Computer Networks 53 (2009) 1737–1750

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A rule-based system for programming self-organized sensor and actor networks

Falko Dressler [a,*], Isabel Dietrich [a], Reinhard German [a], Bettina Krüger [b]

[a] Computer Networks and Communication Systems, Department of Computer Science, University of Erlangen, Martensstrasse 3, 91058 Erlangen, Germany
[b] Department of Cellular and Molecular Physiology, University of Erlangen, Germany

ARTICLE INFO

Article history:
Available online 23 September 2008

Keywords:
Self-organization
Sensor and actor networks
Network-centric operation
Rule-based sensor network
Cellular signaling cascades

ABSTRACT

We describe a programming scheme for massively distributed systems that are assumed to self-organize according to a given set of simple rules. The focus of this investigation is operation and control in sensor and actor networks (SANETs). The main issues addressed by self-organization techniques are scalability, network lifetime, and real-time support. In the literature, biological principles are often cited as inspirations for technical solutions, especially in the domain of self-organization. We developed a system named rule-based sensor network (RSN) according to the observed communication and control behavior in cellular communication. Cellular signaling cascades allow the event-specific reaction initiated by individual cells in collaboration with their direct neighbors. Information between cells are transmitted via proteins and result in the cascade of protein–protein or protein–DNA interactions to produce a specific cellular answer, e.g. the activation of cells or the transmission of mediators. These processes are programmed in every individual cell and lead to a coordinated reaction on a higher organization platform. We transferred these mechanisms to operation and control in SANETs. In particular, a rule-based processing scheme relying on the main concepts of cellular signaling cascades has been developed. It relies on simple local rules and provides problem specific reaction such as local actuation control and data manipulation. We describe this RSN technology and demonstrate comparative simulation results that show the feasibility of our approach.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Wireless sensor networks (WSNs) have become a major research domain in the networking community over the last decade. It has been shown that classical networking techniques are often not suitable or at least insufficient in terms of communication and storage requirements. The main problems are the necessary energy efficiency and the capability to work on low-resource embedded systems. Actually, WSNs are meant to be composed of small battery-driven embedded systems that are communicating over a wireless channel [1,2].

The requirements are becoming even stronger when sensor and actor networks (SANETs) are considered. In many cases, SANETs represent networks similar to WSNs but with inherent actuation facilities. Such actuators can be a heater or a switch – both activated and driven by network-inherent sensor measures. In other cases, actuators can be mobile robot systems able to perform much more complex actuation. In contrast to typical WSNs, SANETs also face critical real-time operation requirements [3].

The coordination and control of SANETs is still an emerging research area. Usually, the applications follow the classical approach as depicted in Fig. 1 (left). Sensor nodes are continuously analyzing the environment (measurement). The measurement data are transmitted to one or more fixed systems for further processing. Then, the actuators are controlled by explicit commands that are

* Corresponding author. Tel.: +49 9131 85 27914; fax: +49 9131 85 27409.
E-mail address: dressler@informatik.uni-erlangen.de (F. Dressler).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.09.007

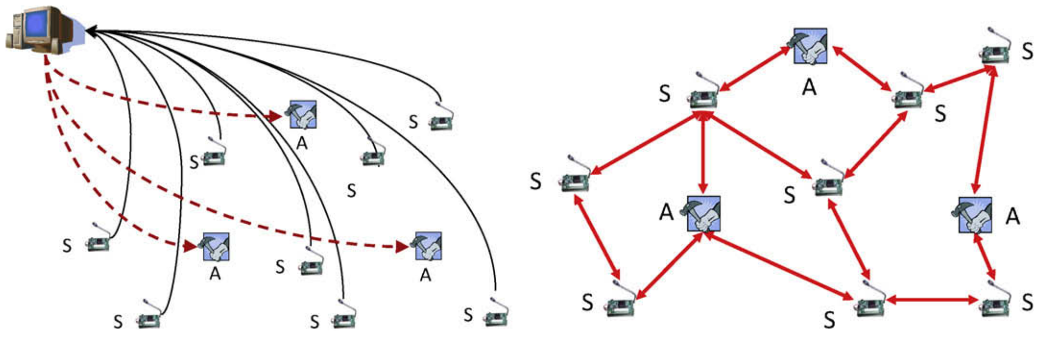


Fig. 1. Operation and control of SANETs: centralized (left), network-centric (right).

finally executed (actuation). The measurement and the control loop are shown by corresponding arrows. Obviously, long transmission distances have to be bridged leading to unnecessarily high transmission delays as well as to a questionable communication overhead in the network, i.e. possible network congestion and energy wastage.

Self-organization of the SANET is considered the final solution to build energy efficient SANETs that allow real-time operation without complex global state maintenance [4]. The favored system behavior is shown in Fig. 1 (right). Self-organization methodologies are used to provide network-centric actuation control, i.e. a processing of measurement data within the network and a direct interaction with associated, i.e. co-located actuators.

A number of approaches related to the main ideas of *autonomic networking*, i.e. the development of self-managing networks, have recently been proposed. One idea is to cluster the available sensor and actor systems into groups that enable simple coordination and control strategies. An example is the distributed coordination framework developed by Melodia et al. [5]. Another approach is to group nodes according to the main objectives of the sensor network such as a given degree of coverage. Gupta et al. [6] have shown that queries into a sensor network can be optimized based on this measure.

Higher level task allocation strategies are also related in the discussed context because actuation represents a specific class of remotely executed tasks. For example, Low et al. [7] employed autonomic networking techniques for task allocation in mobile sensor networks. The use of general self-organization techniques has often been suggested in the domain of communication networks [8]. With respect to SANETs, only few approaches have been published.

In the last few years, we studied some aspects of conceptual similar techniques that have been studied in the domain of cellular biology. These investigations lead to completely different communication and control paradigms in an area that is widely known as *bio-inspired networking*. A great number of solutions are thinkable based on bio-inspired approaches [9].

In this paper, we present a system that we named rule-based sensor network (RSN) [10]. It follows the concept of network-centric operation and control [11] based on adapted mechanisms as known from cell biology. The result is an architecture for data-centric message forwarding, aggregation, and processing. We evaluated the performance of this system using a comprehensive simulation model. According to the simulation results, RSN outperforms classical ad hoc routing techniques by far – in a typical SANET scenario.

RSN was inspired by early rule-based systems that have been developed in the context of active networking solutions [12]. This domain was especially driven by policy based programmable networks [13]. The main idea was to transport data along with code that allows intermediate nodes to process this data. Today, such policy systems are frequently used to describe services on higher layers, e.g. for network management issues [14]. In comparison, RSN relies on data transmission only and requires pre-installed rule sets. The formatting of the messages is – as for any other data-centric networking approach – redefined by the networking service. Other examples for rule systems used in the active networking approach are the mobile object system [15] and communicating rules [16].

Another class of rule-based programming approaches is the control of distributed observation processes. TinyDB [17] was one of the first approaches to access the sensor network in a similar way as accessing a database. For this purpose, SQL-like statements have been defined that periodically trigger the monitoring of sensor data. A rule-based approach for such event triggers was proposed in [18] and in [19]. Both solutions allow more complex statements (rules) to define the selective triggering of performing, preprocessing and transmitting sensor readings. In comparison, RSN is dedicated to generic processing of received data. It can not only be used to generate data according to a given set of rules but also to process received data, to perform routing decisions, and to initiate local actuation devices. A major aspect of the rule-based sensor network system is that is operates on sets of messages instead of applying rules to single data packets.

