CERIAS Tech Report 2010-09
Assessing the Trustworthiness of Streaming Data
by Hyo-Sang Lim, Yang-Sae Moon, Elisa Bertino
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Assessing the Trustworthiness of Streaming Data

Hyo-Sang Lim
Department of Computer Science and CERIAS
Purdue University, Indiana, USA
hslim@cs.purdue.edu

Yang-Sae Moon
Department of Computer Science
Kangwon National University, South Korea
ysmoon@kangwon.ac.kr

Elisa Bertino
Department of Computer Science and CERIAS
Purdue University, Indiana, USA
bertino@cs.purdue.edu

Abstract

The notion of *confidence policy* is a novel notion that exploits trustworthiness of data items in data management and query processing. In this paper we address the problem of enforcing confidence policies in data stream management systems (*DSMSs*), which is crucial in supporting users with different access rights, processing confidence-aware continuous queries, and protecting the secure streaming data. For the paper, we first propose a DSMS-based framework of confidence policy management and then present a systematic approach for estimating the trustworthiness of data items. Our approach uses the data item provenance as well as their values. We introduce two types of data provenance: the *physical provenance* which represents the delivering history of each data item, and the *logical provenance* which describes the semantic meaning of each data item. The logical provenance is used for grouping data items into semantic events with the same meaning or purpose. By contrast, the tree-shaped physical provenance is used in computing trust scores, that is, quantitative measures of trustworthiness. To obtain trust scores, we propose a cyclic framework which well reflects the *inter-dependency* property: the trust scores of data items affect the trust scores of network nodes, and vice versa. The trust scores of data items are computed from their *value similarity* and *provenance similarity*. The value similarity comes from the principle that "the more similar values for the same event, the higher the trust scores," and we compute it under the assumption of *normal distribution*. The provenance similarity is based on the principle that "the more different physical provenances with similar values, the higher the trust scores," and we compute it using the tree similarity. Since new data items continuously arrive in DSMSs, we need to evolve (i.e., recompute) trust scores to reflect those new items. As evolution scheme, we

propose the *batch mode* for computing scores (non)periodically along with the *immediate mode*. To our best knowledge, our approach is the first supporting the enforcement of confidence policies in DSMSs. Experimental results show that our approach is very efficient.

1 Introduction

A large variety of novel applications, like supervisory systems, e-health, and e-surveillance, are characterized by real-time data streaming [17, 18] and continuous query processing [15, 19, 21]. Continuous queries can be issued for a variety of purposes, ranging from real-time decision making to statistical analysis and machine learning applications. These applications often have different requirements with respect to data trustworthiness, and so we need flexible data management systems allowing the various applications to specify "how trustworthy" certain data need to be for each application. As an example, we can consider a data stream management system (*DSMS* in short) for monitoring battlefield. The DSMS gathers enemy locations from various sensors deployed in vehicles, aircrafts, and satellites and processes the continuous queries over those streaming data. In this DSMS, we must limit mission critical applications to access only high confidence data in order to guarantee accurate decisions. By contrast, analytic applications (e.g., network management) can access low confidence data for the purpose of detecting possible sensor errors or malicious sabotage over the sensor network. This example shows that it is important to assess the trustworthiness of data items and control the use of data based on the purpose of use and trustworthiness level of the data.

A possible approach to address such issue is based on two elements. The first element is the association of a *trust score* with each data item (or group of data items). Such score provides an indication about the trustworthiness of the data item and can be used for data comparison or ranking. For example, even though the meaning of absolute scores varies depending on the application or parameter settings, if a data item has the highest trust score in a data set, then we can say that the data item is most trustworthy compared with the other data items in the set. Also, as indicators about data trustworthiness, trust scores can be used together with other factors (e.g., information about contexts and situations, past data history) for deciding about the use of data items.

The second element of the approach is based on the notion of *confidence policy* [8]. Such a policy specifies for an application or task the minimum trust score that a data item, or set of data items, must have for use by the application or task. Our confidence policy language includes many other clauses (not presented in the paper for lack of space) that can be used to formulate articulated conditions about data usage. As such our confidence model subsumes very well known integrity models, like the Biba's model [6] and the Low Water-mark model [12]. Confidence policies are integrated with query processing in that query results are filtered by the policies before being returned to the application. We refer to queries enforcing such policies as *confidence policy-compliant queries*.

The goal of this paper is to develop a DSMS-based framework, focused on sensor networks, that supports confidence policy-compliant queries and, as a core component, provides a systematic approach for estimating and managing trust scores of data. Especially, we focus on assessing trust scores since it is a crucial building block for the confidence policy management. Our approach is based on the concept of provenance, as provenance gives important evidence about the origin of the data, that is, where and how the data is generated. Provenance provides knowledge about how the data came to be in its current state - where the data originated, how it was generated, and the operations it has undergone since its creation.

In order to use data provenance in computing trust scores, we introduce two types of data provenance for each data item: *physical provenance* and *logical provenance*. The physical provenance shows where the data item was produced and how it was delivered; by contrast, the logical provenance represents its semantic meaning or unit in the given application. We use the physical provenance for computing trust scores and the logical provenance for uniquely identifying a logical event for each data item. We then propose a formal method for computing trust scores. Our method is based on the principle that the more trustworthy data a source provides, the more trusted the source is considered. There is thus an interdependency between network nodes and data items with respect to the assessment of their trust scores, i.e., trust score of the data affects the trust score of the network nodes that created and manipulated the data, and vice-versa. To reflect such interdependency property in computing trust scores, we propose a cyclic framework that generates (1) trust scores of data items from those of network nodes and (2) trust scores of network nodes from those of data items. Trust scores are gradually evolved in our cyclic framework.

Our framework works as follows. Trust scores are initially computed based on the values and provenance of data items; we refer to these trust scores as *implicit trust scores*. To obtain these trust scores, we use two types of similarity functions: *value similarity* inferred from data values, and *provenance similarity* inferred from physical provenances. Value similarity is based on the principle that the more data items referring to the same real-world event have similar values, the higher the trust scores of these items are. We observe that most sensor data referring to the same event follow *normal distribution*, and propose a systematic approach for computing trust scores based on value similarity under the normal distribution. Provenance similarity is based on the observation that different physical provenances of similar data values may increase trustworthiness of data items. In other words, different physical provenances provide more independent data items. In the paper we thus present a formal model for computing the provenance similarity and integrating it into the normal distribution framework.

In the paper, we also address the problem of efficiently computing trust scores in a streaming environment. Trust scores in our cyclic framework can be modified whenever a new data item arrives. Such strategy, referred to as *immediate mode*, immediately reflects changes of data values onto trust scores. Such a strategy is not, however, applicable when input rates are very high. To address the problem of performance overhead, we propose a *batch mode*. Under such a mode trust scores are periodically (or non-periodically) evolved for a (possibly very large) set of accumulated data items. The batch mode provides comparable accuracy with respect to the immediate mode.

We again exploit the normal distribution nature of data streams to decide exactly when the evolution should be executed in the batch mode. More specifically, we recalculate trust scores of network nodes only when recent input data items do not follow the normal distributions any longer. Otherwise, trust scores of data items are calculated with old trust scores of network nodes.

We have implemented the provenance-based confidence policy model and the cyclic framework for computing trust scores. Through extensive experiments, we first showcase that our confidence model works correctly in DSMSs, and the cyclic framework gradually evolves trust scores by reflecting changes in data streams. We also show that the batch mode is more flexible than the immediate mode in adjusting accuracy and performance. These experimental results show that our provenance-based model and cyclic framework provide a practical way of supporting the confidence policy in DSMSs.

The rest of the paper is organized as follows. Section 2 presents the provenance-based model for enforcing the confidence policy in DSMSs and introduces notions of physical and logical provenances. Section 3 proposes the cyclic framework for generating trust scores of data items and network nodes based on their values and provenance. Section 4 describes the immediate and batch modes as methods for evolving trust scores. Section 5 reports the experimental results. Section 6 discusses related work on stream data processing and confidence policies. We finally summarize and conclude the paper in Section 7.

2 Provenance-based Model of Confidence Policy Control

In this section, we show an overview of our confidence policy management framework, and then, define physical and logical provenances. In the proposed framework, we exploit our previous work [5, 7, 8] on confidence policy and trustworthiness assessment. However, our work is totally different both in application targets and technical solutions. We will discuss it in the related work section (Section 6).

2.1 Confidence Policy Management over Data Streams

Figure 1 shows our overall framework for confidence policy management on data streams. The figure shows how the sensor data are processed and managed in a DSMS and how they are delivered to users. As shown in the figure, the proposed framework consists of three major components: *trust score computation*, *query and policy evaluation*, and *data quality management*. The role of each component is as follows:

- *Trust score computation* obtains trust scores of data items based on those of network nodes and (periodically) updates the trust scores to reflect the effect of the newly arrived data items. It also maintains and updates trust scores of network nodes based on the scores of data items.
- *Query and policy evaluation* executes continuous queries, each of which has its own *confidence range*. We assume that each continuous query Q is given with its

confidence range $[q_{max}, q_{min}]$. For each Q with $[q_{max}, q_{min}]$, this component first obtains the resulting data items by evaluating the given query Q and then returns data items of which trust scores are in between q_{max} and q_{min}.

