# Dynamic Energy-based Encoding and Filtering in Sensor Networks

Hailong Hou, *Cherita Corbett, Yingshu Li, Raheem Beyah

Department of Computer Science
Georgia State University
Atlanta, GA, USA
Email: {hhou2@student, yli@cs, rbeyah@cs}.gsu.edu

*Computer & Network Security Group
Sandia National Labs
Livermore, CA, USA
Email: clcorbe@sandia.gov

*Abstract*— **In critical sensor deployments it is important to ensure the authenticity and integrity of sensed data. Further, one must ensure that false data injected into the network by malicious nodes is not perceived as accurate data. In this paper we present the Dynamic Energy-based Encoding and Filtering (DEEF) framework to detect the injection of false data into a sensor network. DEEF requires that each sensed event report be encoded using a simple encoding scheme based on a keyed hash. The key to the hashing function dynamically changes as a function of the transient energy of the sensor, thus requiring no need for re-keying. Depending on the cost of transmission vs. computational cost of encoding, it may be important to remove data as quickly as possible. Accordingly, DEEF can provide authentication at the edge of the network or authentication inside of the sensor network. Depending on the optimal configuration, as the report is forwarded, each node along the way verifies the correctness of the encoding probabilistically and drops those that are invalid. We have evaluated DEEF's feasibility and performance through analysis. Our results show that DEEF, without incurring transmission overhead (increasing packet size), is able to eliminate 90% - 99% of false data injected from an outsider within 9 hops before it reaches the sink.**

***Keywords-sensor networks; data filtering; energy-based keying; In-line filtering, sensor security.***

## I. INTRODUCTION

Sensor network technology has rapidly developed in recent years and will be used in a variety of environments. Accordingly, people will come to rely more on sensor networks. For example, in a battlefield scenario, sensors may be used to detect the location of enemy sniper fire [1, 2] or to detect harmful chemical agents before they reach troops. The use of sensors will also evolve from merely capturing data to a system that can be used for real-time compound event alerting [3]. These critical sensor networks must provide authentic and accurate data to surrounding nodes and to the sink to enable time-critical responses (i.e., troop movement, evacuation, first response deployment, etc.). Consequences for propagating false data are costly, depleting limited network resources and wasting response efforts.

Securing sensor networks poses unique challenges because these types of networks are usually unattended and have limited energy, computation, and communication capabilities. In this paper, we propose the Dynamic Energy-based Encoding and Filtering (DEEF) framework, a technique used to verify data in line and drop false packets from malicious nodes, thus maintaining the health of the sensor network. DEEF dynamically updates keys without exchanging messages for key renewals and embeds integrity into packets as opposed to enlarging the packet by appending message authentication codes (MACs).

The contribution of this paper is three-fold. First, we discuss a technique for ensuring authenticity of data by encoding data using the results of a keyed hash. Next, we present a dynamic keying scheme (Energy-based Keying) that uses the perceived energy of a sensor to generate a unique key for the hash function every $n$ transmissions. Finally, we present DEEF which uses the encoding and energy-based keying concepts to perform in-line filtering of unwanted data from the network.

## II. RELATED WORK

As security has become more of a focus in sensor networks, researchers are interested in providing privacy, authentication, and integrity, all through some form of cryptographic algorithms (e.g., encryption, MAC, etc.). In [4], the authors proposed a link-layer security architecture, TinySec, with a distributed focus on security, performance, and usability. In [5] the authors provide new node authentication and message encryption using the RC5 algorithm in counter (CTR) mode.

Following the need for secure sensor networks using cryptography, key management became a significant focus. Nodes in a sensor network use either pre-distributed keys or a dynamic keying scheme, whereby nodes are re-keyed periodically post deployment. The cryptographic keys are normally either group-wise, pair-wise, or network-wise. In [6], the authors propose μTesla, a bootstrapping scheme to enable each sensor to authenticate messages broadcast from the base station in a sensor network with $n$ sensors based on the pre-installation of one key rather than $n$. In [7], a new pairwise key pre-distribution scheme for wireless sensor networks was presented.

W. Du et. al. proposed an efficient technique that replaces the public key authentication process of public key cryptography with a one-way hash function [8]. They use the

sensors' public keys to construct a forest of Merkle trees. By optimally selecting the height of each tree they are able to minimize the computation and communication costs. In [9], the authors propose a half-key scheme based on the well-known multi-key pre-distribution approach to provide data delivery in sensor networks with reduced memory space requirements and better security enforcement.

Given the abundance of potential attacks on routing in sensor networks (e.g., wormhole, Sybil, time synchronization, sinkhole, HELLO flood, etc.) much needed attention has been paid to securing routing in sensor networks. In [10], X. Du et. al. proposed the Secure Cell Relay (SCR) routing protocol. SCR uses a three-way-handshake mechanism (to avoid unidirectional attack) to discover neighbors and generate pair-wise keys as well as broadcast keys (so future secure data in-network aggregation is also supported) at very short intervals after the network is deployed (to avoid attacker tampering). The keys are only known by legal nodes thus guaranteeing the routing protocol's resistance to Sybil and clone attacks, and ensuring confidentiality, integrity, and freshness of the data. Ye et. al. [11] proposed the Statistical En-route Filtering (SEF) mechanism to probabilistically verify data authenticity and to drop false data. The idea is that each sensor detecting the data generates its own message authentication code (MAC) that is attached to the sensor's report and forwarded. Nodes along the path on which a report is forwarded then verify the correctness of those reports probabilistically based on their attached MACs. Deng et al. [12] proposes intrusion-tolerant routing for wireless sensor networks - INSENS. INSENS constructs secure and efficient tree-structured routing with the objective to localize the damage caused by an intruder who has compromised deployed sensor nodes by establishing multiple disjoint paths. Only the base station (BS) has the right to update sensors' routing information using secret pairwise key sharing between the BS and the node. Thus, an attacker cannot modify the routing information, which helps INSENS eliminate sinkhole/wormhole attack. Also, only the BS is allowed to flood the network. INSENS achieves this by using One way-Hash-Chains (OHCs) for each broadcasting packet, limiting the attacker from flooding the network. Zhang proposed a self-adjusting directed random walk technique [13] to enhance source-location privacy. Random walk creates phantom sources and a subsequent routing phase to deliver messages to the sink.

