Neighbor Detection and Crosstalk Elimination in Self-Reconfigurable Robots

David Johan Christensen, David Brandt, Ulrik Pagh Schultz and Kasper Stoy
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
Odense, Denmark
Email: {david, david.brandt, ups, kaspers}@mmi.sdu.dk

***Abstract*— This paper addresses two issues concerning communication between neighbor modules in self-reconfigurable robots. The first issue is automatic neighbor detection that is due to modules self-reconfiguring, whereby the local communication network topology dynamically changes. The second issue is crosstalk between non-neighbor modules, where data packages send through an infrared communication channel are received by a non-neighbor module because of reflections. In this paper, we proposed algorithmic solutions to automatic neighbor detection and crosstalk elimination. The algorithms are simple, distributed, self-organizing and robust. For validation, they are implemented and evaluated on the physical ATRON system. In conclusion, the algorithms are efficient and effective and we argue that these algorithmic contributions may be applicable on other systems as well.**

I. Introduction

By connecting, moving, and disconnecting the modules, of a self-reconfigurable robot, can autonomously change their configuration. Such modules are often homogeneous in both hardware and software and their control completely distributed. Hence, modules may be mass-produced and high numbers of modules can be assembled into a large range of different functional robots. Further, a robot can adapt its morphology to the task if required. Distributed control may ensure scalability and homogenous modules ensures redundancy where one module may takeover the functionality of another modules, thereby, enabling fault-tolerance and self-repair.

This paper considers some challenges related to the communication in self-reconfigurable robot. In such systems, local communication between two neighbor modules is desired. In addition, modules must be able to communicate without being physically connected, e.g. to negotiate when to connect. These requirements imply directional and wireless communication, why infrared communication is often used. However, emitted infrared light can be reflected off other modules or objects to be received by a non-neighbor module. Such crosstalk has serious effects on the functionality of the system, just consider a *disconnect from me* command arriving at the wrong module. In general, such crosstalk can be hard to avoid and care must be given to design the communication system of the modules to reduce it.

Here, we consider the ATRON module which mechanical design did not allow its infrared transceivers and receivers to be shielded off. Hence, the emitted infrared light can be reflected and received as crosstalk by non-neighbor modules. This paper proposes a distributed algorithmic solution to detect and remove crosstalk without the need of globally unique IDs. Further, we propose an algorithmic strategy for automatic detection of neighbors, which is necessary due to self-reconfiguration of modules and non-trivial because of imperfect communication. The algorithms are implemented and validated on the physical ATRON modules. We believe that the algorithms may have applications beyond the ATRON modules, since similar challenges exist for other similar modular robots and in the context of mobile multi-robot systems.

II. Related Work

Hardware prototypes of self-reconfigurable robotic systems [3], [7], [9], [13], [14], [17], [21] consist of dozens of centimeter-scale modules. From such modules robots with various capabilities have been assembled, e.g. robots able to move [8], [19], [20] and self-reconfigure [5], [11].

These systems all use neighbor-to-neighbor communication to coordinate the modules (in addition, some systems such as the MTRAN system also have a global bus system). Since the modules can self-reconfigure, the communication system must automatically detect the appearance or removal of neighbors. Distributed control strategies can be used to ensure scalability [2], [10], [15]. In such distributed systems, the use of global IDs should be avoided to allow any module to replace any other module in the system (e.g. replace a broken module).

An adaptive communication protocol for the self-reconfigurable robot CONRO has been presented by Shen et al. [18]. This protocol can detect changes in topology, due to self-reconfiguration. As in this work, the detection is performed using a special detection message, however, it assumes a reliable communication and no crosstalk. In this paper, we present a practical implementation of automatic neighbor detection and crosstalk elimination as required by the ATRON hardware.

III. ATRON Self-Reconfigurable Modules

The ATRON self-reconfigurable robotic system [7] is a homogeneous modular system. This means that all modules are identical in both hardware and software - functional differentiation can be achieved using roles. Modules can be assembled into a variety of robots: Robots for locomotion (like snakes, cars, and walkers), robots for manipulation (like

ROBOCOMM 2007, 14th – 16th Oct 2007, Athens, Greece.

