COL: A data collection protocol for VANET

Yoann Dieudonné, Bertrand Ducourthial, Sidi-Mohammed Senouci

COL: a Data Collection Protocol for VANET

Yoann Dieudonné, Bertrand Ducourthial, Sidi Mohammed Senouci

***Abstract*— In this paper, we present a protocol to collect data within a vehicular ad hoc network (VANET). In spite of the intrinsic dynamic of such network, our protocol simultaneously offers three relevant properties: (1) It allows any vehicle to collect data beyond its direct neighborhood (i.e., vehicles within direct communication range) using vehicle-to-vehicle communications only (i.e., the infrastructure is not required); (2) It tolerates possible network partitions; (3) It works on demand and stops when the data collection is achieved. To the best of our knowledge, this is the first collect protocol having these three characteristics.**

All that is chiefly obtained thanks to a specific tool, namely Operator ant, borrowed from the self-stabilization area which confers to our algorithm the nice property to recover by itself from topology changes. In addition to a theoretical proof of correctness, our protocol has been implemented and tested through the Airplug Software Distribution: Road and lab experiments are presented and discussed.

I. Introduction

A. Context

Following the current trend in Automotive Engineering, motor vehicles are equipped with more and more sensors in order to improve the safety of the driver and passengers as well as ride comfort.

Taken individually, local information provided by sensors gives an imperfect knowledge to cars and to their passengers. However, by comparing information collected from several vehicles, the knowledge can be built up. Some works have begun on the diagnosis or the collaborative perception among vehicles. In other terms, by exchanging information and estimations from vehicle sensors and analog computers, information becomes more accurate and relevant and thus, the confidence level is increased. For instance, it has been demonstrated [1], [2] that collaborative localization uncertainty in groups of agents or vehicles is less compared to the situation where individual agents estimate their position separately. Various techniques have been proposed to integrate relative observations, like maximum likelihood estimation [3], particle filters [4], Kalman filters [5], [6], and Monte-Carlo simulation. Although the design of the previous schemes have led to practical implementations and have demonstrated their effectiveness in certain settings through extensive simulations or experiments, before doing so, vehicles need to collect data from the whole network or to a lesser extent in their neighborhood. Through a network operator (NO), we can envisage to directly send and receive collected data on a large scale: Recipients are then vehicles but also the infrastructure managers and users of cellular networks. Various applications are possible, such as traffic estimation, average speed, available parking spaces, etc. While a mere 3G connection can transmit data arising from a single car, it seems more appropriate to aggregate data before forwarding them. This *modus operandi* helps to analyze information from vehicles, to contextually filter and merge them with respect to different selection criteria. As a result, it becomes possible to send only useful information so as to save bandwidth. In this context, the architecture has to be compounded of a data collection protocol specific to highly dynamic networks best known as VANET (which stands for Vehicular Ad Hoc Networks) associated with a protocol for forwarding aggregated data to the core network of the NO. In this paper, we aim to focus on the process of collecting data in VANET.

This work has been supported by Orange Labs (FTR&D), France.

Y. Dieudonné was with Université de Technologie de Compiègne and is now at Université de Picardie Jules Vernes, 80000 Amiens, France. Yoann.Dieudonne@u-picardie.fr

B. Ducourthial is with Lab. Heudiasyc UMR CNRS-UTC, Université de Technologie de Compiègne, 60200 Compiègne, France. Bertrand.Ducourthial@utc.fr

S.-M. Senouci was with Orange Labs and is now with the Institut Supérieur de l'Automobile et des Transports, Université de Bourgogne, France. Sidi-Mohammed.Senouci@u-bourgogne.fr

B. Related works

Most of the data collections in vehicular ad hoc networks are tackled by using a mechanism of dissemination which is a process whereby each vehicle periodically broadcasts information about itself. A large number of data dissemination protocols have been recently proposed within the framework of VANET [7], [8], [9], [10], [11], [12]. For instance, we can refer to opportunistic dissemination, such as [7], as well as geographical dissemination [9]. In the first case, propagation is performed with the use of opportunistic diffusion of data: In particular, messages are stored in each intermediate node and forwarded to every encountered node until the destination is reached. The second one consists in sending the message to the closest vehicle toward the destination until it reaches it. Likewise, many other types of dissemination exist: Thereby, we can mention peer-to-peer [9] and cluster-based dissemination [10]. Notwithstanding, all the dissemination protocols, and by extension every data collection relying on them, are generally not upon request. Consequently, information is recurrently diffused even if it is not necessary, leading to bandwidth waste. Moreover, since vehicles do not know which data will be relevant, they will tend to broadcast more than expected.