Each of the aforementioned protocols requires a significant amount of packet overhead and/or an increase in message exchanges. This sort of overhead is insufficient for resource constrained sensor networks; security is provided at the expense of the significant amount of energy consumed for this transmission overhead. We propose a lightweight technique for detecting outsider attacks, which consists of an encoding scheme and dynamic energy-based keying, obviating the need to re-key. Our scheme is an efficient method to ensure authenticity and integrity of data in the sensor network. The closest proposed security work to ours is that in [11]. However, the authors suggest they require 14 bytes of overhead per packet compared to our scheme that does not incur transmission overhead. Further, their scheme does not consider message free re-keying.

## III. Threat Models and Global Assumptions

Due to the broadcast nature of the wireless medium used in sensor networks, attackers may try to eavesdrop, intercept, or inject false messages. In this paper we mainly consider the false injection of messages from an outside malicious node. The attacker is thought to have the correct frequency, protocol, and possibly a spoofed valid node ID.

Throughout this work, the following assumptions are made:

a) All of a sensors' sensing components turn on/off together.
b) A typical routing algorithm such as [14-21] is used.
c) The routing algorithm is deployed on a reliable MAC.
d) The sensor network is densely populated such that multiple sensors observe and generate reports for the same event.
e) Sensors may have different communication ranges and initial battery supplies.

## IV. Encoding Process

Due to the resource constraints of sensor networks, traditional digital signatures requiring expensive cryptography is not a viable option. The scheme must be simple, yet effective. In this section we discuss a very simple encoding process that can be used to ensure the authenticity and integrity of sensed data without incurring transmission overhead of the traditional schemes.

Our scheme uses a keyed hashing approach. Each node sends its ID ($i$-bits), type ($t$-bits) (assuming each node has a type identifier), and data ($d$-bits) (Figure 1) to its next hop.

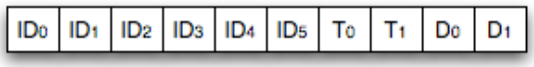


Figure 1. $i$+$t$+$d$ bit string before permutation.

The sensors' ID, type, and the sensed data are transmitted in a pseudo random fashion according to the result of the hashing function H(*key*, ID). An additional copy of the ID is also transmitted in the clear along with the encoded message. The format of the final packet to be transmitted becomes:

$$\text{Packet} = [\text{ID}, \{\text{ID, type, data}\}_k],$$

where $\{x\}_k$ constitutes encoding $x$ with key $k$.

Instead of the traditional approach of sending the hash value along with the information to be sent, we use the result of the hash value locally. The computed hash code is used to encode the <ID|type|data> message (Figure 2).

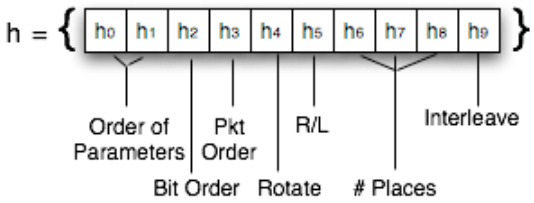

Figure 2. Resulting hash code with example operations associated.

The hash code **h** can be mapped to a set of actions to be taken on the data stream combination. The actions can include:

- Rotate
  - Left - 1
  - Right – 0
  - # Places – 000 → 111
- Interleave
  - Yes – 1
  - No - 0
- Order of parameters
  - ID, T, D – 00
  - ID, D, T – 01
  - D, ID, T – 10
  - D, T, ID - 11
- Order of bits within each parameter
  - Little Endian - 0
  - Big Endian - 1
- Order of bits in packet
  - Little Endian - 0
  - Big Endian - 1

For example, if a node computed the following hash code **h** = {1 0 1 0 1 1 0 0 1 1} the string in Figure 2 becomes the string in Figure 3 before it is transmitted. The receiver will perform the same hash operation (since the inputs to the hash function are stored and updated on each sensor) and perform the inverse actions to accurately decode the packet. To ensure correctness, the receiver compares the plaintext ID with the decoded ID.

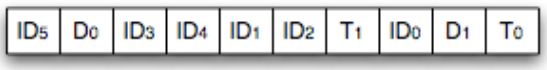

Figure 3. *i*+*t*+*d* bit string after permutation.

Although it is theoretically possible (1 in $2^{i+t+d}$) for a hacker to accurately inject data, it becomes increasingly unlikely as the packet grows. The benefits of this scheme are the following: 1) since there is no hash code to transmit, the packet size does not grow, avoiding bandwidth overhead on an already resource constrained network, thus increasing the network lifetime; 2) the technique is simple, thus ideal for devices with limited resources; and 3) the internal state of the hashing function, namely the key, changes dynamically without re-keying. This will be discussed in detail in the next section.

## V. ENERGY-BASED KEYING

One of the primary contributions of this paper is the method used for handling the keying process. The transient energy of a node depends on the energy required for: event detection, node keep alive, packet reception/transmission, and packet encoding/decoding. As each of these actions occur, the energy in a node is modified (depleted). The current energy, $E_c$, of the node will be used as the key to the hash function (discussed in detail below). It is likely that during the initial deployment each node will have the same energy level, therefore the initial key, $K_1$, is a function of the current energy plus an initialization vector (IV). The IV is pre-distributed. Subsequent keys, $K_j$, are a function of the current energy, $E_c$, and the previous key $K_{j-1}$.

$$K_1 = F(E_c, IV) \quad (1) \text{ (initial transmission)}$$

$$K_j = F(K_{j-1}, E_c) \quad (2) \text{ (subsequent transmissions)}$$

Accordingly, the key used as input to the hash function is dynamic and thus there is no need to refresh or update keys. The keys will change making it difficult for attackers to intercept enough packets to break the encoding algorithm.

### *A. Previous uses of Power/Energy for Security*

Research has been conducted in estimating battery life [22] measuring power consumption [23, 24] and in creating software that uses lower power than standard software [25]. This work served as the foundation for using energy in security schemes and led to the initial work in this area discussed below.