- *Data quality management* tries to control data quality (manually or automatically) by adjusting data rates, increasing/decreasing the number of sensor nodes, or changing delivery paths. Obviously, data quality affects trust scores, and many approaches [9, 15, 21] on sensor networks have addressed the issue of controlling data quality.

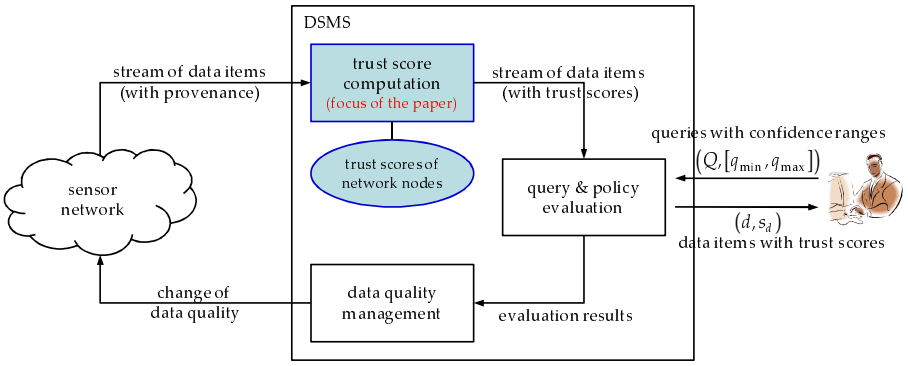


Figure 1: An overall framework of confidence policy control over data streams.

In this paper we focus on the first component so as to get reasonable trust scores of data items and network nodes.

2.2 Physical Provenance and its Representation

The physical provenance of a data item shows where the item was produced and how it was delivered to the server. We show in Section 3 how to exploit this physical provenance to compute trust scores of data items and network nodes.

A network is usually modeled as a graph, and we thus model the physical (sensor) network as a graph of $G(N, E)$. Figure 2 (a) shows an example of a physical sensor network. In the graph $G(N, E)$, a set of nodes, N, and a set of edges, E, are defined as follows:

- $N = \{n_i \mid n_i$ is a network node of whose identifier is $i.\}$: a set of network nodes
- $E = \{e_{i,j} \mid e_{i,j}$ is an edge connecting nodes n_i and $n_j.\}$: a set of edges connecting nodes

Regarding the network nodes in N, we categorize them into three types according to their roles.

Definition 1 A *terminal node* generates a data item and sends it to one or more intermediate or server nodes. An *intermediate node* receives data items from one or more terminal or intermediate nodes, and it passes them to intermediate or server nodes; it may also generate an aggregated data item from the received data items and send the aggregated item to intermediate or server nodes. A *server node* receives data items and evaluates continuous queries based on those items. □

Without loss of generality, we assume that there is only one server node, denoted by n_s, in G.

Prior to formally defining the physical provenance, we introduce two assumptions related to data values and intermediate node operations.

Assumption 1 A data item d has only one numeric attribute, and it carries a numeric value v_d and a provenance p_d for that attribute. □

Assumption 1 is reasonable due to the following reasons. In general, a sensor node has only one purpose, and it thus reports only one numeric value. Even for a data item with two or more attributes, we can regard it as two or more different data items since different attributes may have their own trust scores. With this observation, we can easily extend our solution to multiple attributes. Each attribute can be handled separately by assigning independent scores for each attributes. We can also use multi-attribute distributions and Euclidian distance to directly extend our solution for multiple attributes.

Assumption 2 Single-attribute operations such as selection and aggregation are allowed in *intermediate* nodes, but multi-attribute operations such as join and projection are not allowed. □

In sensor networks, some intermediate nodes may execute data operations such as selection (or shedding), projection, join, and aggregation to reduce the network load or the server processing load. However, to simplify the presentation, we focus on handling selection and aggregation which are the most used operations in sensor networks. Join operations are similar to aggregation since they both combine data from multiple nodes. We will explore additional other operations in our future work.

We now define the *physical provenance* of a data item d, denoted it as p_d. The physical provenance p_d shows where and how the data item d was generated and how it was passed to the server n_s.

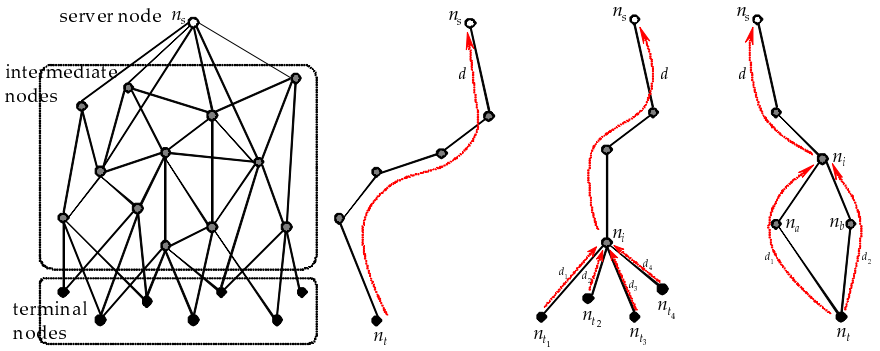
Definition 2 The *physical provenance* p_d of a data item d is a rooted tree satisfying the following properties: (1) p_d is a subgraph of the physical sensor network $G(N, E)$; (2) the root node of p_d is the server node n_s; (3) for two nodes n_i and n_j of p_d, n_i is a child of n_j if and only if n_i passes the data item d to n_j. □

Based on the tree nature, we can also categorize the types of nodes for physical provenance trees as: root, internal, and terminal nodes. Each of these nodes has the form of a (node, a set of edges for children) pair as follows:

- Root node $n_s = (n_s, \{e_{s,i} \mid n_i \text{ is a child of } n_s\})$, where n_i can be an internal or leaf node.
- Internal node $n_i = (n_i, \{e_{i,j} \mid n_j \text{ is a child of } n_i\})$, where n_j can be another internal or a leaf node.
- Leaf node $n_i = (n_i, \emptyset)$.

We also categorize intermediate nodes into two types based on their operations. First, we call the internal node having one child a *simple node*, which simply passes a data item from its child to its parent. Simple nodes are typical in ad-hoc sensor networks which relay data items to a server to address the insufficient capability of data transmission. Second, we call an intermediate node having two or more children an *aggregate node*, which receives multiple data items from multiple children, generates an aggregated data item, and passes it to its parent.

Figures 2 (b) and 2 (c) show some examples of the two different physical provenances. As shown in the figures, physical provenances are subgraphs of the physical sensor network of Figure 2 (a), and they are trees rooted at the server node n_s. In Figure 2 (b) every node in the physical provenance p_d is a simple node, which means that the data item d is generated in a terminal node n_t and simply passed to the server n_s. We call this type provenance a *simple provenance*, which can be represented as a simple path. On the other hand, in Figure 2 (c) an internal node n_i is an aggregate node, which means that n_i generates a new data item d by aggregating multiple data items $d_1, \ldots, d_4$ from $n_{t_1}, \ldots, n_{t_4}$ and passes d to the server n_s. We call this type provenance an *aggregate provenance*, which is represented as a tree rather than a simple path.



(a) A physical sensor network (b) A simple provenance (c) An aggregate provenance (d) An exception

Figure 2: A physical sensor network and physical provenance examples.

According to Definition 2, a physical provenance should be a tree. However, there could be cycles in which the provenance is not a tree. Consider the example in Figure 2 (d), a terminal node n_t generates two data items d_1 and d_2; d_1 is passed to n_i through n_a while d_2 is passed to n_i through n_b; the aggregate node n_i generates a new data

item d by aggregating d_1 and d_2; the data item d is finally passed to the server n_s. In this case the provenance is still a subgraph of $G(N, E)$, but it is not a tree. We do not consider this case because of two reasons. First, it rarely occurs in real environments. Second, computing tree similarity can be done in $O(n^3 \log n)$ [16]; in contrast, computing graph similarity is known as an NP-hard problem [14] in general (refer to Section 3 for details). We note that basically there is no much difference between tree-shaped and graph-shaped provenances. Only minor changes are required to support graph-based provenance. In practice, the system will support both types of provenance and let the application tradeoff between efficiency and accuracy.

2.3 Logical Provenance and its Representation

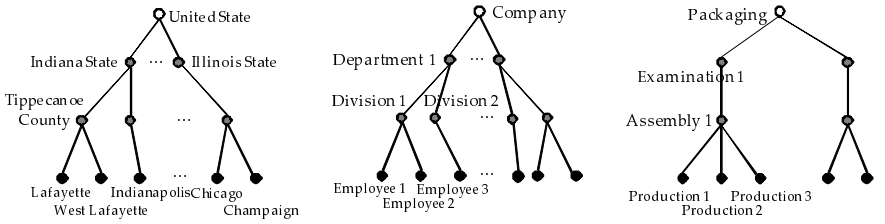
The logical provenance of a data item represents the semantic meaning of the data item in the context of a given application, and it generally differs from the physical provenance which represents the actual transmission information of the data item. For example, the logical provenance can be a tracking of locations by which the data item was issued, a chain of employers who used the data item, or a trail of business logics that processed over the data item. In some applications, the logical provenance can be identical to the physical provenance. In a business workflow system, for example, if the physical network is constructed by following the workflow graph, the logical provenance becomes identical to the physical provenance. To deal with more general applications, however, we assume that the logical provenance in the logical network differs from the physical provenance in the physical network. We use this logical provenance to identify logical events and to assign data items to those events.