To circumvent this problem, data collection would have to issue from a demand started by some initiator. In a fixed network, such a data collection can be achieved with no difficulty through a wave algorithm, the PIF algorithm

being certainly the most emblematic example [13]. The PIF algorithm, like most of general wave algorithms [14], works in two steps –both of them being suggestive of a wave.

The first step corresponds to a *broadcast phase* started by a sink which is called an initiator. During this step, the sink sends a broadcast message to all its neighbors. In particular, in the context of a data collection, this message has to contain the types of data to be collected. The neighbor, that receives the broadcast message for the first time, considers the node that has sent it as its parent and forwards the broadcast message to all its neighbors with the exception of its parent. Behaving like this, a spanning tree is built.

The second step corresponds to a *feedback phase* started by the leaves of the spanning tree. More precisely, when the leaves receive broadcast messages from all theirs neighbors, they send their own data to their parents as feedback for the broadcast message. Obviously, data are related to the types of data which appear in the broadcast message. The other nodes, which are not leaves, will receive the feedback messages with the collected data from their children. These nodes will join their own data to those of their children through a mechanism of aggregation and send them to theirs parents until the sink has taken in all the aggregated data of the complete network.

Unfortunately, due to their intrinsic dynamic, the PIF algorithm is doomed to failure in dynamic networks such as VANET. The deep reason stems from the fact that the PIF algorithm requires the spanning tree to remain invariant: However such a property cannot be fulfilled because links between nodes are subject to incessant breakages.

In [15], the authors bypass the problem by adapting a decentralized wave algorithm from [16] to a vehicular network. Nevertheless, their protocol relies on the assumption that the network remains permanently connected, otherwise their algorithm would be unable to terminate. In particular, it is assumed that no node can disappear but, in reality, this frequently occurs in dynamic networks such as VANET.

C. Contribution

In this paper, we describe a protocol for collecting data in VANET using vehicle-to-vehicle communications only. In our design, every data collection follows a demand including, among others, types of data as well as the maximal duration and depth for the collect process. Contrary to [15], it is not required for vehicular networks to remain continuously connected (the network can temporarily split). To achieve this, each vehicle recurrently confronts its local network view with the other views so as to update it by involving an *r-operator*. An r-operator is a specific tool which brings the nice property of self-stabilization to our algorithm. A self-stabilizing algorithm has the ability to recover by itself from an inconsistent state caused by transient failures. Since topology changes can be viewed as transient failures, our protocol is guaranteed to sustain the dynamic of the vehicular network. In particular, every local view will tend to be quite accurate in spite of the network dynamic.

We proved that our algorithm named COL can be practically implemented. In this way, we built a prototype using the Airplug Software Distribution (ASD) in order to test it via real experimentions on road. ASD is a software suite which allows to develop distributed protocols suited to dynamic networks and allows to validate them via field or lab experiments [17], [27]. In addition to that, we tested our protocol over several ranges of wireless communication scenarios by varying several parameters notably the network dynamic and the communication reliability to demonstrate the robustness of our algorithm.

Due to the lack of space several proofs and experiments are omitted. However, a full version of the paper is available on-line. In addition, a pedagogical movie relates the field experiments [18], [19].

D. Roadmap

The rest of this paper is organized as follows. First, some preliminaries are given in Section II. Then, Section III introduces our data collection protocol. Finally some field and lab experiments are presented in Section IV before concluding the paper in Section V.

II. Preliminaries

In this section, we first introduce the specifications of the data collection problem considered in this paper. Next, we present two concepts, namely local network view and Operator ant, which are used in our protocol in Section III.

A. Collecting data: specifications

A data collection application can be divided into four phases namely, 1) a preparing phase, 2) a gathering phase, 3) an aggregating phase and 4) a sending phase. In this paper, we focus only on the distributed algorithm (second phase) able to gather data in the vehicular network. For more details about Phases 1, 3 and 4 the inquiring reader is referred to [18], [20], [21].