In general event-based coordination systems tend to face three core problems of scale, stability, and extensibility. RSN has been developed with these problem domains in mind. So, RSN is explicitly designed for supporting large scale networks as well as complex rule sets. Stability is a key problem in all self-organizing systems, i.e. mechanisms have to be included that prevent oscillations and similar effects. This must be part of the program itself and, thus, is not in the scope of RSN. However, extensibility is needed in two directions. First, parts of the rule set need to be replaced during runtime and the programmer must be able to implement completely new actions. Both are inherently supported by RSN.

The rest of the paper is organized as follows: Section 2 introduces the concepts of cellular signaling cascades. Section 3 outlines the ideas, the internal system aspects, and the rule language of RSN. The simulation model as well as the obtained results from the performance evaluation are depicted in Section 4. Finally, Section 5 concludes the paper.

2. Cellular signaling

The focus of this section is to briefly introduce the information exchange in cellular environments [20–22]. Information exchange between cells, called *signaling pathways*, follows the same principles that are required by network nodes. A message is sent to a destination and transferred, possibly using multiple hops, to this target.

Within complex organisms, such as mammals, cells are organized according to their physiological function. Neighboring cells have to inform each others that everything is normal, e.g. by sending growth factors, telling the neighbor: "Keep on growing". But also information from far away in the body can be received via a "telephone wire" called the blood. Via these pathways, information can be received and sent and have to be processed by the receiving cell. From a local point of view, the information transfer works as follows. One way is that the reception of signaling molecules via receptors. The receptor can be located on the surface of the cell. Typically, these receptors can bind an information molecule on the outside of the cell and during this binding it is activated, e.g. by a change in its sterical or chemical conformation (phosphorylation of defined amino acids). The activated receptor molecule is able to further activate signaling molecules inside the cell resulting in a "domino effect", because these activated signaling molecules in turn can activate further downstream signaling molecules see Fig. 2 (1-a). As an example the signaling via several growth factors can be mentioned.

Another example for the information transfer via receptors is the following. Small molecules like steroid hormones reach the cell of destination via the blood. This remote information exchange works as follows. A signal is released into the blood stream, the medium that carries it to distant cells. The hormone can pass the cell membrane and enter the cell. Within the cell the receptor binds the hormone. The ligand (hormone)-receptor complex can enter the nucleus of the cell and initiate gene transcription which leads to the production of an "answer".

This answer can be a different behavior of the cell. One example is the signaling via the hormone aldosterone binding to mineralocorticoid receptor expressed in e.g. some cells of the kidney (e.g. the Renin–Angiotensin–Aldosteron system [23]). A schematic construction is shown in Fig. 2 (1-b). Another example is the activation of the immune system.

The interesting property of this transmission is that the information itself addresses the destination. During differentiation a cell is programmed to express a subset of receptors in order to fulfill a specific function in the tissue. In consequence, hormones in the bloodstream affect only those cells expressing the correct receptor. This is the main reason for the specificity of cellular signal transduction. Of course, cells also express a variety of receptors which regulate the cellular metabolism, survival, and death.

In principle these signaling pathways are not as simple as described here. Many of these signaling pathways are interfering and interacting. Different signaling molecules are affecting the same pathway. Inhibitory pathways are interfering with the straightforward signal transduction. To sum up, the final effect is dependent on the strongest signal. The effect of such a signal transduction pathway is

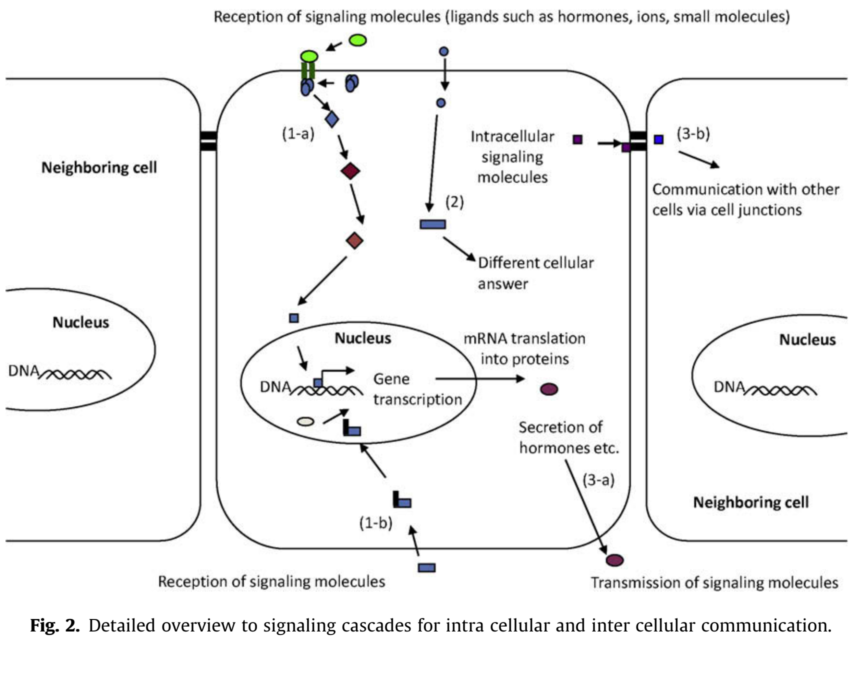


Fig. 2. Detailed overview to signaling cascades for intra cellular and inter cellular communication.

mostly gene transcription. Gene transcription means that the cell respond to the incoming signal by translation of specific mRNA into new proteins, which are then secreted (transported out of the cell), where it can induce signaling processes in the cell's direct environment. The cellular answer is a specific response according to the received signaling molecules and the current constitution of the cell. For example, signaling molecules can be created to send messages to other cells. Additional signaling molecules may affect the established signaling cascade towards the nucleus. The cellular answer is relying on the nucleus to initiate the desired process. Other possibilities are the reorganization of intracellular structures of the cell as a response to the received message.

This *specific response* is the key to information processing. It depends on the type of the signal and the state of the cells (which receptors have been built and which of them are already occupied by particular proteins). Finally, a specific cellular response is induced: either the local state is manipulated and/or a new messaging protein is created. In this scheme different possibilities are shown as to how cells can transfer answers. In Fig. 2 (3-a) the response to a received information particle is gene transcription and the production of a specific protein serving as a new message. This protein can be submitted into the extracellular space, e.g. secretion of hormones into the blood stream to activate cells far away as described above.

Additionally, messages can be forwarded to a neighboring cell via a paracellular pathway. In this case intracellular signaling molecules are transferred via junctions between cells. Congeneric cells develop several forms of junctions. One example are so-called "gap-junction" which represent tunnels where small molecules such as calcium ions or cAMP (cyclic-adenosine-mono-phosphate) can be transferred to the neighboring cell. This pathway is shown in Fig. 2 (3-b).

Finally, other non-protein molecules such as nitric oxide can enter the cell which are directly processed in a biochemical reaction. The resulting product of the reaction directly changes the behavior or state of the cell. For example, nitric oxide leads to smooth muscle contraction, schematically shown in Fig. 2 (2).

The lessons to learn from biology are the efficient and, above all, the very specific response to a problem, the shortening of information pathways, and the possibility of directing each problem to the adequate helper component. Therefore, the adaptation of mechanisms from cell and molecular biology promises to enable a more efficient information exchange. Besides all the encouraging properties, bio-inspired techniques must be used carefully by modeling biological and technical systems and choosing only adequate solutions.

3. Rule-based sensor network

Inspired by the capabilities of cellular signaling, i.e. the specific reaction to received information and the possibility to build signaling networks defining complex reaction pattern, we developed a rule-based programming system for application in SANETs. The primary design goals were a small footprint to enable the application of RSN on small embedded systems, easily transferable code, flexibility, and scalability for network-wide operations (basically, RSN provides the tools and concepts but the specific application needs to be designed properly as well). The rule-system greatly helps in designing distributed algorithms for use in self-organizing massively distributed systems. As described in the introduction, RSN was inspired by early rule-based systems that have been developed in the context of active networking solutions [12].

