Space-Time Derivative-Based Prediction: a novel Trickling Mechanism for WSN

César Huegel Richa, Antônio Augusto Fröhlich, Giovani Gracioli
Software/Hardware Integration Lab
Federal University of Santa Catarina
Florianópolis, SC, Brazil - 88040-900
{huegel,guto,giovani}@lisha.ufsc.br

***Abstract*—Time series prediction techniques reduce the number of messages generated at the application level, saving energy spent in the communication and, consequently, extending the network lifetime. Trickle is a well-known time series prediction mechanism commonly used to decrease the number of transmitted messages in Wireless Sensor Networks (WSN) and thus save energy. This paper presents the Space-Time Derivative-Based Prediction (ST-DBP), a novel Trickling mechanism to suppress data transmission in space-time regions in WSNs. We integrate ST-DBP with the Trustful Space-Time Protocol (TSTP), an application-oriented, cross-layer communication protocol, and compare two variations of the ST-DBP with the original DBP using real data from a Solar Farm in terms of suppression data ratio. Our results show that the two variations of the ST-DBP outperform the original DBP.**

***Index Terms*—Wireless Sensor Networks, Trickling Mechanism, Data Suppression, Data Prediction, Space-time Regions.**

I. Introduction

Wireless Sensor Networks (WSNs) have been the focus of intense research mainly in the last two decades, involving a diverse set of topics, from communication protocols to applications. The reason why WSNs have gained attention is their applicability in areas such as environment monitoring, agriculture, controlling and monitoring of industrial process, and so on.

The main characteristic of a WSN is the limited resources available (*i.e.,* processing power, energy, and memory). Thus, one of the challenges to design WSNs solutions is the maximization of the network lifetime. In this context, communication plays a critical role, because it is the most power consuming task of the network nodes [1].

Common to many WSN applications is the periodic read of sensors data. Usually, sensor nodes periodically read their data and send them to a sink node [1]. In this case, a communication approach based on the data suppression technique employed by the Trickle Algorithm emerges as a potential solution to maximize the lifetime of the network nodes. However, the application must tolerate a small variation on the data precision, which is a valid assumption for most of the WSN applications [1].

The central idea of the Trickle's suppression technique is a time series prediction mechanism, which is embedded into the sensor and sink nodes. The mechanism estimates the next points of a time series. This estimative is carried out in both sensor and sink nodes. Thus, it is possible to avoid data transmissions, reducing the energy consumption without decreasing the application QoS. At each data measurement period, both sink and sensor nodes compute their prediction models, so they both see the same data. If there is significant deviation of the prediction, according to a threshold defined by the application, the sensor node constructs a new model and sends it to the sink, keeping the synchronization of their models.

Time series prediction mechanisms are presented and evaluated in recent publications [1], [2], [3], [4]. These studies clearly show the potential of this approach, which is able to suppress more than 90% of the transmitted messages, and, in some cases, it can reach a suppression rate of up to 99% [1]. However, these works do not consider the localization of the sensor nodes to reduce the energy consumption even more. When the spatial variation of a sensed physical quantity is very low, the models generated by different and closely located sensor nodes tend to be near from each other. In this case, a generalization of the suppression technique to also consider regions can reduce the energy consumption of the nodes and thus increase the network lifetime.

In this paper, we propose a mechanism to generalize the data suppression technique. Instead of applying data suppressing node by node, we propose to apply the suppression technique in a region containing several nodes. Our method, named *Space-Time Derivative-Based Prediction* (ST-DBP), is based on the *Derivative-Based Prediction* (DBP) algorithm [1]. We integrate ST-DBP with the *SmartData* [5] and *Trustful Space-Time Protocol* (TSTP) [6], which is a cross-layer trustful protocol specifically designed for WSN applications. We evaluate ST-DBP in a scenario composed of Solar Farms. Our results indicate that ST-DBP further increases data suppression without degrading application quality requirements when compared to the original DBP.

The remainder of this paper is organized as follows. Section II presents the related work. Section III presents an overview of Smart Data and TSTP. Section IV explains the original DBP. Section V shows the ST-DBP mechanism, a new technique for trickling space-time regions. Section VI presents the implementation aspects of this technique. In Section VII, we evaluate the ST-DBP with real data from solar farms. Finally, Section VIII concludes the paper.

II. Related Work

Several proposed approaches to save energy in WSN focus on optimizing the network stack, without being aware of the transmitted data. Data suppression, in contrast, is an application-level technique to reduce energy in the communication [3]. Recent works have used data suppression based on tendency models to maximize the lifetime of a WSN (i.e., to reduce the energy consumption) [1], [3], [7], [8], [2], [9].

Derivative-based Prediction (DBP) is a data suppression technique where a sensor node builds a linear model, based on real measurements, and then sends the model to the sink node [8]. The linear model is built through the calculation of the derivate of a line between two points. These points are obtained by the average of a set of points located in the edges of a learning window. When the prediction cannot represent real data (the model deviates from reality beyond what is allowed by the application), a new model is generated and sent to the sink node. Thus, this new model is considered in the next predictions. If the prediction represents data according to application threshold levels, no data transmission is required, saving energy.