We model the logical network as a tree of $T(M, F)$ where a set of nodes, M, and a set of edges, F, are defined as follows:

- $M = \{m_i \mid m_i$ is a semantic node whose identifier is $i.\}$: a set of semantic entities (i.e., each semantic entity is mapped to a node in the tree T.)
- $F = \{f_{i,j} \mid f_{i,j}$ is a (directed) edge from node m_i to $m_j.\}$: $f_{i,j}$ means that the semantic meaning of m_i includes that of m_j, i.e., there is an inclusion relationship of $m_i \subseteq m_j$.

We model the logical network as a tree because many semantic structures have their own hierarchies that represent semantic inclusion relationships among application entities. Figure 3 shows some typical examples of logical networks. Figure 3 (a) shows a hierarchy of geographic locations where the area of a parent node contains that of its child node. Similarly, an organization chart and a business process in Figures 3 (b) and 3 (c) are modeled as rooted trees based on their own semantic hierarchies.

Based on the logical network, we now define the *logical provenance* of a data item d, which is denoted as l_d. The logical provenance l_d, which shows the semantic meaning of the data item d, can be simply represented as a set of nodes (i.e., semantic entities) in the logical network. Considering the tree structure of the logical network we define the logical provenance as a simple path as follows:



(a) Geographic location (b) Organization chart (c) Business process

Figure 3: Examples of the logical network.

Definition 3 The *logical provenance* l_d of a data item d is a simple path of nodes in the logical network where (1) every node in l_d includes the semantic meaning of d, (2) the start node of l_d is the lowest node in the tree $T(M, F)$, and (3) the end node of l_d is always the root node of $T(M, F)$. □

In Definition 3 the start node of l_d is generally a leaf node in the tree, but it can also be an internal node if the data item is issued from an aggregate node in the *physical* network. For example, if a temperature data item is generated from a sensor of "West Lafayette" (Figure 3 (a)), its logical provenance will be the path of (West Lafayette, Tippecanoe County, Indiana State, United States); if throughput measures of employees 1 and 2 are aggregated in Division 1 (Figure 3 (b)), the logical provenance of the aggregated data item will be (Division 1, Department 1, Company).

We use the logical provenance for event identification which is a process of grouping data items into semantic events. This process is essential to assess trust scores since the score means how much a data item truly reflects a specific event. Event identification consists of two steps: (1) choosing a set of semantic events from the logical network and (2) identifying events for data items. Both steps are executed by the sever node.

Choosing a set of semantic events: We choose a set of semantic nodes, M_{event} ($\subset M$) from the logical network $T(M, F)$. M_{event} represents events that uniquely identify the semantics of data items. For example, in the logical network for geometric location (in Figure 3(a)), if we want to uniquely identify events in the city level, $M_{event} = \{$Lafayette, Indianapolis, Chicago$\}$. If in the state level, $M_{event} = \{$Indiana, Illinois$\}$. M_{event} can be arbitrary chosen according to application requirements by either humans or systems, but there are two rules that should be satisfied to identify events uniquely and completely.

- Rule 1 (uniqueness): There should be no common nodes among subtrees whose root node is $m_i \in M_{event}$.
- Rule 2 (completeness): In all possible logical provenances l_d (i.e., a path from a leaf node to the root node in $T(M, F)$), there should be at least one $m_i \in M_{event}$ in l_d.

Identifying events: We identify which event is reflected in a data item d. This can be done with a simple procedure to find a common node m_{event} between l_d and M_{event} and to assign the semantic meaning (i.e., event) of m_{event} to d. According to the rules 1 and 2 for M_{event}, we can always uniquely identify the event of a data item.

As we already described, the logical provenance is only used to identify semantic events. We note that, if there is no logical provenance in an application, we use the physical provenance for this purpose (e.g., grouping similar physical provenances as an event).

3 Value- and Provenance-based Trust Score Computation

3.1 Cyclic Framework for Incremental Update of Trust Scores

We derive our cyclic framework based on the *interdependency* [5, 8] between data items and their related network nodes. The interdependency means that trust scores of data items affect trust scores of network nodes, and similarly trust scores of network nodes affect those of data items. In addition, trust scores need to be continuously evolved in the stream environment since new data items continuously arrive to the server. Thus, a cyclic framework is adequate to reflect these interdependencies and continuous evolution properties. Figure 4 shows the cyclic framework according to which the trust score of data items and the trust score of network nodes are continuously updated. In Section 2.3 we have already explained how to classify data items into events, and thus computing trust scores will be done for the data items of the same event in a given streaming window.

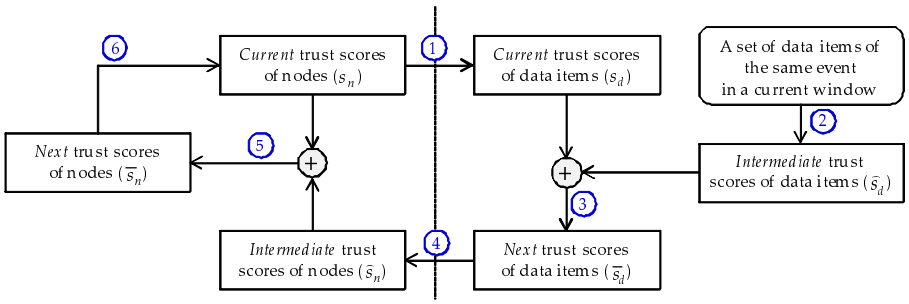


Figure 4: A cyclic framework of computing trust scores of data items and network nodes.

As shown in Figure 4, we maintain three different types of trust scores, *current*, *intermediate*, and *next trust scores* to reflect the interdependency and continuous evolution properties in computing trust scores. Trust scores of data items and network nodes well reflect those properties as many as cycles are repeated. The computation of

trust scores could be iterated until their score becomes less than the given threshold. Repeating multiple cycles for a given streaming window, however, is not adequate in most DSMSs due to the real-time processing requirement. We also note that, since new data items are continuously added the stream, executing the cycle once whenever a new data item arrives is enough to reflect the interdependency and continuous evolution properties in the stream environment. In this paper, we thus execute the cycle of computing trust scores whenever a new item arrives, and we explain in detail the advanced computation modes, i.e., immediate and batch evolution modes in Section 4.

In Figure 4, the current, intermediate, and next trust scores of a data item d are computed as follows: the *current (trust) score* of d, denoted by s_d, is computed from the current trust scores of its related nodes (①); the *intermediate (trust) score* of d, denoted by $\hat{s}_d$, is computed from a set of data items of the same event with d (②); the *next (trust) score* of d, denoted by $\bar{s}_d$, is gradually evolved from its current and intermediate scores (③). We present the detailed computation process for those trust scores of data items in Section 3.3. Next, the current, intermediate, and next trust scores of a network node n are computed as follows: the *current (trust) score* of n, denoted by s_n, is the next trust score assigned to that node at the last stage (⑥); the *intermediate (trust) score* of n, denoted by $\hat{s}_n$ is computed from the next trust scores of data items (④); the *next (trust) score* of n, denoted by $\bar{s}_n$, is computed from its current and intermediate scores (⑤), and becomes its current trust score in the next stage. We explain the detailed computation process for those trust scores of network nodes in Section 3.2.

We note that these scores are used only for comparison purpose. For example, let s_1 and s_2 be trust scores of data d_1 and d_2. If $s_1 > s_2$, d_1 is more trustworthy than d_2. The meaning of absolute scores varies in applications or parameter values.

3.2 Trust Scores of Network Nodes

For a network node n whose current score is s_n, we are about to compute its next score $\bar{s}_n$ In more detail, the trust score of n was computed as s_n in the previous cycle, and we now recompute the trust score as $\bar{s}_n$ using a set of recent data items in a streaming window in order to determine how the trust score is evolved in a new cycle. For a network node n, we compute its next score based on the following two principles: the first principle is that, to consider the interdependency property, the intermediate score $\hat{s}_n$ reflects the trust scores of its related data items; the second principle is that, to gradually evolve trust scores of network nodes, the next score $\bar{s}_n$ reflects its current and intermediate scores s_n and $\hat{s}_n$.

