Experiments with Self-stabilizing Distributed Data Fusion

B Ducourthial, Véronique Cherfaoui

Experiments with Self-stabilizing Distributed Data Fusion

B. Ducourthial and V. Cherfaoui
Sorbonne universités, Université de Technologie de Compiègne,
UMR CNRS UTC Heudiasyc 7253, CS 60 319, 60 203 Compiègne Cedex, France

***Abstract*—The Theory of Belief Functions is a formal framework for reasoning with uncertainty that is well suited for representing unreliable information and weak states of knowledge. In a previous work, a distributed algorithm for computing data fusion on-the-fly has been introduced, avoiding gathering the data on a single node before computation. In this paper, we present an experimental study of its properties. This algorithm is self-stabilizing and runs on unreliable message passing networks. It converges in finite time whatever is the initialization of the system and for any unknown topology. First we explain the algorithm implementation on an unreliable message passing environment and we implement a simple use-case. Then, by experimenting with this distributed application on a realistic network emulator, we show its interest for enforcing local confidence using close nodes, saving bandwidth and warning dangers. Moreover, we focus on the interesting connections between the data fusion operator and the self-stabilizing properties and we highlight the importance of the discounting.**

I. Introduction

Algorithms for gathering data spread out over a network of communicating processing units are well known [22], [27], [6]. However, in the real world, information is almost always tainted with various kinds of imperfections, such as imprecision, uncertainty, ambiguity, etc. Following [9], a variable X taking its values in Ω (domain or *frame of discernment*), could be represented as a pair $(value, confidence)$. The $value$ component corresponds to a subset of Ω while the $confidence$ component is an indication on the reliability of the item of information. Imprecision is related to the $value$, uncertainty is related to the $confidence$. For instance, when using the output of any device (sensor, algorithm, model, expert...), it would be preferable to distinguish between the following pieces of information: "the value is between 15 and 25", "the value is probably 20", "the value is probably between 15 and 25". The first one is imprecise but certain, the second is precise but uncertain while the last one is both imprecise and uncertain.

Exchanging data in real world applications requires managing the confidence in both the messages and the nodes of the network. Different methods were developed to this end. Some of them manage the trust model with reputation mechanisms [30], [17] or as a result of interactions between different network nodes [28], [29]. Others aggregate the data to take advantage of the redundancy of information without taking into account the nature of the source [16], [21]. However when the node is highly mobile (as in vehicular networks for instance), the same information can be received several times by the same node. In this case, data fusion is an interesting approach for managing uncertainties.

Data fusion is a set of methods for combining data from different sources to make a decision and reduce uncertainty on the final result. It allows modelling and taking into account the uncertainties in both the data and the sources of information. In particular, the Theory of Belief Function (or Dempster-Shafer Theory) is well suited for representing unreliable information and weak states of knowledge [23], [4]. It offers a diversity of combination operators.

Data fusion can be centralized or decentralized (distributed) [18]. In [15], a protocol for distributed data aggregation is proposed. It relies on the belief functions framework but requires a spanning tree for dealing with the loops in the networks. In [31], the belief functions framework is used to build a distributed confidence over a dynamic network without spanning tree. However in all these applications, the network is supposed to be reliable. In [2], a self-stabilizing algorithm is studied for computing the average of sensors values. However it does not rely on the belief function framework and does not take uncertainties into account. We proposed in [12] a self-stabilizing distributed algorithm reasoning with uncertainties in the belief functions framework. In [20], we report a complete application of our algorithm for detecting icy roads.

Our distributed data fusion algorithm allows each node processing received information locally using discounting [24] and cautious rule [5] operators. It computes the distributed confidence on each node, taking into account local and remote information, giving more importance to closer sources of information. This algorithm is designed for unreliable message passing environments. As it is self-stabilizing [7], [8], it recovers a correct behavior after finite time starting from an arbitrary global state caused by a transient fault, such as a fake message. In this paper, we present an experimental study of the intrinsic properties of this algorithm.

We begin by explaining the implementation of the algorithm on an unreliable message passing environment. For experimentation purpose, we implement a simple use-case using the Airplug Software Distribution [11]. This use-case (borrowed from the meteorology) is generic enough to inspire new applications; it is simple enough to facilitate our study. Next, by using the realistic network emulator of the Airplug suite, we show the interest of the algorithm to enforce the local confidence using the close nodes, to warn an approaching danger and to save the bandwidth. Finally, we show the

importance of the discounting. A weak discounting enlarges the nodes influence while a large discounting ensures rapid convergence. In any case, discounting is required for self-stabilization.

The rest of the paper is organized as follows. In Section II, we describe the distributed system and our distributed data fusion algorithm. In Section III, we explain its implementation in unreliable messages passing environment and we introduce the simple use-case application. The experimental study is detailed in Section IV. Section V ends the paper with some concluding remarks.

II. Self-stabilizing Distributed Data Fusion Algorithm

A. Distributed System

We consider a distributed system $\mathcal{S}$ composed of communicating computing nodes. Each node includes a local memory and a sequential computing unit so that it is able to run a local algorithm. Nodes are not synchronized. The local memory of node v is composed by its *private memory* $\texttt{PRIV}_v$, an *input memory* $\texttt{IN}_v$ and an *output memory* $\texttt{OUT}_v$. The private memory of v contains its *direct confidence*, regularly updated with an external local device (e.g., a sensor). The input memory is used to store all received messages. The output memory will store the result of the local computation on v, namely its *distributed confidence*.

