BS to execute them.

Studying the various attacks in WSNs and corresponding remedies, we have come up with a rigorous set of response actions, which are listed in Table II.

Table III shows an example of response policy for *data_alteration* incident where $nodeID$ is the ID of the

suspect node.

TABLE III. 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

C. Policy Matching and Response Selection

Since response policies in Kinesis are specified for particular incidents, it is quite straightforward to match the response policy specific to an incident, once reported by the IDS. However the action to be executed is selected dynamically from the action set specified by the matched policy, depending on the *impact* of the incident and the *security assessment* of the suspect node. Such a strategy is adopted to ensure that Kinesis takes the most effective action at any incident.

According to the design of Kinesis, a node monitors its neighbors and continuously updates per-neighbor security state records, reflecting the neighbor behavior observations. The security assessment of a neighbor node is quantified by a numeric, referred to as *Security Index (SI)*, and is updated on each behavior observation. Whenever an incident is reported (i.e. a misbehavior is observed), SI is updated based on three factors:

(i) Incident Confidence: The confidence with which the monitor node detects the incident, denoted by a *Confidence Index (CI)*;
(ii) Impact of the Incident: A numerical representation of the impact of the incident on the sensor network, denoted by an *Impact Index (II)*;
(iii) Neighbor behavior observations: The continuous behavioral observation of the neighbor, reflecting how much the monitor node believes the suspect node.

However, when the neighbor behaves correctly, SI only depends on behavior observations. In what follows, we discuss in details how Kinesis computes these indices and then selects the appropriate response action based on the security index.

1) Confidence Index: The IDS associates a confidence value with each anomaly or attack reported to indicate how likely the incident has occurred. We utilize it for selecting a response action since it measures how effective the IDS is in identifying an incident and how severe the response action 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 [13].
(ii) For *Attacks*: In this case, CI is computed as a *false alarm rate* based on the past performance of the IDS about successfully detecting attacks. CI is computed using the following equation

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

The details about how Kinesis gets feedback about false alerts are discussed in section IV-F.

2) Impact Index: The *II* estimates the overall impact of an attack and indicates the urgency and extremity of the action to uproot the cause of that attack. Despite extensive work on vulnerability scoring in enterprise networks [14], little attention is paid to WSNs. Few works [4] present mathematical risk modeling and analysis for WSNs, but they do not provide a complete framework considering the WSN specific attacks and practical concerns. In this work, we propose a simple mechanism to estimate the impact of an anomaly or attack.

Table IV summarizes the consequences of attacks to the WSN services. The BS maintains a list of possible incidents and their corresponding impacts. Based on the priority of the WSN and risk assessment, the BS assigns static scores to these impacts and pre-configures the nodes with the incident-impact mapping and impact scores. Upon receiving a report of incident x, Kinesis computes the impact of the incident as follows:

$$I^k(x) = \frac{\sum_{j=0}^{n} impact_x^k[j] \times r^k[j]}{\sum_{j=0}^{n} r^k[j]} \quad (1)$$

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. 1, Kinesis computes the *Data Impact (I^d), Network Impact (I^n), Node Impact (I^s)* of the incident and then the II as a linear combination of these three impacts:

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

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

TABLE IV. 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

3) Neighbor Behavior Observations: The neighbor behaviors give a perception to a watchdog monitor about how vulnerable the neighbor is and how likely it is going to make an attack. Hence, we consider the behavior observations of the suspect node as a factor to determine the intensity of the response action. The history of behaviors and trust scores are usually maintained by IDSes [6]. However to conform with IDSes without such facilities, we provide a design to record the neighbor behaviors from various aspects and to compute trust scores, security score/state based on the behavior history. Here, Kinesis depends on IDS only for the notifications on good/bad behaviors which is the basic functionality of an IDS.

To justify the accuracy of the response action, we depend on the history of neighbor behaviors rather than the most recent single behavior. Kinesis maintains per-neighbor *sliding window* of size W to keep track of the neighbor's most recent W behaviors. When a good/bad behavior notification about that neighbor is received from IDS, the sliding window pushes out the oldest behavior and stores the recent one. The monitor

nodes keeps watching two types of neighbor behaviors:
(i) *Service Behavior*: How sincere a neighbor node is in providing sensor network services, such as in-time packet forwarding, transmitting no false data, etc.
(ii) *IPRS Behavior*: The efficiency/honesty of the neighbor in taking response actions i.e. how often the neighbor is taking required and desired actions.

4) Security State Update: A monitor u computes SI for each neighbor k $N(u)$ 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 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 whereas its SI is updated on behavior observations according to Eq. 2. 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 action selection, no aggressive action is taken against the node i.e. the node will not be revoked, reprogrammed, etc. After time t_f, the neighbor transitions to either *Suspicious* or *Secure* state based on its SI. A node in the *Suspicious* state can move to the *Secure* state if it behaves well for long and lowers its SI, and vice-versa. On the other hand, if a node in the *Suspicious* state continues illegitimate behavior, its SI goes above a pre-defined threshold $_2$ and moves to the *Malicious* state. Whenever a neighbor node goes to *Malicious* state, the monitor initiates an aggressive action against the node. However, a neighbor can be revoked from the network anytime due to the monitor's own decision or action initiated by neighboring monitors. In such a case, the monitor enlists the suspect node as *Revoked* and discards any further request/data from this 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.

$$g(SI) = \begin{cases} 1 & ; SI \quad _1 \text{ i.e } k \text{ is } \textit{Fresh/Secure} \\ 1\ 5 & ; \ _1 \quad SI \quad _2 \text{ i.e } k \text{ is } \textit{Suspicious} \\ 2 & ; SI > \ _2 \text{ i.e. } k \text{ is } \textit{Malicious} \end{cases}$$

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

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

$$SI = \begin{cases} \frac{\sum_{j=1}^{i} f(w_k[j])}{i} & , \text{if } i \quad W \\ \frac{\sum_{j=1}^{W} f(w_k[j]) \quad f(w_k[0]))}{W} & , \text{if } i > W \end{cases} \tag{2}$$

D. Response Set Computation

If the IDS reports a single anomaly/attack on an incident, Kinesis computes the *SI*, matches the response policy and selects the *SI* dependent response set from the matched policy, as stated above. In case of multiple attacks reported on an incident, we can follow the same procedure to compute the response set for each attack and then compute the final response set by finding the union of these sets. However,