The gathering phase consists in gathering the data spread out in the vehicles to a given one, called *initiator*. It is started by the initiator vehicle and involves necessarily a limited number of vehicles around the initiator. Obviously several concurrent collects from different initiators could run simultaneously but to simplify the algorithm presentation, we consider a single collect, launched by a single initiator.

We limit the number of vehicles involved by using a `maxdst` parameter, representing the *maximal distance* in number of hops from the initiator. Indeed, each hop (vehicle-to-vehicle communication) increases the total duration of the collect as well as the number of messages in the network; `maxdst` is then an interesting parameter, impacting directly the performances. Note that we may use complex data type to limit the collect to a specific geographical area for instance (*e.g.*, identity of vehicles in the road *r*).

Note that, in a dynamic network, defining the termination of the algorithm using the distance from the initiator is not sufficient (values may change and new vehicles may appear in the area and contribute to the collect with new values).

Besides the difficulty, it is not always desirable. Indeed, for quickly meeting requests from vehicles about possible

proximity events such as traffic jams, road covered in snow or covered in ice, fog, *etc.*, the collect would also have to be restricted in terms of duration. We then introduce two parameters: `maxdur` and `maxstb`. The first one gives the *maximal duration* of the algorithm on each node. The second one allows to optimize the duration; it gives the *maximal* number of successive *stable* values produced by a node before locally ending: if a node always produces the same value, it could locally end the algorithm before `maxdur`. By the way, even if new vehicles enter into the area of collect defined by `maxdst` during the collect, they will not delay the collect because other nodes will stop to propagate these values after `maxdur` units of time.

Parameters `maxdst`, `maxdur` and `maxstb` are used all together. Obviously, some combinations are not pertinent. For instance, too short values of `maxdur` may avoid to explore the network up to `maxdst` hops, and `maxstb` should be smaller than `maxdur` to be useful. Durations are measured in multiple of `aTimer` which is a constant for the duration of a timer (see hereafter).

The type of data to be collected is given by the initiator and is included in the collect messages: parameter `typedt`. For instance, in our experiments, `typedt` is equal to `ident` to collect the id of vehicles or to `speed` to collect their speed (the first case allows to compute the density of the vehicles in a given area, while the second allows to compute the average speed in a given road). It is worth noting that, depending on their type, data can be unstable (*e.g.*, `typedt=speed`), and can change during the gathering phase. The collect algorithm should then include a *conflict operator* denoted $\odot$ to deal with the case where a vehicle receives two different data (or more) coming from a single vehicle. This operator depends on the application; it may for instance return the latest of the values in conflict or the smallest one, *etc.*

B. Local view

At a given time t, a *vehicular network* can be viewed as a directed graph in which each vehicle is viewed as a vertex and such that there exists a directed edge from u to v if, and only if, v can send a message to u at time instant t. In the remainder, $\mathscr{G}(t)$ will indicate the graph at Time t (or $\mathscr{G}$ when no ambiguity arises). As the vehicular network is dynamic, it is modeled by a sequence of graphs $\mathscr{G}_1$, $\mathscr{G}_2,\ldots$ where two consecutive graphs are not necessarily different. A *dynamic path* in a sequence of graphs $\mathscr{G}_1$, $\mathscr{G}_2,\ldots$ is a sequence of directed edges (u_0,u_1), $(u_1,u_2),\ldots,(u_{n-1},u_n)$ such that (u_0,u_1) belongs to $\mathscr{G}_1$, (u_1,u_2) belongs to $\mathscr{G}_2$, ..., (u_{n-1},u_n) belongs to $\mathscr{G}_n$. The length of this dynamic path is n. The *distance* from u to v in a graph $\mathscr{G}$, denoted by $dist(u,v)$, is the length of the shortest path from u to v. The *dynamic distance* from u to v at step k in a sequence of graphs $\mathscr{G}_1$, $\mathscr{G}_2,\ldots$, denoted by $ddist_k(u,v)$, is the length of the shortest dynamic path from u to v among those having the last edge in $\mathscr{G}_k$. Let us denote by x_v the piece of data in vehicle v which has to be collected. Let consider a graph $\mathscr{G}$. We call *local view of depth p* of vehicle v the list $\mathscr{V}_v=(N_0,N_1,\ldots,N_p)$ such that, for all $j\in\{1,\ldots,p\}$, N_j is the set of couples (u,x_u) satisfying $dist(u,v)=j$ and $N_0=\{(v,x_v)\}$. Similarly, in a sequence of graphs $\mathscr{G}_1$, $\mathscr{G}_2,\ldots$, we call *dynamic local view of depth p at step k* of vehicle v the list $(N_0,N_1,\ldots,N_p)$ such that, for all $j\in\{1,\ldots,p\}$, N_j is the set of couples (u,x_u) satisfying $ddist_k(u,v)=j$ and $N_0=\{(v,x_v)\}$.

