Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation

Recognizing Traffic Jams with Hovering Data Clouds

Sándor P. Fekete*, Christiane Schmidt*, Axel Wegener†, and Stefan Fischer†
*Institute for Mathematical Optimization, Braunschweig University of Technology,
Email:{c.schmidt,s.fekete}@tu-bs.de,
http://www.math.tu-bs.de/mo
†Institute of Telematics, University of Lübeck,
Email:{wegener,fischer}@itm.uni-luebeck.de,
http://www.itm.uni-luebeck.de

***Abstract*— Many complex structures in our modern world exist independent of the individual entities they are composed of, giving them an "organic" quality. Important examples include traffic phenomena, e.g., traffic jams; despite of strong efforts over many years, centralized computing has been unable to deal with the resulting problems in a satisfactory manner. With the growing power of sensing devices and wireless communication, participants in traffic are no longer restricted to display passive, particle-like behavior; instead, local data exchange makes it technically feasible to aim for decentralized coordination between cars.**

One fundamental concept for making use of these possibilities comprises Hovering Data Clouds, which consist of relevant information that is kept by ever-changing carriers; a prototypical scenario arises in a traffic jam, where data is maintained by passing it on to newly arriving cars. In this study, we present algorithmic methods for this concept.

This paper is part of project AutoNomos[1] (www.autonomos.de), which aims at traffic control in a decentralized manner.

I. Introduction

A. Complex Systems

A standard scientific method to *understanding* complicated situations is to analyze them in a top-down, hierarchical manner. This approach also works well for *organizing* a large variety of structures; that is why a similar hierarchical, centralized approach has worked extremely well for employing computers in so many aspects of our life.

On the other hand, our world has become increasingly complex. The resulting challenges have become so demanding that it is impossible to ignore that a large variety of systems have a very different structure: the stability and effectiveness of our modern political, social and economic structures relies on the fact that they are based on decentralized, distributed and self-organizing mechanisms. (In this context, see [1] for a current non-fiction bestseller.)

Until very recently, scientific efforts for studying computing methodologies for decentralized complex systems have been very limited. A current approach is based on the paradigm of *organic computing*, which is described in the following section.

B. Organic Computing

Traditional computing systems are based on a centralized algorithmic paradigm: data is gathered, processed, and the result is administered by one central authority. Each of these aspects is subject to obstructions. On the other hand, "Living organisms (...) are to be treated as dynamic systems and contain all infrastructure necessary for their development, instead of depending on coupling to a separate thinking mind. We call this computing paradigm *Organic Computing* to emphasize both organic structure and complex, purposeful action" [2].

The importance of Organic Computing has been motivated as follows: "The advantages of self-organizing systems are obvious: They behave more like intelligent assistants, rather than a rigidly programmed slave. They are flexible, robust against (partial) failure, and are able to self-optimize. The cost of design decreases, because not every variant has to be programmed in advance" [3].

C. Traffic as a Scenario for Organic Computing

When studying a new scientific paradigm, it is tempting to use yet another top-down approach, by focusing on classifications, definitions, categories, terms, etc. Obviously, this would be somewhat paradox (and hence, self-defeating) when applied to distributed, decentralized systems. Moreover, the success of a new paradigm hinges critically on convincing objectives and actual results, in the context of scenarios for which the appropriateness of the paradigm is evident.

A natural scenario for studying organic computing for complex, decentralized systems is traffic, which combines a number of important features: traffic and transportation affect the daily lives of billions of people; they are of enormous economic, social and political importance; they involve state-of-the-art technology, so that the use of computing devices and methods are possible and required; and they provide a good mixture of sub-scenarios of varying complexity, all of which display very clear self-organizing and decentralized features.

D. Project AutoNomos and Hovering Data Clouds

The key point of our project AutoNomos is a decentralized method for traffic analysis and control, based on a bottom-up, multilevel approach. Beyond the motivations described above, it should be stressed that an aspect of particular relevance is *scalability:* while the computational effort for a centralized approach increases prohibitively with the number of verhicles, a decentralized method relies on neighborhood interaction of constant size.