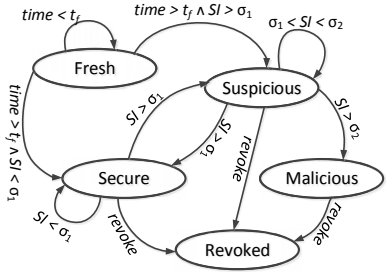


Fig. 3. Security State Diagram of a Monitored Node

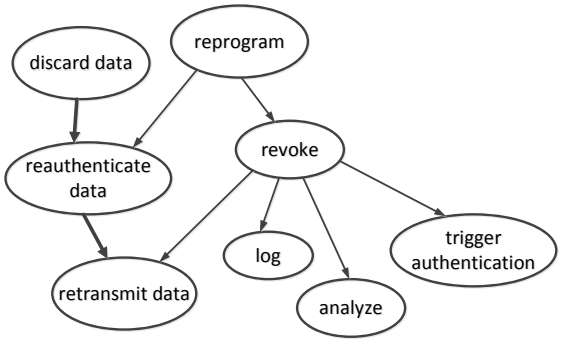


Fig. 4. Example of an Attack Precedence Graph

the individual response sets may be inclusive, overlapping or inconsistent with respect to other sets. In addition, before considering a new action set to be executed, we should check it with the on-line responses to find out the same relationships. As a key to resolve this issue for a limited resource system, we introduce the concept of action precedence graph.

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 a_j denotes that the parent action a_i invalidates the child action a_j, and (iii) we define a new type of edge called *black edge* where a_i a_j indicates that a_i and a_j are contradictory actions and in case of 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 of APG is shown in Figure 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. For computational efficiency, the nodes store this APG graph in an adjacency matrix representation and identify the connectivity between nodes once at the very beginning.

By utilizing the APG, we formalize the computation of equivalence, independence, intersection, and coverage relationships between two action sets in Algo. 1. To compute the optimized response set from n different response sets A_1 A_2 A_n (each specific to an individual attack), Kinesis runs a recursive algorithm that is initialized with

optimized set $O_1 = A_1$. It then continues by computing $O_i = cors(O_{i-1}, A_i)$ for $i = 2, 3, \ldots, n$. Similarly, before executing a new action set, we check its relationship with the on-line responses using Algo. 1 and then find the optimized response set to add to the execution queue.

E. Execution of a Response Action

In Kinesis, the response actions are executed in a fully distributed manner. The low/medium severity actions are executed by the monitor nodes solely based on the own decisions whereas the high severity actions against convicted nodes require consensus among the monitors in a neighborhood. In the latter case, a selected monitor node (*demon*) triggers a message in the neighborhood 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 *demon* 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 responsibilities related to it. Kinesis dynamically selects this *demon* node as the most competent one to take the action. This design ensures the effectiveness of the action as well as avoids the same node doing all the job all the time.

1) Selection of the Demon: A node is selected dynamically as the *demon* for executing an action via a self-organized competition among neighboring monitor nodes. The novelty of our scheme is that we **do not require any synchronization or message exchanges** among the neighbors. Each node in a neighborhood participates in the competition independently through a locally managed back-off timer, called *action timer*. The timer value depends on the *action effectiveness* (AE) of the node, which is estimated locally based on three factors: (i) neighborhood size, (ii) one-hop link qualities, and (iii) time since last action. Intuitively, if a node is connected to more neighbor monitors with good link qualities, it can interact with more nodes and help minimizing the redundant actions. Again, if the node has been idle for a long time, it should take the action to effectively distribute the load in the neighborhood. Hence, this node should be the next demon. The value of AE for node u can be calculated as follows:

$$AE(u) \propto c_1 * t_l + c_2 * \sum_{\substack{k \in N(u) \\ k \in N(s)}} L(u,k) \tag{3}$$

where 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 node u and the neighbor monitor k, and t_l denotes the *time since last action* by u. The higher the $AE(u)$ value, the more effective node u's action is.

The node u joins the competition for being next demon by setting the *action timer*, ActionTimer (u), inversely proportional to its *action effectiveness*.

$$ActionTimer(u) \propto \frac{1}{AE(u)} \tag{4}$$

Thus, a node with better AE has lower back-off period and wins the competition and executes the action. If the action involves a transmission and a neighbor k overhears the message, it cancels the running timer for any action against the same suspect for same incident and updates its t_l and AE value.

2) Consensus among the monitors: To perform high severity operations, the monitors consult with each other and decide an action based on **majority voting**. After selecting the appropriate response action, the *communication* module in the *demon* 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 MAC computed on data using the group key K_g.

Based on the received attack report and local analysis/response decision, other monitor nodes generate and broadcast authenticated *status_reply_msg*-es. Each monitor node computes the majority voting result and the *demon* node broadcasts again the voting decision. Based on the majority voting result, each of the monitor nodes as well as any other neighboring nodes execute the agreed upon action. The BS is also notified with an authenticated report and triggers any action, if needed. The monitor nodes locally observe the neighbors to check whether they abide by the majority decision and otherwise stores a *bad* behavior in the *IRS trust* monitoring window for that misbehaving node.

Algorithm 1 : 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

F. Response Feedback

The majority voting decision provides 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 *Confidence Index (CI)* accordingly. To be noticed that we **do not consider the false negative**s here.

The response feedbacks can also be beneficial from other perspectives. For example, they can be utilized to determine the effectiveness of an action based on which we can adapt the response policies or to estimate the *action effectiveness* of the demon, etc. However, we have not investigated these directions fully and have not integrated them in our design.

G. Secure Policy Storage and Dissemination

The naive approach is to store the response policies as a file or in a policy database which will be an input to Kinesis. Whenever an incident is detected, Kinesis would read the policy file/database to match the response policy. Besides simplicity, this model has other advantages, such as, policy update (e.g., adding a new rule) can be done in an incremental fashion resulting in smaller data transfer over networks. However, there are some significant drawbacks of this approach as well. Most of the operating systems for sensor nodes do not provide a mechanism for file or memory protection. So, malicious modules may get access to the rule file and manipulate it according to their needs. Also, each time Kinesis has to manage an incident, it has to read the policy file resulting in a large number of read operations in its life time. Such operations are prohibitively expensive for resource constrained sensor environment.

Kinesis overcomes these difficulties by generating a binary code from the input policy file and uploading the binary to the nodes. The BS generates the policy binary file from the response policies specified according to our policy language and disseminates or updates the binary throughout the WSNs in the form of standard code dissemination. However, disseminating this binary is likely to be more expensive than doing an incremental update of the rule set according to the naive approach. We assume that such **policy changes are infrequent** and hence do not become a serious concern. It also eliminates the need for expensive read operations from the flash memory at run-time.