Raza et al. present application-level experiments using real data and DBP [1], [8]. In their experiments, DBP obtained a data suppression rate of up to 99%. It also had a better performance in five of seven data sets, when compared to other state-of-the-art techniques, such as Piecewise Linear Approximation (PLA) [10], Similarity-based Adaptable Framework (SAF) [7], and Polynomial Regression (POR) [1]. However, the work shows that when the network stack is considered, DBP triplicates the network lifetime. After some modifications in the MAC and routing protocols, which are responsible for a great number of control messages in the network, the authors have obtained an improvement of about seven times in the network lifetime [1].

Recently, two DBP variations were proposed by Barton and Musilek, named Delayed DBP and DBP with look ahead [2]. Delayed DBP proposes a delay in the learning window, based on the premise that calculating the slope of a line that begins in the past and ends in the future results in better predictions. In this way, it is possible to achieve a smaller number of model updates, since Delayed DBP is aware of the near future. This technique has obtained a reduction of 60% and 70% of model updates when compared to the original technique, with a delay overhead of 7.5 and 2.5 minutes, respectively. Thus, this technique is not adequate for real-time systems, due to the sink's slightly delayed perception of reality. DBP with look ahead changes the Delayed-DBP to make it feasible for real-time applications [2]. Instead of delaying the learning window, it uses a Recurrent Neural Network (RNN) to predict future points that will be the base to construct the model. This approach reduced about 90% of the model updates.

Istomin et al. evaluated the impact of data prediction at the application-level, considering optimizations in the network stack of a new protocol, named Crystal [3]. The authors show that is possible to achieve low energy consumption only by the usage of specialized hardware. As the conclusion presented in [1], application-level prediction techniques have a bound in the lifetime maximization.

However, by considering the sensor nodes localization we show that it is possible to achieve better data suppression ratios. Our proposed technique (ST-DBP) is based on the recent proposed DBP [1], because it presents high data suppression ratios. The idea behind ST-DBP (use the node localization) can also be applied in other algorithms that use prediction models.

III. Smart Data and TSTP Overview

Trustful Space-Time Protocol (TSTP) [6] is an application-oriented, cross-layer communication protocol for *Cyber-Physical Systems* (CPS) on a WSN. TSTP focuses on efficiently delivering functionality recurrently needed by such systems: trusted, timed, geo-referenced, SI-compliant data that is resource-efficiently delivered to a sink or gateway, instead of focusing on keeping the original protocol interfaces in a modular, layered architecture with shared data. TSTP delivers this functionality to applications through the *SmartData* construct [5], which promotes a data-centric view of the network. Both artifacts, described in this section, are fundamental for the ST-DBP.

In TSTP, communication occurs among sensors and a specific node called sink. Sink is the node of a WSN that is interested in information of space-time regions. Sensors are nodes that measure one or more types of environment variables (e.g., temperature, luminosity, etc) with a certain precision and frequency, and are able to report these measurements to the sink [6]. The sink's interest in information of a space-time region is announced by an interest message, according to the semantics of SmartData, explained in this section.

A *SmartData* is a piece of data enriched with enough metadata to make it self-contained in terms of semantics, spatial location, timing, and trustfulness. It is meant to be the only application-visible construct in the sensing/actuating platform and therefore implicitly mediates all system-level services, including communication, synchronization, and the interaction with transducers and actuators. The data contained in a SmartData is automatically updated from the network according to the parameters specified at instantiation time. Therefore, an application reads a sensor, simply accessing the data inside SmartData through its interface. Similarly, the actuation occurs by changing this data. In this case, messages are automatically propagated over the network to command actuators accordingly, until the new value is observed.

The SmartData interface is depicted in Figure 1. The SmartData interface has two constructors[1]: the first one is used to abstract local transducers, while the second is used to create local proxies of remote transducers. In both cases, the binding of a SmartData object with the corresponding transducer is done via the `Transducer` class parameter. Every transducer is supposed to declare constants, named `UNIT` and `ERROR`. The first one is used to personify the corresponding SI quantity,

[1]In this work, we use the C++ language.

while the second is used to define the transducer absolute error. For instance, a SmartData object instantiated with a transducer specifying K (Kelvin) as `UNIT` and specifying 1.5 as `ERROR` represents the SI quantity temperature with an accuracy of $\pm 1.5\ K$. This SmartData can abstract a temperature sensor or an air conditioning (i.e., a temperature actuator). In this paper, we focus on the use of SmartData as an abstraction of a sensor.

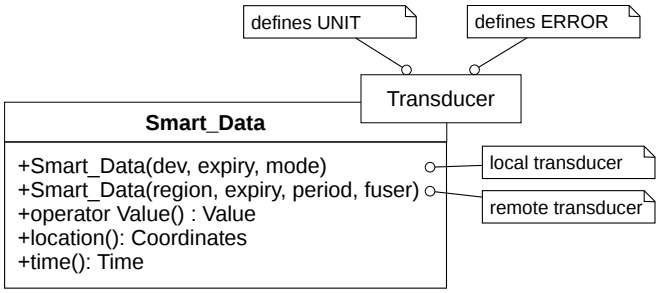

Fig. 1: SmartData interface.