The key objectives motivating the development of RSN were improved scalability and real-time support for operation in sensor and actor networks. RSN is based on the following three design objectives that enable the mentioned objectives:

- *Data-centric communication* – Each message carries all necessary information to allow data specific handling and processing without further knowledge, e.g. about the network topology.
- *Specific reaction on received data* – A rule-based programming scheme is used to describe specific actions to be taken after the reception of particular information fragments.
- *Simple local behavior control* – We do not intend to control the overall system but focus on the operation of the individual node instead. Simple state machines have been designed, which control each node (being either sensor or actor).

In the following, the concepts of RSN are outlined and the intended use is depicted according to some examples relevant in the domain of SANETs.

3.1. Data-centric operation

The RSN architecture has been developed for SANET programming and operation that consequently follows the data-centric communication approach and enforces a complete network-centric operation [11]. Thus, instead of carrying address information, each message is encoded using a (type, content) pair. The type describes the message and the attached content. The data itself will usually include a value and application-specific meta information such as a geographical position or priority information.

Similar data-centric communication schemes have been proposed in the context of probabilistic data dissemination. The best known approach is gossiping [24]. Its key objective is essentially reduced communication overhead compared to other approaches – whereas the probability that messages reach the destination might be very low in specific scenarios such as linear setups. Optimized gossiping approaches are available but out of scope of this article.

The message encoding and processing in RSN are similar to the ones suggested by directed diffusion [25]. Even though the communication scheme is completely different, directed diffusion and RSN both rely on the identification of messages according to representative type information.

Each message will be encoded similar to the following structure:

`M := {type, region, confidence, content}`

At least `type` and `content` are needed for every message processed by RSN. Additional parameters such as a geographical region or a confidence level can be added in order to provide meta information for optimized data processing. Examples for such messages are:

- `{temperatureC, [10, 20], 0.6, 20}`A temperature of 20 °C was measured at the coordinates `[10, 20]`. The confidence is 0.6, therefore, a low-quality sensor was employed.
- `{pictureJPG, [10, 30], 0.9, "binary JPEG"}`A picture was taken in format JPEG at the coordinates `[10, 30]`.

3.2. RSN architecture

An extensible and flexible rule system is used to evaluate received messages and to provide the "programming" in a similar way as performed for the cellular response. Even though the message handling in biological cells is more sophisticated, the basic principles including the processing instructions (the DNA) are modeled.

The local behavior is controlled by a rule interpreter in form of simple state machines. The interpreter applies the installed rules to previously received messages. It uses a queuing subsystem that acts as a generic receptor for all messages and keeps them for a given time. This time control is necessary to prevent queue overflows due to received messages of unknown type.

Fig. 3 depicts the working behavior of a single RSN node. After receiving a message, it is stored in a message buffer. The rule interpreter is started periodically (after a fixed Δt) or after the reception of a new message. The period Δt is critical for particular applications such as data aggregation: the longer messages are stored before being processed, the better the possible aggregation ratio (more messages can be aggregated into a single one); and the longer the period, the longer the artificially introduced per hop delay.

Each rule that is used to process the received messages consisting of two parts, a condition and an action, as shown in Fig. 4. Starting with this overview, we will continue to use the specific RSN syntax to outline rules in the examples. The condition is intended to associate messages to a given rule, i.e. an action. In RSN, the specific reaction on received data is achieved by means of predicates. RSN is able to select all messages of a given type or messages with specific content attributes. All selected messages are stored in so-called working sets.

The predicate work on parameters of the received messages or on local state information. All parts of a message can be accessed, e.g. to select all messages of a given type (`$type = = "Temperature"`), to identify important messages (`$priority > 0.8`), or to test whether given thresholds have been exceeded (`$value > THRESHOLD`). Local state information includes the current time or parameters specific to the current message evaluation such as the number of messages in the buffer (`:count > 1`) or just a random value (`:random > 0.5`).

3.3. Available actions

The following actions have been implemented in the current version of RSN. Basically, the following categories of actions can be distinguished: *rule execution*, i.e. operations on the received messages; *node control*, i.e. control of the local node behavior (e.g. addition of sensors); and *simulation control*, i.e. actions needed for experiment control without influence on the node behavior.

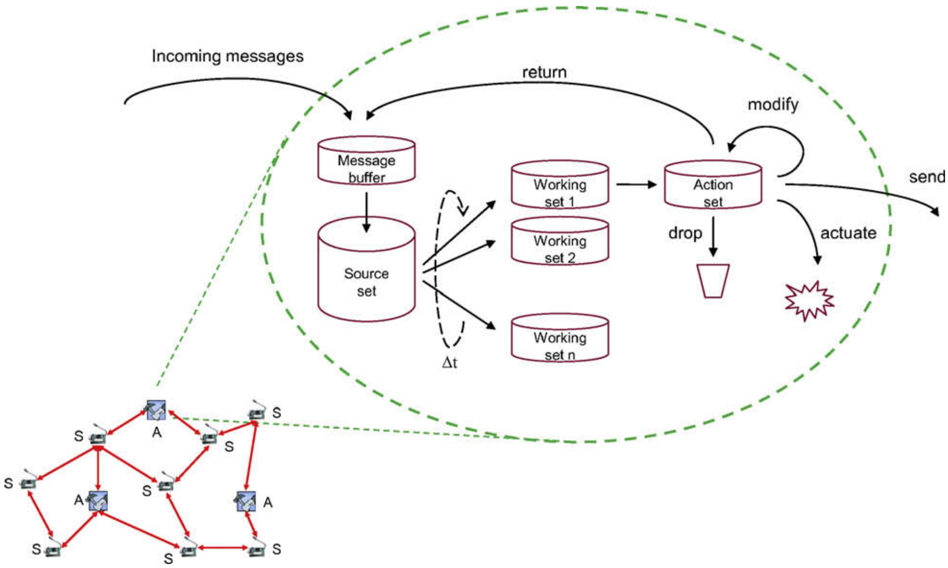


Fig. 3. The working behavior of a single RSN node. Received messages are stored in a buffer, selected for a working set according to specific criteria, and finally processed, i.e. forwarded, dropped, etc.

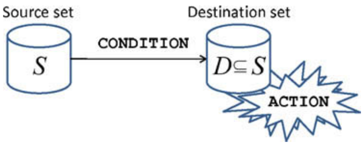


Fig. 4. Each rule selects a number of messages from the source set (CONDITION) and applies a (set of) actions to the selected messages (ACTION).

3.3.1. Rule execution

The following actions are meant to be used for network-centric processing of messages. All these actions work on the source message set that has been created by the condition element, i.e. by selecting messages according to a well-defined specific pattern. Examples for the application of the described actions are provided in the next section.

- `!stop` – Early termination of the rule execution. Depending on the current state (i.e. the number and kind of received messages), it may be necessary to stop the current processing of the rule set. The next iteration will start with the first available rule.
- `!drop` – Erases all messages in the current set. Needs to be called if messages have been successfully processed.
- `!dropDuplicates` – All duplicates are discarded according to a unique identifier in each message. This command is needed to emulate, for example, standard gossiping algorithms.
- `!return` – A new message is created and appended to the source message set.
- `!returnAll` – Copies of all messages in the current set are created and stored in the source message set.
- `!send` – A new message is created and submitted to the lower layer protocol for transmission to neighboring nodes.
- `!sendAll` – Copies of all messages in the current set are created and submitted to the lower layer protocol for transmission to neighboring nodes.
- `!actuate` – A message is sent to locally connected actuators.