To maintain the integrity of policies, the code dissemination process must be secure. We can utilize any of the secure code dissemination protocols proposed for WSNs [7]. Since the policy dissemination/update is performed through a secure mechanism and an attacker cannot modify the sensor ROM (that contains the policy binary) even if it is compromised [16], the integrity of the response policy is ensured.

H. Kinesis Implementation and Configuration

We implemented the Kinesis modules and the policy rules in TinyOS 2.x. According to the policy language we define in Sec. IV, policy rules are implemented as *switch-case* based on incident. This strategy optimizes the implementation. An alternative might be to automatically generate and optimize C codes for policy rules from the definition grammar using standard compliers and then include the binary in Kinesis package. However, we did not focus on various optimizations, such as fusing code blocks to reduce the codebase, optimization on the input rule set, etc. 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 severity tolerance.

To configure Kinesis, the network administrator has to configure the sensor nodes with the response policy binary, the attack risk scores, the action priorities, and the real coefficients. We assume that the sensors are configured after the deployment and changes to these data are infrequent.

V. Simulation Results

A. Simulation Setup

At first, we simulate the performance of Kinesis in the TinyOS simulator TOSSIM. As a routing protocol, we use the standard *Collection Tree Protocol*. In the experiments, we consider anomalies and attacks at various protocol layers: (i) data loss, (ii) data alteration, (iii) selective forwarding, and (iv) sinkhole attacks. The policies considered for these incidents are shown in Table V. To detect the relevant incidents, we implement a simple watchdog monitor based IDS in TinyOS 2.x.

We generate the topologies with symmetric links. The source periodically sends out data every 2 seconds. In each run of simulation, the results are averaged over $4,000$ data transmissions. Unless otherwise stated, we use the above default values in simulation.

TABLE V. Considered Response Policies

on 'data_alteration' if *severity*(data_alteration, suspect) IN (0,0.3] then *retransmit_data* if *severity*(data_alteration, nodeID) IN (0.3,0.5] then *retransmit_data*, *trigger_route_change* if *severity*(data_alteration, suspect) $>$ 0.5 then *retransmit_data*, *revoke* nodeID
on 'data_loss' if *severity*(data_loss, suspect) IN (0,0.3] then *retransmit_data* if *severity*(data_loss, nodeID) IN (0.3,0.5] then *retransmit_data*, *trigger_route_change* if *severity*(data_loss, suspect) $>$ 0.5 then *retransmit_data*, *revoke* nodeID
on 'selective forwarding' *retransmit_data*, *revoke* nodeID
on 'sinkhole' *revoke* nodeID

B. Performance Metrics

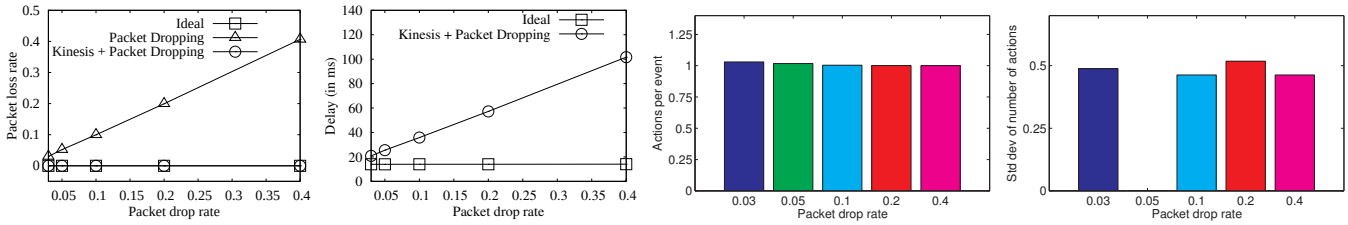
The performance metrics considered to evaluate Kinesis are:

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

- *Data Loss Rate at the BS*: The frequency with which the BS experiences the effect of an anomaly or attack. For example, in case of *data_loss* incidents, it implies 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.
- *Average Data Transmission Delay*: The average time needed for a packet 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 number of actions taken per incident by the monitors in a neighborhood. It justifies our *action timer* design based distributed scheme to trigger the response action for an incident.

(3) **Load Balance:** How evenly the response action executions are distributed in the neighborhood. This is indicated by



(a) Packet reception failure rate at the BS (b) Average data transmission delay (c) Average number of actions per *data_loss* incident (d) Load balance between neighbor monitors

Fig. 5. Kinesis Performance for *data_loss* incidents in Controlled Network Experiments

the standard deviation among the number of actions taken by the monitors in a neighborhood.

(4) **Energy Consumption:** The energy consumption by Kinesis in defending against various attacks.

C. Controlled Experiments

To show the near-perfect behavior of Kinesis, we first run the simulations in a small network of 10 sensors where the source node has a 2-hop routing path to the BS and the forwarder is a data dropping attacker. We control the links between the intermediate nodes so that they all become neighbors to each other with good link qualities and can observe both the source and the attacker. Another implication is that, when a monitor transmits an action message, all other awaiting monitors can overhear it and stop their actions.

Fig. 5(a) displays the performance of Kinesis in case of data recovery on *data_loss* events. The data loss rate is varied from low (0.03) to high rates (0.4). The figure shows that in a network without Kinesis, the rate of data reception failure at the BS increases linearly with the data drop rate by the attacker. On the other hand, Kinesis counteracts the attack and reduces the data loss rate to 0, which is equal to the natural data loss rate of the attack free WSN we considered.

As shown in Fig. 5(b), Kinesis introduces an average data latency within a range of [20,101] ms with respect to the attack free WSN. When Kinesis is notified about a data loss, the response execution is controlled by the *action timer* value which adds a delay to the retransmission of the dropped data packet. Hence, it takes longer for the packet to reach the BS. It also explains the linear increasing trend of average data latency with the drop rate. The higher the data rate, the more data packets experience action execution delays which increases the average latency over all the transmissions. However, the latency increases at most by 5% with respect to inter-packet delays at the BS.

Fig. 5(c) shows the average number of actions taken per event and fig. 5(d) shows the standard deviation among total actions executed by the neighboring monitors. As expected, on average about 1 action is taken for each event. It implies that Kinesis maintains a perfect synchronization among the neighbors on action executions. The very small standard deviation $\sim [0, 0.5]$ indicates the high success of Kinesis in distributing the response executions amongst the neighbors.

D. 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, the source and the attacker are randomly selected and the results are averaged over 10 runs. The attack rate is set to 0.1. For concurrent attacks, a second attacker is placed both in the same and different neighborhood than the first one. The attackers are equally likely to make an attack.