Local transducers are abstracted using SmartData objects instantiated with the first constructor depicted in Figure 1. The constructor's first parameter, `dev`, is used to differentiate multiple instances of the same transducer available on a given node. The second parameter, `expiry`, defines the validity of the data expressed relatively to the time they were produced, so the expiry is refreshed on every SmartData update, just like the period. Different components, tasks, or even functions can instantiate SmartData objects bound to the same physical transducer with different expiry times. It is, therefore, a per-instance attribute. Application tasks in a system using SmartData can be mostly aperiodic, with a periodic behavior being implicitly accomplished by the traditional OS Periodic Task's `wait()` blocking method that is used to wait for a SmartData update [11]. If a value different from zero is given as the `period`, then the associated transducer will be periodically sampled to update the SmartData and `wait()` will block until the next period. Otherwise, the object encapsulating the transducer will be invoked on every SmartData access.

Specifying an expiry smaller than the period is invalid. If distinct SmartData objects using the same Transducer are created with different periods, then the greatest common divisor is used for sampling, but `wait()` on each object respects the originally specified periods and so does the expiry. The last parameter, `mode`, designates SmartData's visibility: *Private* to the process that created it, inaccessible from the network; *Advertised* on the network for remote sensing; or *Commanded* remotely over the network, which is used by actuators. In this paper, we consider only instances of SmartData with the *Advertised* mode.

Remote transducers are also abstracted using instances of SmartData. Objects created with the second constructor depicted in Figure 1 are local proxies to remote transducers. All the system-level procedures needed to ensure a consistent semantic for such objects is transparently carried out by the implementation, including communication, synchronization, and scheduling. The constructor's first parameter is used to specify a Space-Time region of interest as:

$$Space\text{-}Time\ Region(x, y, z, r, t_0, t_f)$$

where x, y, z designates the center of the interest region, r designates its radius, and $[t_0, t_f]$ designates the interest's time interval. The interest region is used in combination with the SI Physical Quantity associated with a SmartData object through the `UNIT` constant defined by the Transducer given as parameter. The instantiation of a remote SmartData implies the announcement of an interest for an SI Quantity in a given region of space for a given interval of time. SmartData objects (created with the first constructor) matching the criteria will reply to the interest in accordance with parameters `period` and `fuser`. A zero period means that all updates on the remote nodes will be reported, determining an event-based mode of operation. Other values require matching objects to periodically send a response every `period` units of time, starting at t_0 and repeating until t_f. This defines a time-triggered operation for the network. Different periods are handled just like for local transducers, so values for this parameter must be carefully chosen. The last parameter, `fuser`, is used for data fusion and is better explained in Section VI (it is extended).

Similarly to local transceivers, remote ones abstracted as SmartData also have an *Expiry*. Once again, it is a client-side information, so several SmartData objects acting as proxies to the same transducer can have different expiration times. Nodes with SmartData exported as Advertised will manage proper response messages to match all interests. The shortest of the expiries is used in the network packet and can be used for expedite routing on the network while expired messages are simply dropped, and therefore avoiding wasted resources to produce invalid data that will be discarded at the destination.

Independently of how a SmartData is created, three common methods can always be invoked on it: a native type conversion operator (summarized as `operator Value()` in Figure 1), `location()`, and `time()`. The first enables implicit conversions from and to the type of the encapsulated data (e.g., integer, real), so SmartData can be transparently operated. The second returns the location where the data was produced, while the third returns the time data was produced. In this paper, we focus on scenarios where sensor nodes have local transducers and the sink node has remote transducers.

IV. Derivative-based Prediction Overview

Derivative-Based Prediction (DBP), used in this work, is a data prediction technique compatible with applications that have data quality restrictions [1]. It is adequate for resource-constrained WSN applications, because it is simpler than other existing Trickle methods [1].

DBP has an assumption that measured data of a sensor node can be approximated, considering short and medium intervals and a linear precision model. This linear model represents data with an error within the tolerance margin, which must be part of the application requirements. A time interval, in

which the deviation from the model can occur, is defined as $\mathcal{E}_T$. The tolerance is defined as $\mathcal{E}_V = (\epsilon^{rel}, \epsilon^{abs})$, where ϵ^{rel} is the tolerable relative error and ϵ^{abs} is the tolerable absolute error. DBP tolerance and time interval are formally defined as follows [1]:

Definition 1. Let V_i be a sensed value taken at time t_i and $\mathcal{E}_V$ the tolerance value. From the application perspective, reading a value V_i becomes equivalent to reading any value $\hat{V}_i$ in the range R_V defined by the maximum error, $\hat{V}_i \in R_V = [V_i - \epsilon, V_i + \epsilon]$, where $\epsilon = max\{\frac{V_i}{100}\epsilon^{rel}, \epsilon^{abs}\}$. In other words, the application considers a value $\hat{V}_i \in R_V$ as correct.

Definition 2. Let $\Delta T = (t_k - t_j)$ be a time interval, and $\hat{V}_{\Delta T} = \{\hat{V}_j, ..., \hat{V}_k\}$ the set of values reported to the application during ΔT. The time tolerance $\mathcal{E}_T$ is the maximum acceptable value of ΔT such that all the values reported in this interval are incorrect, i.e., $\hat{V}_i \notin R_V, \forall \hat{V}_i \in \hat{V}_{\Delta T}$.

In many applications, the amplitude variation of a time series can be high, that is, the measured values can be very small or very large. An example of this is the global irradiation, which can have an amplitude from 1 W/m^2 to more than 1000 W/m^2. This is the reason why a tolerance value is defined in terms of relative and absolute errors. Thus, it is possible to specify maximum absolute and relative errors, for both small and large values.