We now show how to compute $\hat{s}_n$ and $\bar{s}_n$. First, we let D_n be a set of data items that are issued from or passed through n in the given streaming window. That is, all data items in D_n are identified to the same event, and they are issued from or passed through the network node n. We adopt the idea that "higher scores for data items ($\in D_n$) result in higher scores for their related node (n)" [5, 7]. Thus, $\hat{s}_n$ is simply computed as the average of $\bar{s}_d$'s ($d \in D_n$), which are the next trust scores of data items in D_n. That is, it is computed as follows:

$$\hat{s}_n = \frac{\sum_{d \in D_n} \bar{s}_d}{|D_n|} \tag{1}$$

In Eq. (1) we note that the trust score of a network node is determined by trust scores of its related data items, and this satisfies the first principle, i.e., the interdependency property. Also, based on s_n and $\hat{s}_n$, the next score $\bar{s}_n$ is computed as follows:

$$\bar{s}_n = c_n s_n + (1 - c_n)\hat{s}_n, \\ \text{where } c_n \text{ is a given constant of } 0 \leq c_n \leq 1. \quad (2)$$

The next score $\bar{s}_n$ is evolved from the current score s_n by the intermediate score $\hat{s}_n$, and it will be used as the current score s_n in the next computation cycle. Likewise, we consider both s_n and $\hat{s}_n$ to obtain $\bar{s}_n$, and this satisfies the second principle, i.e., the consideration of current and intermediate scores.

The constant c_n in Eq. (2) represents how fast the trust score is evolved as the cycle is repeated. The meaning of c_n can be explained as follows. If c_n has a larger value, especially if $c_n > \frac{1}{2}$, we consider s_n to be more important than $\hat{s}_n$, and this means that the previously accumulated historic score (s_n) is more important than the latest trust score ($\hat{s}_n$) recently computed from data items in D_n. On the other hand, if c_n has a smaller value, especially if $c_n < \frac{1}{2}$, we consider the latest score $\hat{s}_n$ to be more important than the historic score s_n. In summary, if c_n is large, the trust score will be evolved slowly; in contrast, if c_n is small, the trust score will be evolved fast. In the experiment we set $c_n = \frac{1}{2}$ to equally reflect the importance of s_n and $\hat{s}_n$, and we assume that the first value of s_n is set to 1.

3.3 Trust Scores of Data Items

Basically we compute the trust score of a data item d using its value v_d and physical provenance p_d. As we explained in Section 2.3, data items with the same purpose (property, region, etc.) are classified into the same event. To compute trust scores of data items, we introduce the following assumption that all data items in the same event follow a *normal (Gaussian) distribution.*

Assumption 3 For data items in a set D of the same event, their values are normally distributed with the mean μ, variance σ^2, and probability density function $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where x is the attribute value v_d of a data item $d\,(\in D)$. □

We note that the normal distribution assumption is not a limit of our solution. We can adapt other distributions, histograms, or correlation information with simple changes of the data similarity models. We use the normal distribution since it well reflects natural phenomena. Especially, values sensed for one purpose in general follow a normal distribution [10, 22], and thus Assumption 3 is reasonable for streaming data items in sensor networks. Based on Assumption 3, we present the computation methods for s_d, $\hat{s}_d$, and $\bar{s}_d$, which are the current, intermediate, and next trust scores of d, respectively.

3.3.1 Current trust score s_d

For a data item d, we first compute its current score s_d based on current scores of network nodes maintained in its physical provenance p_d (see ① in Figure 4). This process

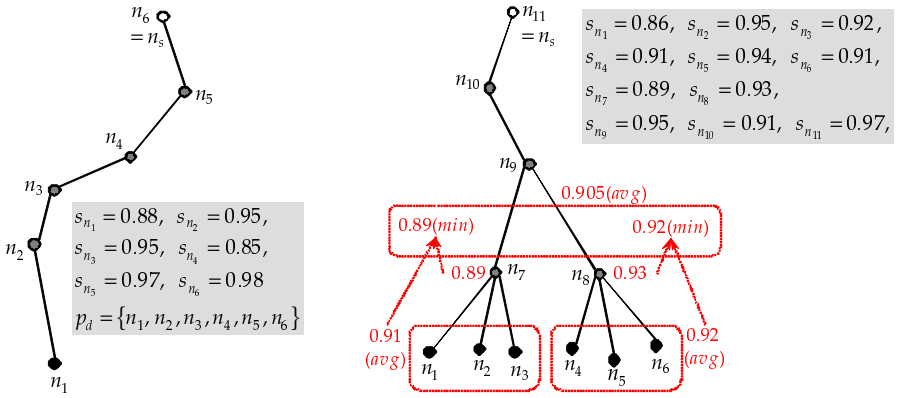
reflects the interdependency property because we use trust scores of network nodes for those of data items. In Section 2 we explained two different physical provenances: one was the simple provenance of a path type; another was the aggregate provenance of a tree type. According to this classification, we first present how to compute the current score s_d for the simple provenance and then extend it for the aggregate provenance.

In case of the simple provenance (like in Figure 2 (b)), we can represent its physical provenance as $p_d = (n_1, n_2, \ldots, n_k = n_s)$, that is, a sequence of network nodes that d passes through. In this case, we determine the current score s_d on the minimum score of the network nodes in p_d. This is based on an intuition that, if a data item passes through network nodes in a sequential order, its trust score might be dominated by the worst node with the smallest trust score[1]. That is, we compute s_d as follows:

$$s_d = \min\{s_{n_i} |\ n_i \in p_d\} \tag{3}$$

Example 1 shows how to compute the current score s_d if the data item d has a simple provenance.

Example 1 Suppose that a data item d has the simple provenance p_d in Figure 5 (a). There are six network nodes in p_d, and their current scores are 0.88, 0.95, 0.95, 0.85, 0.97, and 0.98, respectively. Thus, its current trust score s_d is computed as 0.85, that is, the minimum current score of network nodes in p_d. □



(a) A simple provenance (b) An aggregate provenance

Figure 5: Physical provenance examples and current scores of network nodes.

If a data item d has an aggregate provenance p_d, we need to consider the tree structure (like in Figure 2 (c)) to compute its current score s_d, since p_d is represented as a tree rather than as a simple path. Unlike the simple provenance, there are aggregate nodes in the aggregate provenance (for example, see n_7, n_8, and n_9 in Figure 5 (b)).

[1]We can also use an average score or weighted average score of network nodes to compute the current score. In this case, we obtain the score by simply changing the minimum function to the average or weighted average function in Eq. (3).

Thus, for an aggregate node, we first obtain a representative score by aggregating current scores of its child nodes and then use that aggregate score as a current score of child nodes. We use an average score of child nodes as their aggregated score[2]. By recursively executing this aggregating process, we simplify a tree to a simple path of aggregated scores, and we finally compute the current score s_d by taking their minimum score as in Eq. (3).

Algorithm 1 shows a recursive solution for computing the current score s_d from its physical provenance p_d, which can be either a simple or aggregate provenance. To obtain the current score s_d of a data item d, we simply call *CompCurrentScore*(n_s) where n_s is the root node of p_d. Example 2 shows how to compute the current score s_d if the data item d has an aggregate provenance.

Example 2 Consider a data item d that has the aggregate provenance p_d in Figure 5 (b). Network nodes n_1, n_2, and n_3 send their data items to n_7, and n_7 generates a new data item by aggregating their items. Similarly, n_8 generates a new item for n_4, n_5, and n_6; n_9 makes an item for n_8 and n_9. Current scores of n_1, n_2, and n_3 are averaged to 0.91; this average is compared to the current score 0.89 of n_7; the minimum score 0.89 is selected in that path. Similarly, 0.92 is selected to represent n_4, n_5, n_6, and n_8. Next, by averaging two scores 0.89 and 0.92, we obtain 0.905 as a representative score of all child nodes of n_9. Eventually, we obtain the current score s_d as 0.905 since the minimum of $\{0.905, 0.95, 0.91, 0.97\}$ is 0.905. □

Algorithm 1 *CompCurrentScore* (n_i: a tree node in p_d)

```
1:  if n_i is a simple node (i.e., n_i has only one child) then
2:     Let n_j be the child node of n_i;  // an edge e_{i,j} connects two nodes.
3:     return MIN(s_{n_i}, CompCurrentScore(n_j));
4:  else if n_i is an aggregate node with k children then
5:     Let n_{j_1}, ..., n_{j_k} be k child nodes of n_i;
6:     return MIN(s_{n_i}, AVG(CompCurrentScore(n_{j_1}), ...,
7:                             CompCurrentScore(n_{j_k})));
8:  else  // n_i is a leaf node.
9:     return s_{n_i};
10: end-if
```

3.3.2 Intermediate trust score $\hat{s}_d$

An intermediate trust score $\hat{s}_d$ of a data item d is computed from the latest set of data items of the same event with d in the current streaming window (see Figure 4). Let the set of data items in the same event with d be D. In general, if set D changes, i.e., a new item is added D or an item is deleted from D, we recompute the trust scores of data items in D. We obtain $\hat{s}_d$ through the initial and adjusting steps. In the initial step, we use the *value similarity* of data items in computing an initial value of $\hat{s}_d$. In

[2]According to the aggregate operation applied to the aggregate node, we can use different methods. That is, for `AVG` we can use an average of children, but for `MIN` or `MAX` we can use a specific score of a network node that produces a resulting minimum or maximum value. An aggregation itself, however, represents multiple nodes, and we thus use the average score of child nodes as their representative score.

the adjusting step, we use the *provenance similarity* to adjust the initial value of $\hat{s}_d$ by considering physical provenances of data items.