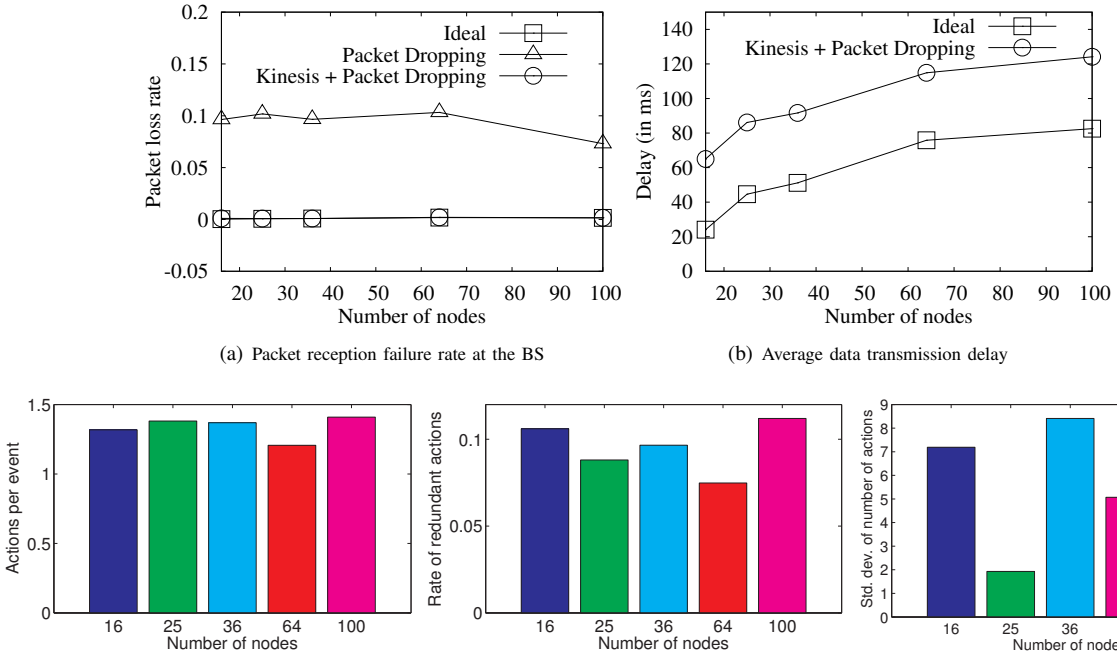
1) Single Attack: In this section, we show the performance of Kinesis in case of a single attacker in the network.

***data_loss* incident**: Fig. 6 illustrates the performance of Kinesis in networks of various sizes, from 16 to 100. As shown in Fig. 6(a), Kinesis reduces the data loss rate of a network under attack from [0.073, 0.103] to ~ 0.002, which is similar to the natural data loss rate (~ 0.0018) in a network without attack. It proves the effectiveness and scalability of Kinesis, both in small and large networks.

Fig. 6(b) reveals the linearly increasing trend in average transmission latencies with network sizes. However, the average amount of delay Kinesis adds due to action execution is almost invariant ([39.03, 41.607] ms) for different networks. The delay incurred by Kinesis is mostly because of the *action timer*. As we see from Eq. 3,4, the *action timer* value doesn't directly depend on the network size, rather depends on the number of neighbors and the qualities of links with them. In the experiments, neighborhood sizes vary from 3 to 5 in different networks and the range of link qualities lies in [0.8, 0.976]. The combined effect of neighborhood size and link qualities made the *action timer* values almost invariant in different networks.

Unlike the controlled experiments, Fig. 6(c) shows that Kinesis is not always able to take a single action per incident. Occasionally, it triggered as high as 1.4 actions per event on average. We also determined the rate of redundant actions taken per incident, which is computed by normalizing the number of actions with the number of possible actors. As shown in Fig. 6(d), the rate of redundant actions is bounded by 0.11 for different actions. The phenomena of redundant actions may occur due to two reasons

1. Hidden node problem: The problem occurs when the monitors of the source and the attacker are not connected or weakly connected to each other. Let's explain the scenario with Fig. 8 - a segment of the attacker's neighborhood found from a simulation topology. Here, node 8 is the source, 18 is the attacker and others are their watchdog monitors. When 18 drops a packet, all the monitors 7, 9 and 29 starts their



(a) Packet 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 between neighbor monitors

Fig. 6. Kinesis Performance for *data_loss* incidents with rate 0.1 in grid networks of various sizes

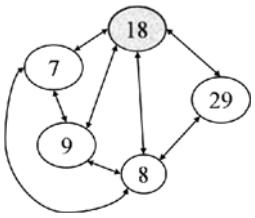


Fig. 8. A segment of the attacker's neighborhood in the simulation topology

action timers to execute the response. When the timer in one of the nodes 7 and 9 fires, for example 7 wins, it retransmits the dropped data and node 9 stops its timer whenever it overhears the action. However, 29 does not possess link to either 7 or 9 and cannot overhear whoever takes the action. Being unaware of other actions on the same event, 29 will execute the action when its timer fires. On the contrary, in controlled experiments, all the monitors of the source and the attacker were connected to each other with high quality links. So they could immediately learn about any other action in the neighborhood on the same event. However, this kind of **redundancy is not a sole problem of Kinesis**, but will be a problem to any **overhearing based solutions**.