At this point, we focus on the key concept of *Hovering Data Clouds* (HDCs), which are virtual structures that exists independent of particular carriers. Hovering Data Clouds are somewhat related to Virtual Autonomous Mobile Nodes (AVMN), as introduced by Dolev et al. [4], [5], [6]; such an AVMN is "an automaton that can make autonomous on-line decisions concerning its own movement". Note that the model of an AVMN assumes that the current position is described by a known external function; and while an HDCs corresponds to a distributed structure that is carried by collective of individual processors, considerable effort is spent on making sure that at

[1]Supported by DFG Focus Program "Organic Computing" (SPP 1183), grant numbers Fe 407/11-1, Fi 605/9-1.

978-0-7695-3071-0/07 $25.00 © 2007 IEEE
DOI 10.1109/ISoLA.2006.30

any given time there is precisely one AVMN. Furthermore, the concept of AVMNs focuses on the automaton-like properties, which is realized by making sure that "each participating processor keeps a replica of the AVMN's current state and a buffer of input events waiting to be applied to the state." This necessitates a continuing process of updates to ensure that global properties of the AVMN are maintained.

The main point of this paper is to present the general concepts of our project, and demonstrate their concrete usefulness for a first, relatively basic scenario: understanding and controlling traffic jams for a single-lane highway. We show how Hovering Data Clouds can be used to model critical features like the front and back of a traffic jam, describe how they can be formed and maintained in a self-organizing manner, and discuss further extensions. It should be noted that this involves aspects of decentralized data structures and communication in mobile ad-hoc networks, which are interesting in their own right.

The rest of this paper is organized as follows. Section 2 presents aspects of self-organization in traffic and communication networks. This is followed by a detailed description of Hovering Data Clouds and how they can be maintained in the context of a basic traffic jam (Section 3). Finally, Section 4 gives a summary and outlook to future perspectives.

II. Traffic, Self-Organization, and Communication

Traffic is one of the most important complex systems of our modern world, with several levels of complexity reaching from individual actions of a driver, over local phenomena like density fluctuations and traffic jams, regional and temporal traffic patterns, all the way up to long-range traffic development and regulation.

In recent years, tremendous progress has been made in understanding the dynamics of traffic flow and traffic congestion; however, a number of serious obstacles still prevent efficient coordination and regulation of traffic in large-scale networks. Three of the most serious impediments have been the incompleteness of input data, the computational intractability of forecasting the behavior of real-life traffic consisting of huge numbers of vehicles, and the lack of local communication (and thus: cooperation) between drivers.

With the advances of modern communication technologies, it has become possible to keep track of virtually all data of driving vehicles. Understanding traffic as a complex system that is based on local interaction suggests studying distributed computing approaches for simulating and forecasting traffic phenomena. Finally, wireless ad-hoc networks allow real-time interaction and data exchange between adjacent vehicles.

A. Traffic as a Self-Organizing Organic System

The structure of traffic is a phenomenon that is self-organizing at several levels; see [7] for a philosophical discussion of self-organization in multi-level models. But even though the behavior of and the interaction between motorists has been observed for a long time, the possibilities arising from modern technology allowing direct and decentralized complex interaction between vehicles has hardly been studied. The only efforts we are aware of combine game theory with traffic simulations. (For example, see the symposium organized by traffic physicist Schreckenberg with game-theory Nobel laureate Selten [8].) Neither make use of mobile ad-hoc networks and distributed algorithms in large networks.

To the best of our knowledge, the idea of combining the above aspects, i.e., self-organizing traffic by combining ad-hoc networks with distributed decentralized algorithms has not been studied so far. This may be because it requires combining a number of different aspects, each of which has only been developed by itself in recent years. Mobile ad-hoc networks, models for large systems of self-driven multi-particle systems, as well as algorithmic aspects of decentralized distributed computing, possibly with elements of game-theoretic interaction.

B. Computing Methodologies in Traffic and Telematics

As the interest in guiding and organizing traffic has been growing over recent years, the scientific interest in traffic as a research topic has developed quite dramatically. For an overview ("Traffic and related self-driven many-particle systems"), see the excellent survey [9]. Obviously, research on traffic as a whole is an area far too wide for a brief description in this short overview; we focus on a particular strain of research that is particularly relevant for our proposed work, as it appears to be most suited for simulation and extension to decentralized, self-organizing systems of many vehicles.