(1) Initial score of $\hat{s}_d$ based on value similarity

First, we explain the underlying idea of computing an initial value of $\hat{s}_d$ based on the value similarity of data items. Recall Assumption 3 which assumes that data items in D are normally distributed, and their mean and variance are μ and σ^2, respectively. Based on this assumption, we observe that, for a set D of a single event, its mean is the most representative value that well reflects the value similarity. This is because the mean is determined by the majority values, and obviously those majority values are similar to the mean in the normal distribution. Thus, we conclude that the mean has the highest trust score; if the value of a data item is close to the mean, its trust score is relatively high; if the value is far from the mean, its trust score is relatively low.

Based on those observations, we propose a method to compute the intermediate score s_d in the initial step. In obtaining $\hat{s}_d$, we assume $v_d \geq \mu$. We can easily extend it to the case of $v_d \leq \mu$. For simplicity, we omit that case.

As the intermediate score $\hat{s}_d$, we use the cumulative probability of the normal distribution. In this method, we use "$1 -$ the amount of how far v_d is from the mean" as the initial score of $\hat{s}_d$, and here "the amount of how far v_d is from the mean" can be thought as the cumulative probability of v_d. Thus, as in Eq. (4), we obtain the initial $\hat{s}_d$ as the integral area of $f(x)$.

$$\hat{s}_d = 2\left(0.5 - \int_{\mu}^{v_d} f(x)\,dx\right) = 1 - \int_{2\mu - v_d}^{v_d} f(x)\,dx = 2\int_{v_d}^{\infty} f(x)\,dx \tag{4}$$

Figure 6 shows how to compute the integral area for the initial intermediate score s_d. In the figure, the shaded area represents the initial score of $\hat{s}_d$, which is obviously in (0,1]. Here, the score $\hat{s}_d$ increases as v_d is close to μ.

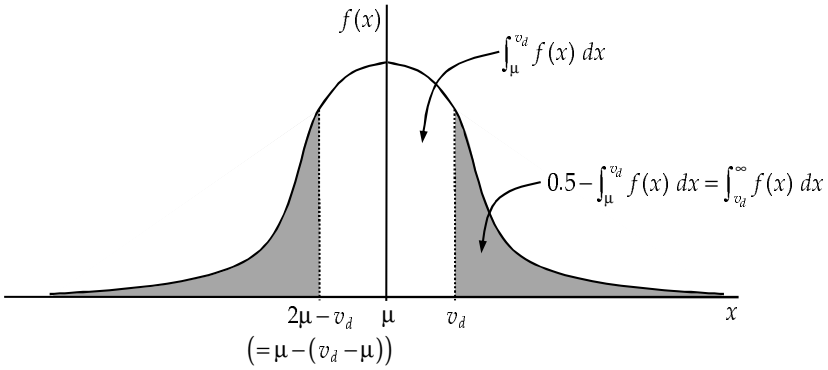


Figure 6: Computing the intermediate score of $\hat{s}_d$.

We note that our cyclic framework also well handles sudden changes of true sensing values. For example, in our data similarity model, if a sensing value is quite different from the other recent values, it can be estimated as false even though it is actually true. In our framework, this case is gradually handled as time goes on. When a sensor collects a different value, the score of this data is low since it is far from the mean of the

normal distribution. If the data is true, adjacent sensors start collecting similar values, and then, these values will get higher scores.

(2) Adjusted score of $\hat{s}_d$ based on provenance similarity

The initial score of $\hat{s}_d$ computed from the sensing values has the problem that it does not consider the effect of provenance similarity. Thus, we need to adjust the intermediate score $\hat{s}_d$ by reflecting the provenance similarity of data items. To achieve this, we let a set of physical provenances in D be P and the similarity function between two physical provenances $p_i, p_j\ (\in\ P)$ be $sim(p_i, p_j)$[3]. Here, the similarity function $sim(p_i, p_j)$ returns a similarity value in $[0, 1]$, and it can be computed from the tree or graph similarity [14, 16]. Computing graph similarity, however, is known to be an NP-hard problem [14], and we thus use the tree similarity [16], which is an edit distance-based similarity measure.

Our approach to take into account provenance similarity in computing the intermediate score $\hat{s}_d$ is based on some intuitive observations. In the following, notation '$\sim$' means "is similar to", and notation $\nsim$ means "is not similar to." Given two data items $d, t \in D$, their values v_d, v_t, and their physical proveances $p_d, p_t \in P$,

- if $p_d \sim p_t$ and $v_d \sim v_t$, the provenance similarity makes a *small positive* effect on $\hat{s}_d$;
- if $p_d \sim p_t$ and $v_d \nsim v_t$, the provenance similarity makes a *large negative* effect on $\hat{s}_d$;
- if $p_d \nsim p_t$ and $v_d \sim v_t$, the provenance similarity makes a *large positive* effect on $\hat{s}_d$;
- if $p_d \nsim p_t$ and $v_d \nsim v_t$, the provenance similarity makes a *small positive* effect on $\hat{s}_d$;

Table 1 summarizes these observations. As shown in the table, the provenance similarity makes positive or negative effect on the trust score according to the corresponding value similarity, and we are going to reflect this property to adjusting the intermediate score $\hat{s}_d$.

Table 1: Effect of provenance similarity in adjusting $\hat{s}_d$.

	$p_d \sim p_t$ (provenances are similar)	$p_d \nsim p_t$ (provenances are not similar)
$v_d \sim v_t$ (values are similar)	$+$ (small positive effect)	$+++$ (large positive effect)
$v_d \nsim v_t$ (values are not similar)	$---$ (large negative effect)	$-$ (small negative effect)

[3]Data items in the same event may have similar physical provenances, so we may assume that the number of possible provenances in an event is finite and actually small. Thus, for the real-time processing purpose, we can materialize all $sim(p_i, p_j)$'s in advance and maintain them in memory.

Based on the above observations, we introduce a measure of *adjustable similarity* to reflect the provenance similarity in adjusting $\hat{s}_d$. Given two data items $d, t\,(\in D)$, we first define the adjustable similarity between d and t, denoted by $\rho_{d,t}$, as follows:

$$\rho_{d,t} = \begin{cases} 1 - sim(p_d, p_t), & \text{if } dist(v_d, v_t) < \delta_1; \text{// positive effect} \\ -sim(p_d, p_t), & \text{if } dist(v_d, v_t) > \delta_2; \text{// negative effect} \\ 0, & \text{otherwise. // no effect} \end{cases} \tag{5}$$

In Eq. (5), $dist(v_d, v_t)$ is a distance function between v_d and v_t; δ_1 is a threshold indicating when v_d and v_t are to be treated as similar; δ_2 is a threshold indicating when v_d and v_t are to be treated as dissimilar. In the experiment we set δ_1 and δ_1 to 20% and 80% of the average distance, respectively. The adjustable similarity $\rho_{d,t}$ in Eq. (5) well reflects the effect of provenance and value similarities in Table 1. That is, if v_d and v_t are similar, $\rho_{d,t}$ has a positive value of "$1 - sim(p_d, p_t)$" determined by the provenance similarity; in contrast, if they are not similar, $\rho_{d,t}$ has a negative value of "$-sim(p_d, p_t)$." To consider adjustable similarities of all data items in D, we now obtain their sum ρ_d as follows:

$$\rho_d = \sum_{t \in D, t \neq d} \rho_{d,t} \tag{6}$$

We exploit the adjustable similarity ρ_d in the framework of normal distribution in order to reflect the provenance similarity to adjusting the intermediate score $\hat{s}_d$, Simply speaking, we adjust the value v_d by considering ρ_d and use the adjusted value, denoted by $\bar{v}_d$, to compute $\hat{s}_d$ instead of v_d. In more detail, we first normalize ρ_d into $[-1, 1]$ using its maximum and minimum similarities, $\rho_{\max}$ and $\rho_{\min}$. The normalized value of ρ_d, denoted by $\bar{\rho}_d$, is thus computed as follows:

$$\bar{\rho}_d = 2\frac{\rho_d - \rho_{\min}}{\rho_{\max} - \rho_{\min}} - 1, \quad \text{where } \rho_{\max} = \max\{\rho_t |\, t \in D\} \quad \text{and } \rho_{\min} = \min\{\rho_t |\, t \in D\} \tag{7}$$

We then adjust the data value v_d to a new value $\bar{v}_d$ as follows:

$$\bar{v}_d = \min\{v_d - \bar{\rho}_d(c_p \cdot \sigma), \mu\}, \quad \text{where } c_p \text{ is a constant greater than 0.} \tag{8}$$

Figure 7 shows how the value v_d changes to $\bar{v_d}$ based on the adjustable similarity $\bar{\rho}_d$ in the framework of a normal distribution. As shown in the figure, if $\bar{\rho}_d > 0$, i.e., if the provenance similarity makes a positive effect, v_d moves to the left in the distribution graph, i.e., the intermediate score $\hat{s}_d$ increases; in contrast, if $\bar{\rho}_d < 0$, i.e., if the provenance similarity makes a negative effect, v_d moves to the right in the graph, i.e., $\hat{s}_d$ decreases. In Eq. (8), c_p represents the important factor of provenance similarity in computing the intermediate score. That is, as c_p increases, the provenance similarity becomes more important. We use 0.2 as the default value of c_p, i.e., we move the data value v_d in $\pm$20% range of the standard deviation σ.