3.3.2. Node control

Besides the actions for message processing, actions have been integrated to control the local node behavior. Such node control actions allow to enable/disable locally attached sensors and actuators as well as to modify the current rule set, i.e. the local programming of a node.

- `!controlSensor` – A control message is sent to all attached sensors. According to the submitted attributes in `$control`, the behavior of the sensors can be controlled: `rsnSensorEnable` and `rsnSensorDisable` enable or disable the sensor, `rsnSensorSetType` updates the type field of the sensor, and `rsnSensor-SetMeasuringInterval` changes the sampling frequency.

  `!controlSensor ($control := rsnSensorSet-MeasuringInterval, $text := "1s");`
- `!controlActuator` – Similarly, this command controls locally attached actuators. The attribute `$control` defines the action: the actuator is enabled or disabled by `rsnActuatorEnable` and `rsnActuatorDisable`, respectively, and `rsnActuatorSetType` updates the type field of the actuator.

  `!controlActuator ($control := rsnActuator Enable);`
- `!controlManagement` – The management plane defines the rule set itself. Again, the `$control` attribute is used to specify the intended action: the rule interpretation can be started or stopped by `rsnManagementEn-able` and `rsnManagementDisable`, respectively, the rule set can be replaced in order to modify the behavior of this node using `rsnManagementFromRsnString` or `rsnManagementFromRsnFile`, and the evaluation interval can be configured by `rsnManagement Set-Evaluation Interval`.

  `!controlManagement ($control := rsnManage-ment FromRsnFile, $text := "filename. rsn").`

3.3.3. Simulation control

The following actions have been integrated for simplified control of simulation experiments. These actions are not working on a given set of messages. Nevertheless, it is possible to initiate these actions based on the current state of the node, e.g. after the reception of a specific message.

- `!recordAll` – Statistics are recorded for all messages in the current working set. In particular, the following information is stored: ID of the current node, ID of the node that generated the message, node specific ID of the message, globally unique ID of a message, hop count, current time, and delay (elapsed time since message creation).
- `!endSimulation` – This action terminates an experiment. In our implementation for OMNeT++, the simulation core is notified accordingly (see below).

3.4. Variables and variable handling

All the described conditions and actions work on a set of message parameters or local variables describing the state of the node. In the following, some of the most important variables are introduced. Additionally, selected statistical preprocessing techniques for data aggregation have been integrated into the current version of RSN in order to enable selected application examples. In the following section, we describe and analyze two application examples that inherently benefit from the network-centric preprocessing features provided by RSN.

3.4.1. Message attributes

Each message is specifically encoded to allow receiving nodes to determine the meaning of the message and the necessary behavior. This encoding can be changed according to the application scenario. Possible parameters (currently used in the RSN implementation) are listed in Table 1.

3.4.2. Node attributes

Each node can store and update state information locally. In the context of self-organization, this refers to the local state of an autonomous system. Such information can be updated according to received messages or by other local observations. Table 2 lists the currently implemented node attributes.

3.4.3. Preprocessing features

Data aggregation is an important issue in massively distributed systems. Usually, statistical measures are used to describe results received from several nearby nodes. RSN supports such data aggregation techniques by providing a set of preprocessing techniques as summarized in Table 3. All the listed operations process the messages in the current working set.

3.5. Application examples

In order to demonstrate the capabilities, two simple application examples are depicted in the following. First, the probabilistic data forwarding technique gossiping is reproduced in RSN. The algorithm according to [24] forwards packets with a given gossiping probability p. In order to cope with special cases (problems) such as linear networks, flooding is used for the first n hops.

Each message is assumed to be encoded in the following way:

```
M := {hopCount, content}
```

Then, the gossiping algorithm can be formulated as follows (again, we are using the implemented RSN syntax for the examples):

```
# infinite loop prevention
if $hopCount >= networkDiameter then {
   !drop;
}
# flooding for the first n hops
if $hopCount < n then {
   !sendAll;
   !drop;
}
# gossiping
if :random < p then {
   !sendAll;
   !drop;
}
# clean up
!drop;
```

In the first block, all messages are selected that have a `hopCount` greater or equal to `networkDiameter`. These messages are silently dropped (`!drop`). This command is included to prevent infinite loops. The second block selects all messages with `hopCount` smaller than n and forwards these messages (`!sendAll`). After processing the messages, they are discarded. The third block selects all remaining messages in the working set if an on-demand calculated random value (`:random`) is smaller than the gossiping probability p. These messages are forwarded and all remaining messages are dropped.

From this simple example, two mechanisms become obvious. First, each command operates on sets of messages instead of single messages. Secondly, messages remain in the working set until they are dropped. Thus, multiple commands may be applied to particular messages.

A second example should demonstrate more sophisticated applications. In this example, the sensors are used to measure the temperature. Data aggregation is performed to reduce the number of messages in the system. Additionally, critical temperature values are observed and alarm messages are created if a threshold has been exceeded.

The message encoding is similar to the previous example:

```
M := {type, position, content, priority}
type := ( temperature || alarm )
```

Table 1
Currently implemented message attributes

Attribute	Description
`$name`	Descriptive name of the message
`$type`	Type of the message; describes the content
`$position`	Position of the source node
`$hopCount`	Number of traversed nodes
`$priority`	Importance factor of this message
`$length`	Length of the message
`$creationTime`	Timestamp describing the creation of the message
`$value`	Message type specific value
`$text`	Further informative text, e.g. to qualify the value

Table 2
Currently implemented node attributes

Attribute	Description
`:count`	Number of messages in the current working set
`:totalMessageCount`	Number of all messages received by the node
`:hostName`	ID of the current host
`:position`	Position of the node
`:random`	Random value for probabilistic decisions

Table 3
Implemented preprocessing features

Command	Description
`@minimum`	Minimum of the selected value
`@maximum`	Maximum of the selected value
`@sum`	Sum of the selected value
`@average`	Average of the selected value
`@median`	Median of the selected value
`@lowerQuartile`	Lower quartile of the selected value
`@upperQuartile`	Upper quartile of the selected value
`@count`	Number of the selected value

The complete algorithm can now be written as follows:

```
# test for exceeded threshold and
# generate an alarm message
if $type = temperature &&
   $content > threshold then {
   !actuate(buzzerOn);
   !send($type := alarm, $priority = 1);
}
# perform data aggregation
if $type = temperature &&
   :count > 1 then {
   !send($content := @median of $content,
   $priority := 1 − @product of $priority);
   !drop;
}
# message forwarding, e.g. according
# to a simplified gossiping algorithm
if :random < $priority then {
   !sendAll;
   !drop;
}
!drop;
```

In this example, the three command blocks actually perform different operations. The first block tests the temperature value and, if the threshold is exceeded, a local actuation is enforced (a buzzer is turned on – `!actuate`) and a new alarm message is generated with message priority set to one (`!send`). In the second block, all temperature messages are aggregated (if more than one has been received – `:count`). The content is set to the median of all temperature values and the message priority is increased. Finally, the last block is in charge of message forwarding.

From these two examples, it can be seen that RSN provides a powerful set of commands to enable in-network operation and control for SANETs. Nevertheless, a number of open issues still exist:

- Handling of unknown messages – Which action should be performed if unknown messages, i.e. messages of unknown type, have been received? Basically, two decisions are possible, drop vs. seamless forwarding, while not being appropriate in all application scenarios.
- Period of RSN execution Δt – The duration of messages stored in the local node introduces an artificial per-hop delay. The optimal value for Δt affects the aggregation quality vs. real-time message processing.
- Rule generation and distribution – So far, we considered homogeneously programmed nodes. This is not necessarily the optimal case. Also, new rules may be required during the lifetime of the network. The rule deployment needs further research in terms of diffuse or random distribution vs. global optimization.