Each sensor node executing DBP constructs a learning window $\mathcal{W}$, which is formed by the last w measured values. Be $\mathcal{W}_t$ the learning window at a time t, w' the current window size, and p the given period, we have: $\mathcal{W}_t = \{V_{t-((w'-1)*p)}, ..., V_t\}$, for every $w' > 1$. When $\mathcal{W}$ reaches the maximum size w, the sensor node constructs the first model $\mathcal{M}$, calculating the line inclination α that crosses two defined points. The first point is the average of the oldest l points of $\mathcal{W}$ and the second point is the average of the l most recent points of $\mathcal{W}$, named border points. Thus, a linear model $\mathcal{M}$ generated at a time t_0 can be described as a function $y(t)$ such as: $\hat{V}_t = y(t) = \alpha * (t - t_0) + \beta$, where β is the average of the oldest l points of $\mathcal{W}$.

This first model is then sent to the sink node, which is considered valid. At each period, the sensor node updates the learning window with a new measure V_t and compares it with the DBP predicted value $\hat{V}_t$ using the current model $\mathcal{M}$. If the DBP predicted value is within the tolerance $\mathcal{E}_V$, $\mathcal{M}$ is still a valid model and the sink continues generating values based on this model. Otherwise, if the measures are out of the tolerances $\mathcal{E}_V$ and $\mathcal{E}_T$, a new model $\mathcal{M}'$ is built by the sensor node, based on the last w measured values, and sent to the sink node.

V. ST-DBP For Trickling a Space-Time Region

The original DBP, described in the previous Section, is limited to only one sensor. Thus, it is adequate for a node by node monitoring. From the region-to-sink perspective of TSTP, however, if the obtained information of a desired space-time region is trustful and within the defined tolerance levels, it does not matter from which sensor the information is generated. TSTP was designed in such a way that the sink node announces its interests for sensing data and sensor nodes deliver it whenever possible [6]. This characteristic requires DBP to be expanded in order to reach data suppression for space-time regions and not for individual nodes.

We propose an extension of the DPB, named Space-Time Derivative-Based Prediction (ST-DBP). In the ST-DBP model, each node $n \in \mathcal{N}^{\mathcal{R}}$ belongs to a region of interest $\mathcal{R}$. Each region $\mathcal{R}$ has a finite number of nodes. Each node $n \in \mathcal{N}^{\mathcal{R}}$ creates a tuple $\langle \alpha, \beta, \tau, \theta \rangle$, where α and β are the model coefficients, τ is the instant in which the model was generated, and θ is a (x, y, z) coordinate of the model origin. The difference of the ST-DBP in comparison with DBP is the spatial component in the model and the way nodes synchronize the model (all nodes in a region of interest $\mathcal{R}$ and the sink node synchronize the model).

ST-DBP introduces the concept of a `trickling space-time region`, which is a node agglutination represented by the same model. This concept is formalized by the Definition 3.

Definition 3. Be $\mathcal{N}^{\mathcal{R}}$ a finite set of nodes in a space-time region $\mathcal{R}$. A `trickling space-time region` is a set of nodes $T_i \in 2^{\mathcal{N}^{\mathcal{R}}} \setminus \varnothing$ such that $\forall n \in T_i, \neg\exists j : \mathcal{M}_j^T$ was generated closer to n than $\mathcal{M}_i^T$, where $\mathcal{M}_i^T$ is the model of the `trickling space-time region` T_i.

ST-DBP initially assumes that $T_i = \mathcal{N}^{\mathcal{R}}$ and $\mathcal{M}_i^T$ is the first generated model and announced by a node $n \in T_i$. Thus, $\forall n' \in T_i$ that listens to this announcement, the `trickling space-time region` of n' is $\mathcal{M}_n^L$. Consequently, the first node to announce a model will represent its `trickling space-time region` by its own model. The node will also update and announce its model in each updating in order to keep the synchronization with other nodes. On the other hand, all the other nodes verify, at every data period, the model of their own `trickling space-time regions`, dropping their model announcements until a region partitioning occurs.

The partitioning of a region occurs in a distributed way, when a model $\mathcal{M}_i^T$ from the `trickling space-time region` T_i deviates from the reality of a node $n \in T_i$, as defined by the Definitions 1 and 2. When this occurs, a node n builds and announces a new local model $\mathcal{M}_n^L$, based on its learning window, and starts to represent a new `trickling space-time region`. In this case, every node that listens to the announce of a new `trickling space-time region` must define whether it stays in the current region or changes to the newly created one. The node will only change its `trickling space-time region` if the distance to the origin of the new region is smaller than the distance to the origin of the current region.

As a consequence, every node will have the origin of its `trickling space-time region` as close as possible to its location. In the worst-case, there will be as many regions as nodes. In this case, ST-DBP degrades into the original DBP in

terms of suppressed messages (*i.e.,* it applies data suppression node by node, instead of regions).

The sink node has more computational power and less energy restrictions than sensor nodes. Thus, it is able to analyze every models from a region of interest and decide how and when to perform an agglutination of the partitioned regions. When the sink detects that two or more regions have close and stable models, it computes a new model based on those from the regions and announces it as part of a new region of interest (that includes all analyzed regions). When the sensor nodes receive the model from the sink, they assume this new model.