By using the adjusted data value $\bar{v}_d$, we finally recompute the intermediate score $\hat{s}_d$. By simply changing v_d to $\bar{v}_d$, we can also obtain Eq. (9) from Eq. (4) in which

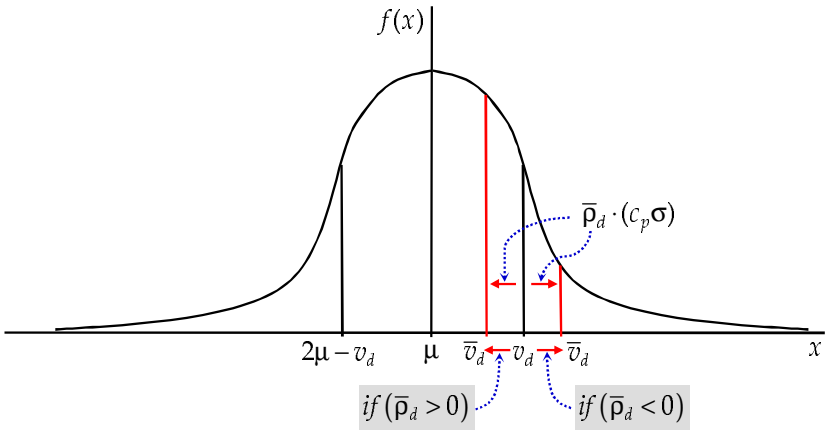


Figure 7: Effect of provenance similarity on a data value.

the integral area for the intermediate score may increase or decrease by the provenance similarity.

$$\hat{s}_d = 2 \int_{\bar{v}_d}^{\infty} f(x)\, dx = 1 - \int_{2\mu - \bar{v}_d}^{\bar{v}_d} f(x)\, dx \tag{9}$$

3.3.3 Next trust score $\bar{s}_d$

For a data item d we eventually compute its next trust score $\bar{s}_d$ by using the current score s_d and the intermediate score $\hat{s}_d$. In obtaining $\bar{s}_d$, we use s_d for the interdependency property since s_d is computed from network nodes, and we exploit $\hat{s}_d$ for the continuous evolution property since $\hat{s}_d$ is obtained from the latest set of data items. Similar to computing the next score $\bar{s}_n$ of a network node n in Eq. (2), we compute $\bar{s}_d$ as follows:

$$\begin{aligned} \bar{s}_d &= c_d s_d + (1 - c_d)\hat{s}_d, \\ &\text{where } c_d \text{ is a given constant of } 0 \leq c_d \leq 1. \end{aligned} \tag{10}$$

As shown in Eq. (10), the next score $\bar{s}_d$ is gradually evolved from the current and intermediate scores s_d and $\hat{s}_d$. We also note that $\bar{s}_d$ will be used to compute the intermediate scores (i.e., $\hat{s}_n$) of network nodes in the next computation cycle (see ④ in Figure 4) for the interdependency and continuous evolution properties.

Similar to the constant c_n used in computing s_n for a network node n in Eq. (2), the constant c_d in Eq. (10) represents how fast the trust score evolves as the cycle is repeated. That is, if c_d has a larger value, the previously accumulated historic score (s_d) is more important than the recently computed score ($\hat{s}_d$); in contrast, if c_d has a smaller value, the recent score is more important than the historic score. In summary, if c_d is large, trust scores of data items evolve slowly; in contrast, if c_d is small, they evolve fast. In the experiment we set $c_d = \frac{1}{2}$ to equally reflect the importance of s_d and $\hat{s}_d$.

In this section, instead of calibrating our model with real data sets, we present general principles for choosing parameter values(e.g., confidence ranges control the

tradeoff between the number and quality of results, c_n controls how fast scores are evolved). We believe these principles can be used in most applications.

4 Incremental Evolution of Trust Scores

Since new data items continuously arrive in DSMSs, we need to evolve (i.e., recompute) trust scores to reflect those new items. In this section, we propose two evolution schemes: *immediate mode* and *batch mode*. We compare these two modes in Section 4.1, and then explain the batch mode in detail in Section 4.2.

4.1 Immediate Mode vs. Batch Mode

The immediate mode evolves current, intermediate, and next trust scores of data items and network nodes whenever a new data item arrives at the server. It means that all the steps in the cyclic framework are conducted for every new data item d. The immediate mode provides high accurate trust scores since scores reflect the latest data items. However, this mode incurs a heavy computation overhead since all the trust scores of data items in the given streaming window and network nodes included in the physical provenance should be recomputed whenever a single data item arrives. Especially, the immediate mode is not feasible when the arrival rate of data items is very fast.

The batch mode accumulates a certain amount of input data items, and then evolves trust scores only once for the accumulated data items. Even though the current trust score of data item d is calculated whenever a new data item d arrives (i.e., ① in Figure 4), the other scores are not recalculated until the system reaches a certain condition. By reducing the number of evolutions, the batch mode reduces the computation overhead so as to make the cyclic framework scalable over the input rate of data items and the size of sensor networks. However, the accuracy of trust scores can be low compared with the immediate mode, since between evolutions the system uses old information.

We thus can see that there is a tradeoff between accuracy and efficiency. In the next section, we describe the batch mode in detail, and then, explain how can we balance the tradeoff. We omit the explanation of immediate mode since we already described the cyclic framework in the context of the immediate mode.

4.2 Batch Mode in Detail

The batch mode consists of two stages: a *stall* stage and an *evolution* stage. In the stall stage, for each input data item d, the current trust score s_d is calculated from the current trust scores of its related nodes (see Eq. (3)), and d is accumulated into a buffer. In the evolution stage, the other trust scores, i.e., $\bar{s}_d$, $\hat{s}_d$, s_n, $\bar{s}_n$, and $\hat{s}_n$, are computed based on the data items accumulated in the buffer. The batch mode starts with the stall stage and triggers the evolution stage when a threshold is reached. Here, managing the threshold is the key concept for balancing efficiency and accuracy. A higher threshold means a higher efficiency but a lower accuracy; in contrast, a lower threshold means a lower efficiency but a higher accuracy.

The simplest way to manage the threshold is to use a counter or a timer. In this naive approach, the evolution stage begins for every certain number (i.e., threshold) of data items or for every certain time period. This approach is easy and efficient. However, the naive approach cannot promptly adapt the current status of the sensor network since the static counter or timer cannot reflect the dynamic changes of the current status.

We propose an advanced approach to manage the threshold. The approach uses the concept of confidence interval of the normal distribution to reflect the current status of the sensor network. Let t_1 be a time point when the previous evolution was conducted and t_2 be the current time point. To determine when the evolution stage is triggered, we compare $N_{t_1}(\mu_{t_1}, \sigma_{t_1})$, the normal distribution at t_1, and μ_{t_2}, the mean of data items whose arrival times are in (t_1, t_2). Here, the threshold is given as a confidence level γ. If μ_{t_2} falls out of the confidence interval of γ in $N_{t_1}(\mu_{t_1}, \sigma_{t_1})$, we trigger the evolution stage and recompute $N_{t_2}(\mu_{t_2}, \sigma_{t_2})$ for the next evolution. This approach increases the accuracy of the batch mode since it dynamically discards the probabilistic model that is not any longer accurate and reconstructs a new model reflecting the current status.

Figure 8 shows the advanced approach where the confidence level (i.e., threshold) γ is equal to 95%. Here, n_1 denotes the number of accumulated data items used in generating $N_{t_1}(\mu_{t_1}, \sigma_{t_1})$ at time point t_1. If $\mu_{t_2} < \mu_{t_1} - 1.96 \cdot \frac{\sigma_{t_1}}{\sqrt{n_1}}$ or $\mu_{t_2} > \mu_{t_1} + 1.96 \cdot \frac{\sigma_{t_1}}{\sqrt{n_1}}$ (i.e., μ_{t_2} is out of the confidence interval), the evolution stage starts since the current probabilistic model is not correct with a confidence level of 95%.

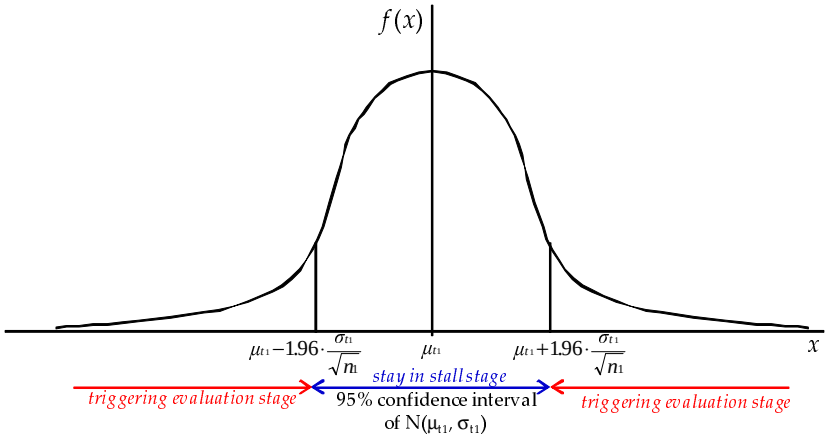


Figure 8: Managing the threshold in the advanced batch mode.