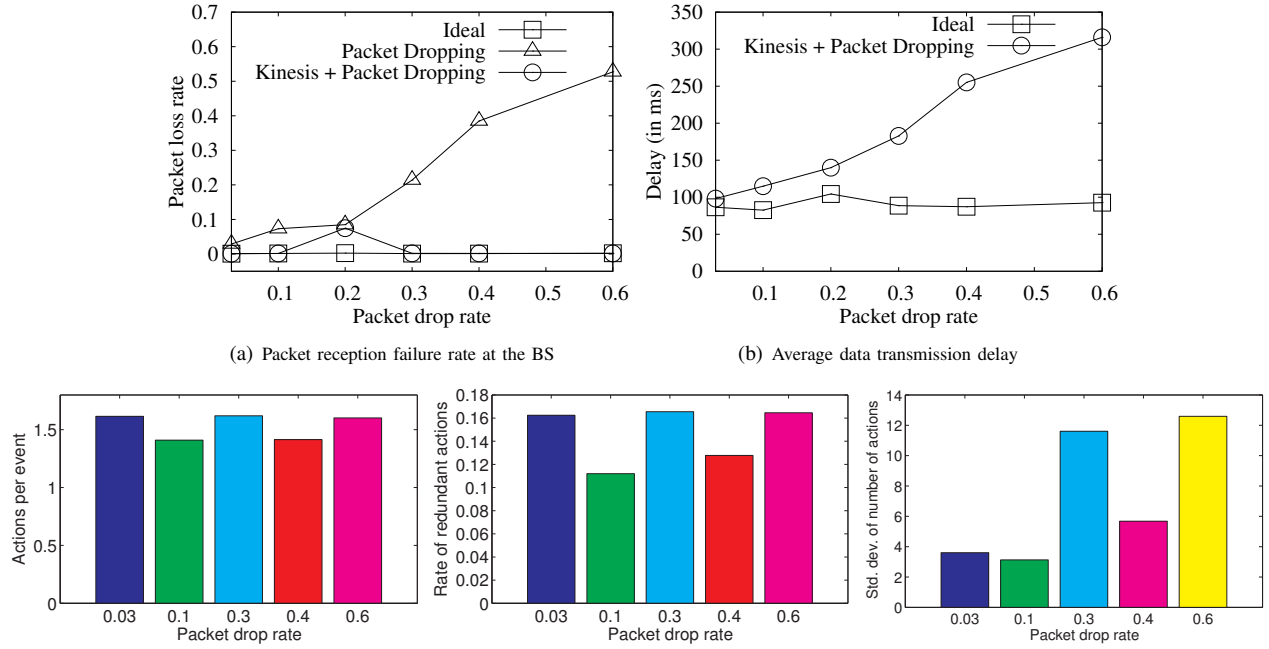
2. Action Timer Value: The locally computed action timer values at two monitors may be close when the *load balancing factor* (i.e. time since last action) is same in both the nodes and the *link quality factor* with the neighbors cannot make a big difference, and vice versa. Since a monitor executes response actions when the timer fires, it may take a redundant action when it does not get enough time to hear others' actions, even if it has good connectivity to other actor(s).

The small standard deviation ([1 93 8 41]) in the number of actions taken by the neighboring monitors, as shown in Fig. 6(e), indicates the high success of Kinesis in balancing load.

To further analyze the scalability of Kinesis, we measure its performance under various data loss rates in a 100-node network and show in Fig. 7 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(a) shows that Kinesis reduces the data loss rate of a network under attack from [0.02, 0.52] to 0 0001, which proves its effectiveness and scalability, even under higher attack rates. Fig. 7(b) reveals the linearly increasing trend in average transmission latencies (similar to what is shown in Fig. 5(b)) with higher rate attacks. Even the range of average latencies introduced by Kinesis with varying attack rates is negligible ([12,223] ms).

Fig. 7(c) and 7(d) show that the average number of actions per incident and redundancy per incident are invariant with respect to attack rates. As discussed above in this section, the number of actions depend on how well the monitoring neighbors are connected to each other and how well the action timer values differ at these nodes. It explains why the number of total and redundant actions per incident does not vary with different attack rates. With small standard deviation in the number of actions taken by the monitors in a neighborhood, Fig. 7(e) shows how well the distributed mechanism of Kinesis works in triggering the response actions.

We also vary the number of attackers from 2% to as high as 20% of the total nodes in a 100-node network. Fig. 10(a) shows that Kinesis still keeps the data loss rate $<$ 0 009. Due to Kinesis operations, average transmission latencies vary within [122.33,189.46] ms as shown in Fig. 10(b). The results are consistent to earlier results.



(a) Packet reception failure rate at the BS

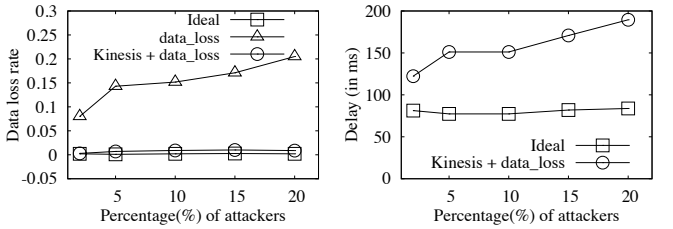
(b) Average data transmission delay

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

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

(e) Load balance between neighbor monitors

Fig. 7. Kinesis Performance for *data_loss* incidents of various rates in a 10×10 grid network



(a) Data reception failure rate at the BS

(b) Data reception failure rate at the BS

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

***data_alteration* attack**: We also run simulations for *data_alteration* attacks and find similar trends in the results. Later on, we show the performance of Kinesis for concurrent incidents of *data_loss* + *data_alteration*, hence we do not report the graphs here.

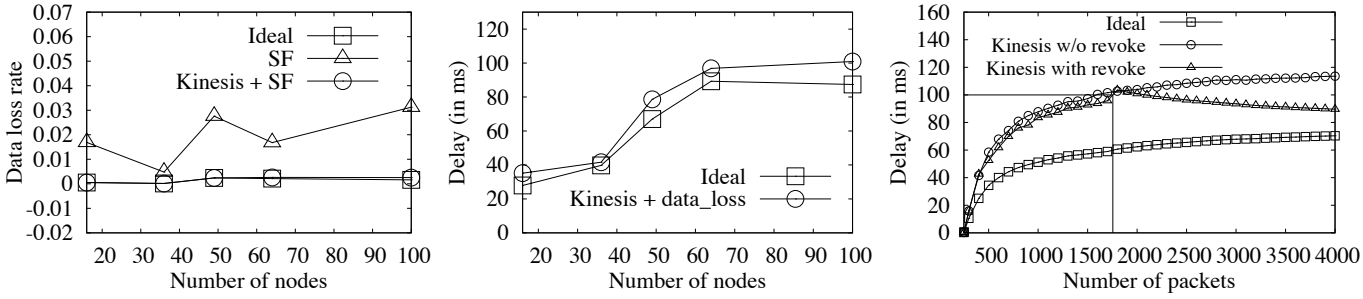
***selective_forwarding* attack**: In a selective forwarding attack, the monitor nodes initially observe data loss by the attacker and hence retransmits dropped data. Once they detect a selective forwarding attack, a monitor node (selected as the next daemon) issues a *state_req_msg* to the neighborhood. The neighboring monitors reply with their own action decision about the suspicious node 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. 9 reports the performance measurements of Kinesis under selective forwarding attack 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. 9(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.