Communications are done through a simple atomic action called `push`: when a *sender* node u executes `push(m)`, the value m stored in its output memory is copied into the input memories of some *receiver* nodes $v_1, v_2, \ldots, v_k$. Nodes can move, disturbing the communications. The receivers of a `push` action on v are not known from the sender v and do not know v. They are determined by the current topology of $\mathcal{S}$ and could be different from those of a previous `push` by the same node v.

This communication scheme can be implemented on a wireless network with a link capacity of a single message: a `push` is implemented using a local broadcast followed by an idle period longer than the maximal communication duration (which is bounded in wireless protocols such as IEEE 802.11). Nodes movements and collisions add/delete links according to the communication range. The framework used for our experiments implements such a local broadcast.

We assume transient faults sometimes occur to memories or messages. They are circumvented by the self-stabilizing property of our algorithm.

B. Distributed data fusion algorithm

On each node v, the private memory $\texttt{PRIV}_v$ is regularly updated using a local external device (sensor, other algorithm...). However, such piece of information is uncertain and imprecise and it is then represented as a basic belief assignment, called the *direct confidence* of v. Thanks to Algorithm 1, it will be combined on-the-fly with direct confidences of other nodes to produce the *distributed confidence* of v. The aim is to circumvent uncertainty and imprecision of the local external device by using information provided by other nodes.

a) Data set: The state of belief of a node is expressed on a *frame of discernment* Ω using a *basic belief assignment* (BBA for short). Such a BBA can be represented by several means, the most common one being with a *mass function*. A mass function m^Ω is a mapping from the set of subsets of Ω, denoted $\mathcal{P}(\Omega)$, to the set of *masses* $[0,1] \subset \mathbb{R}$ such that $\sum_{X \subset \Omega} m^\Omega(X) = 1$.

A set $X \subset \Omega$ such that $m(X) > 0$ is called a *focal set*. If every focal set X satisfies $|X| = 1$, m is said to be Bayesian and it corresponds to a probability mass function. However, the main interest of the theory of belief functions is to consider every subset X of Ω.

The more confident in X a node is, the higher $m^\Omega(X)$ is. If the empty set $\emptyset$ is not a focal set, the mass is *normal*. A mass on $\emptyset$ is used to model conflict between pieces of evidence on which m is based. If Ω is not a focal set, the mass is *dogmatic*. A mass on Ω is used to model lack of knowledge. The higher $m^\Omega(\Omega)$ is, the less informative the mass function m^Ω is. If $m^\Omega(\Omega) = 1$, the mass function is *vacuous*. Finally, a mass function is *simple* if it admits at most two focal sets including Ω.

Besides classical mass functions, a basic belief assignment can be represented by other functions, such as commonality and weights functions, though their definition is less intuitive. Our algorithm works with weights, which are obtained from masses using commonalities [25] [5], as summarized in Figure 1.

As we consider non dogmatic and *separable* mass functions ([23] Chapter 4), the weights belong to $(0,1]$, which is discretized [12] from ϵ (the smallest weight) to 1. Our data set (denoted by $\mathbb{K}$ in the following) is then the set of vectors of discretized weights, one component per subset of Ω except Ω.

b) Cautious operator: To combine vectors of weights, we use the cautious operator denoted by $\textcircled{\scriptsize 1}$ [5]. This idempotent operator solves the *data incest* problem which appears when the information from a given source is taken into account several times. More generally, idempotency is required for ensuring convergence in a network with circuits [10].

The cautious operator is based on the *Least Commitment Principle*, which states that: "when several belief functions are compatible with a set of constraints, the least informative one should be selected". It is easily computed on two vectors of weights by taking the minimum of each component. We denote by $\mathbf{w}_\perp$ (resp. $\mathbf{w}_\top$) the element of $\mathbb{K}$ composed only with the smallest weight ϵ (resp. the largest weight 1), which are respectively the absorbing element and the neutral element of the cautious operator $\textcircled{\scriptsize 1}$.

c) Discounting: When a node u sends its distributed confidence using `push`, the vector of weights will be copied into the input memories of some receivers. On such a receiver v, it will be *discounted* using a mapping $\mathbf{r} : \mathbb{K} \to \mathbb{K}$ before the computation with $\textcircled{\scriptsize 1}$.

As the operator $\textcircled{\scriptsize 1}$ is associative, commutative and idem-

mass function	commonality function	weight function
$m: \mathcal{P}(\Omega) \to [0,1]$	$q: \mathcal{P}(\Omega) \to [0,1]$	$\mu: \mathcal{P}(\Omega) \setminus \Omega \to \mathbb{R}^+$
$A \mapsto m(A)$	$A \mapsto q(A)$	$A \mapsto w(A)$
$\sum_{A \subset \Omega} m(A) = 1$	$q(A) = \sum_{B \subset \Omega, A \subseteq B} m(B)$	$\mu(A) = \Pi_{B \subset \Omega, A \subseteq B} q(B)^{(-1)^{\lvert B \rvert - \lvert A \rvert + 1}}$

Fig. 1. Definition of the weight functions obtained from masses using commonalities

potent, it defines an order relation denoted $\prec_{\textcircled{\scriptsize}}$ on $\mathbb{K}$. If $\mathbf{r}(\mathbf{w}) \prec_{\textcircled{\scriptsize}} \mathbf{w}$, the algorithm always converges to $\mathbf{w}_\perp$ except on acyclic networks; if $\mathbf{r}(\mathbf{w}) = \mathbf{w}$ the algorithm stabilizes on any network but does not support transient fault [14]. When $\mathbf{w} \prec_{\textcircled{\scriptsize}} \mathbf{r}(\mathbf{w})$, the discounting ensures the self-stabilizing property of Algorithm 1 [12].