VI. ST-DBP Design and Implementation

The ST-DBP Trickle mechanism is implemented as a new class `ST_DBP`, together with an aggregation of the SmartData class, as demonstrated by Figure 2. Both classes abstract all tasks related to the Trickle mechanism, which are transparent to the application.

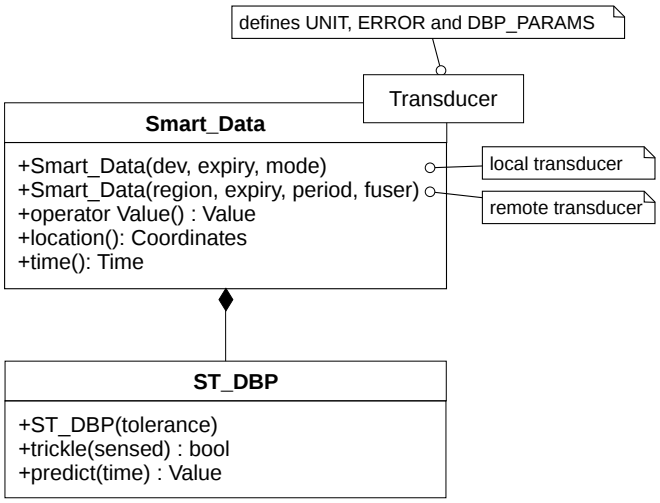

Fig. 2: Trickling SmartData.

The `ST_DBP` class constructor (see Figure 2) has a parameter ($tolerance$) that defines the acceptable time and value tolerance levels by the application, as described in Section IV. This parameter is extracted from the requested precision of the `TSTP Interest Message`.

`Transducer` defines the ST-DBP parameters, i.e., the learning window size w and the amount of points l that make up the window borders. Thus, the `ST_DBP` abstracts the operation carried out in a learning window, implemented as a circular queue (the type and size of the queue are defined at compile-time by the `Transducer`).

The `ST_DBP` class implements the `trickle(sensed)` and `predict(time)` methods. The former abstracts the required actions for the ST-DBP correct functioning in the sensor node. This method is used to create a SmartData object and to decide whether Trickle must act or not. The latter computes the current model $\mathcal{M}$ to obtain the predicted $\hat{V}_i$ at the instant t_i (`time`). This method is used in the sink node within a SmartData object, which abstracts a remote `Transducer`, to obtain a predicted data when requested by the application.

A. ST-DBP Trickle Algorithm

`ST-DBP Trickle Algorithm` (Algorithm 1), implemented in `ST-DBP::trickle(sensed)`, is executed by each sensor node at each data period. Initially, the algorithm inserts a new sensor data ($sensed$) into the learning window. Then, it verifies if the sensor node already knows its region model (line 4) and, if it does not (*i.e.,* $\mathcal{M}_i^T = null$), the node creates a first local model (line 6) and considers its model as the region model just if the learning window is complete (line 5). In this case, the sensor node starts to automatically represent its region.

Algorithm 1 ST-DBP Trickle Algorithm

```
1:  procedure Trickle(sensed)
2:      sendModel ← false
3:      W_i.insert(t, sensed)
4:      if M_i^T = null then
5:          if W_i.size ≥ w then
6:              M_i^T ← M_i^L ← makeModel(W_i)
7:              sendModel ← true
8:          end if
9:      else
10:         if checks(M_i^T, sensed, t) then
11:             M_i^T ← M_i^L ← makeModel(W_i)
12:             sendModel ← true
13:         end if
14:     end if
15:     if sendModel then
16:         Q_i.insert(makeModelMsg(M_i^L))
17:     end if
18: end procedure
```

If the sensor node already knows its region model $\mathcal{M}_i^T$, the algorithm evaluates this model by comparing the prediction result with the sensed value at the instant t. This evaluation is performed by the $checks$ function (line 10). It returns $true$ when all requirements of the Trickle mechanism are satisfied and $false$ otherwise. When the requirements are not satisfied, it means that the current region model deviated beyond the tolerance defined by the interest, generating and announcing a new local model. Moreover, the sensor node assumes this model as the model of a new region, in which it starts to automatically represent.

At the end of its execution, the algorithm verifies if there is a new model, which is a candidate to represent the region of the sensor node. If so, a message containing the model is built and inserted into the Q_i messages queue to be later transmitted by the TSTP (line 16).

B. ST-DBP Model Update Algorithm

`ST-DBP Model Update Algorithm` is executed in a sensor node whenever a model or interest message is received by `TSTP` to update the model of the sensor node. The updating process is implemented according to the Algorithm 2, which deals with two possible cases:

1) Case 1: if m is a model message, the algorithm evaluates the received model and its origin (line 7). If the origin

of the received model is closer than the origin of the current model in the sensor node, the algorithm updates the model.

2) Case 2: if m is an interest message and has a model, the algorithm verifies if the sensor node is within the interest region (line 11). If so, the received model is obligatorily considered as the region model of the sensor node. This is the expected behavior when the sink node sends an interest with a model in order to agglutinate regions.