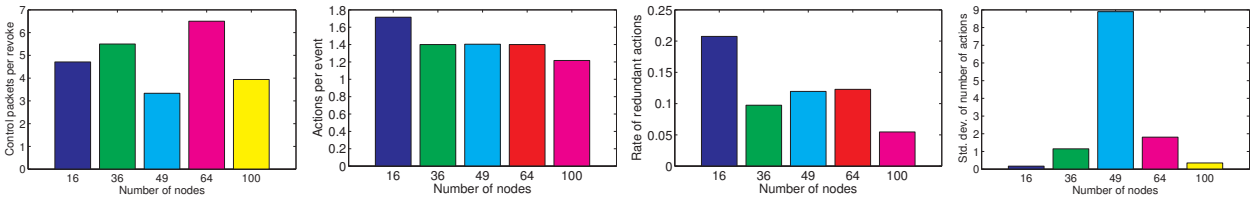
Fig. 9(b) shows an interesting and significantly different trend in transmission latencies with Kinesis under *selective_forwarding* attack. In this case, the average transmission delays are much lower compared to that of *data_loss* incidents and quite closer to the natural data transmission delays. This is due to the revoke operation after which all nodes exclude the attacker from the routing path. Before the revocation, Kinesis only retransmits dropped data and adds latency to data transmissions. However, after the revocation of the attacker, there is no attack and hence no delay is incurred due to response action execution.

To analyze the performance better, we show the average transmission delays over time in Fig. 9(c). Initially when the monitors do not detect the selective forwarding attack yet but only observe data losses, they retransmit dropped packets and hence add latencies to data transmissions. However after the revocation of the malicious node at packet 1755, there is no attack and hence no delay is incurred due to response action execution.

Fig. 9(d) shows the average number of control messages (*state_req_msg* + *status_reply_msg*) exchanged in a neighborhood when a monitor detects a selective forwarding attack and goes for majority voting, and possibly revokes. The number of control messages is proportional to the size of neighborhood, hence it does not vary significantly with the network size. However, the number of control messages per majority voting is bounded by 6.2 packets. To be mentioned that the size of *state_req_msg* is 27 bytes and of *state_reply_msg* is 35 bytes.



(a) Packet reception failure rate at the BS (b) Average data transmission delay (c) Average transmission delays over packets



(d) Average number of control message exchanges in a neighborhood on *revoke* (e) Average number of actions per *data_loss* incident (f) Rate of redundant actions per *data_loss* incident (g) Load balance between neighbor monitors

Fig. 9. Kinesis Performance for *selective_forwarding* attacks in grid networks of various sizes

Fig. 9(e), 9(f), and 9(g) show consistent results with the earlier experiments and hence 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 executing the decided action is ~ 96.4 ms, most of which is contributed by the action timer value.

***sinkhole* attack**: For *sinkhole* attack, we modify the routing 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. In Kinesis, a monitor suspects a potential *sinkhole* attack upon hearing an inconsistent path cost advertisement. The following data drop observations confirm the attack, leading to a quick attacker revocation. Thus, Kinesis not only reduces the data loss rate to ~ 0.0015 (Fig. 11(a)), but also makes the transmission delays closer to natural latency (Fig. 11(b)). Note that *sinkhole* attack often created routing loop causing as high as 3.5% data loss. By revoking the attacker, Kinesis made the WSN stable again. Fig. 11(c), 11(d), and 11(e) show consistent results with the earlier experiments.

2) Concurrent Attacks: We first consider two concurrent but independent attackers, one causing *data_loss* attack and the other *data_alteration* attack at various rates. Fig. 12 shows that the performance of Kinesis does not degrade even under concurrent and high rate attackers. As we see in Fig. 12(a) 12(b) 12(c), Kinesis shows behaviors consistent with the single attack scenario, in all the aspects.

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. 13(a) 13(b) 13(c) 13(d) show how Kinesis performs in such scenario. The irregularity for node 16 is due to the temporary routing instability after revocations.

E. Energy Consumption of Kinesis

Table VI shows the energy efficiency of Kinesis by comparing the aggregate energy consumption of an WSN without and with Kinesis. Here, we consider one data source and measure the energy consumption over 3000 packet transmissions.

TABLE VI. AGGREGATE ENERGY CONSUMPTION OF KINESIS

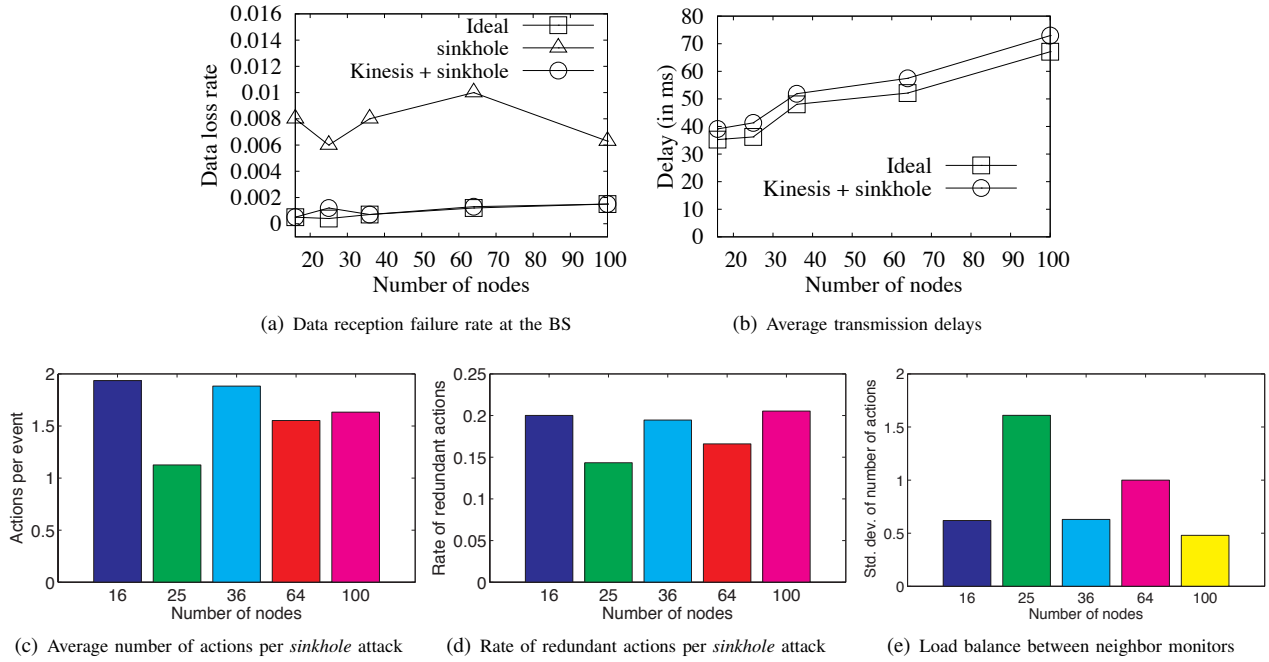
	Ideal	Kinesis		
		data_loss	SF	sinkhole
$\times 10^7$ mJ	1.320488	1.320482	1.320488	1.32048020

F. Action Timer Configuration

Action timer design is a crucial part of Kinesis system 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. 3 and see the impact of timer values on Kinesis performance. Fig. 15 shows the timer impact of these coefficients on the timer and on load balance and redundant actions. 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, 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. 14(a) shows that the optimum values of (c_1, c_2) in terms of load balance is near (3,5) whereas in Fig. 14(b) the optimum value is found after (4.5, 3.5). Thus to optimize both the action redundancy and load balance, (c_1, c_2) should be selected onwards (4.5, 3.5).