ISBN 978-963-9799-08-0
DOI 10.4108/ICST.ROBOCOMM2007.2183

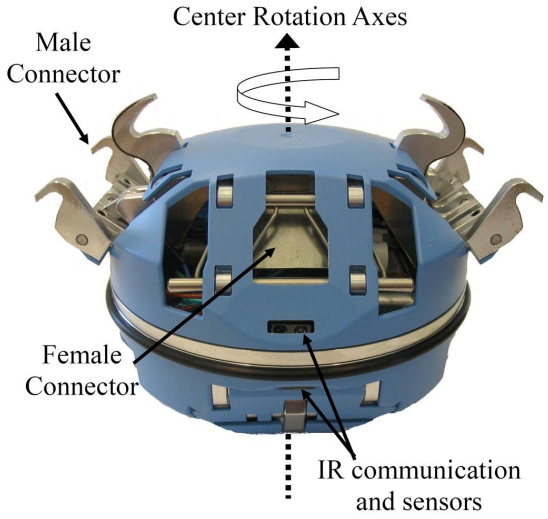


Fig. 1. A single ATRON module: on the top hemisphere the two male connectors are extended, on the bottom hemisphere they are contracted.

small robot arms) or robots that achieve some functionality from their physical shape, such as structural support [1], [4]. By self-reconfiguring, modules can change the shape of the robot, for example from a car to a snake and then to a walker.

An ATRON module has a spherical appearance composed of two hemispheres, which can actively be rotated relative to each other. On each hemisphere, a module has two actuated male connectors and two passive female connectors. In the HYDRA project [16] we have manufactured 100 ATRON modules, a single module is shown in Figure 1.

Rotation around the center axes is, for self-reconfiguration, always done in 90-degree steps. This moves a module, connected to the rotating module, from one lattice position to another. One full 360-degree rotation takes about 6 seconds, without the load from other modules. Encoders sense the rotation of the center axes. Male connectors are actuated and shaped as three hooks, which grasp on to passive female connector bars. A connection or a disconnection takes about two seconds. Located next to each connector are an infrared transmitter and receiver, which allow modules to communicate with neighbor modules and sense distance to nearby objects, more on this in Section IV. Connector's positions and orientation are such that the ATRON modules sit in a global surface-centered cubic lattice structure. Furthermore, each module is equipped with tilt sensors that allow the module to know its orientation relative to the direction of gravity.

IV. ATRON Communication System

The ATRON communication system must allow two neighbor modules reliably to exchange data packages with each other. Such local (neighbor-to-neighbor) communication scales well with the number of modules since the load on individual communication channels is constant, if well-designed distributed control strategies are used. In contrast global communication, such as wireless, would have scaling problems since the same medium is use by all the modules [22].

The ATRON communication system is implemented using a infrared receiver and transmitter for each connector position, see Figure 1. Infrared is preferred over wired communication since this allows two modules to communicate even if they are not physically connected.

However, crosstalk occurs because the physical spherical design of the ATRON modules did not allow the IR communication channels to be shielded off. This cause the transmitted IR light to be reflected off the metallic surfaces of the modules which then reach other non-neighbor modules as illustrated on Figure 2. This *crosstalk* communication is highly undesired but has proven extremely hard to remove with solutions such as: non-reflexive film, IR caps for making light more diffuse, lower transmission power, etc. Some of these solutions have reduced the crosstalk problem to some extend; however, no solution has come anywhere close to removing the problem entirely.

A number of experiments have been performed on the unmodified hardware to measure the communication quality between different (neighbor and non-neighbor) modules as shown in Figure 2. This is the only configuration where crosstalk occurs in the ATRON system. Because of the ATRON lattice configuration, any communication channel will be arranged in such a four-module loop. The measured results showed a perfect communication (no noise or byte loss) between both M1 and M2 and between M1 and M3. Note, that communication between M1 and M3 is crosstalk and therefore undesirable. We found no communication between M1 and M4. However, we believe that it can occur in some rare cases. In addition, we cannot rule out that small uncertainties in the mounting of the IR transmitter and receiver on the PCBs, which sometimes will make non-neighbor modules communicate even better than two neighbor modules.