4. Simulation experiments

In order to evaluate the efficiency of RSN, we compared it to the typical setup used in sensor network scenarios. Multiple sensor nodes are continuously measuring environmental conditions and transmit this information to a central base station. This, in turn, will analyze the received results and engage the installed actors accordingly. For the communication, we chose Dynamic MANET on Demand (DYMO), which is a popular routing protocol used in the ad hoc and sensor network community. We also created the same setup with RSN for a direct comparison.

4.1. Setup and scenario

For the simulations, we developed a simulation model using OMNeT++ version 3.3 [26], a simulation environment free for non-commercial use, and the INET Framework 20060330, a set of simulation modules released under the GPL. OMNeT++ runs discrete, event-based simulations of communicating nodes on a wide variety of platforms and is getting increasingly popular in the communications community. Scenarios in OMNeT++ are represented by a hierarchy of reusable modules written in C++. Their relationships and communication links are stored as Network Description (NED) files. Simulations are either run interactively in a graphical environment or executed as command-line applications.

We implemented RSN in form of a C++ library. This library contains all functionality that is necessary to process RSN statements. RSN statements are formulated in a flexible script language. We integrated the RSN library into the OMNeT++ simulation framework in order to execute intensive tests and experiments with different algorithms for data aggregation, probabilistic data communication, and distributed actuation control.

For comparison of network-centric actuation control with classic base-station approaches, we investigated the following scenario. A large number of sensor nodes are considered to measure environmental conditions such as the temperature. If measurements exceed a given threshold, actuation devices are triggered. Such actuators are able to interact with the environment or to initiate secondary events and actions.

In order to evaluate the communication behavior in this scenario, we created a simulation model in which 100 sensor nodes are placed on a rectangular playground. The nodes are either distributed in form of a regular grid or in a random pattern. In addition to these sensor nodes, four actor nodes are included in the middle of each quadrant.

In our example, we configured all sensors to periodically send their sensor readings towards the actuators. In the base-station scenario, the central base station checks the received measures to determine whether they exceed the given threshold and forwards only appropriate messages to the actors. In the network-centric operation scenario, all messages are distributed with a certain gossiping probability and only the actors are able to check the threshold. All the variable parameters used in the simulation are summarized in Table 4.

For the base-station scenario, we used the DYMO routing protocol [27], which can be considered a de-facto standard in the ad hoc and sensor networking community. In particular, we used the implementation of DYMO available for OMNeT++ [28].

Table 4
Variable simulation parameters

Parameter	Values
Sensor reading period	60 s, 600 s
Sensor readings	uniform in [0, 100]
Threshold for actuation	50, 70, 90
RSN gossiping probability	0.2, 0.5, 0.8

For all communications, wireless networking modules working according to the IEEE 802.11b standard have been used. All simulation parameters used to parameterize the modules of the INET Framework are summarized in Table 5.

In the RSN scenario, the sensor nodes have been configured with the following program. It ensures that all messages are forwarded with a probability of `GOSSIP-PROB` (set to 0.2, 0.5, and 0.8, respectively) over a maximum distance of `DIAMETER` (for the presented simulation results, we configured the maximum hops count to four). The `!recordAll` command is used for statistical purposes only.

```
!recordAll;
if $hopCount >= DIAMETER then {
   !drop;
}
if :random <= GOSSIP-PROB then {
   !sendAll;
   !drop;
}
   !drop;
```

The actors have a much simpler programming. For each received message, they check whether the `THRESHOLD` (set to 50, 70, and 90, respectively) was exceeded and, if necessary, local actuation is initiated.

Table 5
INET framework module parameters

Parameter	Value
net.headerLengthByte	20 byte
net.ROUTE_TIMEOUT	120 s
net.ROUTE_DELETE_TIMEOUT	200 s
net.NET_DIAMETER	10
mac.address	Auto
mac.bitrate	2 Mbit/s
mac.broadcastBackoff	31 slots
mac.maxQueueSize	14 Pckts
mac.rtsCts	True
decider.bitrate	2 Mbit/s
decider.snirThreshold	4 dB
snrEval.bitrate	2 Mbit/s
snrEval.headerLength	192 bit
snrEval.snrThresholdLevel	3 dB
snrEval.thermalNoise	−110 dBm
snrEval.sensitivity	−85 dBm
snrEval.pathLossAlpha	2.5
snrEval.carrierFrequency	2.4 GHz
snrEval.transmitterPower	1 mW
channelcontrol.carrierFrequency	2.4 GHz
channelcontrol.pMax	2 mW
channelcontrol.sat	−85 dBm
channelcontrol.alpha	2.5

```
!recordAll;
if $value > THRESHOLD then {
!actuate($type := rsnActuatorLightSource,
$value := @average of $value,
$priority := 2);
!drop;
}
!drop;
```

4.2. Measurement results

A number of simulations have been executed with the primary objective to analyze the following characteristics of both evaluated communication and control approaches:

- Real-time support, i.e. the overall latency between measuring a value higher than the particular threshold and the time the message successfully arrived at the actuators. In this context, also the path length is of interest, which is directly proportional to the end-to-end latency and to the message loss probability.
- Overhead, i.e. the number of messages that need to be processed by all the nodes to transmit the necessary data messages. This includes protocol overhead from routing protocols as well as overhead due to duplicated messages for gossiping approaches.

In order to increase the statistical significance of the simulation experiments, all simulations have been executed five times (runs). In each experiment, all the 100 sensor nodes send exactly 200 packets. After starting the simulation, the time for each sensor to start its local activities is uniformly distributed over the first 60 s. This behavior first models the initialization of real sensor nodes at arbitrary times and, secondly, it prevents collisions on the MAC layer due to synchronization effects.

All results are shown as boxplots. For each data set, a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the edges of the box towards the minimum and maximum of the data set, but no further than 1.5 times the interquartile range. Data points outside the range of box and whiskers are considered outliers and drawn separately. Additionally, the mean value is depicted in form of a small filled square. In most graphs, the overall mean and median are shown in the middle bar.

4.2.1. Real-time support

First, the end-to-end latency of the application messages has been analyzed. We measured the time from creating a sensor message until it was successfully received by the actor. Thus, the term end-to-end latency describes the time from measuring an effect at a sensor until the measurement has been received (and processed) by an actor. This includes the propagation delay between the forwarding nodes as well as the processing (and queuing) delay at all relaying nodes. Because only messages exceeding a given threshold are of interest for the actors, we just analyzed the latency after identifying the message as matching this criterion.

Figs. 5 and 6 show the measurement results. In all the shown graphs, all setups as depicted in the previous subsection and all the simulation runs are integrated to show the statistical effects of single parameters. In Fig. 5, results for the RSN scenario are shown. The graphs differentiate between the deployment scenarios and the gossiping probability. If only the first reception of the first copy of the message is considered, the end-to-end delay slightly oscillates around 1.4 ms. The measured maximum is at about 16 ms. The results are nevertheless only meaningful, if all sensor messages can be differentiated, e.g. by a unique id. If this is not possible, the reception of further copies cannot be distinguished form the first one. The measurement results taking this effect into account slightly oscillate around 2.2 ms with a maximum peak at 33 ms.

If we compare these results to the DYMO scenario as shown in Fig. 6, we obviously see that the delays in this scenario are significantly higher (median: 20 ms, mean: 55 ms, and max: 5700 ms). There are two reasons for this behavior. First, the mean path length is essentially longer as discussed below and, secondly, the on-demand routing protocol takes some time for setting up the routing path before being able to transmit a message. This effect is shown by the comparison between the 60 s and 600 s message generation setups. The route timeout of DYMO has been configured to 120 s. Thus, in the 600 s scenario, almost always the route towards the base and towards the actor nodes will timeout and needs to be reestablished.