Algorithm 2 ST-DBP Model Update Algorithm

```
1: procedure ModelUpdate(m)
2:     modelUpdate ← false
3:     fromSink ← false
4:     if m.isModelMessage then
5:         D ← distance(here(), m.M.θ)
6:         D_i ← distance(here(), m.M_i^T.θ)
7:         if D < D_i or M.θ = M_i^T.θ then
8:             modelUpdate ← true
9:         end if
10:    else if m.isInterestMessage and m.M ≠ null then
11:        if here() is inside m.sphere then
12:            modelUpdate ← true
13:            fromSink ← true
14:        end if
15:    end if
16:    if modelUpdate then
17:        if ∃e_Q ∈ Q_i : e_Q is a ModelMsg and e_Q.M = M_i^L then
18:            if fromSink or checks(m.M, getSensed(), t) then
19:                M_i^T ← m.M
20:                delete Q_i.remove(e_Q)
21:            end if
22:        else
23:            M_i^T ← m.M
24:        end if
25:    end if
26: end procedure
```

C. ST-DBP Bootstrap

Given an interest of the sink node in a determined space-time region, all sensor nodes that answer to this interest measure a data from the same unity, and consider the same data period. At the ST-DBP bootstrap, all sensor nodes answer to this interest and start measures in their respective learning windows, until they are filled. When the windows are filled, sensor nodes verify the existing of a model to represent the region, as the model still does not exist, each node creates a local model and assumes it as being its region model, announcing it to other nodes.

However, as all nodes have the same data period and their clocks are synchronized by `TSTP`, there is a possibility of all nodes execute these tasks at the same time, causing a generalized partitioning of the region at the instant t_0. To solve this, we explore the cross-layer design of `TSTP`, which allow us to search and remove a message from the MAC's transmission queue before it is real transmitted. `TSTP-MAC` verifies the channel before starting a transmission and then detects a message that is being transmitted. Thus, if this message is for a model, `TSTP` notifies a `Trickle` component, that executes the Algorithm 2. When the algorithm realizes a new model (line 16), it verifies if there is a recent message containing its local model stored in the transmission queue (line 17). If so, the algorithm verifies if the new model respects the acceptable tolerance, removing the message from the transmission queue if it does. Otherwise, the algorithm keeps the local model as being the region model and the message is not removed from the transmission queue.

D. Backward and Forward

`Greedy Forwarding Algorithm` [12] is proposed to avoid the retransmission of messages for incorrect destinations. The routing algorithm used by `TSTP` performs a geographically controlled flooding, where transmissions always have a positive progress towards the destination. This behavior can introduce problems in ST-DBP, since a generated model in any point of a interest region and sent to the sink can not be listened by one or more nodes that are in the opposite direction of the sink node. A solution for this problem is a geographically controlled flooding in all directions, dropping a message whenever it is one hop out of the interest area and with a negative progress towards the destination.

E. ST-DBP Regions Agglutination Algorithm

`ST-DBP Regions Agglutination Algorithm` (Algorithm 3) is executed in the sink node and is responsible for agglutinating partitioned regions by evaluating the current region models.

Initially, the algorithm generates a list $L = \{\ S' \mid S' \in 2^S \ \wedge \ |S'| \ \geq \ \phi\ \}$, where S is the set of models in the current regions and ϕ is the minimum amount of regions for agglutination, descendingly sorted by $|S'|$. For each $S' \in L$, the algorithm evaluates pairs of models $(\mathcal{M}_i^T, \mathcal{M}_j^T) \in P = \{(x, y) \in S' \times S' |\ x \neq y\}$. For each pair of model $p \in P$, the algorithm verifies if the greatest deviation from one model to another within a window of w predictions, deviates beyond a portion ψ of the tolerance $\mathcal{E}_V$, defined by the interest in which the models were generated.

A set of regions corresponding to a set of model S' is agglutinated when: (i) there is at least one model $\mathcal{M}_x^T$ of this set whose the maximum deviation when compared to other models is equal to or less than a portion ψ of tolerance $\mathcal{E}_V$; and (ii) this deviation is equal to or less than the deviation of any other model in the set when performed the same comparison, i.e.,

$$\exists x \neg \exists y \{x, y \in S' \mid f(y, S', w) < f(x, S', w)\}$$

where $f(x, S', w)$ is the function that calculates the maximum deviation of the model x in relation to all models of S', within a window of w predictions.

When the algorithm finds a condition to perform an agglutination of a set of regions, it builds and stores a new message of interest in a region that encompasses all sensor nodes related to the evaluated set of models. This message has the model $\mathcal{M}_x^T$, representing the agglutinated region. After that, the algorithm removes the models replaced by $\mathcal{M}_x^T$ and generates a new list L. This process is repeated until the generated set L is empty