V. Design Goals for Communication in Self-Reconfigurable Robots

The overall design goal of this communication protocol is to provide secure peer-to-peer communication between neighbor ATRON modules and automatically detect if a neighbor module is present on a given IR channel. However, because of self-reconfiguration (neighbors can appear/disappear) and crosstalk (IR can be reflected) a standard communication protocol cannot be used. Here, we summarize the design goals of a communication system for self-reconfigurable robots.

Scalable: State-of-the art self-reconfigurable robots consist of dozens of modules. We are, however, concerned with robots consisting of hundreds, thousands and eventually billions of microscopic size modules. To ensure scalability, the communication protocol must utilize distributed secure peer-to-peer communication between neighbor modules. Further, modules are homogenous in hardware and should be the same in software, again to ensure scalability. Thus, the modules do not have globally unique IDs.

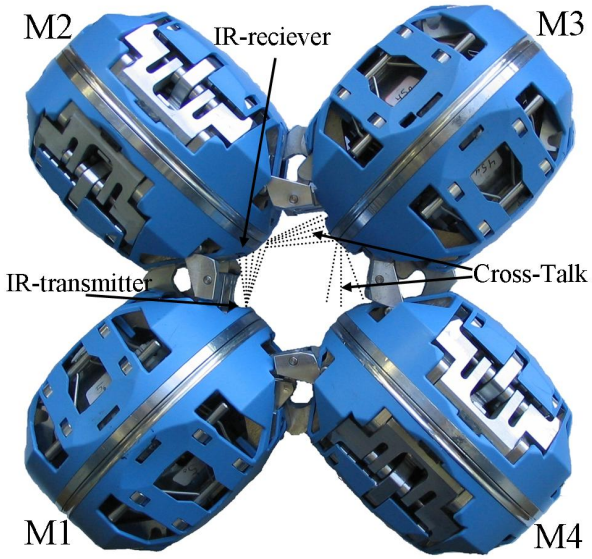


Fig. 2. Crosstalk between 4 ATRON modules. One module, M1, sends messages through the IR transmitters that are received by M2. However, the infrared light is also reflected by mechanical parts of M2 causing M3 also to receive the messages from M1 (crosstalk). We did not observe any crosstalk from M1 to M4, but believe it to be feasible that it occur in rare cases.

Robust: Since dealing with large numbers of modules, the communication protocol must be robust to module failures, like resetting or dead modules. Further, it must be robust to self-reconfiguration, crosstalk, noise etc.

Self-organizing: Changes in the network topology due to self-reconfiguration must be detected and dealt with locally while ensuring secure peer-to-peer communication with neighbor modules. Simultaneously, the system must adapt to changes in the communication load and filter out crosstalk messages. These adaptations must be based on local available information only.

Minimal: The ATRON and self-reconfigurable robots in general are small embedded system with limited resources. The communication protocol must therefore have a minimum of memory, bandwidth and computational overhead.

VI. Basic Communication Protocol

For comparison and as a foundation for further expansions a simple, custom build, communication protocol has been implemented to provide peer-to-peer communication between two neighbor modules. This basic protocol assumes no crosstalk and a prior knowledge of the local network topology (which channels has neighbors).

Each ATRON module contains two micro-controllers, one on each hemisphere. The two micro-controllers use RS485 to communicate with each other through a center slip-ring. Each micro-controller manages the communication of the four IR channels on its hemisphere by multiplexing a single UART. The micro-controller can listen for IR activity on all four channels at the same time but can only receive and transmit data on one channel at a time.

The basic communication protocol therefore manages four channels at a time. It can be in one of three states: listening (all channels), sending (one channel) or receiving (one channel). When listening, the protocol will as soon as some IR activity is detected switch to that channel and start to receive. Errors in data packages are detected using a cyclic redundancy check (CRC). Correctly received data packages will be acknowledged (single byte) by the receiver. Non-acknowledged packages will timeout and be resent. The packages are unnumbered, so if an acknowledge byte is lost a package may be sent and received several times. The process of switching between the listening and sending state is randomized, with some back-off if package collisions occurs in the communication (e.g. data error or receiving while sending).

In the following subsections, we will present extensions to this basic communication protocol that enables it to automatically detect neighbors and eliminate crosstalk from non-neighbor modules.