It is remarkable that until the early 90s, efforts for simulating traffic were based on complex multi-parameter models of individual vehicles, resulting in complex systems of differential equations, with the hope of extending those into simulations for traffic as a whole. Obvious deficiencies of this kind of approach are manifold:

1) Because the behavior of even just an individual vehicle is guided by all sort of factors influencing a driver, the hope for a closed and full description appears hopeless.
2) Determining the necessary data for setting up a simulation for a relevant scenario is virtually impossible.
3) Running such a simulation quickly hits a wall; even with today's computing power, simulating a traffic jam with a few thousand individual vehicles based on such a model is far beyond reach.

A breakthrough was reached when physicists started to use a different kind of approach: instead of modeling vehicles with ever-increasing numbers of hidden parameters, try to consider them as systems of many particles, each governed by a very basic set of rules. As Nagel and Schreckenberg managed to show [10], even a simple model based on cellular automata can produce fractal-like structures of spontaneous traffic jams, i.e., complex, self-organizing phenomena. Over the years, these models [11] were generalized to two-lane highway traffic [12], extended for simulating commuter traffic in a large city [13], and have grown [14], [15] to the point of being used for real-time traffic forecasts for the 2250 km of public highways in the state of North Rhine-Westphalia [16], [17], [18] (see `http://www.autobahn.nrw.de/`.) Also, see the book chapter by Nagel [19].

Parallel to the scientific developments described above, the interest in and the methods for obtaining accurate traffic data has continued to grow. The employment of induction loops and traffic cameras has been around for quite a while, but despite of enormous investments, e.g., 200 Mio. Euros by the German Federal Ministry for Traffic, Construction and Housing (BMVGW) [20] for putting up systems for influencing traffic, the limits on tracking individual vehicles, as well as following particular traffic substructures are obvious. A more recent development is the use of *floating car data:* By keeping track

of the movements of a suitable subset of vehicles (e.g., taxis in Berlin city traffic), the hope is to get a more accurate overall image of traffic situations, both in time and space [21]. However, even this approach relies on the use of the central processor paradigm, and does not allow the use of ad-hoc networks for the active and direct interaction and coordination between vehicles.

C. Self-Organization in Communication Networks

In today's communication world, wireless networks such as GSM in the wide area and WLAN in the local area range have become ubiquitous. Still, most applications using these networks rely on a thoroughly managed infrastructure such as, for instance, base stations in GSM or access points in WLAN. Many research activities, however, already go one step further and make use of the fact that more and more mobile devices with radio communication capabilities are available. These devices are not necessarily bound to communication infrastructures, but can instead create a spontaneous network, a so-called ad-hoc network, in order to allow communication among all participating processor (and not necessarily to the outside). In its sophisticated form, some of the processor in an ad-hoc network act as relay stations and transport data from a source to a destination that is not in direct radio range (multihop ad-hoc network).

III. Hovering Data Clouds for Traffic Jams

A. Hovering Data Clouds

A dynamically changing system such as a traffic jam consists of many ever-changing objects, as cars located at different positions keep moving with respect to back and front of the queue. If we want to maintain useful information related to the back of a traffic jam, we have to keep shifting the related roles from one car to the next, together with all relevant data. Thus, we look for a *local* data structure with the following properties:

- The structure self-organizes with the onset of a traffic jam, and it ceases to exist when the jam disappears.
- It is located at a useful virtual location, which is defined by the traffic jam, e.g., its back.
- The structure continues to exist, even as their current carriers move or change their role.
- It contains up-to-date information that describes the traffic jam.

We call such a structure a *Hovering Data Cloud* (HDC). In this section, we describe how to deal with the above properties in the context of a relatively simple traffic scenario.

B. Scenario

We consider a single-lane highway on which cars move from right to left (positions on the right of the road having lower values on a coordinate axis than locations on the left). Any vehicle carries a computing and wireless communication device, each with a unique identifier, and has a reliable way of measuring time and location, e.g., by using GPS. Cars can communicate if they are within broadcast range of each other; this range is denoted by R. Communication delays are relatively small, we assume 10ms for a single communication.