In the latter case, the function $\mathbf{r}$ is called a *discounting* because it is used to *decrease* the information in a given basic belief function. It allows taking information from remote nodes into account while giving a larger importance to close nodes. As a consequence, the distributed confidence computed on a node reflects the local situation of the node. On the contrary, without discounting ($r(\mathbf{w}) = \mathbf{w}$), a single result is obtained per connected component in the network. When the information admits a local meaning (such as the weather forecast in our use-case application), the result on a node u should differ from the result of a far away node v except if all the nodes agree on their direct confidence. Hence, while it is useful to take into account remote information, all the nodes should not always converge to the same belief function.

d) Algorithm: Now that we have defined the data set and the operator used to combine the data, the algorithm is described as follows. The direct confidence of each node is regularly updated by an external mean, as explained previously, and stored in the private memory $\texttt{PRIV}_v$. It is coded (as all other variables) by a vector of weights belonging to $\mathbb{K}$. The input memory $\texttt{IN}_v$ on node v stores all data pushed by some ancestors[1] since the last timer expiration. The output memory $\texttt{OUT}_v$ contains the distributed confidence computed by v.

Nodes are not synchronized. Timers are given by local clocks and may have an unbounded drift. Upon timer expiration, each node computes its distributed confidence by combining its own direct confidence with those it has received since the last timer expiration, using the operator $\textcircled{\scriptsize}$ and after discounting of the received vectors using $\mathbf{r}$. It also pushes its result.

Algorithm 1: **Distributed Confidence, node** v

1 Upon timer expiration:
2 $\quad \texttt{PRIV}_v \leftarrow$ current direct confidence
$\quad \triangleright$ *Initializing the iterative computation*
3 $\quad \texttt{OUT}_v \leftarrow \texttt{PRIV}_v$
4 $\quad$ **for each** entry u in $\texttt{IN}_v$ **do**
$\quad\quad \triangleright$ *Iterative computation of the output*
5 $\quad\quad \texttt{OUT}_v \leftarrow \texttt{OUT}_v \textcircled{\scriptsize} \mathbf{r}(\texttt{IN}_v[u])$
6 $\quad$ **end for**
$\quad \triangleright$ *Sending the distributed confidence to neighbors*
7 $\quad$ **push(** $\texttt{OUT}_v$ **)**
8 $\quad$ Reset $\texttt{IN}_v$
9 $\quad$ Restart the timer

Assuming the topology is stable and the direct confidences stabilized, Algorithm 1 converges in finite time to the legitimate configuration, whatever are the values in memories IN and OUT and the messages in transit [12]. In the legitimate configuration, the output memory of each node v is equal to the combination of all the private memories of its ancestors u (belonging to Γ_v), discounted as many times as the number of hops from u to v:

$$\texttt{OUT}_v = \textcircled{\scriptsize}_{u \in \Gamma_v} \mathbf{r}^{\text{dist(u,v)}}(\texttt{PRIV}_u)$$

Let k be the smallest integer satisfying $r^k(\mathbf{w}_\perp) = \mathbf{w}_\top$. Supposing a synchronous system, the stabilization time is $O(k)$ because a node builds its result with only nodes at less than k hops and any incorrect value disappears after k hops.

III. Implementation and application

In this section, we present a generic implementation of our algorithm that we use for designing a simple use-case: a basic distributed meteorological application. Besides the illustration purpose, this application will be used to experiment with Algorithm 1 in the next section. We begin by the basic meteorological application.

A. A basic distributed meteorological application

We consider a distributed system where each node is able to measure the local atmospheric pressure (using a sensor or any other device), allowing us to deduce a weather forecast. Such a device could be damaged or disturbed; α is the proportion of time the sensor is not working correctly.

Instead of working with measurements (intervals of $\mathbb{R}$), a simple frame of discernment $\Omega = \{\texttt{rainy}, \texttt{cloudy}, \texttt{sunny}\}$ is used, where rainy corresponds to low athmospheric pressure and sunny to high pressure. Each node computes its direct confidence as a mass function on Ω using sigmoid functions (Figure 2) so that their sum is equal to $1 - \alpha$, leaving α for the Ω component representing the uncertainty[2].

The mass function is then converted to a vector of weights (Figure 3). It is used to compute the distributed confidence (Algorithm 1). Periodically, the last computed distributed confidence is sent to the neighbors.

When a message is received by a node, it is saved, erasing the last message stored for this sender. Figure 4 shows the

[1] Neighbors in the directed graph.

[2] For the sake of simplicity, here we use sigmoids only on singletons.

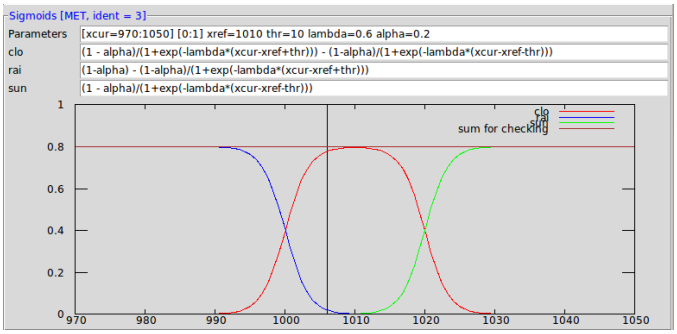

Fig. 2. Using sigmoids to determine the BBA from local pressure (here 1006 hPa).