Algorithm 3 ST-DBP Regions Agglutination Algorithm

```
1: procedure RegionsAgglutination
2:     done ← false
3:     L ← { S' | S' ∈ 2^S ∧ |S'| ≥ φ } ordered by |S'| desc
4:     Agg ← { }
5:     while |L| > 0 and ¬ done do
6:         done ← true
7:         for each S' ∈ L do
8:             minDeviation ← ∞
9:             model ← null
10:            for each s ∈ S' do
11:                maxDeviation_S ← 0
12:                for each p ∈ {(x, y) ∈ S' × S' | y ≠ x ∧ x = s} do
13:                    deviation ← checksPair(p.x, p.y, w)
14:                    if deviation > maxDeviation_S then
15:                        maxDeviation_S ← deviation
16:                    end if
17:                end for
18:                if maxDeviation_S < minDeviation then
19:                    minDeviation ← maxDeviation_S
20:                    model ← s
21:                end if
22:            end for
23:            if checkDeviation(minDeviation, ψ, E_V) then
24:                I ← makeInterest(makeSphere(S'), model)
25:                Agg ← Agg ∪ {I}
26:                S ← S \ S'
27:                done ← false
28:                Break
29:            end if
30:        end for
31:        L ← { S' | S' ∈ 2^S ∧ |S'| ≥ φ } ordered by |S'| desc
32:    end while
33:    if |Agg| > 0 then
34:        Agg ← Agg order by (4πr^3)/3 desc
35:        for each A_i ∈ Agg do
36:            Q_i.insert(A_i)
37:        end for
38:    end if
39: end procedure
```

or none agglutination has been carried out in the L models. At the end, the algorithm inserts the generated interests, in descending order by the region volume, into the transmission queue to be later announced.

VII. ST-DBP Evaluation

In this Section, we compare ST-DBP with the original DBP technique. This evaluation is carried out in an environment that simulates the communication among nodes and all mechanism properties described in Sections IV and V.

Our evaluation uses real temperature data, collected during 10 days by 10 sensors on a Solar Farm. The sampling period is one minute, giving a total number of 14,400 samples. We use the suppression ratio (SR) [1] as the first comparison metric:

$$SR = 1 - \frac{messages\ generated\ with\ prediction}{messages\ generated\ without\ prediction} \quad (1)$$

SR varies from 0 to 1. The greater is its value, the more efficient is the technique.

We use the same parameters as in [1] for the value $\mathcal{E}_V$ and time $\mathcal{E}_T$ tolerances, and the size of the learning window w and its borders l. The only exemption is the relative error ϵ^{rel}, which defines an interval of values. We also define the parameters ϕ and ψ of the ST-DBP agglutination algorithm, executed in our evaluation at every 15 minutes, whenever the number of current models is greater or equal to ϕ. All parameters used during the evaluation are defined in Table I.

The network deployment consists in a sensor distribution along a single solar array string. We consider that all models are delivered to all nodes within an interest region. This assumption is made, because the ST-DBP is sensitive to the behavior of the lower-layer network protocol, especially routing. It because, in an ideal case, all model messages must be delivered to all sensors nodes inside of the interest region. Since all sensor nodes have the same `dataperiod` and their clocks are synchronized, the end-to-end delay is a factor of impact too for the mechanism performance. However, it is out of this work, and future actions will focus on this evaluation.

TABLE I: Parameters used during the evaluation.

ϵ^{rel}	ϵ^{abs}	$\mathcal{E}_T$	w	l	ϕ	ψ
(0 to 10 at every 0.5)%	0.5°C	2	6	3	2	0.2

We compare two ST-DBP versions with the original DBP. The first one, named ST-DBP W/oAgg, does not implement the agglutination algorithm (Algorithm 3). So, it only uses the geographic proximity of the announced models. The second version, named ST-DBP Full (or only ST-DBP), is the complete version of the ST-DBP with all algorithm described in Section VI. These two versions allow us to compare the performance gains related to the agglutination mechanism.

Figure 3 shows the obtained results. ST-DBP has presented significant better results compared to the original DBP. With a low ϵ^{rel} of 1.5%, ST-DBP has a difference in the suppression ratio of 3.17%, totaling 97.94%, which is a significant gain because the DBP suppression ratio is already high. With ϵ^{rel} of 10%, ST-DBP Full has present the higher suppression ratio, of 99.7%. When compared to the previous work [1], with ϵ^{rel} of 5%, ST-DBP has reached a suppression ratio of 99.32% against a ratio of 98.18% from the original DBP.

Moreover, the ST-DBP without agglutination (ST-DBP W/oAgg) has also shown better performance than the original DBP. For instance, although small, with a ϵ^{rel} of 5%, the difference between the two approaches was 0.48%.

The results of simulations of 10 days, with ϵ^{rel} of 1.5%, are summarized in Table II. With the ST-DBP Full, in this configuration, occurred 2838 models updates, 428 region partitions, and 130 agglutinations. The ST-DBP Full has reached a maximum suppression interval of 5 hours and 20 minutes (19200 s), against a maximum of 2 hours and 47 minutes (10020 s) of the original DBP. In average, ST-DBP Full increased the suppression intervals by 0.92 times when compared to the original DBP.

An explanation to this gain promoted by the ST-DBP is the main characteristic, of grouping the sensor nodes in regions. The frequency plot depicted in Figure 4, presents a comparison of the distribution of partitioned regions number along of the simulation.

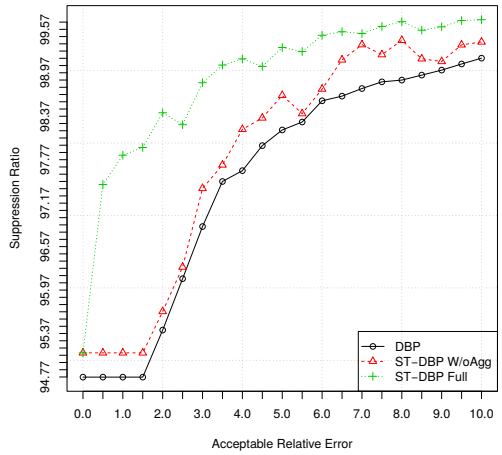

Fig. 3: Suppression ratio of the evaluated parameters in the DBP, ST-DBP W/oAgg, and ST-DBP Full.