Now we examine traffic patterns. Depending on speed and traffic density, a traffic jam may form. This gives rise to two HDCs, one each at the front and back location of the queue, cp. Figure 2; these HDCs are maintained while the jam continues to exist. In the following we describe the details of how this can be achieved; note that the same basic variables and processes are maintained in each processor.

C. Variables

We will make use of the following variables and parameters for each processor.

$status_i$ (with possible values *idle*, $joining_i$, $active_i$) describes the current state of a processor, where the index $i = 1, 2$ refers to the HDCs marking back (1) and front (2) of the traffic jam; initially, all processors are *idle*. In the absence of a traffic jam, *idle* processors in the range of the HDC become *active* immediately. If a jam exists, processors become *joining*, if they lie within r_{HDC} of the current HDC position; an *active* processor becomes *idle*, if it ceases to be near the boundary of the jam, i.e., if its distance to HDC position exceeds r_{HDC}. Note that the status *joining* is only necessary in the presence of information that is not known to all processors. *state* describes the values of all variables stored on a processor.

$location_1$ and $location_2$ describe the current positions of the HDCs marking back (1) and front (2) of the traffic jam. For clarity we should mention that the variables *loc, v* and ID refer to the actual processor, *l, g* and *ident* refer to the processor from which the actual message m was received. *buffer* is a memory location for storing arriving messages. *clock* is used for keeping track of real time. *env* is a matrix for storing the data of all broadcasting processors within distance R (maximum size: $3 \times \lceil 2R/(\text{minimum distance}) \rceil$).

If a processor has a right neighbor (i.e., a following car) within congestion radius CR, the flag p is set to 1. Analogously, $q = 1$ indicates a left neighbor (i.e., a preceding car) within congestion radius CR.

The auxiiliary variables *ahead, congestion_counter, congestion_participant, congestion_participant_before, s, q_before, p_before,* P_w, Q_w, p_w, q_w, *t, front, back_id, front_id* are all intitialized with 0; *back* is initilaized with the largest possible value of a location on the road.

If status *joining* is necessary, then v_{0_1}, v_{0_2} are the initial states of the HDCs.

t_{data}, t_{smdata}, $t_{information}$, $t_{aheadInfo}$ are time slots that are used for updating the indexed variables. t_{data}, t_{smdata} occur alternately. $t_{information}$, $t_{aheadInfo}$ describe times with evenly spaced intervals. The function settimer used with the argument next-multiple(t_x) ($x \in \{information, aheadInfo\}$) sets the timer on the next time of t_x (even if some time elapsed since the last timer was set), cf. Figure1. t_{ab} is the time period that passes until all processors within $2R$ have started their *data* interval.

A parameter that is related to the presence of a traffic jam is the *congestion radius* CR that is a function of the current speed of vehicles; if two cars violate this critical distance, it may be an indication of a traffic jam.

Conversely, the *congestion velocity* CV, considers the velocity in relation to the current distance; if two cars move slower than apropriate for their current distance, this may also indicate a traffic jam. (Note that this second parameter filters out cars that tailgate at high speed. As pointed out by [22], it works best to consider speed for recognizing traffic jams.)

Finally, d is the critical bound on the latency in *LBcast*.

D. Algorithm Description

We give details of the algorithmic steps; see Algorithm 1.

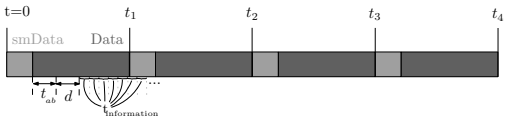

Fig. 1: Example for time slots and timer.

We consider a discrete sequence of time slots, $t_i, t_{i+1}, \ldots$. The interval between to consecutive time slots is divided into two subintervals: a small interval (*smData*), and a bigger one (*Data*), cf. Figure 1. Thus, in the end of *smData (onTimer(smData))* the timer for *Data* is initiated and vice versa.