Direct confidence [MET, ident = 3]

	rai	rai clo	clo	clo sun	sun	sun rai	rai clo sun
Masses	0.0213	0	0.7785	0	0.0002	0	0.2
Weights	0.9038	1.0000	0.2044	1.0000	0.9990	1.0000	

Fig. 3. Direct confidence computed from the local measurement (here 1006 hPa) using sigmoids (Fig. 2), given as mass and weight vectors, with components belonging to $\mathcal{P}(\Omega)$.

distributed confidences sent by Nodes 1, 5 and 7 and received by Node 3 (nodes may also send their direct confidence for computing the neighborhood confidence; this is not described here, see [12]).

Receiving [MET, ident = 3]

Host	Nseq	Last direct confidence sent by this host (as masses)
5	494	clo,rai,sun:0.2;clo,sun:0;rai:0.7980;:0;rai,sun:0;clo,rai:0;sun:0.0000;clo:0.0020
7	494	clo,rai,sun:0.2;clo,sun:0;rai:0.7621;:0;rai,sun:0;clo,rai:0;sun:0.0000;clo:0.0379
1	495	clo,rai,sun:0.2;clo,sun:0;rai:0.0000;:0;rai,sun:0;clo,rai:0;sun:0.4000;clo:0.4000
Host	**Nseq**	**Last distributed confidence sent by this host (as weights)**
5	494	clo,rai,sun:;clo,sun:1.0000;rai:0.2004;:1.0000;rai,sun:1.0000;clo,rai:1.0000;sun:1.0000;clo:0.5333
7	494	clo,rai,sun:;clo,sun:1.0000;rai:0.2079;:1.0000;rai,sun:1.0000;clo,rai:1.0000;sun:1.0000;clo:0.4333
1	495	clo,rai,sun:;clo,sun:1.0000;rai:1.0000;:1.0000;rai,sun:1.0000;clo,rai:1.0000;sun:0.3004;clo:0.3333

Fig. 4. Node 3 stores the last messages received from its neighbors (here Nodes 1, 5 and 7); they contain the distributed confidence computed by the sender.

Periodically, each incoming distributed confidence is discounted and then combined with the local direct confidence using the cautious operator. Figure 5 shows the vectors of weights extracted from the last received messages (Fig. 4) and the distributed confidence computed using a term-to-term minimum, after applying the discounting function (here $x \mapsto \min(1, x + 0.1)$) to all received vectors. The direct confidence of the local node (here 3) is not discounted. This computed distributed confidence is periodically sent to the neighbors (Figure 7).

Distributed confidence [MET, ident = 3]: 224 updates

Update | Stop | delay (ms): 2000 | Discounting function f(wcur): min(1,wcur+0.1)

WEIGHTS	rai	rai clo	clo	clo sun	sun	sun rai
Distributed	0.3004	1.0000	0.2044	1.0000	0.4004	1.0000
3	0.9038	1.0000	0.2044	1.0000	0.9990	1.0000
1	1.0000	1.0000	0.3333	1.0000	0.3004	1.0000
5	0.2004	1.0000	0.5333	1.0000	1.0000	1.0000
7	0.2079	1.0000	0.4333	1.0000	1.0000	1.0000

Fig. 5. Computing the distributed confidence on node 3. Vectors of weights received from nodes 1, 5 and 7 are first discounted (increased by 0.1 here). Then a term-to-term minimum is computed, giving the result, as shown by red arrows.

By considering focal sets of cardinality larger than one (e.g., $\{\text{rainy}, \text{cloudy}\}$), the theory of belief functions generalizes the Bayesian probability theory and is well adapted for representing weak states of knowledge. Nevertheless, when a decision has to be taken, one needs to go back to focal sets of cardinality one. For this purpose the computed distributed confidence (expressed as a vector of weights) is converted in a mass function m which is then mapped to a *pignistic probability* function [26], defined by: $BetP(A) = \sum_{\emptyset \neq B \subset \Omega} m(B) \frac{|A \cap B|}{|B|}$.

Figure 6 shows the pignistic probabilities computed from the direct and distributed confidence (neighborhood confidence is not described here, see [12]).

Confidences summary [MET, ident = 3]

MASSES	rai	rai clo	clo	clo sun	sun	sun rai	rai clo sun
Direct	0.0213	0	0.7785	0	0.0002	0	0.2
Neighborhood	0.5655	0.0000	0.3680	0.0000	0.0443	0.0000	0.0221
Distributed	0.2673	0.0000	0.4464	0.0000	0.1716	0.0000	0.1147

PIGN. PROBA.	rai	clo	sun
Direct	0.0880	0.8452	0.0669
Neighborhood	0.5729	0.3754	0.0517
Distributed	0.3055	0.4846	0.2098

Fig. 6. Pignistic probability computed from the distributed confidence.

B. Implementation

We implemented Algorithm 1 using the Airplug Software Distribution [1], [11], dedicated to the design, study and experiment of applications in dynamic networks. The Airplug framework allows implementation of distributed applications in message passing environment with simple conventions. Any language can be used providing that the application is able to asynchronously read from its standard input for receiving messages and to write to its standard output for sending messages.