5 Experimental Evaluation

In this section, we present our performance evaluation. In what follows, we first describe the experimental environment, and then present the experimental results.

5.1 Experimental Environment

The goal of our experiments is to evaluate the *efficiency* and *effectiveness* of our approach for the computation of trust scores. To evaluate the efficiency, we measure the elapsed time for processing a data item with our cyclic framework in the context of a large scale sensor network and a large number of data items. To evaluate the effectiveness, we simulate an injection of incorrect data items into the network and show that trust scores rapidly reflect this situation. Finally, we compare the immediate and batch modes with respect to efficiency and effectiveness.

We simulate a sensor network (i.e., physical network) and a logical network for the experiments. For simplicity, we model our sensor network as an f-ary complete tree whose fanout and depth are f and h, respectively. We vary the values of f and h to control the size of sensor networks for assessing the scalability of our framework. We also simulate a logical network as a tree with a number of leaf nodes equal to N_{event}. This parameter represents the number of unique events.

We use synthetic data that has a single attribute whose values follow a normal distribution with mean μ_i and variance ${\sigma_i}^2$ for each event $i\,(1 \leq i \leq N_{event})$. To generate data items, for each event, we assign N_{assign} leaf nodes of the sensor network with an interleaving factor $N_{interleave}$. This means that the data items for an event are generated at N_{assign} leaf nodes and the interval between the assigned nodes is $N_{interleave}$ (e.g., if $N_{interleave} = 0$, then N_{assign} nodes are exactly adjacent with each other). To simulate the incorrect data injection, we randomly choose an event and a node assigned for the event, and then generate a random value which is not following the normal distribution for the event.

For the similarity function between two physical provenances p_i and p_j (i.e., *sim*(p_i,p_j)), we use a path edit distance defined as follows:

$$sim(p_i, p_j) = 1 - \frac{1}{h}\sum_{k=1}^{h} \frac{\text{node distance between } p_i \text{ and } p_j \text{ at the } k\text{-th level}}{\text{total number of nodes at the } k\text{-th level}}$$

Here, the node distance is defined as the number of nodes between two nodes at the same level. All the experiments have been conducted on a PC with a 2.2GHz Core2 Duo processor and 2GB RAM running Windows/XP. The program code has been written in Java with JDK 1.6.0. Table 2 summarizes the experimental parameters and their default values. In all experiments we use the default values unless mentioned otherwise.

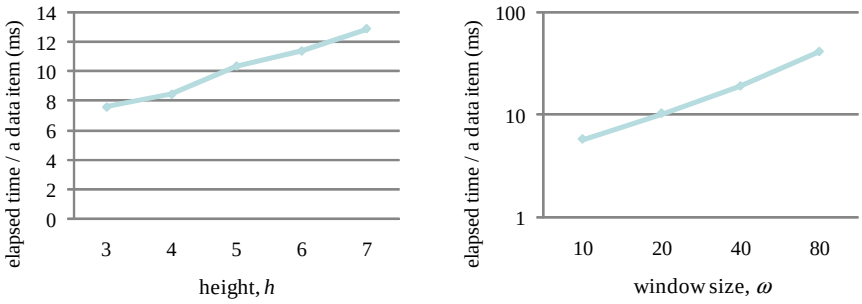
Table 2: Summary of notation.

Symbols	Definitions	Default
h	height of the sensor network	5
f	fanout of the sensor network	8
N_{event}	# of unique events	1000
N_{assign}	# of nodes assigned for an event	30
$N_{interleave}$	interleaving factor	1
ω	size of window for each event	20

As can be seen in Table 2 we only vary some application insensitive parameters. The other parameters (e.g., weights, thresholds) may be more sensitive to application contexts (e.g., data distributions, attack patterns). We will consider these parameters with specific applications in our future work and mention this as future research.

5.2 Experimental Results

(1) *Computation efficiency*: We measured the elapsed time for processing a data item. Figure 9 reports the elapsed times for different values of h's and ω's.



(a) Varying height of the sensor network

(b) Varying the window size

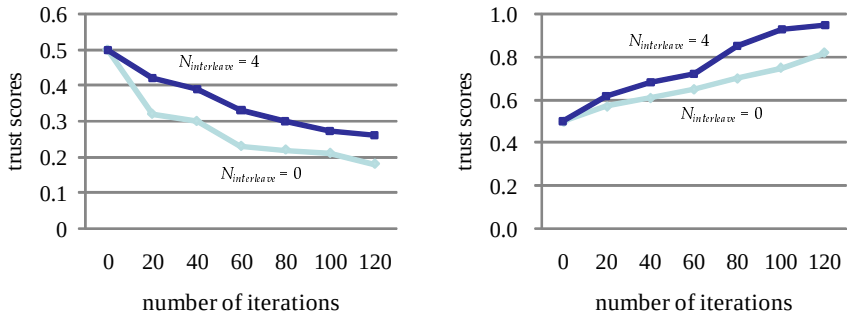
Figure 9: Elapsed times for computing trust scores.

From Figure 9 (a), we can see that the elapsed time increases as h increases. The reason is that, as h increases, the length of physical provenance also increases. However, the increasing rate is not high; for example, the elapsed time increases only by 9.7% as h varies from 5 to 6. The reason is that the additional operations for longer physical provenance linearly increase when computing trust scores for both data items and network nodes. For data items, only s_d and $\hat{s}_d$ are affected by the length of the physical provenance, i.e., an additional iteration is required to compute a weighted sum for s_d and a provenance similarity comparison for $\hat{s}_d$. For network nodes, the computation cost increases linearly with the height (not with the total number of nodes), since we consider a very small number of network nodes related to the provenance of the new data item.

From Figure 9(b), we can see that the elapsed time increases more sharply as ω increases. The reason is that the number of similarity comparisons (not an iteration) for $\hat{s}_d$ linearly increases as ω increases. However, we can see that the performance is still adequate for handling high data input rates; for example, when ω is 80, the system can process 25 data items per second.

(2) *Effectiveness*: To assess the effectiveness of our approach, we injected incorrect data items into the sensor network, and then observed the change of trust scores of data items. Figure 10 shows the trend in trust score changes for different values of the interleaving factor $N_{interleave}$. Here, $N_{interleave}$ affects the similarity of physical provenances for an event, i.e., if $N_{interleave}$ increases, the provenance similarity decreases.

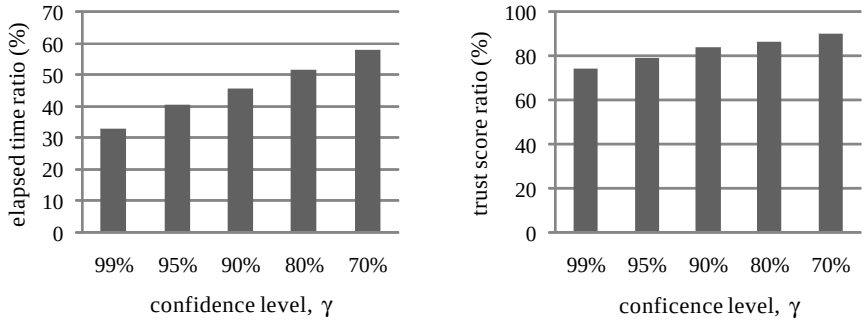
Figure 10 (a) shows the changes in the trust scores when incorrect data items are injected. The figure shows that trust scores change more rapidly when $N_{interleave}$ is smaller. The reason for this trend is explained by the principle "different values with similar provenance result in a large negative effect." In contrast, Figure 10 (b) shows the changes when the correct data items are generated again. In this case, we can see that the trust scores are modified more rapidly when $N_{interleave}$ is larger. The reason for this trend is explained by the principle "similar values with different provenance result in a large positive effect."



(a) With untrustworthy data items (b) With trustworthy data items

Figure 10: Changes of trust scores for incorrect data items.

(3) *Immediate vs. Batch*: To compare the immediate and batch modes, we measured the average elapsed time for processing a data item and the average difference of trust scores when the confidence level γ of the batch mode varied. Here, γ determines the threshold for triggering the evaluation stage, i.e., the smaller γ means a more frequent invocation of the evolution stage since the confidence interval becomes narrow. Figure 11 reports the comparison results.



(a) Efficiency of the batch mode (b) Effectiveness of the batch mode

Figure 11: Comparison of the intermediate and batch modes.

Figure 11(a) shows the relative elapsed time for a data item. We can see that the performance advantage of the batch mode is high when γ is large. The reason is that the number of executions becomes small for large values of γ. In contrast, Figure 11(b)

shows that the accuracy of the batch mode decreases for large values of γ. The reason is the use of decayed network trust scores for computing new data trust scores between evaluations stages. However, we can see that the error rate is not high; for example, when γ = 99%, the error is only 74.5%. This means that the batch mode does not significantly reduce the effectiveness (accuracy) compared with the immediate mode.

In summary, the experimental results show that our approach is a practical technique for computing trust scores in sensor networks. Especially, the batch mode significantly reduces the computing overhead without losing accuracy and so it is well suited for handling fast data streams.