C. Operator Ant

In our protocol, each vehicle periodically updates its local view with respect to the local view of its neighbors, by using Operator ant [22], [23]. This operator belongs to the family of *r*-operator [24], [23]. When used for distributed computation on networks, these operators have interesting properties for fault tolerance, providing they fulfill some requirements. By modeling a local algorithm with such an operator, global properties of the distributed algorithm operating on the whole distributed system can be stated by simply checking the algebraic properties of the operator.

Operator Ant has already be defined in previous work [22], [24], [23]. We give here the intuition behind its construction in order to explain its use for computing local views.

Let $\mathbb{S}$ be the set of well formed local views (views with empty sets N_i or with repetitive couples (x,x_v) are discarded by the nodes at the reception). We consider the operator $\oplus$ on $\mathbb{S}$ that merges two views while deleting needless or repetitive information so that a vehicle appears only once in a local view. To do so, for each vehicle v, we keep only the couple (v,x_v) having the lowest level *i.e.*, the leftmost. In case of conflict between two couples (v,x_v) and (v,x'_v) of same level, the ambiguity is resolved by using a conflict operator $\odot$, as explained in Section II-A: only the couple $(v,x_v\odot x'_v)$ is kept. Providing that the binary operator $\odot$ is associative ($a\odot(b\odot c)=(a\odot b)\odot c$), commutative ($a\odot b=b\odot a$) and idempotent ($a\odot a=a$), we can show that operator $\oplus$ is associative, commutative and idempotent on $\mathbb{S}$ (note that this is for instance the case when $\odot$ gives the latest or the smallest value produced by a vehicle in case of unstable data).

Since it is associative, commutative and idempotent, operator $\oplus$ defines an order relation $\preceq_\oplus$ on $\mathbb{S}$ by: $\mathscr{V}_1\preceq_\oplus\mathscr{V}_2\equiv\mathscr{V}_1\oplus\mathscr{V}_2=\mathscr{V}_1$. By the way, when using such operator, vehicles compute the smallest view of all those they received, preferring then small paths from ancestors instead of longer.

Nevertheless, each time a view is sent to a neighbor, its sets of couples have to be shift to the right because distances increase by one. This is done by an endomorphism r of $\mathbb{S}$, that inserts an empty set at the beginning of the view: $r(\mathscr{V})=(\emptyset,N_0,N_1,\ldots,N_p)$ where $\mathscr{V}=(N_0,N_1,\ldots,N_p)$.

Hence, any vehicle v periodically updates its local view $\mathscr{V}_v$ by computing the smallest view among $\mathscr{V}_v$ and $r(\mathscr{V}_u)$ for any view $\mathscr{V}_u$ sent by a neighbor u since the last local computation. The result operator is named *ant*; it is defined by: $ant(\mathscr{V}_v,\mathscr{V}_u)=\mathscr{V}_v\oplus r(\mathscr{V}_u)$. We can show that it is a strictly idempotent r-operator inducing a partial order relation on $\mathbb{S}$ and the resulting distributed algorithm supports transient faults [22], [24]. Note that, to keep views of at most depth p, it is sufficient to truncate them just after the *ant* computation.

III. COL Algorithm Design

In this section, we present our collect algorithm called *COL*, corresponding to the second phase in Section II-A.

A. Algorithm Intuition

The intuition underlying the algorithm is simple and we briefly describe it here (see Algorithm 1 for details). At a high level, as soon as it is implicated in the collect, every vehicle periodically broadcasts its local view to all its neighbors. Of course, at the beginning of the data collection, only the initiator is concerned by the collect. However, during the process of the messages propagation, the number of concerned vehicles will grow whilst complying the maximum distance criterion (`maxdst`).