VI. TESTBED EVALUATION

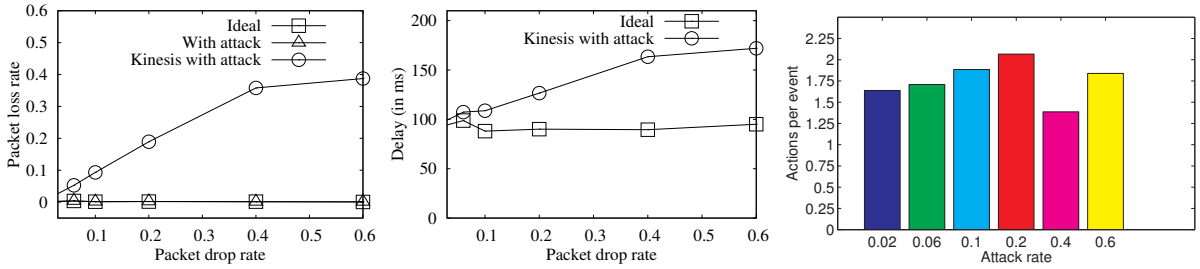
We ported the implementation of Kinesis to the TelosB platform. Our motes have a 8 MHz TI MSP430 micro controller, 2.4 GHz radio, 10 kB RAM, and 1 MB external ash for data logging. We evaluated the performance of Kinesis



(a) Data reception failure rate at the BS (b) Average transmission delays

(c) Average number of actions per *sinkhole* attack (d) Rate of redundant actions per *sinkhole* attack (e) Load balance between neighbor monitors

Fig. 11. Kinesis performance for *sinkhole* attack



(a) Data Reception Failure Rate at the BS (b) Data Transmission Delay (c) Action per Event

Fig. 12. Kinesis performance for *data_loss* + *data_alteration* incidents with various rates in a 10×10 grid network

for two attacks (i) *data_loss* (ii) *selective_forwarding*. For the evaluation, we consider the same performance metrics as in TOSSIM simulations: (i) Data Loss Rate at the BS, (ii) Average Data Transmission Delay, (iii) Average Actions per Incident.

A. Experimental Setup

We placed TelosB motes in an indoor environment and controlled the transmission power of the motes to ensure multihop communication in the network. All motes are battery-powered and a special mote is used as the root node and to collect statistical information. A source node sends out data packets every 1 second. For the purpose of performance analysis, we collected information about the number of transmitted data packets, action packets and transmission delays. The root node is connected to a laptop in a USB port and passes the statistical data information through the serial forwarder. We run the experiments for 10000 packets and average the results.

B. Multihop Indoor Experiments

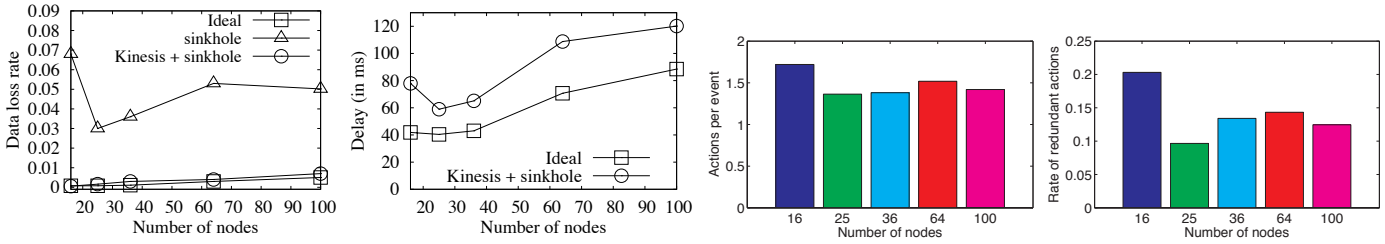
We build a 250×150 cm topology consisting of 20 TelosB sensors deployed randomly in a home environment. In order to ensure multihop communication, we use the lowest power level 1. Fig. 15(b) shows a part of the testbed and fig. 15(a) shows the coordinates of the nodes, where nodes are labeled from 2 to 21. Node 2 is selected as the source node and 20 is the root node.

***data_loss* incident**: For data drop attacks, node 12 is set as the attacker which drops packets at the rate of 0.1. Table VII summarizes the performance of Kinesis and shows the comparison with a network without attack.

TABLE VII. PERFORMANCE OF KINESIS IN TESTBED ON *data_loss* INCIDENTS

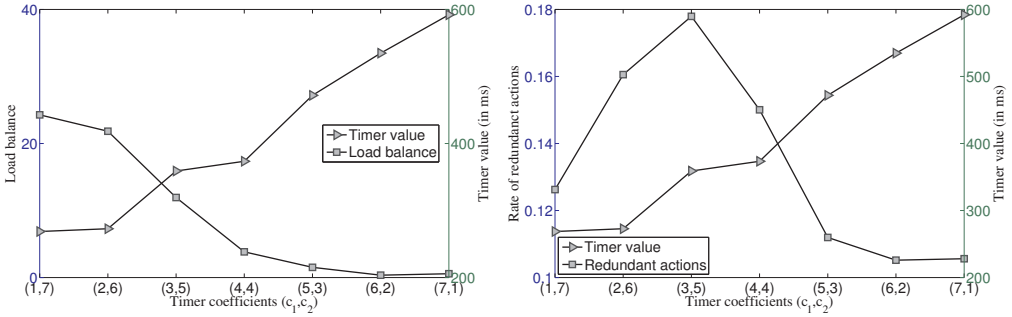
	Ideal	Packet drop	Kinesis + Packet drop
Packet loss rate	0.00029	0.103	0.00058
Average transmission delay (ms)	98.20	N/A	122.87
Average actions per incident	N/A	N/A	1.66