Secondly, we analyzed the path length, i.e. the hop count, in the same experiments. The results are depicted in Figs. 7 and 8. Obviously most messages are transported over only two hops in the RSN scenario. This also explains the log latency communication. In the DYMO scenario, each message needs to be transmitted first to the base station (which requires on average about six hops) and then it is forwarded to the actor nodes (requiring on average seven hops). Thus, we could expect a factor of about 3-7 for the latency difference between the RSN and the DYMO scenario. Nevertheless, as shown in Fig. 6, the factor is about

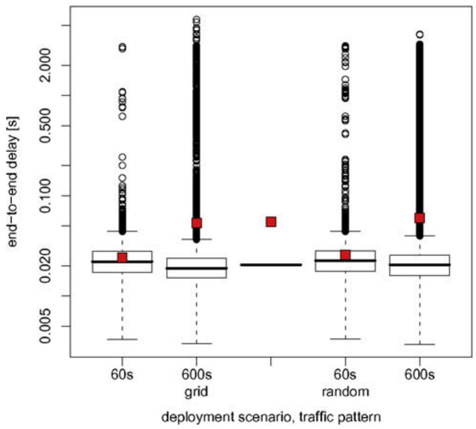


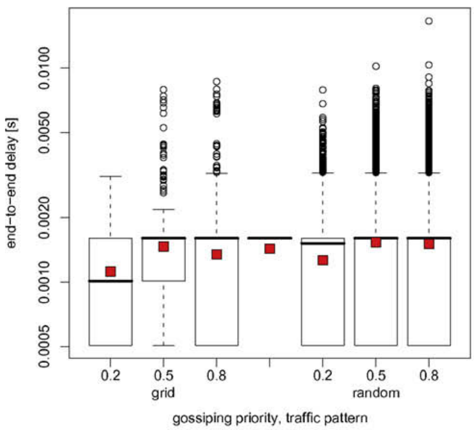
Fig. 6. End-to-end latency in the DYMO scenario as observed from the application.

10–55. The only explanation for this high factor is the overhead resulting from the on-demand routing.

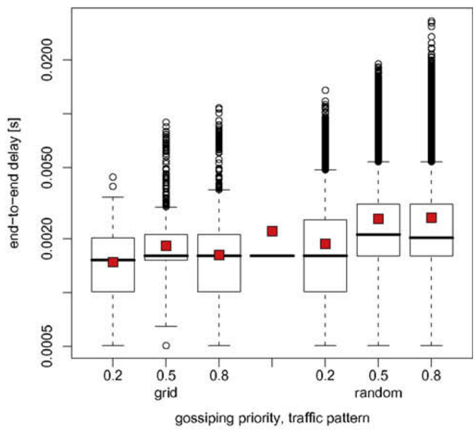
4.2.2. Overhead

In the previous paragraph, we have shown that the overhead may essentially affect the real-time support of the employed communication techniques. Especially in the context of SANETs, the overhead also characterizes the energy efficiency of the entire system, and thus, the possible *network lifetime* [29].

In Fig. 9, the protocol overhead is depicted. For the DYMO scenario with 60 s sampling period, we observed that each node needs to send on average between 20 and 1300 DYMO messages in order to transmit 200 data messages in the random and grid deployment scenario, respectively. If the sampling rate becomes too small, i.e. if the route timeouts of DYMO are triggered, on average between 5200 and 76,000 DYMO messages need to be sent for delivering 200 data messages. The primary reasons for these high numbers are the high probability of multiple nodes

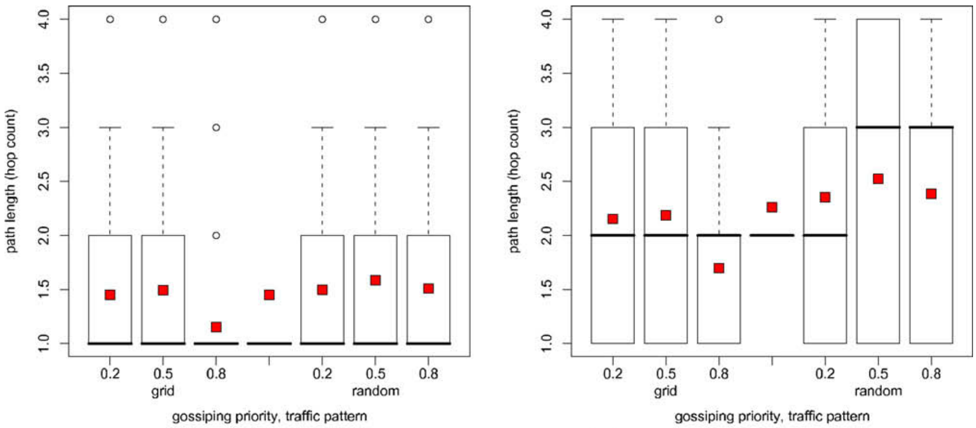


(a) time until the first copy of a message arrives



(b) time until any copy arrives

Fig. 5. End-to-end latency in the RSN scenario.



(a) path length for the first copy of a message

(b) path length for any copy

Fig. 7. Path length (number of hops) for the RSN scrnario.

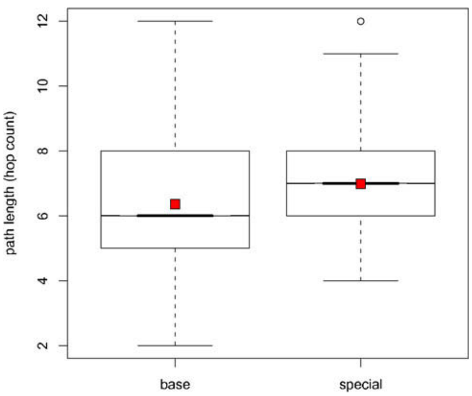
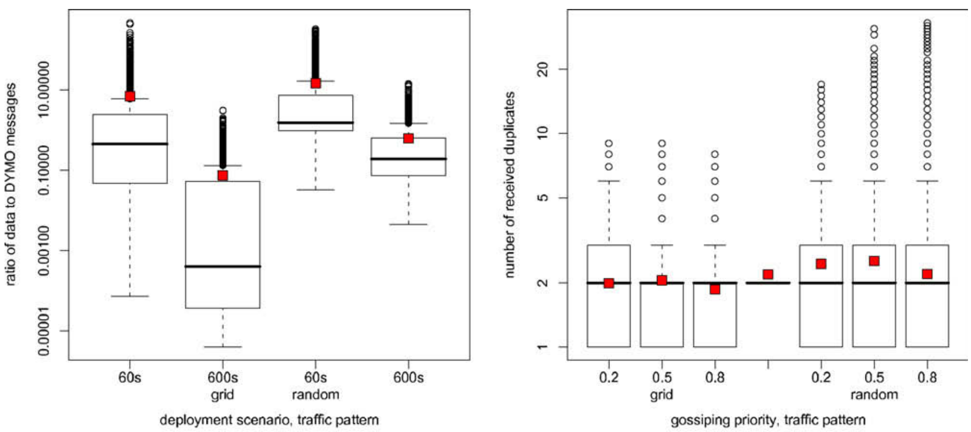


Fig. 8. Path length towards the base station and between the base and the actors (DYMO).

searching simultaneously for a given destination and the increased collision probability (see below). The ratio of data to DYMO messages is shown in Fig. 9a – this figure takes all data messages into account, whether generated at the local node or forwarded on behalf of other nodes.