In the same way, every vehicle periodically recomputes its own view by applying Operator ant to the received local views. This recurrent process allows to take into account possible new vehicles implicated in the collect as well as possible topology changes due to the dynamic of the network. In particular, obsolete local views will be rectified thanks to Operator ant and its intrinsic property of self-stabilization.

When the data collection draws to a close, the result corresponds to the local view of the initiator.

So, to achieve that, our protocol considers two aspects namely handling the dynamic vicinity and collecting data.

B. Handling the dynamic vicinity

To handle the instability of the vicinity, each time a node v receives a message from a node u, it locally grants a lifetime of `maxloss` timers to u (lines 15 and 23). In this way, if ever v does not receive another message from u at the end of `maxloss` timers, v will consider u is no longer in its vicinity. More precisely, after a timer expires, the lifetime of u will be decremented by one. When the lifetime reaches the value of zero, all the data relative to u is erased from v (lines 28-30).

Algorithm 1: **Collect protocol COL, for any node** v

```
1   Starting_action(typedt, maxdst, maxdur, maxstb):
        ▷ Starting the collect on the initiator
2       col_state ← active                ▷ A collect is now running
3       col_initiator ← v
4       col_param ← (typedt, maxdst, maxdur, maxstb)
5       local_data ← item of data of v of datatype typedt
6       local_view ← {(v, local_data)}
7       count_dur ←0                      ▷ Count until maxdur
8       count_stb ← 0                     ▷ Count until maxstb
9       tab_views ← ∅                     ▷ List of last received views
10      send( col_initiator, col_param, local_view )
11      start timer with duration aTimer

12  Upon message arrival:
13      receive( rcv_init, rcv_par, rcv_view ) from u
14      if col_state == active then
            ▷ Node has already been reached by the collect
15          tab_lifetime[u] ← maxloss
16          tab_views[u] ← rcv_view       ▷ Store the received view
17      else if col_state ≠ terminated
            ▷ Node enters into the collect
18          col_state ← active
19          col_initiator ← rcv_init
20          (typedt, maxdst, maxdur, maxstb) ← rcv_par
21          count_dur ← 0                 ▷ Count until maxdur
22          count_stb ← 0                 ▷ Count until maxstb
23          tab_lifetime[u] ← maxloss
24          tab_views[u] ← rcv_view       ▷ Store the received view
25          start timer with duration aTimer
26      end if
        ▷ Messages are ignored when the node has leaved the collect.

27  Upon timer expiration:
        ▷ Detecting neighbors disappearance
28      tab_lifetime[u] -= 1 for any u in tab_lifetime
29      for each u such that lifetime[u] == 0 do
            Delete entry u in tab_lifetime and tab_views
31      end for
        ▷ Computing the new local view
32      old_local_view ← local_view
33      local_data ← item of data of v of datatype typedt
34      local_view ← {(v, local_data)}
35      for each u such that tab_views[u] exists do
36          local_view ← ant(local_view, tab_views[u])
37      end for
38      Truncate local_view to the first maxdst elements
        ▷ Termination detection
39      count_dur += 1
40      if old_local_view and local_view are equivalent then
41          count_stb += 1
42      else
43          count_stb ← 0
44      end if
45      col_state ← terminated
46      if col_initiator ∈ local_view then
            ▷ Valid view regarding the collect
47          send( col_initiator, col_param, local_view )
48          if count_stb ≠ maxstb and
                                count_dur ≠ maxdur then
49              restart timer with duration aTimer
50              col_state ← active
51          else if col_initiator == v
                ▷ End of the collect on the initiator. Aggregating phase,
                see Section II-A)
52              Compute the final result using local_view
53          end if
54      end if
```

C. Collecting data

1) Starting the collect: Every data collection is triggered by a single node called the initiator. We consider here a single collect to simplify. Nevertheless the algorithm can easily be extended to allow several concurrent or successive collects.

As soon as the initiator decides to start a collect, it sends a message in its neighborhood made up of three fields (line 10): Identity of the initiator (col_initiator); Collect parameters (col_param) ; Its current local view (local_view).