Algorithm 1 has been implemented as a generic Airplug application (called MET) written in Tcl/Tk, accepting any frame of discernment, sigmoid model for BBAs, discounting and parameters. It has been used to program the simple meteorological application (Figures 2 to 7). Libraries for sets, masses, weights, conversions and operators represent about 750 lines. The MET application including the GUI represents 2020 lines. It can receive measurements from another local Airplug application connected to a sensor [19] or can generate itself values for testing purposes, using a given function and an initial value. We use this functionality in the next section.

Airplug applications can be used in several *modes* without having to modify them. Airplug-term allows rapid prototyping using a Linux-based computer; communications are managed using the shell facilities. Airplug-live allows rapid deployment in embedded computers; communications are done using wireless broadcast [13]. Airplug-emu emulates the network in order to prepare or replay real tests and to easily study the properties of an application by varying some parameters [3]. We use this mode for studying MET in the next section.

When using Airplug-emu, the scenario is described in an XML file; it includes the number of nodes, their initial positions and movements, their applications and options... Nodes movements can be given as GPS traces obtained during real experiments (among other means). Airplug-emu creates

or deletes the communication links depending on the relative positions of the nodes, the given wireless communication range and the link reliability. When a MET instance sends a message, all MET instances running in the neighborhood (defined by the range and the reliability) will receive it. Additionally, as shown in Figure 7, it is possible to send forged distributed confidence to neighbors for studying the robustness of the application against erroneous messages.

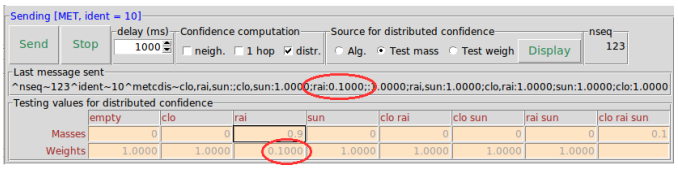

Fig. 7. A node can push either its last computed distributed confidence or a forged vector of weights for testing purpose, entered as masses or weights (see Figure 10).

IV. Experiments and results

Obviously the basic meteorological application described in the previous section relies on a too simple model to be useful for weather prediction. However this simplicity is interesting to understand and study the properties of distributed data fusion, knowing that Algorithm 1 and its Airplug implementation can be applied to any measurement and more complex frames of discernment Ω. We first illustrate the behavior of the MET application in a network, explaining how to interpret the results. Then we explain the interest of distributed data fusion. Finally, we show the interesting self-stabilizing properties of Alg. 1 to tolerate message failures.

A. Result interpretation

Figure 8 presents two screenshots of Airplug-emu on a fixed network (with full-duplex links). Seven instances of the MET application compute their direct and distributed confidences by exchanging messages, as explained in Section III (neighborhood confidence is also computed though it is not described in this paper). On each node, the bars represent the direct, neighborhood and distributed confidence, expressed as pignistic probabilities (grey for `rainy`, white for `cloudy` and yellow for `sunny`).

In the Figure 8-top, there is no discounting. Hence, on this connected network, all nodes converge to the same distributed confidence, equal to the minimum vector of weights (computed term-to-term). As Node 4 measures a high pressure and announces sunny weather, Node 3 cloudy weather and Node 5 rainy weather, the resulting distributed confidence gives the same importance to each, leading to three right-most bars of equal height on each node.

In the Figure 8-bottom, there is a discounting. The local measurements are the same in both networks, leading to the same direct confidences (three left-most bars on each node). Local measurements differ in the network from node to node but the influence of a node decreases with the distance, thanks to the discounting. For example, the rainy weather announced by Node 5 influences the distributed confidence of Nodes 3, 1 and 4 but less and less (the right-most grey bar is smaller and smaller on these nodes).