In [26-28] the authors presented a host-based IDS, Battery-based Intrusion detection (B-BID), which uses battery behavior anomalies based on certain pre-determined threshold levels to determine if an attack (network-based or host-based) is taking place. The basic concept behind this work is that normal CPU operation is not characterized by persistent battery use, while an attack may have that characteristic. After their system learns what behavior is deemed normal for applications running on the node, abnormal behavior can be detected using statistical and spectral analysis. They were also able to develop energy signatures for certain attacks. The authors of [26-28] present a similar network-based approach in [29].

In [30] the authors introduce several types of denial-of-service attacks on battery-powered mobile hosts (service request power attacks, benign power attacks, and malignant power attacks). They propose a power-secure architecture that uses multi-level authentication and discuss using energy signatures to validate trusted programs.

In [31] the authors explore modification of battery polling rates (for an IPS proposed in their previous work [28]) in conjunction with the variance of malicious network activity. A dynamic polling rate algorithm was presented in order to provide maximum detection while conserving battery charge life.

Though there is minimal overlap with the previous work discussed above and our work, it sets the precedence for using an energy-based approach in security.

*B. Calculating Energy Costs*

In this paper we focus on the following actions that consume energy in the sensor network: *event detection, packet reception, packet transmission, packet encoding, packet decoding, and the energy required to keep a node alive in the idle state*. We denote the energy costs for these actions as:

- Event detection: $E_d$
- Packet reception: $E_r = n * e_r$ ($e_r$ denotes the energy cost of receiving one bit of data. $n$ is the size of packet).
- Packet transmission: $E_t = n * e_t$ ($e_t$ denotes energy cost of transmitting one bit of data.).
- Packet encoding: $E_{enc} = n * e_{enc}$ ($e_{enc}$ denotes cost of encoding one bit).
- Packet decoding: $E_{dec} = n * e_{dec}$ ($e_{dec}$ denotes cost of decoding one bit).
- Keep-alive energy: $E_a = t * e_a$ ($e_a$ is the energy to keep the node alive within $t$ units of time).

Now, the function to compute energy cost is:

$$F(E) = SUM(E_d, E_r, E_t, E_{enc}, E_{dec}, E_a)$$

Using this simple function we can compute the energy cost associated with the actions performed by each node. For example, sensors perform two primary actions – sense data and forward data. Accordingly, the equations below represent $E_S$, the cost to send a sensed report; $E_{FW}$, the cost to forward the report if the node is watching this particular sensor; and $E_F$, the cost to forward the report if the node is not watching the sensor. Here "watching" refers to a receiving node that has locally stored knowledge of the keying material of the sending node. This will allow the receiver to authenticate a message from the source. This is discussed in more detail in section F.) After an event is detected, with the node alive for $t$ units of time the following would be the cost associated with the sending node:

$$E_S = E_d + E_t + E_{enc} + E_a$$
$$= E_d + n * (e_t + e_{enc}) + t * e_a$$

When this packet passes through a forwarding node (that is watching the source node), it will cost:

$$E_{FW} = E_r + E_t + E_{dec} + E_{enc} + E_a$$
$$= n * (e_r + e_t + e_{enc} + e_{dec}) + t * e_a$$

When this packet passes through a forwarding node (that is not watching the source node), the above equation reduces to:

$$E_F = E_r + E_t + E_a = n * (e_r + e_t) + t * e_a$$

*C. Percieved and Bridge Energy*

In order to authenticate a packet, a node must keep track of the energy of the sending node to derive the key needed for decoding. Ideally, once the authenticating node has the initial energy value of the sending node, the value can be updated by decrementing the cost associated with the actions performed by the sending node using the cost equations defined above. However, the energy value used to encode data at the sending node may differ from the energy value that is stored for the sending node at its corresponding watching node. For example, if a malicious node (spoofing the ID of a valid node) attempts to inject false data into the sensor network it will be discarded at the first node that is watching the node whose ID was spoofed. Assuming that the first node the false data encounters is watching the node whose ID was spoofed, that node will immediately discard the packet. At this point the node that discarded the data and the nodes above that node on the path to the sink lose synchronization (because the upper portion never sees the malicious packet and does not know to decrement the energy associated with servicing the malicious transmission). If a valid packet were to be generated by the current watching node using its current energy, the upstream node(s) that watch this particular node would discard the packet. To resolve this issue, the *perceived* energy value rather than the actual energy value is used to encode the data. The perceived energy is the energy value that the upstream nodes perceive the node(s) they are watching have. So in this example, the **actual** residual energy of the watching node is $E_{cj} = E_{c(j-1)} - E_r - E_d$, however, the node additionally stores its **perceived** energy $E_p = E_{c(j-1)}$ (before malicious node transmits) because it knows that it dropped the previous packet(s) and that upstream nodes are completely unaware of the energy spent handling the malicious transmission. The perceived energy $E_p = E_{c(j-1)}$ is used to encode subsequent packets (generated by the watching node) and is updated as valid (or assumed valid) packets are transmitted. Since the upper nodes currently have $E_{c(j-1)}$ stored as the energy for the current watched node, the network remains synchronized.

It is important to mention a more complicated scenario. Assume the same scenario as mentioned above, however assume that the packet enters the network and spans several hops before the appropriate watching node filters the data. Now, the entire upper portion (portion above intermediate watching node) has lost synchronization with the lower portion of the network (portion below the watching node). This is because the lower portion of the network assumes that the packet that was forwarded will make it to the sink and the upper watching node will decrement energy levels of the nodes it watches accordingly. However, the upper portion never receives the data that was dropped by the intermediate watching node so it does not know that a packet ever traversed that path and does not update the energy of the nodes it watches. The intermediate watching node is the only node that knows the state of both portions of the network. Accordingly, to resolve potential loss of synchronization in this scenario, the watching node must store an additional energy value, the *bridge energy* value, to allow

resynchronization (bridging) of the network at the watching node that dropped the packet. That is, as subsequent packets generated from the node of interest pass through the watching node that dropped the packets, the watching node will decode the packet with the perceived energy key of the originating node and re-encode the packet with the bridge energy key, thus keeping the entire network synchronized.