The collect parameters are selected by the initiator and are 4 in number (see Section II-A): `typedt` (datatype to be collected), `maxdst` (maximal distance from the initiator for which the collect is desirable), `maxdur` (local maximal duration), `maxstb` (local maximal duration in case of stable view).

At the time of the first emission from the initiator, its vicinity knowledge is reduced to the empty set. In particular, its local view contains only its identity and its local data. However, it is to be expected that its local view will expand.

2) Receiving a message: At the arrival of a message (line 13), in case *v* has already been reached by the collect, the view of the sender is stored (line 16). In the converse case, we have to check whether the node has not yet been reached by the collect, or has been reached but has already terminated (line 17). If it is the first reception of a message of the collect, the parameters are stored and the variables are initialized (lines 18-22); the timer is then started (line 25).

3) Periodic computation: Node *v* computes a new local view at timer expiration. The new view of *v* depends only on views sent by neighbor nodes considered still present in its vicinity (*i.e.*, nodes for which lifetime is not null, as explained in Section III-B): lines 28-31 delete entries for nodes that did not sent a message recently.

The new view is computed using Operator ant introduced in Section II-C (line 36). The resulting new local view is truncated to the first `maxdst` elements in order to respect the distance from the initiator (line 38).

The last step consists in detecting the termination of the collect (lines 39-54). The number of computations since the beginning of the current collect is increased (line 39); the number of successive computations with the same result is increased or reset, depending on the successive views (lines 40-44) . Then, col_state is set to terminated (line 45) and will be set to active only if the collect has to continue (line 50).

If the initiator is not in the new local view (line 46), then Node *v* is not concerned by the collect: it is too far from the initiator. In that case, Node *v* no longer participates in the data collection. In the converse case, the message is broadcast in the neighborhood (line 47). This allows to prevent excessive messages in the network, propagated by nodes that left the collection area.

If the number of computations (count_dur) has not reached `maxdur` and the number of identical successive views (count_stb) has not reached `count_stb` (line 48), the timer is restarted for a new computation (line 49). In the converse case, the node has locally terminated. If it is the Initiator, the final result is obtained. Phases 3 and 4 can begin (see Section II-A): aggregating the collected data, sending the result.

D. Sketch of proof

A self-stabilizing algorithm has the property to recover from transient faults [25]. More details about correctness can be found in [18].

Property 3.1: The COL protocol is self-stabilizing and builds a local view of the network centered on the initiator, providing it has enough time to converge.

This property is due to the fact that the aforementioned collection protocol is mainly based on Operator ant which confers to COL the property of self-stabilization. Indeed, Operator ant leads to a large range of self-stabilizing tasks, such as computing local views, in a kind of distributed systems which admit bounded communication links. As stated in [26], since wireless communications can be viewed as bounded links and seeing that topology or data changes can be viewed as transient failures, the nice property of self-stabilization can be directly extended to the present framework.

The self-stabilizing property of the COL algorithm ensures that it can support any transient failure providing it has enough time to converge. However, in order to obtain a result in bounded time, the COL algorithm includes a termination detection using both `maxdur` and `maxstb`. Nevertheless, even when the convergence has not be reached, the algorithm outputs a *dynamic* local view centered on the initiator:

Property 3.2: The COL protocol builds a dynamic local view of the network, centered on the initiator (providing there is no corruption of volatile memories).

These properties ensure that the COL protocol always returns a valid dynamic local view. Moreover it supports transient faults when the parameters are well chosen.

IV. Road and Lab Experiments

Some road and lab experiments were conducted using the Airplug Software Distribution (ASD) [17], [27]. Full details can be found in [18]. A short movie is also available on [19].

A. Proof of Concept: Road Experiments

To demonstrate the feasibility of our collect protocol, we have tested Algorithm 1 via some road experiments after implementing it into a Tcl/Tk Airplug application (see [19] for screenshot movies). For this purpose, five cars have been mobilized. Within each of them, exactly one PC (Dell mini-9 Model DP118) under Ubuntu and running the Airplug core program as well as the COL application was installed. All the PCs are equipped with an external WiFi card with USB connectors (Alfa AWUS036EH), allowing to connect an antenna on the roof of the vehicles (D-LINK ANT24-0700, 2.4 GHz, 7 dBi, omni-directional).