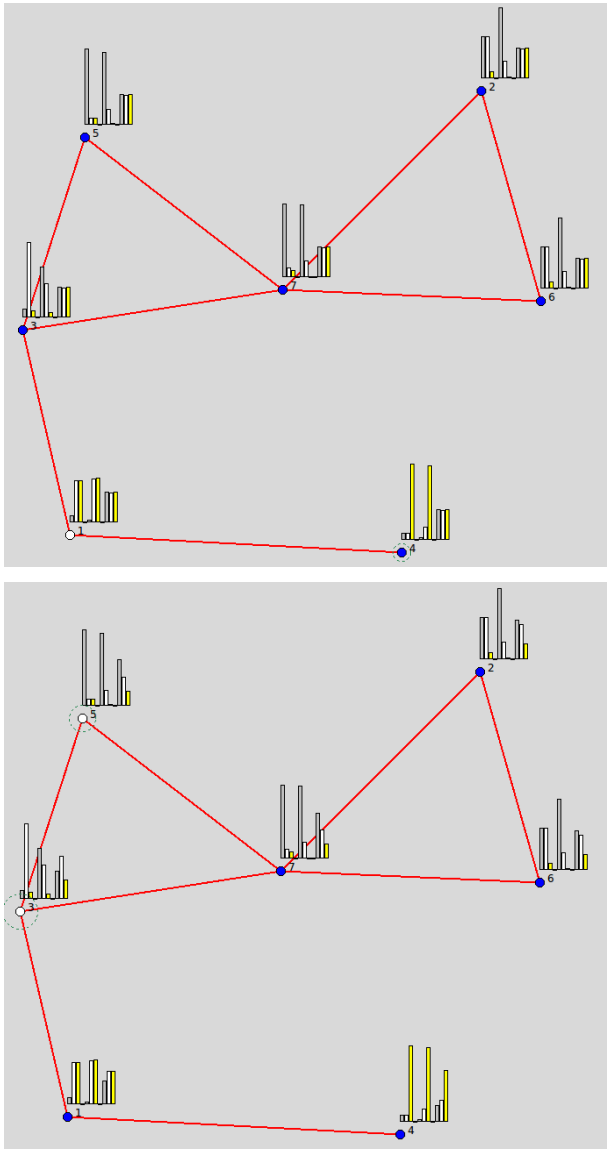

Fig. 8. Simple distributed meteorological application (MET), without discounting (top) and with discounting (bottom). The bars represents the pignistic probabilities (grey for `rainy`, white for `cloudy` and yellow for `sunny`) of the direct confidence (left-most three bars), the neighborhood confidence (middle bars, not described here) and the distributed confidence (right-most three bars).

Consider Node 6. Its direct confidence gives the same importance to rainy and cloudy weather, meaning that it cannot decide between them. It is the same for its neighbor, Node 2. However the direct confidence of their neighbor, Node 7, clearly indicates the weather is rainy. Then the distributed confidence of Nodes 2 and 6 announces rainy weather (their right-most grey bar are larger than their right-most white bar). Hence, the discounting enforces the collaboration between close nodes, which helps to decide.

B. Interest of distributed data fusion

As shown in Figure 8, when using a discounting each node stabilizes on a distributed confidence related to its own local measurement as well as those of other nodes, discounted according to the distance. The more the discounting function $\mathbf{r}$ increases the component of the incoming vector of weights, the smaller the area of influence of a node is. With the discounting $\mathbf{r}(x) = \min(1, x + 0.1)$, a node has no influence at more than 10 hops. This makes sense in a network of sensors where the measurement has a local meaning.

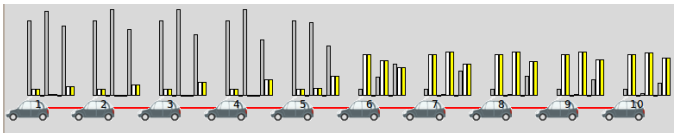

Fig. 9. Using distributed data fusion for prevision. Vehicles 1 to 5 are under rainy weather; other vehicles are warned when approaching (see the three right-most bars).

This approach can also be used to foresee a danger. For instance, Figure 9 displays a screenshot of Airplug-emu with a convoy of vehicles, where vehicles 1 to 5 are experiencing rainy weather, to the contrary of vehicles 6 to 10. This can be shown with the three left-most bars on each vehicle. However, the distributed confidence computed by the vehicles on the left warns about the rain, with an intensity increasing with the distance to the rainy area (see three right-most bars).

Besides these qualitative properties, Algorithm 1 allows saving bandwidth compared to the classical approach relying on a centralization of the data. Indeed, it avoids collecting data using a distributed collect algorithm [22], [6]. The message size is given by the weights discretization and the cardinal of the frame of discernment Ω. These are parameters of MET, set to ten thousandth for the former and 3 for the latter in our studies, leading to about 32 bytes for encoding the distributed confidence. This is much less than in the self-stabilizing collect algorithm for dynamic networks presented in [6] where each node up to distance k would include its identity and its distributed confidence in the message.

The convergence time is equal to k timers, where k is the smallest integer satisfying $r^k(\mathbf{w}_\perp) = \mathbf{w}_\top$ (10 in our case). Computations and sending are done periodically with a timer set to 1 s here (see Figures 5 and 7).

C. Discounting versus self-stabilization

The self-stabilizing property of Algorithm 1 is related to the operator used for local computation [12], [10]. The algorithm stabilizes despite transient faults on memories or messages if and only if the discounting $\mathbf{r}$ is strictly increasing according to the order relation defined by the cautious operator on the vectors of weights. Moreover, the convergence time is related to the smallest integer k satisfying $r^k(\mathbf{w}_\perp) = \mathbf{w}_\top$.

In Figure 10-top, the distributed system converged to the legitimate configuration, defined by the local measurements of each node (here all equal to 1030 hPa). On the bottom, an illegitimate configuration has been reached because Node 10