To minimize the potential for loss of synchronization, the perceived energy is only updated if a node is encoding. This means that the node is either the originating node or a forwarding node that has had to bridge the network. Generally, the potential for the network to lose synchronization occurs when lower nodes do not perform actions that upper nodes assume were performed or the converse. Depending on the technique required (traditional or non-traditional) there are many scenarios that could arise, however using perceived and bridge energy will assure the continued synchronization of the network as whole.

*D. Energy Binning*

Energy can be considered as a somewhat imprecise value given that there is a percentage error associated with different actions taken by a sensor. This error may vary based on internal or external elements (temperature, etc.) placed on the node as well as on the precision used when obtaining the energy value.

The imprecise nature of the sensor's energy value presents problems when attempting to use the energy value as the key to the encoding function. When considering the imprecise value it is possible for nodes to lose synchronization. Thus, we propose the use of discrete values of energy by introducing binning to obtain approximate values. For example, if the energy value of a node is 11.75 Joules and we have bins of width ten Joules starting from zero, the energy would be reported as 20 Joules because 11.75 Joules falls in the second bin.

Although binning (using approximate discrete values) helps significantly with synchronization, it is still possible for borderline values to cause nodes to lose synchronization. This depends on the bin size, which will affect the frequency of key regeneration. The smaller the bin size the higher the frequency of key regeneration but the higher the probability of error. In this paper we assume that the optimal bin size has been determined and that energy is discrete when passed to our algorithm. We discuss this further in the future work section.

*E. Authentication Information Stored in Nodes*

Since we will authenticate packets in line, every node in the sensor network will store one or more record(s) (Figure 4) based on the algorithm chosen (DEEF-T or DEEF-NT) and the vigilance required. For each node being watched, a record contains the following state information:

- Node ID
- Node type
- $E_c$ (current energy level)
- $E_b$ (bridge energy level)
- Keep-alive timer

Each node must also maintain local state information for itself, which includes:

- IV (stored until after 1st decode)
- Actual energy level, $E_c$
- Perceived energy level, $E_p$

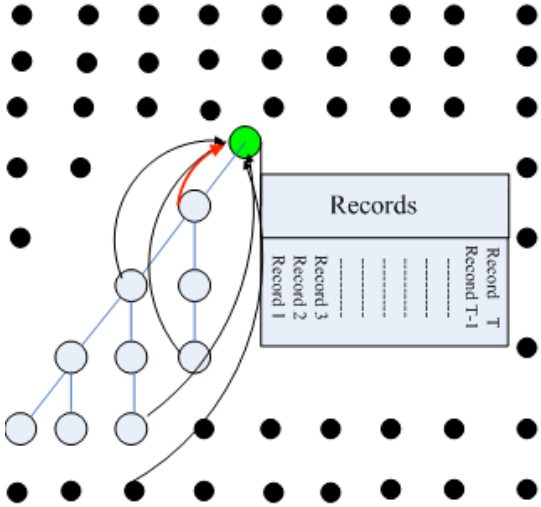


Figure 4. Record array stored in arbitrary node.

*F. Monitoring Node Energy Status*

In this section we discuss the general process of monitoring a watched node. Deciding which nodes to watch and how many depends on the configuration of the algorithm used (DEEF-T or DEEF-NT) and the aggressiveness threshold associated and is discussed in section VI.

Nodes in sensor networks normally play one or two of the following roles. Nodes will either behave as the *originating node* that generates a report or a *forwarding node* that forwards a report. The report traverses the network through forwarding nodes and finally to the *terminating node*, the sink.

When an event is detected by an *originating node* the next step is for the report to be secured. The originating node uses the perceived energy value (discussed in Section V-C) and an initialization vector (or previous key value if not first transmission) to construct the next key. The key is used as input into the keyed hashing function for encoding the <ID|type|data> message. The encoded message and the cleartext ID of the originating node is transmitted to the next hop (forwarding node or sink) using the following format: [ID, $\{ID, type, data\}_k$], where $\{x\}_k$ constitutes encoding $x$ with key $k$. The perceived energy value is updated and stored for use with the transmission of the next report, only if this nodes has performed encoding – either as an originating node or a node that is currently bridging the network.

Once the *forwarding node* receives the packet it will first check its watch-list to determine if the packet came from a node it is watching. If the node is not being watched by the current node, the packet is forwarded without modification or authentication. Although this node performed actions on the packet (received and forwarded the packet), its local perceived energy value is **not** updated (as mentioned in section VI-C).

This is done to maintain synchronization with nodes watching it further up the route. If the node is being watched by the current node, the forwarding node checks the current record stored that is associated with the sending node and extracts the energy value to derive the key and authenticates the message by decoding the message and comparing the plaintext node ID with the encoded node ID. If the packet is authentic, an updated energy stamp (calculated as discussed in section V-B) is stored in the record associated with sending node. If the packet is not authentic it is discarded. Again, the energy value associated with the current sending node is only updated if this node has performed encoding on the packet. That is, if this node is the originating node or is currently bridging the network.

## VI. DYNAMIC ENERGY-BASED ENCODING AND FILTERING (DEEF)[1]

### A. Authentication Flexibility in DEEF

DEEF has the ability to provide traditional authentication where authentication is performed at every node. Accordingly, filtering is performed at the boundary nodes (first node the packet encounters) and it is very difficult ($\frac{1}{2^{packetsize}}$) for malicious nodes to inject data into the network. In non-traditional mode the network statistically filters illegitimate traffic that has entered the network. Accordingly, it is possible for illegitimate traffic to enter the network and span several hops before being dropped.

### B. Traditional Authentication

In the traditional authentication scheme we assume there exist a short window of time at initial deployment that an adversary is not able to compromise the network, because it takes time for an attacker to capture a node or get keys. During this period, route initialization information is used by each node to decide which node to watch and a record *r* is stored for each of its 1-hop neighbors in its watch-list (using a binary tree structure routing algorithm, $r = 3$). To obtain a neighbor's initial energy value, a network-wise master key can be used to transmit this value during this period.