In *onTimer(smData)* the current processor position and its velocity is only broadcast within CR. If a processor receives such a message, it is treated in *onTimer(NewMessage)* (as all received messages)—after an additional delay of d (see *LBrecv(m)*). In case a processor receives such a message from ahead, we set p equal to 1. Analogously, a message from behind results in setting q equal to 1.

For sending more data (*position, velocity*, p, q) in a wider range in *onTimer(Data)* we distinguish several situations. Only if the processor is participating in a HDC or if it is neither caught up in a traffic jam (t_i) nor was so before (t_{i-1}) but falling below a certain velocity, it will send such a message. This enables us to reduce the amount of transmitted messages. However, non-active processors inside of a traffic jam or with sufficient high speed do not send, the information is either not important yet or the processor is not a potential participant of a traffic jam.

Describing the consequences of being a congestion participant, we need to consider how a processor achieves this situation. A processor receives *data* messages from its surrounding processors within R. The data of each such processor is stored in *env*. Afterwards, it is checked whether the sending processor is located close (less than CR) to the receiver and if the velocity is sufficient low, e.g., clearly below 60 km/h (see [22].) If so, the back is computed from the position of the two processors and the previous back. Furthermore, the processor becomes a participant of the traffic jam. Similarly we check for all processor in *env* whether they are located close to the sending processor and fall below a velocity of CV. In both cases we increment a counter for the congestion (indicating a positive number of vehicles).

If the counter was incremented, the processor lies close (less than r_{HDC}) to the back, the back has no right neighbor (thus, it is really the back) and all messages from processor within the range of R were received, then *Congestion* is invoked. The front of the jam is treated analogously; *CongestionAhead* is invoked here. If the HDC at the back has yet to receive information from the HDC at the front, then the position of *front* is set to the position of the most advanced processor within R.

The status of a processor is maintained in *Congestion*: if it is *active*, we set a timer for *Information*, i.e., as long as the processor is *active*, the position of the HDC at the back of the jam, the position of the HDC at the front, and the current speed of the back are broadcast. Only if the position of the back HDC has a value greater than the current processor location, the processor continues to broadcast, and processors approaching this position become *joining*.

In *CongestionAhead*, $status_2$ is updated; if the processor is *active*, the timer for *AheadInfo* is set. This means that the position of the front HDC is broadcast regularly, as long as the processor is *active*. When such a message *hdcdistance* is processed, the back HDC variable *front* indicating the position of the front HDC is updated for an *active* processor: inactive processors between front and back HDC pass on the message towards the back.

With increasing distance, clustering and updating data is performed by the HDC transportation layer. However, this is not the focus of this paper.

Thus, messages are broadcast to:

- relate the positions of the processors (messages broadcast in onTimer(smData), onTimer(Data), processed in onTimer(NewMessage)),
- transmit the information of the HDC at the front of the traffic jam to the one at the back (messages in onTimer(AheadInfo)),
- transmit the information of the back HDC to following cars (messages in onTimer(Information)).

IV. Summary and Conclusions

We have described a new concept for dealing with challenges in Organic Computing that arise in the context of mobile self-organizing structures. These Hovering Data Clouds are particularly natural and relevant in the context of traffic, more specifically, for traffic jams.

There are various extensions and generalizations. An obvious next step is to extend our ideas to two-lane highways (possibly stretching over several exits and even highway crossings), or more refined HDCs that reflect substructures in a taffic jam; other extensions and variants include the recognition of bottlenecks in traffic, e.g., caused by a convoy of slow trucks, accidents or emergency vehicles.

A qualitatively more challenging step is required when considering more advanced structures that consist of several HDCs: an HDC simply marks front or back of a traffic jam, but eventually we are interested in more complex interaction between all involved vehicles, e.g., when trying to smoothen out complex stop-and-go patterns, mark advisable exits for following cars, or even map possible detours. We call such high-level structures *Organic Information Complexes* (OICs). They will be pursued and discussed in future work.

Acknowledgment

We thank Elad Schiller for helpful discussions of Autonomous Mobile Virtual Nodes.