TABLE II: Summary of the simulations with ϵ^{rel} of 1.5%.

Metric	DBP	ST-DBP	
		W/oAgg	Full
Mean of partitioned regions	–	9.17	3
Agglutinations / 2h	–	–	1.08
Model Updates / 5 min	2.62	2.45	0.98
Partitions / 30 min	NA	0.02	0.89
Maximum suppression interval	10020 s	19200 s	19200 s
Mean suppression interval	413.79 s	430.28 s	639.29 s

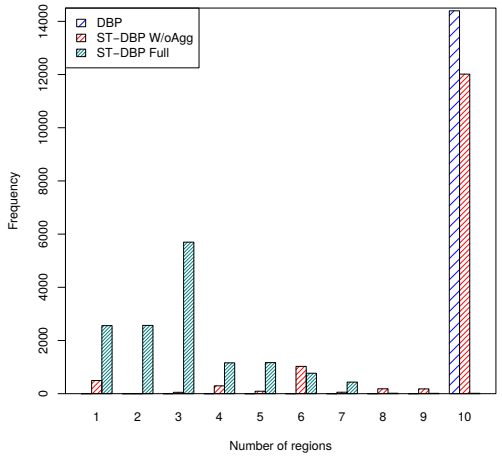

Fig. 4: Distribution of partitioned regions number, with ϵ^{rel} of 1.5%.

For this comparison, we consider the number of regions partitioned with the original DBP, the maximum possible for the entire simulation interval, because there is one model for each of the ten sensor nodes. With ST-DBP W/oAgg the region of interest reached the maximum partitioning after 39.66 hours, which corresponds to 83.47% of the simulation time. ST-DBP Full remained 75.18% of the time with a regions number less than or equal to three, 24.52% between four and seven regions, and less than 0.3% above seven regions.

VIII. Conclusion

None of the recent proposed data suppression techniques based on time series prediction explores the spatial variation of time series. In this paper, we present an extension of the DBP data suppression technique, named Space-Time Derivative-based Prediction (ST-DBP). ST-DBP represents data from a set of sensors within a region in a single model. We evaluated ST-DBP using real data from a Solar Farm and compared it with the original DBP in terms of suppression ratio. ST-DBP outperformed DBP for up to 3.17%. As future work, we plan to evaluate the energy consumption gains of ST-DBP using a real WSN platform.

References

[1] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, "Practical data prediction for real-world wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2231–2244, Aug 2015.

[2] T. Barton and P. Musilek, "Derivative based prediction with look ahead," in *2016 International Joint Conference on Neural Networks (IJCNN)*, July 2016, pp. 2118–2123.

[3] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data prediction + synchronous transmissions = ultra-low power wireless sensor networks," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: ACM, 2016, pp. 83–95. [Online]. Available: http://doi.acm.org/10.1145/2994551.2994558

[4] B. Ghaleb, A. Al-Dubai, and E. Ekonomou, "E-trickle: Enhanced trickle algorithm for low-power and lossy networks," in *2015 IEEE International Conference on Ubiquitous Computing and Communications*, Oct 2015, pp. 1123–1129.

[5] A. A. Frohlich, A. M. Okazaki, R. V. Steiner, P. Oliveira, and J. E. Martina, "A Cross-layer Approach to Trustfulness in the Internet of Things," in *9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, Paderborn, Germany, 2013, pp. 1–8. [Online]. Available: http://www.lisha.ufsc.br/pub/Frohlich_SEUS_-2013.pdf

[6] D. Resner and A. A. Fröhlich, "Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks," in *20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2015).*, Luxembourg, Luxembourg, Sep. 2015, pp. 1–8. [Online]. Available: http://www.lisha.ufsc.br/pub/Resner_ETFA_-2015.pdf

[7] D. Tulone and S. Madden, "An energy-efficient querying framework in sensor networks for detecting node similarities," in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '06. New York, NY, USA: ACM, 2006, pp. 191–300. [Online]. Available: http://doi.acm.org/10.1145/1164717.1164768

[8] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco, "What does model-driven data acquisition really achieve in wireless sensor networks?" in *2012 IEEE International Conference on Pervasive Computing and Communications*, March 2012, pp. 85–94.

[9] X. Xu and G. Zhang, "A hybrid model for data prediction in real-world wireless sensor networks," *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–1, 2017.

[10] T. Palpanas, M. Vlachos, E. Keogh, and D. Gunopulos, "Streaming time series summarization using user-defined amnesic functions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 7, pp. 992–1006, July 2008.

[11] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time os," *Real-Time Systems*, vol. 49, no. 6, pp. 669–714, Nov 2013. [Online]. Available: https://doi.org/10.1007/s11241-013-9183-3

[12] D. Resner, G. M. de Araujo, and A. A. Fröhlich, "On the Impact of Dynamic Routing Metrics on a Geographic Protocol for WSNs," in *Brazilian Symposium on Computing Systems Engineering.*, João Pessoa, Brazil, Nov. 2016. [Online]. Available: http://www.lisha.ufsc.br/pub/Resner_SBESC_2016.pdf