When an event occurs and a report is generated, it is encoded as a function of a dynamic key based on the perceived energy of the originating node and transmitted. When the packet arrives, the forwarding node extracts the key of the sending node (this could be the originating node or another forwarding node) from its record and decodes the packet. After the packet is decoded the plaintext ID is compared with the decoded ID. If the packet is authentic, and this hop is not the final hop, the packet is re-encoded by the forwarding node with its own key derived from its current perceived energy level. If the packet is illegitimate, the packet is discarded. This process continues until the packet reaches the sink. Accordingly, illegitimate traffic is filtered before it enters the network.

Re-encoding at every hop refreshes the strength of the encoding. Recall that the general packet structure is [ID,{ID, type, data}$_k$]. To accommodate this scheme the ID will always be the ID of the current node and the key is derived from the current node's local energy value. If the location of the originating node that generated the report is desired, the packet structure can be modified to retain the ID of the originating node and the ID of the forwarding node. The structure of the packet becomes:

$$\text{Packet} = [ID_f, ID_o, \{ ID_f, ID_o, \text{type}, \text{data}\}_k]$$

Where $ID_f$ is the ID of the forwarding node and is rewritable by subsequent forwarding nodes. $ID_o$ is the ID of the originating node and the *key* is derived from the energy level of the current forwarding node. Initially, at the originating node $ID_f = ID_o$.

If a hacker does manage to make a packet look legitimate the packet will make it to the sink and we can employ a weighted voting algorithm where the sink can use the majority vote. To trick the sink the hacker would have to compromise at least half the nodes in the network and send data simultaneously. DEEF-T reduces the transmission overhead but increases processing overhead because of the decode/encode at each hop. Analysis of traditional authentication is in the performance evaluation section.

### C. Non-traditional Authentication

In the non-traditional authentication scheme, each node randomly picks *r* nodes to monitor and stores the corresponding state before deployment. Therefore, no routing information and no communication using a network-wide master key is necessary to obtain neighbors' energy values.

As a packet leaves the source node (originating node or forwarding node) it passes through node(s) that watch it probabilistically. If the current node is not watching the node of interest, the packet is forwarded. If the node of interest is being watched by the current node the packet is decoded and the plaintext ID is compared with the decoded ID. If the packet is authentic, and this hop is not the final destination, the original encoded packet is forwarded unless the node is currently bridging the network. In that case, the original packet is encoded with the bridge energy and forwarded. Since this node is bridging the network the local perceived energy is decremented accordingly. If the packet is illegitimate, the packet is discarded. This process continues until the packet reaches the sink. This technique consumes more transmission overhead but in contrast to the traditional DEEF scheme it reduces processing overhead (because less re-encoding is performed and decoding is not performed at every hop). The tradeoff is that an illegitimate packet may traverse several hops before being dropped. The effectiveness of this scheme depends primarily on the value *r*, the number of nodes that each node watches. A discussion of general performance of this algorithm efficiency as a function of *r* is given in the performance evaluation section.

Note that in this scheme re-encoding is not done at forwarding nodes unless they are bridging the network. Accordingly the packet structure is [ID,{ID, type, data}$_k$], where the ID will always be the ID of the current node and the key is derived from the current node's local energy value.

If a hacker does manage to make a packet look legitimate, the packet will make it to the sink. Again, we can employ weighted voting. Analysis of non-traditional authentication is in the performance evaluation section.

[1] Due to space limitations the detailed algorithm was not presented.