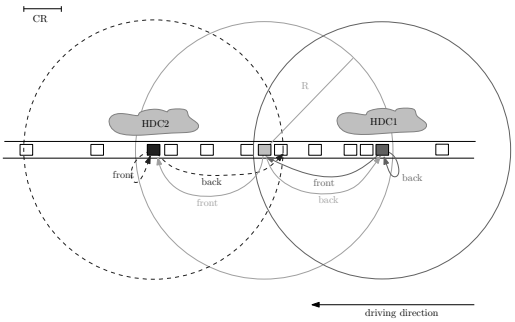

Fig. 2: Example for the two HDCs: different cars (rectangles) determine front and back (assumption: all drive slow enough), HDCs arise if all further conditions are met.

Algorithm 1: HDCs for traffic jams

initialization:
i=0; *etc.* ;
buffer = $\emptyset$;
settimer(clock, smData);

onTimer(smData)
q_before = Q_w;
p_before = p_w;
back = ∞;
front = 0;
$P_w = 0$;
$Q_w = 0$;
$p_w = 0$;
$q_w = 0$;
$p = 0$;
$q = 0$;
LBcast-CR($\langle$*smdata*, ID, *loc, v*$\rangle$); `//broadcast in CR`
settimer(next-multiple(t_{data}), Data); `//as Data follows smData`
s = 0;

onTimer(Data)
t = *clock*;
congestion_counter = 0;
if *(congestion_participant = 1)* $\vee$ *((p = 1)* $\wedge$ *(congestion_participant_before = 1))* $\vee$ *((q = 1)* $\wedge$ *(congestion_participant_before = 1))* **then**
  congestion_participant_before = 1;
else
  congestion_participant = 0;
  p-before = 0;
  q-before = 0;
if *($status_1$ = $active_1$)* $\wedge$ *(congestion_participant = 0)* **then**
  $status_1$ = idle;
  ahead = 0;
if *($status_2$ = $active_2$)* $\wedge$ *(congestion_participant = 0)* **then**
  $status_2$ = idle;

if *(congestion_participant = 0* $\wedge$ $v \leq CVmax$*)* $\vee$ *($status_1$ = $active_1$)* $\vee$ *($status_2$ = $active_2$)* **then**
  LBcast($\langle$*data*, ID, $loc, v, p, q$$\rangle$); `//broadcast in R`
  settimer(next-multiple(t_{smdata}), smData); `//as smData follows Data`
  congestion_participant_before = congestion_participant;
  congestion_participant = 0;
else if *(congestion_participant = 1)* $\vee$ *(congestion_participant = 0* $\wedge$ $v > CVmax$*)* **then**
  congestion_participant_before = congestion_participant;
  congestion_participant = 0;
  settimer(next-multiple(t_{smdata}), smData); `//as smData follows Data`
$i = 0$;

LBrecv(m)
buffer = buffer $\cup$ $\langle$ *m, clock*$\rangle$;
settimer(*clock* + *d*, NewMessage);

Congestion
if $status_1 = active_1$ **then**
  if *($location_1$ < back)* $\vee$ *($location_1$ > back)* **then**
    $location_1$ = back;
  settimer(next-multiple($t_{information}$), Information);
if $status_1 = joining_1$ **then**
  $status_1 = active_1$;
  settimer(next-multiple($t_{information}$), Information);
  $location_1$ = back;
if $status_1 = idle$ **then**
  if *q_before = 0* **then**
    $location_1$ = back;
    state = v_{0_1};
    $status_1 = active_1$;
    settimer(next-multiple($t_{information}$), Information);
  else
    $status_1 = joining_1$; `//status to collect all dates`

onTimer(Information)
if $status_1 = active_1$ **then**
  LBcast($\langle$*Congestion*, $location_1$, HDC2_location, v $\rangle$);
  settimer(next-multiple($t_{information}$), Information);
  if $|loc\text{-}location_1| > r_{HDC}$ **then**
    $status_1 = idle$; `//If the distance is too big, the processor //becomes idle. Thus, on the next call of` *Information* `no //further timer is set`
    ahead = 0;