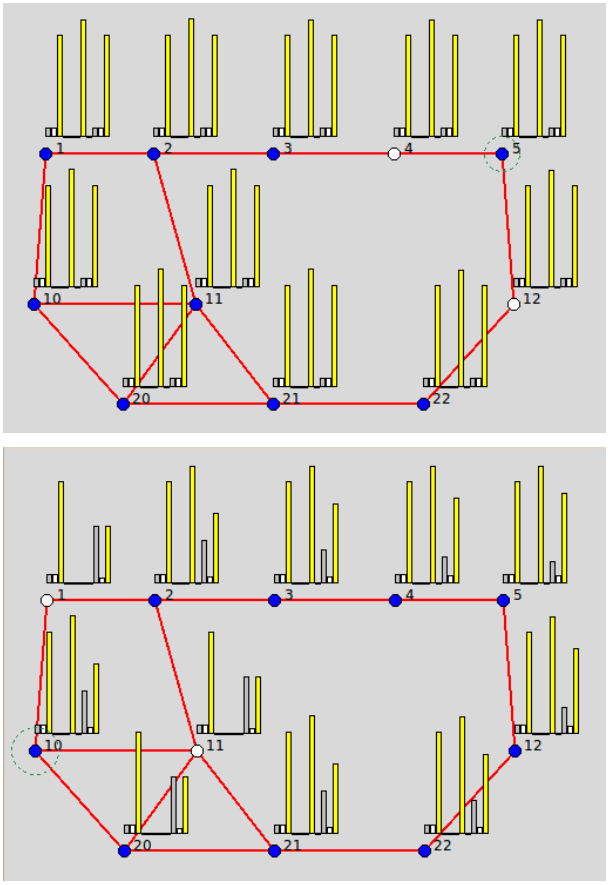

Fig. 10. The illegitimate configuration (bottom) has been obtained from the legitimate configuration (top) after Node 10 sent an erroneous distributed confidence to all its neighbors, as shown in Figure 7. Distributed confidence is represented by the pignistic probabilities with the three right-most bars in each node.

broadcasts an erroneous distributed confidence instead of the one it computes, as shown in Figure 7. Then its neighbors computes an erroneous distributed confidence which propagates through the network and also back to Node 10 itself, which then includes the erroneous confidence in its own computation, after a discounting.

After the transient failure ceases, the system converges from the illegitimate configuration of Figure 10-bottom to the legitimate configuration of Figure 10-top when using the discounting $\mathbf{r}(x) = \min(1, x + 0.1)$. On the contrary, when using $\mathbf{r}(x) = x$, the system remains in the illegitimate configuration of Figure 10-bottom.

Figure 11 shows the evolutions of the distributed confidence, represented by the pignistic probabilities on some of the nodes of the network displayed in Figure 10. While the system has stabilized to the legitimate configuration of Figure 10-top, Node 10 begins to broadcast at date 8 an erroneous distributed confidence, indicating a strongly rainy weather (see Fig. 7). The other nodes then update their distributed confidence and the system stabilizes to the illegitimate configuration of Figure 10-bottom. The farther from Node 10 the node is, the less influenced it is, thanks to the discounting which applies

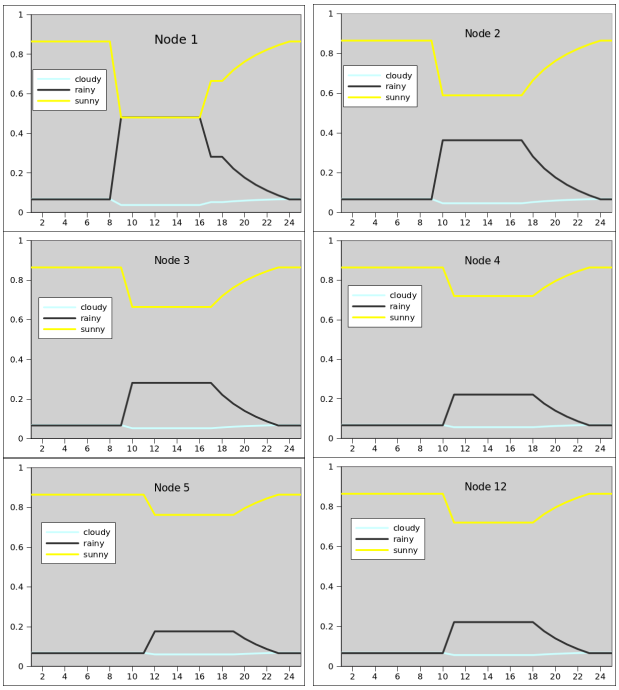

Fig. 11. Self-stabilization with the discounting $\mathbf{r}(x) = \min(1, x + 0.1)$. Node 10 broadcasts the erroneous message of Fig. 7 from date 8 to 18.

at each hop. At date 18, Node 10 ceases to broadcast the forged distributed confidence; this is the end of the transient failure. Then Figure 11 shows that nodes converge back to their legitimate distributed confidence[3].

With a discounting leading to a smaller k, such as $\mathbf{r}(x) = \min(1, x + 0.25)$, the convergence is faster. However the influence of a node is limited to nodes at distance less than 4. This could be a drawback in some applications as the one shown in Figure 9.

Finally, the algorithm is able to stabilize after a topology modification, created in Airplug-emu either by changing the range of the nodes or by using GPS traces for the nodes.

V. Conclusion

In this paper, we presented the self-stabilizing distributed data fusion algorithm first introduced in [12]. Then we explained how it can be implemented as a useful application accepting many parameters to adapt to several contexts. The MET application is an Airplug application accepting any frame of discernment, sigmoids and discounting. It computes the direct confidence thanks to external or internal measures; it exchanges messages to compute the distributed confidence using the discounting and the cautious operator.

We used MET to program a basic distributed meteorological application. Though our model is too simple to be useful for weather prediction, it permitted to illustrate the interest of our approach. Distributed data fusion allows dealing with imprecise data given by cheap sensors. It enforces the local confidence with those of close nodes. Moreover it can be used to foresee a danger. It is well adapted to sensors or vehicular networks, where the measurements have a local meaning. It saves bandwidth compared to centralized approaches with shorter messages.

We highlighted the importance of the discounting. We showed by experimentation that it is required for ensuring the convergence of the algorithm despite transient failures. Moreover, the discounting impacts the influence area of a node on the network as well as the convergence time of the algorithm. This parameter has then to be carefully chosen.

Future work will focus on the use of other data fusion operators in this distributed context.