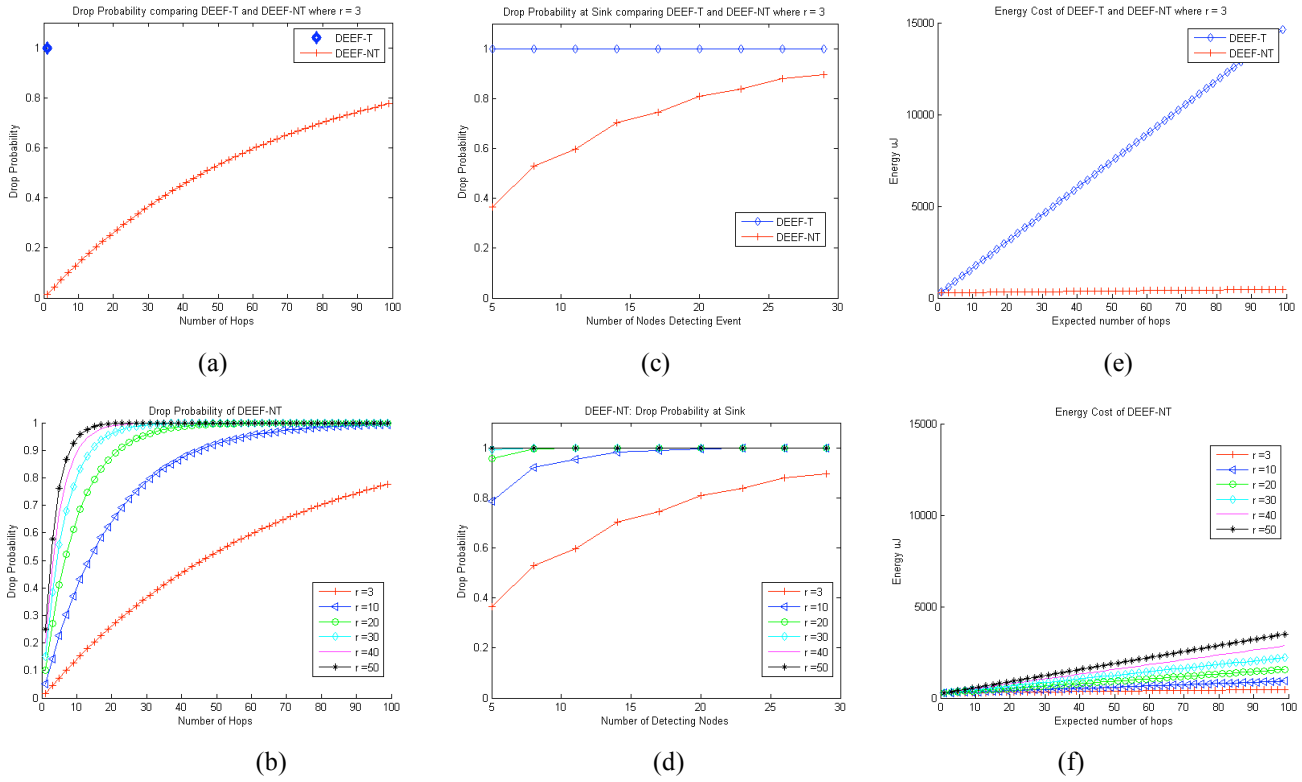


Figure 5. (a) Effectiveness of DEEF-T vs. DEEF-NT where $r = 3$; (b) Effectiveness of DEEF-NT with varying $r$; (c) Sink drop probability considering event detection set size DEEF-T vs. DEEF-NT where $r = 3$; (d) Sink drop probability considering event detection set size for DEEF-NT with varying $r$; (e) Average cost to send valid packet in DEEF-T and DEEF-NT where $r = 3$; (f) Average cost to send valid packet in DEEF-NT with varying $r$.

## VII. Performance Evaluation

In this section we use analysis to quantify the effectiveness of DEEF-T and DEEF-NT. Next we examine how the density of the network affects the chances of false packets successfully traversing the network and being processed by the sink when the sink employs a simple majority vote approach. Finally, we examine the cost of our algorithm on legitimate packets considering current sensor applications.

When considering the effectiveness of the algorithms we target the case where the sink is considered an average node with average resources. That is, the sink is not assumed to store (and maintain) keys for every node in the network. In some situations it may be impossible to have a powerful sink. In taking this approach, we attempt to ascertain the impact (effectiveness/energy costs/consequences) on distributing the resource load.

For analysis, we use the following values in evaluating our equations: number of nodes in the network, $N = 200$, and packetsize = 32.

### A. *Effectiveness of DEEF-T and DEEF-NT*

In this section we compare DEEF-T and DEEF-NT considering the drop probability vs. number of hops. We also take a closer look at DEEF-NT and vary the number of records $r$.

In DEEF-T, in order for an attacker to be able to successfully inject a false packet, an attacker must forge the packet encoding which is a result of the keyed hash. Given that the complexity of the packet is $2^{packetsize}$ the probability of an attacker correctly forging the packet is:

$$\frac{1}{2^{packetsize}} = \frac{1}{2^{ID+TYPE+DATA}} \quad (1)$$

Accordingly, the probability of the hacker incorrectly forging the packet and therefore the packet being dropped ( $p_{drop-T}$ ) is:

$$1 - \frac{1}{2^{packetsize}} = 1 - \frac{1}{2^{ID+TYPE+DATA}} \quad (2)$$

Since DEEF-T authenticates at every hop, forged packets will always be dropped at the first hop with a probability of $p_{drop-T}$ .

Since DEEF-NT statistically drops packets along the route, the drop probability for DEEF-NT ( $p_{drop-NT}$ ) is a function of the effectiveness of the watching nodes as well as the ability for a hacker to correctly guess the encoded packet structure. Accordingly, the probability of detecting and dropping a false packet at one hop when randomly choosing *r* records (nodes to watch) is:

$$p_{drop-NT} = \frac{\mathrm{r}}{N} * \left(1 - \frac{1}{2^{ID+TYPE+DATA}}\right) \quad (3)$$

Thus, the probability to detect and drop the packet when choosing *r* records after *h* hops is:

$$1 - (1 - p_{drop-NT})^h \quad (4)$$

With the assumption that the packet size is 32 bits and if we use the same storage requirements for both DEEF-T and DEEF-NT of $r = 3$ we can compare both schemes in Figure 5a when using the same amount of storage. Recall from section VI-B that DEEF-T requires $r = 3$. Figure 5b shows the effectiveness of DEEF-NT as the number of stored records vary.

*B. Probability packets span network and dropped at sink*

As discussed in sections VI-A and VI-B, we assume that the sink has the ability to take a majority vote to determine which data to process and accept. Therefore, even if one false packet successfully makes it to the sink it is still far less likely for the sink to process the data because multiple (more than half) nodes must be compromised during an event detection in order to make the sink process false data. The number of nodes necessary to compromise the integrity of the processed data for DEEF-T and DEEF-NT is:

$$x = floor\left(\frac{\#nodes\_detect\_event}{2}\right) + 1 \quad (5)$$

In order for packets to successfully traverse the network and trick the sink during an event detection sequence an attacker would have to simultaneously forge data from *x* nodes in the detection set. The probability of unsuccessfully injecting traffic simultaneously from *x* compromised nodes (and thus the sink dropping packet) for DEEF-T (6) and DEEF-NT (7) is:

$$1 - \left(\frac{1}{2^{ID+TYPE+DATA}}\right)^x \quad (6) \qquad 1 - \left[(1 - p_{drop-NT})^h\right]^x \quad (7)$$

Figure 5c compares DEEF-T and DEEF-NT with $r = 3$. Figure 5d presents DEEF-NT varying *r*. As the local node density increases, either by increasing the number of physical nodes or the sensing range of nodes, it becomes increasingly difficult for false data to successfully traverse the network and trick the sink. This is particularly useful in DEEF-NT because the drop probability increases with the number of hops the packets span. However, if the hacker injects malicious packets a few hops away from the sink (e.g., $h = 10$), DEEF-NT without employing a majority vote at the sink becomes less effective. Using the majority vote approach given a low hop count (e.g., $h = 10$) increases the overall drop probability from 14% (Figure 5b) to 90% (Figure 5d) when $r = 3$ and the number of detecting nodes is 30.