Our experiments were conclusive: the COL algorithm is operational in practice for collecting data issued from vehicles. Moreover we confirmed the formal validation: data are collected despite network dynamic and disconnections, to the contrary of previous known algorithms (that we also implemented for comparison purpose). See [18] for details.

B. Lab experiments

For lab tests, the mobility of the vehicles has been emulated through Airplug-emu [27] in order to study many road scenarios and parameters, with more vehicles. We varied several parameters such as the reliability of the communication links, `aTimer`, `maxloss`, *etc.*

The reliability of the communication links is measured by dividing the number of successful communications over the total number of communications. Airplug-emu allows to vary such a parameter in order to replay conditions observed on the road or to study what would happen in case of poor network conditions. Figure 1 shows that the more reliable the links are, the greater the percentage of collected data is. This experiment shows that our data collection protocol still works in presence of many messages lost due to the dynamic.

To study the impact of `maxloss` on the number of collected data, for each `maxloss` $\in \{1,\ldots,10\}$, we run

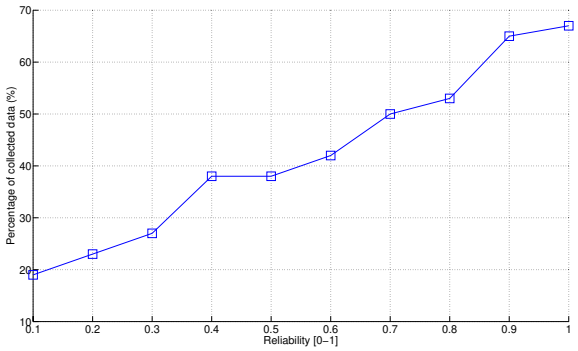

Fig. 1. Percentage of collected data as a function of links reliability

50 simulations and we recorded the average percentage of collected data. Reliability and duration are respectively fixed at 1 and 2000 ms. As expected, when `maxloss`$= 1$, the initiator collects no data except its own because in this case, according to Algorithm 1, all the received data are deleted before computation. On our scenarios with low dynamic, we observed that `maxloss`$= 2$ allowed to collect all the data. However, that is no longer the case through a less stable scenario. Indeed, we tested the impact of `maxloss` via a scenario which consists of two convoys with opposite directions that repeatedly cross each other on a circular route. Due to their opposite directions, the two convoys are repeatedly disconnected and reconnected. As a result, the greater `maxloss` is, the greater the percentage of collected data is.

To conclude all these experiments either on the road or on the emulator, i) the COL algorithm is able to collect data even in case of high network dynamic; ii) parameter `aTimer` depends mainly on the density of vehicles and a value of 1 s is convenient for almost all scenarios; iii) parameter `maxloss` depends mainly on the network dynamic and a value of $3 \times$ `aTimer` (3 s) is a reasonable choice in practice.

V. Conclusion

In this paper, we proposed a distributed embedded protocol which collects information produced by vehicles using inter-vehicle communications only. It is based on the operator `ant` allowing to construct a local view of the network and thus, to collect data in spite of the topology changes. A formal proof is available for its convergence and robustness.

Our protocol has been compared during road tests with previous known algorithms, showing our contribution: it can collect data even on dynamic networks subject to disconnection and messages losses. Complementary experiments were conducted on a network emulator [27] under several road scenarios. Parameters influence has been discussed and pertinent values have been determined for practical cases.

This protocol has been used in an experiment consisting in collecting the average speed of a convoy of vehicles on a web server, by combining it with an Internet connection discovery [21] (see movie and screenshot on-line at [19]). Future work will consist in combining our protocol with on board sensors and data fusion algorithms [20] to increase the perception of a vehicle and build new ITS applications.

References

[1] S. Capkun, M. Hamdi, and J.-P. Hubaux, "GPS-free positioning in mobile ad hoc networks," *Cluster Computing*, vol. 5, no. 2, pp. 157–167, 2002.

[2] S. Roumeliotis and I. Rekleitis, "Propagation of uncertainty in cooperative multirobot localization: Analysis and experimental results," *Autonomous Robots*, vol. 17, no. 1, pp. 41–54, July 2004.

[3] E. Karami and M. Shiva, "Maximum likelihood MIMO channel tracking," in *VTC*, 2004, pp. 876–879.