In the RSN scenario, in almost all measurements about two duplicates are received by the actor nodes. Thus, an overhead factor of two can be noticed as shown in Fig. 9b. According to the probabilistic forwarding scheme, some peaks up to 33 duplicates can be recognized. This effect has been expected and it can, according to the median of two, be neglected. On the other hand, the loss ratio is quite high in the RSN scenario as depicted in Fig. 10b. The primary reason lies in the working principle of probabilistic communication. A number of sensors need to send their messages of three of four hops towards the actors. Thus, the probability of reaching the destination equals



(a) Ratio of data to DYMO messages

(b) Number of received duplicates in the RSN scenario

Fig. 9. Overhead due to protocol characteristics.

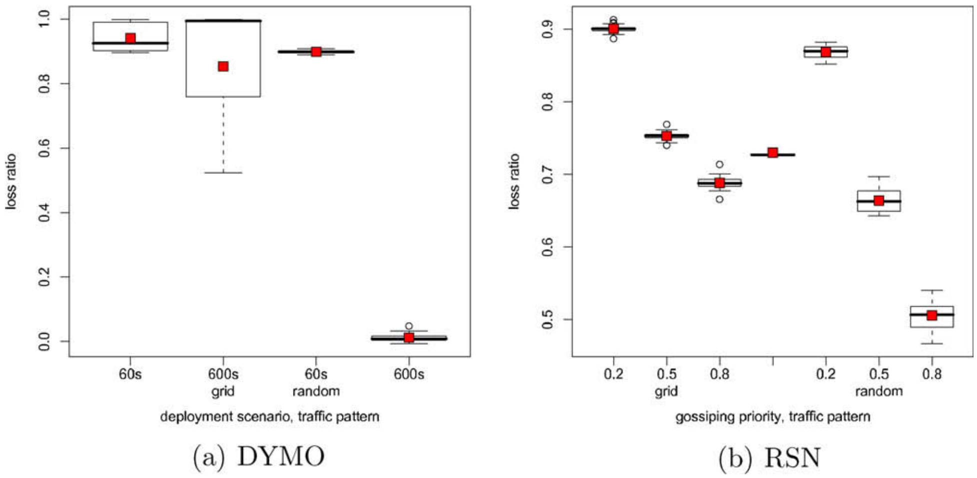


Fig. 10. Loss ratio as observed at the network layer.

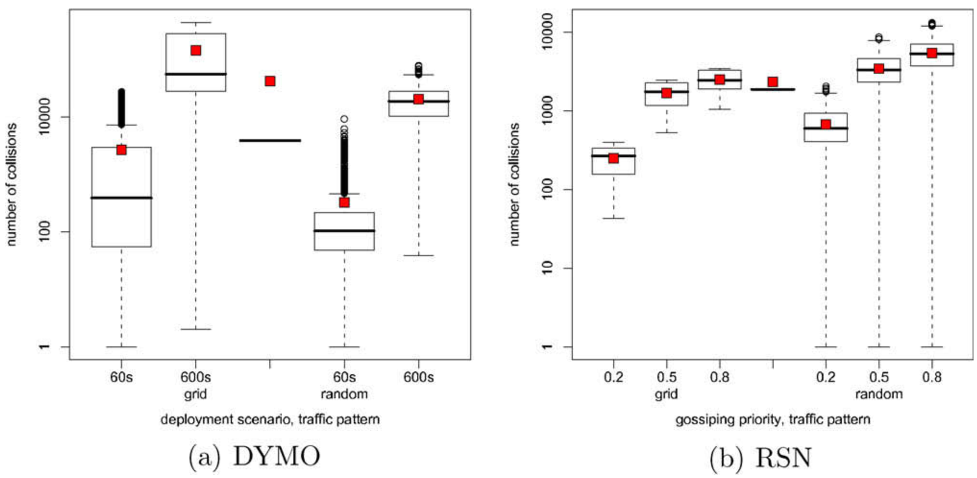


Fig. 11. Number of collisions in the wireless medium.

to p^3 or p^4, respectively, which is quite low for gossiping probabilities p of 0.2, 0.5, and 0.8.

Another reason for the high loss ratios is the unreliable wireless communication. As shown in Fig. 10a, the loss ratio is also high for the DYMO scenario. Thus, we finally analyzed the number of collisions at the MAC layer. This measure allows to determine the load distribution over the time and the ability of the network to afford the necessary number of data and protocol message transmissions. The results are shown in Figs. 11a and b. It can be seen that the use of RSN leads to reduced network congestion (on average we measured 2300 collisions) compared to the DYMO scenario (42,300 collisions).

5. Conclusion and future research

In this paper, we presented and discussed a methodology for network-centric operation in SANETs. Inspired by biological information processing, we developed three easy to handle building blocks: data-centric operation, specific reaction on received data, and simple local behavior. The resulting architecture, which we named rule-based sensor network (RSN), is able to process sensor data and to perform network-centric actuation according to a given set of rules. In particular, this system is able to perform collaborative sensing and processing in SANETs with purely local rule-based programs. The interaction and collaboration between these nodes finally leads to an optimized system behavior in an emergent way. We also developed a simulation model to compare the system performance with classical base-station approaches. In particular we analyzed the performance of the DYMO routing protocol, which can be considered state of the art in ad hoc and sensor network routing, with RSN. It turned out that RSN provides much better scalability and support for real-time operation. This advantage is achieved by reducing the predictability of the system to a certain degree. Depending on the application scenario, this disadvantage might be feasible considering deployments with huge numbers of sensor and actor nodes. Additionally, the possible parameterization of the RSN approach allows adjusting the reliability vs. overhead ratio according to the current needs in the network.

We see RSN as a significant advancement in the domain of SANET research as it provides a comprehensive while simple approach to network-centric operation. Two major aspects should be emphasized. First, each RSN rule operates on sets of messages instead of single messages. This allows performing data aggregation and fusion algorithms without the need of maintaining dedicated meta information describing the message history. Secondly, messages remain in the working set until they are explicitly discarded. This feature provides means of storing local state without reference to additional data structures outside the rule system. Applications can exploit this to gain additional advantage after replacing single rules or even the entire rule set. The new rules can be simply applied to "histories" of previously received data. This mechanism inherently supports the extensibility of RSN in terms of updates of the rule-system itself and for implementing new actions during runtime.

In this paper, we outlined the feasibility mainly based on selected simulation results. Meanwhile, an implementation for real sensor nodes is also available [30] and our future work will include intensive lab experiments with this platform.

Future research directions not only for our group include two steps. First, global rule optimization based for example on an observer/controller system model [31] needs to be investigated. Expected advantages are the strengthened deterministic behavior of the overall system. However, it is not obvious how status information can be collected from a centralized position for adequate optimizations. Thus, a second step is necessary by investigating fully self-controlled behavior based on algorithms for self-optimization of rule sets.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–116.

[2] D. Culler, D. Estrin, M.B. Srivastava, Overview of sensor networks, Computer 37 (8) (2004) 41–49.

[3] I.F. Akyildiz, I.H. Kasimoglu, Wireless sensor and actor networks: research challenges, Elsevier Ad Hoc Networks 2 (2004) 351–367.

[4] V.C. Gungor, Ö.B. Akan, I.F. Akyildiz, A real-time and reliable transport (RT) protocol for wireless sensor and actor networks, IEEE/ACM Transactions on Networking (ToN) 16 (2) (2008) 359–370.

[5] T. Melodia, D. Pompili, V.C. Gungor, I.F. Akyildiz, A distributed coordination framework for wireless sensor and actor networks, in: 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2005), Urbana-Champaign, IL, USA, 2005, pp. 99–110.

[6] H. Gupta, S.R. Das, Q. Gu, Connected sensor cover: self-organization of sensor networks for efficient query execution, in: 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2003), Annapolis, MD, USA, 2003.