*C. Energy cost comparing DEEF-T and DEEF-NT*[2]

In this section we look at the cost to transmit valid data in DEEF-T and DEEF-NT. In both algorithms, there is a single cost to detect the event, encode the packet, and transmit the packet ( $E_{\mathrm{d}}, E_{enc}, E_t$ ). Additionally, there is a recurring cost to marshal the packet through the network depending on the number of hops. In DEEF-T this cost is $\left(E_r, E_{dec}, E_{enc}, E_t\right)$ for all of the intermediate nodes since all of the nodes perform the same operations. In DEEF-NT the cost is $\left(E_r, E_{dec}, E_t\right)$ or $\left(E_r, E_t\right)$ for variable fractions of the forwarding nodes depending on the number of nodes each node chose to watch. Finally, there is a single cost to receive and decode the packet at the sink ( $E_r, E_{dec}$ ). The average cost to transmit a packet using DEEF-T is:

$$E_d + E_{enc} + E_t + \left[\begin{matrix} E[numhops] \bullet \\ \left(E_r + E_{dec} + E_{enc} + E_t\right) \end{matrix}\right] + E_r + E_{dec} \quad (8)$$

The average cost to transmit a packet using DEEF-NT is:

$$E_{\mathrm{d}} + E_{enc} + E_t + \left[\begin{matrix} E[numhops] \bullet \\ \left\langle \begin{matrix} (\%forward\_enc\_dec) \bullet \left(E_r + E_{dec} + E_t\right) \\ +(\%forward) \bullet \left(E_r + E_t\right) \end{matrix} \right\rangle \end{matrix}\right] + E_r + E_{dec} \quad (9)$$

To determine the average energy used for DEEF-NT to transmit a valid packet we must have an idea of the number of nodes along the path that are likely to be watching a specific node at a specific time. We first determine the number of possible record arrays ***y***. We let *r* equal the number of records and *N* equal the number of nodes in the network. This is:

$$y = \left(\frac{(N)!}{r!(N-r)!}\right) \quad (10)$$

Next we determine the possible number of record arrays that do not have a specific record in common. Therefore, the number of possible record arrays ***q*** that exclude a specific record is:

$$q = \left(\frac{(N-1)!}{r![(N-1)-r]!}\right) \quad (11)$$

[2] In order to simplify calculations bridging is not considered in this analysis.

Accordingly, the number of record arrays that contain the same record of interest is $(y-q)$. We can calculate the percentage of nodes in the network that have the record of the node of interest (12) and the percentage that do not contain the record of the node of interest (13). This is:

$$prob(have\_record) = \frac{(y-q)}{y} \quad (12)$$

$$prob(do\_not\_have\_record) = \frac{q}{y} \quad (13)$$

To make the analysis possible, we make the assumption that the network is equally distributed. That is, if a cross-section of the network is sampled, it will have the same node composition as the entire network. Accordingly, to determine the number of nodes along a given path that share a record (encode/decode and forward a packet associated with a watched node) with a particular number of expected hops we can apply equations (12) and (13) to the number of nodes along the path. Taking energy values ($E_t = 65\mu J$, $E_r = 50\mu J$, $E_{dec} = 16\mu J$, $E_{enc} = 15\mu J$)[3] from [32] we can perform analysis. Figures 5e and 5f compares the average cost to transmit a packet through the network for DEEF-T and DEEF-NT assuming $\boldsymbol{r}$ = 3 while varying the number of hops taken.

### D. Discussion

We note the following observations for the above scenarios. While DEEF-T offers 99% drop probability of illegitimate packets at the first hop (Figure 5a), it more than doubles the amount of energy consumed for legitimate traffic in DEEF-NT at the fifth hop (Figure 5e). At the 15th hop, DEEF-T consumes 3 to 6 times more energy than DEFF-NT, and the separation of energy costs significantly increases as the network scales.

However, when storage is a concern and DEEF-NT is watching the same number of nodes as DEEF-T (i.e., $r$ = 3), DEEF-NT can achieve at best 77% drop probability at the 100th hop. At which point the illegitimate packet would have consumed 52% more energy than DEEF-T. On the other hand, DEEF-NT can achieve at least a 90% drop probability within 9 hops if each node is watching at least 25% of the network ($r$ = 50). Requiring more storage, however giving the benefit of a high drop probability within a few hops and lowering energy costs when dealing with legitimate packets.

From this analysis, one could argue that DEEF-T is better suited where storage is the most critical resource and where the network tends to be less chatty, whereas DEEF-NT is most beneficial for chatty networks and/or when storage is not the crippling resource.

[3] We could not find a value for $E_d$ so we chose a reasonable value. This is inconsequential in the comparison of DEEF-T and DEEF-NT as both equations contain $E_d$ with the same frequency.

## VIII. CONCLUSION AND FUTURE WORK

In this paper we present the Dynamic Energy-based Encoding and Filtering (DEEF) framework. Using a simple encoding scheme and dynamic energy-based keying, DEEF allows flexibility in the approach to removing false data from the network. We have evaluated DEEF's feasibility and performance through analysis. Our results show that DEEF-T and DEEF-NT can be configured to provide optimal performance in a variety of network configurations depending largely on the application of the sensor network.

We plan to extend this work by performing simulations to determine optimal bin sizes for energy. Further, we plan to enable our technique to deal with loss of synchronization (from unreliable MAC, retransmissions, imprecise binning, etc.). We also plan to conduct experiments to obtain variability of sensor energy levels when performing certain actions (e.g., encoding, etc.) to ensure optimal binning. Simulations and additional analysis will be performed to determine how packet size affects performance of the authentication technique used (DEEF-T or DEEF-NT). Also, we plan to implement our technique in conjunction with current sensor routing algorithms. Finally, we plan to introduce the concept of meeting nodes (DEEF-MN) in DEEF in order to detect malicious insider attacks.

## REFERENCES


[1] Simon G., Maroti M., Ledeczi A., Balogh G., Kusy B., Nadas A., Pap G., Sallai J., Frampton K.: Sensor Network-Based Countersniper System, SenSys 04, Baltimore, November 2004.

[2] Ledeczi, A., Nadas, A., Volgyesi, P., Balogh, G., Kusy, B., Sallai, J., Pap, G., Dora, S., Molnar, K., Maroti, M., Simon G.: "Countersniper System for Urban Warfare," ACM Transactions on Sensor Networks, Vol. 1, No. 2, pp. 153-177, November, 2005

[3] Chinh Vu, Raheem Beyah, and Yingshu Li. "A Composite Event Detection in Wireless Sensor Networks." To appear in the Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC), April 2007.