[4] N. A. M. Efatmaneshnik, A. T. Balaei and A. Dempster, "A modified multidimensional scaling with embedded particle filter algorithm for cooperative positioning of vehicular networks," in *IEEE International Conference on Vehicular Electronics and Safety*, 2009.

[5] R. Parker and S. Valaee, "Vehicular node localization using received-signal-strength indicator," in *IEEE TVT*, vol. 56, 2007, pp. 3371–3380.

[6] Z. Mo, H. Zhu, K. Makki, N. Pissinou, and M. Karimi, "On peer-to-peer location management in vehicular ad hoc networks," *International Journal of Interdisciplinary Telecommunications and Networking (IJITN)*, vol. 1, no. 2, pp. 28–45, 2009.

[7] H. Wu, R. M. Fujimoto, R. Guensler, and M. Hunter, "Mddv: a mobility-centric data dissemination algorithm for vehicular networks," in *Vehicular Ad Hoc Networks*, 2004, pp. 47–56.

[8] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode, "Trafficview: traffic data dissemination using car-to-car communication," *Mobile Computing and Communications Review*, vol. 8, no. 3, pp. 6–19, 2004.

[9] U. Lee, E. Magistretti, B. Zhou, M. Gerla, P. Bellavista, and A. Corradi, "Efficient data harvesting in mobile sensor platforms," in *PerCom Workshops*, 2006, pp. 352–356.

[10] L. Bononi and M. D. Felice, "A cross layered mac and clustering scheme for efficient broadcast in vanets," in *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*, 2007, pp. 1–8.

[11] I. Salhi, M. O. Cherif, and S.-M. Senouci, "A new architecture for data collection in vehicular networks," in *ICC*, 2009, pp. 1–6.

[12] U. Lee and M. Gerla, "A survey of urban vehicular sensing platforms," *Computer Networks*, vol. 54, no. 4, pp. 527–544, 2010.

[13] A. Segall, "Distributed network protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 1, pp. 23–34, 1983.

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

[15] S.-H. Chen and T.-L. Huang, "A wave algorithm for mobile ad hoc networks," in *Workshop on Algorithms and Computational Molecular Biology co-located with ICS*, 2002.

[16] S. Finn, "Resynch procedures and a fail-safe network protocol," *IEEE Transactions on Communications*, vol. 27, no. 6, pp. 840–845, 1979.

[17] B. Ducourthial and S. Khalfallah, "A platform for road experiments," in *VTC Spring*, 2009.

[18] Y. Dieudonné, B. Ducourthial, and S.-M. Senouci, "Design and experimentation of a self-stabilizing data collection protocol for vehicular ad-hoc networks, extended version," Lab. Heudiasyc UMR CNRS UTC 6599, Université de Technologie de Compiègne, Tech. Rep., 2010.

[19] Airplug website. [Online]. Available: https://www.hds.utc.fr/airplug

[20] V. Cherfaoui, T. Denoeux, and Z. Cherfi, "Distributed data fusion: application to confidence management in vehicular networks," in *Proc. of FUSION 2008*, Cologne, Germany, 2008.

[21] B. Ducourthial and F. Elali, "A light architecture for opportunistic vehicle-to-infrastructure communications," in *ACM International Symposium on Mobility Management and Wireless Access*, 2010.

[22] B. Ducourthial and S. Tixeuil, "Self-stabilization with r-operators," *Distributed Computing*, vol. 14, no. 3, pp. 147–162, 2001.

[23] B. Ducourthial, S. Khalfallah, and F. Petit, "Best-effort group service in dynamic networks," in *SPAA*, 2010, pp. 233–242.

[24] B. Ducourthial, "r-semi-groups: A generic approach for designing stabilizing silent tasks," in *9th Stabilization, Safety, and Security of Distributed Systems (SSS'2007)*, 2007, pp. 281–295.

[25] S. Dolev, *Self-Stabilization*. MIT Press, 2000.

[26] S. Delaët, B. Ducourthial, and S. Tixeuil, "Self-stabilization with r-operators revisited." in *Journal of Aerospace Computing, Information, and Communication*, 2006.

[27] A. Buisset, B. Ducourthial, F. E. Ali, and S. Khalfallah, "Vehicular networks emulation," in *19th International Conference on Computer Communication Networks (ICCCN)*, 2010.