CongestionAhead
if $status_2 = active_2$ **then**
  if *($location_2$ < front)* $\vee$ *($location_2$ > front)* **then**
    $location_2$ = front;
  settimer(next-multiple($t_{aheadInfo}$), AheadInfo);
if $status_2 = joining_2$ **then**
  $status_2 = active_2$;
  settimer(next-multiple($t_{aheadInfo}$), AheadInfo);
  $location_2$ = front;
if $status_2 = idle$ **then**
  if *p_before = 0* **then**
    $location_2$ = front;
    state = v_{0_2};
    $status_2 = active_2$;
    settimer(next-multiple($t_{aheadInfo}$), AheadInfo);
  else
    $status_2 = joining_2$;

onTimer(AheadInfo)
if $status_2 = active_2$ **then**
  LBcast($\langle$*hdcdistance*, $location_2$$\rangle$);
if $|loc\text{-}location_2| > r_{HDC}$ **then**
  $status_2 = idle$;

```
onTimer(NewMessage)
    let m = min(m:⟨ m, t ⟩ ∈ buffer, t = clock - d);
    if m = ⟨data, ident, l, g, p-value, q-value⟩) then
        //checking critical values
        env_i = ⟨ident, l, g⟩;
        if (| loc - l | < CR) ∧ (v < CV) then
            back = min{back, loc, l};
            if back==loc then back_id = ID;
            if back==l then back_id = env_i.ident;
            P_w = p-value of back_id; Q_w = q-value of
            back_id;
            front = max{front, loc, l };
            if front==loc then front_id = ID;
            if front==l then front_id = env_i.ident;
            p_w = p-value of front_id; q_w = q-value of
            front_id;
            congestion_counter++;
            s = 1;
            congestion_participant = 1;
        //comparison: transmitting processor -
        already received messages
        for j, 1, i − 1 do
            if (| l - env_j.l | < CR) ∧ (g < CV) then
                back = min{back, l, env_j.l};
                if back==env_j.l then back_id = env_l.ident;
                if back==l then back_id = env_i.ident;
                P_w = p-value of back_id; Q_w = q-value of
                back_id;
                front = max{front, l, env_j.l };
                if front==env_j.l then front_id =
                env_j.ident;
                if front==l then front_id = env_i.ident;
                p_w = p-value of front_id; q_w = q-value of
                front_id;
                if s = 0 then
                    congestion_counter++;
                    s = 1;

    if (congestion_counter > 0) ∧ (|back - loc| < r_HDC) ∧
    (P_w = 0)
        ∧ (clock ≥t + t_ab + d) then
        invoke Congestion;
        congestion_participant = 1;
    if (congestion_counter > 0) ∧ (|front - loc| < r_HDC) ∧
    (q_w = 0)
        ∧ (clock ≥ t + t_ab + d) then
        invoke CongestionAhead;
        congestion_participant = 1;
    if ahead = 0 then HDC2_location = front;
    i++;
    if m = ⟨Congestion, l, location_hdc_front, g⟩ then
        if l > loc then LBcast(⟨Congestion, l,
        location_hdc_front, g⟩);
        if |l - loc| < r_HDC then status_1 = joining_1;
    if m = ⟨hdcdistance, L_2⟩ then
        if status_1 = active_1 then
            HDC2_location = L_2; ahead = 1;
        else if congestion_participant = 1 then
            LBcast(⟨hdcdistance, L_2⟩);
    if m = ⟨smData, ident, l, g⟩ then
        if l < loc then p = 1; //right neighbor
        if l > loc then q = 1; //left neighbor
```

References

[1] J. Surowiecki, *The wisdom of crowds*. London: Doubleday, 2004.

[2] O. C. Initiative, "A novel computing paradigm," http://www.organic-computing.org.

[3] C. Müller-Schloer, H. Schmeck, and T. Ungerer, "Organic Computing – Proposal for a Focus Program," 2004.

[4] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, and J. L. Welch, "Virtual mobile nodes for mobile ad hoc networks." in *DISC*, 2004, pp. 230–244.

[5] S. Dolev, S. Gilbert, E. Schiller, A. Shvartsman, and J. Welch, "Autonomous virtual mobile nodes," in *3rd Workshop on Foundations of Mobile Computing (DIAL-M-POMC)*, 2005.