[7] K.H. Low, W.K. Leow, M.H. Ang, Autonomic mobile sensor network with self-coordinated task allocation and execution, IEEE Transactions on Systems, Man, and Cypernetics–Part C: Applications and Reviews 36 (3) (2005) 315–327.

[8] C. Prehofer, C. Bettstetter, Self-organization in communication networks: principles and design paradigms, IEEE Communications Magazine 43 (7) (2005) 78–85.

[9] F. Dressler, I. Carreras (Eds.), Advances in biologically inspired information systems – models, methods, and tools, Studies in Computational Intelligence (SCI), vol. 69, Springer, 2007.

[10] F. Dressler, I. Dietrich, R. German, B. Krüger, Efficient Operation in Sensor and Actor Networks Inspired by Cellular Signaling Cascades, in: 1st ACM/ICST International Conference on Autonomic Computing and Communication Systems (Autonomics 2007), ACM, Rome, Italy, 2007.

[11] F. Dressler, Network-centric actuation control in sensor/actuator networks based on bio-inspired technologies, in: 3rd IEEE International Conference on Mobile Ad Hoc and Sensor Systems (IEEE MASS 2006): 2nd International Workshop on Localized Communication and Topology Protocols for Ad hoc Networks (LOCAN 2006), IEEE, Vancouver, Canada, 2006, pp. 680–684.

[12] K.L. Calvert, S. Bhattacharjee, E.W. Zegura, J. Sterbenz, Directions in active networks, IEEE Communications Magazine 36 (10) (1998) 72–78.

[13] M. Sloman, E. Lupu, Policy specification for programmable networks, in: 1st International Working Conference on Active Networks (IWAN'99), Lecture Notes in Computer Science, Berlin, Germany, 1999, pp. 73–84.

[14] L. Lymberopoulos, E. Lupu, M. Sloman, An adaptive policy-based framework for network services management, Journal of Network and Systems Management 11 (3) (2003) 277–303.

[15] J. Vitek, C. Tschudin, Mobile Object Systems – Towards the Programmable Internet, vol. LNCS 1222, Springer, 1997.

[16] L.F. Mackert, I.B. Neumeier-Mackert, Communicating rule systems, in: 7th IFIP International Conference on Protocol Specification, Testing and Verification, 1987, pp. 77–88.

[17] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database Systems (TODS) 30 (1) (2005) 122–173.

[18] S. Sen, R. Cardell-Oliver, A rule-based language for programming wireless sensor actuator networks using frequency and communication, in: 3rd Workshop on Embedded Networked Sensors (EmNets 2006), Cambridge, MA, 2006.

[19] J.C. Castillo, T. Olivares, L. Orozco-Barbosa, Implementation of a rule-based routing protocol for wireless sensor networks, in: 2nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks (PM2HW2N'07), ACM, Chania, Crete Island, Greece, 2007, pp. 19–25.

[20] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J.D. Watson, Molecular Biology of the Cell, Third ed., Garland Publishing Inc., 1994.

[21] T. Pawson, Protein modules and signalling networks, Nature 373 (6515) (1995) 573–580.

[22] G. Weng, U.S. Bhalla, R. Iyengar, Complexity in biological signaling systems, Science 284 (5411) (1999) 92–96.

[23] A. Guyton, Blood pressure control – special role of the kidneys and body fluids, Science 252 (5014) (1991) 1813–1816.

[24] Z.J. Haas, J.Y. Halpern, L. Li, Gossip-based ad hoc routing, in: 21st IEEE Conference on Computer Communications (IEEE INFOCOM 2002), 2002, pp. 1707–1716.

[25] C. Intanagonwiwat, R. Govindan, D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, in: 6th ACM International Conference on Mobile Computing and Networking (ACM MobiCom 2000), Boston, MA, USA, 2000, pp. 56–67.

[26] A. Varga, The OMNeT++ Discrete Event Simulation System, in: European Simulation Multiconference (ESM 2001), Prague, Czech Republic, 2001.

[27] I. Chakeres, C. Perkins, Dynamic MANET On-Demand (DYMO) Routing, Internet-Draft (work in progress) draft-ietf-manet-dymo-10.txt (July 2007).

[28] C. Sommer, I. Dietrich, F. Dressler, A simulation model of DYMO for ad hoc routing in OMNeT++, in: 1st ACM/ICST International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008): 1st ACM/ICST International Workshop on OMNeT++ (OMNeT++ 2008), ACM, Marseille, France, 2008.

[29] I. Dietrich, F. Dressler, On the lifetime of wireless sensor networks, ACM Transactions on Sensor Networks (TOSN) 5 (1) (2009).

[30] F. Dressler, A. Awad, D. Wang, R. German, Rule-based programming of heterogeneous sensor and actor networks, in: 27th IEEE Conference on Computer Communications (IEEE INFOCOM 2008), Demo Session, IEEE, Phoenix, AZ, 2008.

[31] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, H. Schmeck, Organic computing addressing complexity by controlled self-organization, in: 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2006), IEEE Computer Society, 2006, pp. 185–191.

Falko Dressler received his M.Sc. and Ph.D. from the University of Erlangen in 1998 and 2003, respectively. In 2003, he joined the Computer Networks and Internet group at the Wilhelm-Schickard-Institute for Computer Science, University of Tuebingen. Since 2004, he is an assist. professor in the Computer Networks and Communication Systems group at the Department of Computer Science, University of Erlangen, where he coordinates the Autonomic Networking group. He is an Editor for the Elsevier Ad Hoc Networks journal, the ACM/Springer Wireless Networks (WINET) journal, and the Journal of Autonomic and Trusted Computing (JoATC). He was guest editor of special issues on self-organization, autonomic networking, and bio-inspired computing and communication for IEEE Journal on Selected Areas in Communications (JSAC), Elsevier Ad Hoc Networks, and Springer Transactions on Computational Systems Biology (TCSB). He published two books including Self-Organization in Sensor and Actor Networks, published by Wiley in 2007. He co-authored more than 100 reviewed research papers. He is a Senior Member of the IEEE (IEEE Communications Society, IEEE Computer Society), member of ACM and GI (Gesellschaft fnr Informatik). He is actively participating in several working groups of the IETF. His research activities are focused on (but not limited to) Autonomic Networking addressing issues in Wireless Ad Hoc and Sensor Networks, Self-Organization, Bio-inspired Mechanisms, Network Security, Network Monitoring and Measurements, and Robotics.

Isabel Dietrich received her M.Sc. in computer sciences from the University of Erlangen, Germany in 2005. She is currently a PhD student in the Computer Networks and Communication Systems group at the Department of Computer Sciences, University of Erlangen. Her research interests include UML-based discrete-event simulation, and performance analysis of communication networks and wireless sensor networks.

Reinhard German received a diploma in computer science in 1991, his Ph.D. in 1994 and his Habilitation degree in 2000 from the Computer Science Department, Technical University of Berlin. Then he joined the Department of Computer Science at the University Erlangen-Nuremberg first as an associate professor (system simulation) and since 2004 as a full professor (computer networks and communication systems) where he is currently the head of department. His research interests include model-based and measurement-based performance analysis, modeling and simulation paradigms and tools, numerical analysis of Markovian and non-Markovian models, vehicular communications and autonomous sensor/actuator networks.

Bettina Krüger received her M.Sc. in biology from Göttingen University in 1998 and her Ph.D. from the University of Erlangen in 2003. In 2003, she joined the Department for Nephrology, Erlangen University as a Postdoc. Since 2004, she is a Postdoc at the Department for Cellular and Molecular Physiology, Erlangen University. She is a member of the German society of physiology and the German society of biochemical and molecular biology. Her research interests include kidney physiology, especially the molecular mechanisms of the regulation of kidney ion channels.