We can see that the performance of Kinesis in testbed is



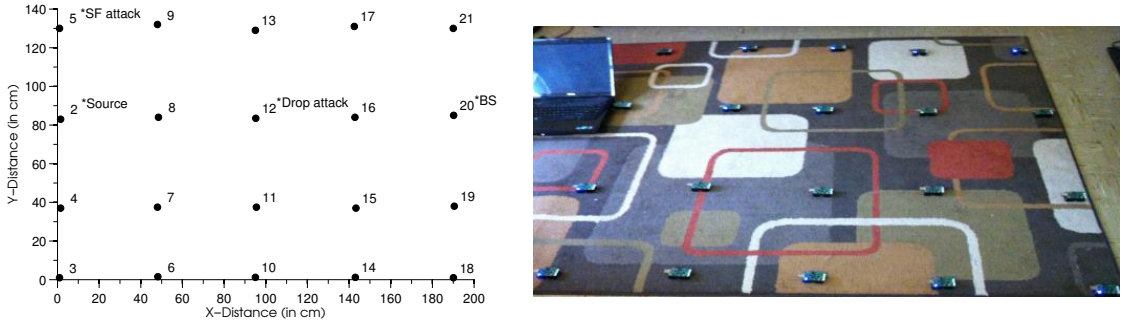
(a) Data loss rate for *sinkhole + SF* (b) Average transmission delays for *sinkhole + SF* (c) Average number of actions per *sinkhole + SF* (d) Rate of redundant actions per *sinkhole + SF*

Fig. 13. Kinesis performance for concurrent attacks



(a) Load balance vs Action timer (b) Action redundancy vs Action timer

Fig. 14. Coefficient configuration for Action Timer



(a) Coordinates of nodes (b) Part of the testbed

Fig. 15. Placement of nodes in indoor multihop network

consistent to that in simulations, which justifies the simulation results.

***selective_forwarding* attack**: For selective forwarding attack, node 5 is set as the attacker which drops packets at the rate of 0.1. However, in this case, instead of revoking the attacker, we let the attacker continue to see how accurately the monitor nodes can take decisions of data retransmit only and revocation. We found that, in all the cases, Kinesis took accurate decisions. The other typical performance, e.g. packet loss rate, etc. shows similar behavior as earlier.

VII. Related Work

Past approaches have focused on anomaly detection in WSNs but very few provided automatic responses to ensure continuous service availability. Asim *et al.* [2] propose an architecture that organize 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 mainly on network failures and energy related issues, rather than on malicious behaviors or attacks. MALADY is a machine learning-based system that enables embedded sensor nodes to use gathered data to make real-time decisions [9]. However, MALADY aims at the detection and learning process rather than response to attacks. Mamun *et al.* propose a policy based intrusion detection and response system with a four level hierarchy architecture [12]. Their intrusion response system has a general scope based on customizable policies, however their only responses are temporary or permanent revocation depending on the misbehavior occurrence, and are only applicable to their hierarchical architecture.

Lim *et al.* proposed rerouting strategies against jamming attacks in WSNs for Microgrids [11]. They recover from such attacks by rerouting their traffic to a chosen path based on the highest RSSI value among multiple candidate paths, without considering other link factors. Some researchers have designed

response systems to isolate faulty nodes from the network communication layer as an initial response.

To the best of our knowledge, our approach is the first able to provide responses to an extensive amount of WSN attacks. It is also extensible to novel anomalies and intrusions, scalable over larger networks, and provides a flexible policy language.

VIII. Conclusion

In this paper, we presented the first incident response and prevention system for WSNs. The system reacts not only after an attack occurs but also on anomalous events so that the WSN is functional while the attack progresses. The system is dynamic as it selects the response actions based on the suspects security status. It is distributed since it does not require any central authority to trigger the actions. The simple yet flexible design of the response policies make the system easily extensible to handle newer attacks. Kinesis is secure in policy dissemination, storage and executions. The experimental results show that Kinesis achieves high effectiveness in terms of data rate and latency, low redundancies in action executions, and most importantly, the scalability.

To further enhance the system, we will

(i) investigate how to improve the redundancy and load distribution in case of *hidden node* problem, discussed in the simulation section. A related problem might be to select the monitors in an optimized way so that all the monitors in a group can listen to each other,

(ii) work towards more extensive risk assessment and policy configuration framework

References

[1] 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.

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

[3] O. Chipara, C. Lu, T. C. Bailey, and G.-C. Roman. Reliable clinical monitoring using wireless sensor networks: experiences in a step-down hospital unit. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems*, pages 155–168, 2010.

[4] 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.

[5] T. Gao, C. Pesto, L. Selavo, Y. Chen, J. Ko, J. H. Lim, A. Terzis, A. Watt, J. Jeng, B. rong Chen, K. Lorincz, and M. Welsh. Wireless medical sensor networks in emergency response: Implementation and pilot results. In *IEEE Conf. on Tech. for Homeland Security*, 2008.

[6] A. Hasswa, M. Zulkernine, and H. S. Hassanein. Routeguard: an intrusion detection and response system for mobile ad hoc networks. In *WiMob (3)*, pages 336–343, 2005.

[7] S. Hyun, P. Ning, A. Liu, and W. Du. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *IPSN*, 2008.

[8] J. Ko, C. Lu, M. Srivastava, J. Stankovic, A. Terzis, and M. Welsh. Wireless sensor networks for healthcare. *Proceedings of the IEEE*, 98(11):1947–1960, 2010.

[9] S. Krishnamurthy, G. Thamilarasu, and C. Bauckhage. Malady: A machine learning-based autonomous decision-making system for sensor networks. In *Proc. of the Intl. Conf. on Computational Science and Engineering - Volume 02*, pages 93–100, 2009.

[10] I. Krontiris, K. M. Ave, T. Giannetsos, and T. Dimitriou. Lidea: A distributed lightweight intrusion detection architecture for sensor networks. In *In Proceeding of SecureComm*, 2008.

[11] Y. Lim, H.-M. Kim, and T. Kinoshita. Traffic rerouting strategy against jamming attacks in wsns for microgrid. *International Journal of Distributed Sensor Networks*, 2012.

[12] M. S. I. Mamun, A. F. M. S. Kabir, M. S. Hossen, and R. H. Khan. Policy based intrusion detection and response system in hierarchical wsn architecture. *CoRR*, abs/1209.1678, 2012.

[13] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. of the Intl. Conf. on Mobile Computing and Networking*, pages 255–265, 2000.

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

[15] Y. Ponomarchuk and D.-W. Seo. Intrusion detection based on traffic analysis in wireless sensor networks. In *Annual Wireless and Optical Communications Conference (WOCC)*, pages 1–7, 2010.

[16] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. Scuba: Secure code update by attestation in sensor networks. In *Proc. of the ACM Workshop on Wireless Security*, pages 85–94, 2006.

[17] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J. A. Stankovic. An advanced wireless sensor network for health monitoring. In *Transdisciplinary Conf. on Distributed Diagnosis and Home Healthcare*, pages 2–5, 2006.