[6] Dolev, Lahiani, Gilbert, Lynch, and Nolte, "Virtual stationary automata for mobile networks (short)," in *PODC: 24th ACM SIGACT-SIGOPS Symp. Princ. Dist. Comput.*, 2005.

[7] C. Gershenson and F. Heylighen, "When can we call a system self-organizing?" in *Advances in Artificial Life, 7th European Conference, ECAL 2003, Dortmund, Germany, September 14-17, 2003, Proceedings*, ser. Lecture Notes in Computer Science, W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, Eds., vol. 2801, 2002, pp. 606–614.

[8] M. Schreckenberg and R. Selten, *Human behavior and traffic networks*. Berlin: Springer-Verlag, 2004.

[9] D. Helbing, "Traffic and related self-driven many-particle systems," *Reviews of Modern Physics*, vol. 73, pp. 1067–1141, 2001.

[10] K. Nagel and M. Schreckenberg, "A cellular automation model for freeway traffic," *Journal de Physique I France*, vol. 2, pp. 2221–2229, 2001.

[11] K. Nagel, "High-speed simulation of traffic flow," Ph.D. dissertation, Center for Parallel Computing, 1995.

[12] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour, "Two-lane traffic simulation on cellular automata," *Physica A*, vol. 231, pp. 534–550, 1996.

[13] M. Rickert and K. Nagel, "Experiences with a simplified microsimulation for the Dallas/Fort Worth area," *Int. J. Mod. Phys. C*, vol. 8, pp. 133–153, 1997.

[14] K. Nagel, M. Rickert, and C. L. Barrett, "Large-scale traffic simulations," in *Proc. 2nd International Conference on Vector and Parallel Processing*, ser. Lecture Notes in Computer Science, vol. 1215, 1997, pp. 3800–402.

[15] K. Nagel, "Cellular automata models for transportation applications," in *Proc. 5th International Conference on Cellular Automata for Research and Industry*, ser. Lecture Notes in Computer Science, vol. 2493, 2002, pp. 20–31.

[16] O. Kaufmann, K. Froese, R. Chrobok, J. Wahle, L. Neubert, and M. Schreckenberg, "Online simulation of the freeway network of NRW," in *Traffic and Granular Flow '99*, D. Helbing, H. J. Herrmann, M. Schreckenberg, and D. E. Wolf, Eds. Springer-Verlag, 2000, pp. 351–356.

[17] S. F. Hafstein, R. Chrobok, A. Pottmeier, J. Wahle, and M. Schreckenberg, "Cellular automation modeling of the Autobahn traffic in North Rhine – Westphalia," in *Proceedings 4th IMCAS Symposium on Mathematical Modelling*, I. Troch and F. Breitenecker, Eds., 2003, pp. 1322–1331.

[18] A. Pottmeier, S. Hafstein, R. Chrobok, J. Wahle, and M. Schreckenberg, "The traffic state of the Autobahn network of North Rhine-Westphalia: An online traffic simulation," in *Proceedings 10th World Congress and Exhibition on Intelligent Transport Systems and Services*, 2003, document Nr. 2377.

[19] K. Nagel, "Traffic networks," in *Handbook of graphs and networks – From the genome to the internet*, ser. Lecture Notes in Computer Science, S. Bornholdt and H. G. Schuster, Eds. Berlin: Wiley-VCH, 2003, ch. 11.

[20] B.-u. W. Bundesministerium für Transport, "Programm zur Verkehrsbeeinflussung auf Bundesautobahnen 2002 bis 2007," 2002, available at http://www.bmvbw.de.

[21] B. Kwella and H. Lehmann, "Floating car data analysis of urban road networks," in *Proceedings Computer Aided Systems Theory - EUROCAST'99*, ser. Lecture Notes in Computer Science, F. Pichler, R. Moreno-Díaz, and P. Kopacek, Eds., vol. 1798. Vienna, Austria: Springer, 2000.

[22] W. Brilon, M. Regler, and J. Geistefeldt, "Zufallscharakter der Kapazität von Autobahnen und praktische Konsequenzen," *Straßenverkehrstechnik*, vol. 03 and 04, 2005.