6 Related Work

Work related to our approach falls into four categories: (i) access control policies, (ii) provenance (or lineage) management, (iii) trustworthiness calculation, and (iv) data quality management in sensor networks.

For access control in a relational database management system, most existing access control models, like Role-Based Access Control (RBAC) [11] and Privacy-aware RBAC [20], perform authorization checking before every data access. We can exploit one of these access control models together with our confidence policy model to implement the query and policy evaluation component of an overall framework. Thus, our confidence policy is complementary to such conventional access control enforcement and applies to query results.

Data provenance, also referred to as lineage or pedigree in databases [25], has been widely investigated. Approaches have been developed for tracking the provenance of the query results, i.e., recording the sequence of steps taken in a workflow system to derive the dataset, and computing confidence levels of the query results [2, 4, 13, 23]. For example, Widom et al. [26] have developed the Trio system which supports management of information about data accuracy and lineage (provenance). Sarma et al. [24] have developed an approach to compute lineage and confidence in probabilistic databases according to a decoupled strategy. These approaches compute the confidence of query results based on the confidence on the base tuples in the database, i.e., they assume that the confidence of each base tuple (i.e., data item) is known whereas we actually compute the trust score for each data item. In addition, very few approaches have been proposed for evaluating trustworthiness of data items and network nodes in DSMSs.

Recently, Bertino et al. [5] introduced a novel notion of confidence policy and presented a conceptual approach for computing trustworthiness of data items and data sources. They pointed out that trust scores can be affected by four factors: data similarity, data conflict, path similarity, and data deduction. Based on these factors, Dai et al. [7, 8] proposed a novel approach for computing trustworthiness and evaluating the confidence policy compliant queries. Even though we exploit the observations in [7, 8], we believe that this paper makes the following novel contributions, which are very innovative with respect to the state of the art in streaming data management: (1) We provide a trustworthiness assessment framework for data streams which have unique characteristics such as fast data arrivals and incremental updates [1, 3]. These charac-

teristics make the problem of assessing data becomes very challenging, and thus the previous results in a static database cannot be directly applied to the streaming environment. (2) We formally present the interdependency and the value and provenance similarities with a statistical model. This model is well suited for data streams since trust scores are computed incrementally and well reflect the current situations of sensor networks. (3) We provide a new approach(batch mode) to handle fast streams. It provides a solution for an important data stream issue, that is, handling the tradeoff between efficiency and accuracy.

There have been many efforts on data quality management in sensor networks [9, 15, 21]. In particular, Hwang et al. [15] and Olston et al. [21] proposed quality management methods for continuous queries of sensor networks. David et al. [9] proposed quality improvement methods in sensor networks by using cache mechanisms. As we mentioned in Section 2.1, we can use these previous results for the data quality management component in Figure 1.

7 Conclusions

In this paper we propose a provenance-based solution for enforcing the confidence policy in DSMSs. Our solution provides a systematic approach for computing and evolving the trustworthiness levels of data items and network nodes and is able to support confidence policy compliant continuous queries in DSMSs. The contributions of the paper can be summarized as follows. First, we propose a provenance-based framework that enforces confidence policies in the evolution of continuous queries over streaming data. Second, based on the notion of physical and logical networks, we introduce the notions of the *physical* and *logical* provenances for data items, respectively. Third, we introduce a cyclic framework of computing actual trust scores of data items and network nodes based on the *value* and *provenance similarities* embedded in data items. Fourth, we propose a batch mode and an intermediate mode to achieve efficiency and accuracy. Fifth, through extensive experiments, we showcase that our confidence model and cyclic framework works well in DSMSs, and the batch mode is more flexible than the immediate mode.

As future work, we plan to further investigate the following issues: (1) consider multiple *dependent* attributes and multi-attribute in-network operations (thus dropping Assumptions 1 and 2) and (2) consider other probability distributions instead of normal distributions (thus dropping Assumption 3).

References

[1] D. Abadi et al., "Aurora: A Data Stream Management System," In *Proc. of Int'l Conf. on Management of Data*, ACM SIGMOD, Demonstration, San Diego, CA, p. 666, June 2003.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the Representation and Querying of Sets of Possible Words," *SIGMOD Record*, Vol 16, No. 3, pp. 34-48, Sept. 1987.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," In *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, Madison, WI, pp. 1-16, June 2002.

[4] D. Barbara, H. Garcia-Molina, and D. Porter, "The Management of Probabilistic Data," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 4, No. 5, pp. 487-502, Oct. 1992.

[5] E. Bertino, C. Dai, H.-S. Lim, and D. Lin, "High-Assurance Integrity Techniques for Databases," In *Proc. of the 25th British Nat'l Conf. on Databases*, Cardiff, UK, pp. 244-256, July 2008.

[6] K. Biba., "Integrity Considerations for Secure Computer Systems," *Technical Report TR-3153, Mitre*, 1977.

[7] C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu, "An Approach to Evaluate Data Trustworthiness Based on Data Provenance," In *Proc. of the 5th VLDB Workshop on Secure Data Management*, Auckland, New Zealand, pp. 82-98, Aug. 2008.

[8] C. Dai et al., "Query Processing Techniques for Compliance with Data Confidence Policies," In *Proc. of the 6th VLDB Workshop on Secure Data Management*, Lyon, France, pp. 49-67, 2009.

[9] Y. David, N. Erich, K. Jim, and S. Prashant, "Data Quality and Query Cost in Wireless Sensor Networks," In *Proc. of the 5th IEEE Int'l Conf. on Pervasive Computing and Communications Workshops*, White Plains, New York, USA, pp. 272-278, Mar, 2007.

[10] E. Elnahrawy and B. Nath, "Cleaning and Querying Noisy Sensors," In *Proc. of the 2nd ACM Int'l Conf. on Wireless Sensor Networks and Applications*, San Diego, California, pp. 78-87, Sept. 2003.

[11] D. F. Ferraiolo, R. Sandhu, S. Gavarila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Trans. on Information and System Security*, Vol. 4, No. 3, pp. 224-274, Aug. 2001.

[12] T. Fraser, "LOMAC: Low Water-mark Integrity Protection for COTS Environments," In *Proc. of the 2000 IEEE Symposium on Security and Privacy*, Berkeley, California, pp. 230-245, May 2000.

[13] N. Fuhr, "A Probabilistic Framework for Vague Queries and Imprecise Information in Databases," In *Proc. of the 16th Int'l Conf. on Very Large Data Bases*, Brisbane, Australia, pp. 696-707, Aug. 1997.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, 1990.

[15] S.-Y. Hwang, P.-Y. Liu, and C.-H. Lee, "Using Resampling for Optimizing Continuous Queries in Wireless Sensor Networks," In Proc. of the 8th Int'l Conf. on Intelligent Systems Design and Applications," Kaohsiung, Taiwan, pp. 107-110, Nov. 2008.

[16] P. N. Klein, "Computing the Edit-distance between Unrooted Ordered Trees," In *Proc. of the 6th annual European Symposium on Algorithms (ESA)*, Venice, Italy, pp 91-102, Aug. 1998.

[17] X. Lian et al., "Similarity Match over High Speed Time-series Streams," In *Proc. of the 23rd Int'l Conf. on Data Engineering*, Istanbul, Turkey, pp. 1086-1095, Apr. 2007.

[18] H.-S. Lim, K.-Y. Whang, and Y.-S. Moon, "Similar Sequence Matching Supporting Variable-length and Variable-tolerance Continuous Queries on Time-series Data Stream," *Information Sciences*, Vol. 178, No. 6, pp. 1461-1478, Mar. 2008.

[19] S. Madden, M. Shah, J. M. Hellerstein, V. Raman, "Continuously Adaptive Continuous Queries over Streams," *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Madison, WI, pp. 49-60, June 2002.

[20] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo, "Provacy Aware Role Based Access Control," In *Proc. of the 12th ACM Symp. on Access Control Models and Technologies*, Sophia Antipolis, France, pp. 41-50, June 2007.

[21] C. Olston, J. Jiang, and J. Widom, "Adaptive Filters for Continuous Queries over Distributed Data Streams," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, San Diego, CA, pp. 563-574, June 2003.

[22] M. Rabbat and R. Nowak, "Distributed Optimization in Sensor Networks," In *Proc. of the 3rd Int'l Symp. on Information Processing in Sensor Networks*, Berkeley, California, pp. 20-27, Apr. 2004.

[23] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom, "Working Models for Uncertain Data," In *Proc. of the 22nd Int'l Conf. on Data Engineering*, Atlanta, GA, p. 7, Apr. 2006.

[24] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting Lineage for Confidence Computation in Uncertain and Probabilistic Databases," In *Proc. of the 24th Int'l Conf. on Data Engineering*, Cancun, Mexico, pp. 1023-032, Apr. 2008.

[25] Y. L. Simmhan, B. Plale, and D. Gannon, " A Survey of Data Provenance in e-Science," *SIGMOD Record*, Vol. 34, No. 3, pp. 31-36, Sept. 2005.

[26] J. Widom, "Trio: A System for Integrated Management of Data, Accuracy, and Lineage," In *Proc. of the 2nd Biennial Conf. on Innovative Data Systems Research*, Asilomar, CA, pp. 262-276, Jan. 2005.