[4] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: A link layer security architecture for wireless sensor networks," in *Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[5] Hannu Sikkil, Mikael Soini, Petri Oksa, Lauri Syd heimo, and Markku Kivikoski, KILAVI Wireless Communication Protocol the Building Environment - Security Issues, Consumer Electronics, 2006.

[6] Wen-Huei Chen and Yu-Jen Chen, A Bootstrapping Scheme for Inter-Sensor Authentication within Sensor Networks, IEEE Communications Letters, Oct. 2005.

[7] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney. A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks, Proceedings of the 10th ACM conference on Computer and Communications Security, 2003.

[8] Wenliang Du and Ronghua Wang,Peng Ning, An Efficient Scheme for Authenticating Public Keys in Sensor Networks, Proceedings of the 6th ACM International Symposium on Mobile Ad hoc Networking and Computing, pp. 58 – 67, 2005.

[9] Chin-Fu Kuo, Yung-Feng Lu, Ai-Chun Pang, Tei- Wei Kuo, Security-Enhanced Data Delivery In Sensor Networks, Proceedings of the 39th Annual 2005 International Carnahan Conference on Security Technology, Oct. 11-14, 2005.

[10] Xiaojiang Du, Yang Xiao, Hsiao-Hwa Chen, Qishi Wu, Secure Cell Relay Routing Protocol for Sensor Networks, Wireless Communications and Mobile Computing, Volume 6, Issue 3, pp. 375-391,May 2006.

[11] F Ye, H Luo, S Lu, L Zhang, Statistical En-Route Filtering of Injected False Data in Sensor Networks, IEEE Journal on Selected Areas in Communications, vol. 23, no. 4, April 2005.

[12] J Deng, R Han, S Mishra, INSENS: Intrusion-tolerant Routing for Wireless Sensor Networks, Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems, Providence, RI, May 2003.

[13] Liang Zhang, A Self-Adjusting Directed Random Walk Approach for Enhancing Source-Location Privacy in Sensor Network Routing, Proceedings of the 2006 International Conference on Communications and Mobile Computing, pp. 33 – 38, 2006.

[14] Young-Jin Kim, Ramesh Govindan, Brad Karp, Scott Shenker, Lazy cross-link removal for geographic routing, In Proc Sensys, 2006.

[15] Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri, Power Aware Routing for Sensor Databases, In Proc IEEE Infocom, 2005.

[16] Antonio Caruso, Stefano Chessa, Swades De, and Alessandro Urpi, GPS free coordinate assignment and routing in wireless sensor networks, In Proc IEEE Infocom, 2005.

[17] Jehoshua Bruck, Jie Gao, Anxiao (Andrew) Jiang, MAP: medial axis based geometric routing in sensor networks, In Proc Mobicom, 2005.

[18] B. Deb, S. Bhatnagar, and B. Nath. ReInForM: Reliable Information Forwarding Using Multiple Paths in Sensor Networks. The 28th Annual IEEE Conference on Local Computer Networks (LCN), October 2003.

[19] Kuhn, R. Wattenhofer and A. Zollinger. worst-case optimal and average-case efficient geometric ad hoc routing. In Proc 4th ACM International Synposium on Mobile Ad-hoc Networking and Computing, pages 267-278, Annapolis, MD, 2003.

[20] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, Fabio Silva, Directed diffusion for wireless sensor networking, IEEE/ACM Transactions on Networking (TON), Volume 11 Issue, February 2003.

[21] Dragos Niculescu, Badri Nath, Routing and forwarding: Trajectory based forwarding and its applications, Proceedings of the 9th annual international conference on Mobile computing and networking, September 2003.

[22] Thomas Martin and Daniel Seiwiorek, "Non-Ideal Battery Behavior and Its Impact on Power Performance Trade-offs in Wearable Computing," Proceedings of the 1999 International Symposium on Wearable Computers, San Francisco, CA, October 18-19, 1999; pp. 101-106.

[23] Jason Flinn, M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," wmcsa, p. 2, Second IEEE Workshop on Mobile Computer Systems and Applications, 1999.

[24] Phillip Stanley-Marbell and Michael Hsiao. Fast, Flexible, Cycle-Accurate Energy Estimation. Proceedings of the International Symposium on Low Power Electronics and Design, Huntington Beach, California, 2001.

[25] Ping-Wen Ong and Ran-Hong Yan. Power-Conscious Software Design – A Framework for Modeling Software on Hardware. IEEE Symposium on Low Power Electronics, 1994.

[26] Grant A. Jacoby, Randy Marchany and Nathaniel J. Davis IV, "Battery-Based Intrusion Detection: a First Line of Defense," Proceedings of the 5th IEEE SMC 2004 Information Assurance Workshop, June 2004.

[27] Grant A. Jacoby and Nathaniel J. Davis IV, "Battery-Based Intrusion Detection," GlobeComm 2004, December 2004.

[28] Timothy K. Buennemeyer, Grant A. Jacoby, Randolph C. Marchany, and Joseph G. Tront, "Battery-Sensing Intrusion Protection System," Proceedings of the 7th IEEE SMC 2006 Information Assurance Workshop, June 2006.

[29] Grant A. Jacoby, Randy Marchany and Nathaniel J. Davis IV, "How Mobile Host Batteries Can Improve Network Security," IEEE Security & Privacy Magazine, September 2006.

[30] Thomas Martin, Michael Hsiao, Dong Ha, Jayan Krishnaswami, "Denial-of-Service Attacks on Battery-powered Mobile Computers," Proceedings of the 2nd IEEE Pervasive Computing Conference, Orlando, Florida, March 2004, pp. 309-318.

[31] Timothy Buennemeyer, Theresa Nelson, Michael Gora, Randy Marchany and Joseph Tront. Battery Polling and Trace Determination for Bluetooth Attack Detection in Mobile Devices. Proceedings of the 5th IEEE SMC 2004 Information Assurance Workshop, June 2007.

[32] Crossbow. www.xbow.com