[3]The stable floor around date 17 in graphic of Node 1 is explained by the fact that data is saved periodically, as for the distributed confidence update. The two timers certainly occur simultaneously and were not scheduled as expected.

Acknowledgment

This work was carried out in the framework of the Labex MS2T, which was funded by the French Government, through the program “Investments for the future” managed by the National Agency for Research (ANR-11-IDEX-0004-02).

References

[1] Airplug web site. https://www.hds.utc.fr/airplug.

[2] J. Bahi, M. Haddad, M. Hakem, and H. Kheddouci. A new reliable and self-stabilizing data fusion scheme in unsafe wireless sensor networks. In *Proceedings of the 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT '10, pages 87–93, Washington, DC, USA, 2010. IEEE Computer Society.

[3] A. Buisset, B. Ducourthial, F. El Ali, and S. Khalfallah. Vehicular networks emulation. In *19^{th} International Conference on Computer Communication Networks (ICCCN)*, August 2010.

[4] A. P. Dempster. Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38:325–339, 1967.

[5] T. Denoeux. Conjunctive and disjunctive combination of belief functions induced by nondistinct bodies of evidence. *Artificial Intelligence*, 172:234–264, 2008.

[6] Y. Dieudonné, B. Ducourthial, and S.-M. Senouci. Col: A data collection protocol for vanet. In *IEEE Intelligent Vehicles Symposium (IV)*, Alcalá de Henares, Spain, June 2012.

[7] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[8] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[9] D. Dubois and H. Prade. Representation and combination of uncertainty with belief functions and possibility measures. *Computer intelligence*, 4:244–264, 1988.

[10] B. Ducourthial. r-semi-groups: A generic approach for designing stabilizing silent tasks. In *9^{th} Stabilization, Safety, and Security of Distributed Systems (SSS'2007)*, pages 281–295, November 2007.

[11] Bertrand Ducourthial. Designing applications in dynamic networks: The Airplug Software Distribution. In Matthieu Roy, editor, *SAFECOMP 2013 - Workshop ASCoMS (Architecting Safety in Collaborative Mobile Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, Toulouse, France, September 2013.

[12] Bertrand Ducourthial, Véronique Cherfaoui, and Thierry Denoeux. Self-stabilizing distributed data fusion. In *Stabilization, Safety, and Security of Distributed Systems*, volume 7596 of *Lecture Notes in Computer Science*, pages 148–162. Springer Berlin Heidelberg, 2012.

[13] Bertrand Ducourthial and Sofiane Khalfallah. A platform for road experiments. In *VTC Spring*, 2009.

[14] Bertrand Ducourthial and Sébastien Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3):147–162, 2001.

[15] Andrea Gasparri, Flavio Fiorini, Maurizio Di Rocco, and Stefano Panzieri. A networked transferable belief model approach for distributed data aggregation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, PP(99), 2011.

[16] P. Golle, D. Greene, and J. Staddon. Detecting and correcting malicious data in vanets. In *1st ACM Workshop on Vehicular Ad hoc Networks (VANET)*, pages 29–37, New York, NY, USA, 2004.

[17] J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. In *2nd International Conference on Trust Management*, pages 48–62, Oxford, UK, 2004.

[18] H.B. Mitchell. *Multisensor Data Fusion: An introduction*. Springer, 2007.

[19] J. Radak, B. Ducourthial, and S. Bonnet. Design and implementation of a vehicular network testbed using wireless sensors. In *8th International Workshop on Wireless Sensor, Actuator and Robot Networks (WiSARN 2014)*, Benidorm, Spain, June 2014.

[20] Jovan Radak, Bertrand Ducourthial, Véronique Cherfaoui, and Stéphane Bonnet. Detecting road events using distributed data fusion: Experimental evaluation for the icy roads case. *IEEE Trans. Intelligent Transportation Systems*, 17(1):184–194, 2016.

[21] M. Raya, P. Papadimitratos, V. D. Gligor, and J-P. Hubaux. On data-centric trust establishment in ephemeral ad hoc networks. In *the 28th IEEE conference on Computer Communications (INFOCOM)*, pages 1238–1246, Phoenix, AZ., USA, April 2008.

[22] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, 29(1):23–34, 1983.

[23] G. Shafer. *A mathematical theory of evidence*. Princeton, N.J, 1976.

[24] P. Smets. Data fusion in the transferable belief model. In *3rd International Conference on Information Fusion*, 2000.

[25] Ph. Smets. The canonical decomposition of a weighted belief. In *Int. Joint Conf. on Artificial Intelligence*, pages 1896–1901, San Mateo, Ca, 1995. Morgan Kaufman.

[26] Ph. Smets. Decision making in the TBM: the necessity of the pignistic transformation. *Int. Journal of Approximate Reasoning*, 38:133–147, 2005.

[27] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[28] G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *ACM Workshop Wireless Security*, pages 1–10, Philadelphia, PA, USA, 2004.

[29] J. Wang and H-J. Sun. A new evidential trust model for open communities. *Computer Standards & Interfaces*, 31:994–1001, 2009.

[30] G. Zacharia and P. Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14:881–907, 2000.

[31] Nicole El Zoghby, Véronique Cherfaoui, Bertrand Ducourthial, and Thierry Denoeux. Distributed data fusion for detecting sybil attacks in VANETs. In Springer-Verlag, editor, *Proceedings of the 2nd International Conference on Belief Functions*, Advances in Intelligent and Software Computing, 2012.