VII. Extended Communication Protocol

In this section, we present a strategy for dealing with self-reconfiguration and crosstalk in a simple, computationally cheap and completely distributed way, which provide the application layer with transparent functionality such as safe sending of a package to a neighbor module and information if a module is present on a given channel. We extend the basic communication protocol, described in Section VI, with these algorithmic strategies.

A. State of Module, Channel and Neighbor Channels

First we expand the basic communication protocol to include some state information about the module and any detected modules within communication range. A module has a locally unique ID, id_{my}, which is used by the communication protocol (Section VII-B describes how local uniqueness of the ID is ensured). In addition, each of the eight communication channels has a state. Hence, the module's communication state is given by its ID and the state of its eight channels:

$$M = \{id_{my}, C_0,\ C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ C_7\}$$

The states of the channels are independent and we can therefore split their states into two, where two protocols manage four channels each on different hemispheres. Only the id_{my} state must be shared between the two protocols.

Further, the state of an ATRON communication channel is given by the state of up to three neighbor channels (due to crosstalk) which can communicate with this channel i. Only one of those neighbors can be its immediate neighbor, others may be sending data to that channel through crosstalk. The states are identified using the neighbor channel's IDs. The channel state also include a state variable, t_{comp}, which is used to compensate the neighbor channel's *strength* for the communication load on that channel:

$$C_i = \{t_{comp},\ S_{id1},\ S_{id2},\ S_{id3}\}$$

Algorithm 1 Receive a *ping* byte on channel i

Require: id_{rec} read from ping byte
  if $S_{id_{rec}} \notin C_i$ **then**
    if C_i is full **then**
      delete $S \in C_i$ with $min(S.strength)$
    end if
    construct $S_{id_{rec}}$ from id_{rec}
    add $S_{id_{rec}}$ to C_i
  end if
  $S_{id_{rec}}.pingCount = S_{id_{rec}}.pingCount + 1$

In general, the state of a channel must include the state of as many neighbor channels as it can to communicate with (inclusive crosstalk).

Finally, the local state of a neighbor channel, i, is given by its ID and two variables (*strength* and *pingCount*) used to measure the communication quality with that neighbor channel:

$S_i = \{id,\ strength,\ pingCount\}$

The use of these states should become clear in the following sections.

B. Automatic Neighbor Detection and Locally Unique IDs

In order to facilitate automatic discovery of neighbor modules a special *ping* byte is sent at random time, with on average one ping per $\Delta t = 20ms$ on each communication channel. The ping byte is encoded as follows:

$$PING = \overbrace{01}^{header}\ \overbrace{\times}^{gender}\ \overbrace{\times\times\times\times\times}^{id}$$

Every module is assumed to have a locally unique ID in the range 1-31 (0 is reserved as a no ID). The gender bit indicates if the sending connector is male of female - such information is useful for the application layer. Below we explain the approach used in detail.

Receive a ping: We can expect to receive a ping byte every $20ms$ from neighbor modules within communication range. Every time a *ping* is received on a channel i, from a neighbor with ID, id_{rec}, the counter, $S_{id_{rec}}.pingCount$, is incremented. If the *ping* was received from an unknown neighbor we construct its corresponding state $S_{id_{rec}}$, potentially removing the neighbor state which has the lowest *strength*. See Algorithm 1.

Update channel state: Approximately every $\Delta t = 20ms$ the state of a channel i is updated, see Algorithm 2. For simplicity this is done in conjunction with the sending of a *ping* on the same channel i. When updating the channel's state we first compute the amount of time spend, since last update, listening on that channel (for *pings* and packages), e.g. $\Delta t_{listen} = 17ms$, and the total amount of time since last update, e.g. $\Delta t_{total} = 23ms$. From this we update the channel's state variable t_{comp} as a moving average of the percentage of time spend listening, e.g. 73% (time not spend sending or receiving packages).

Then, the states of the known neighbor channels are updated. The communication *strength* to a neighbor module

Algorithm 2 Update state, C_i, of channel i

Require: Δt_{total} and Δt_{listen} since last update
  $t_{comp} \Leftarrow \alpha_1 \cdot \Delta t_{listen}/\Delta t_{total} \cdot + (1-\alpha_1) \cdot t_{comp}$
  for all $S_j \in C_i$ **do**
    $S_j.strength \Leftarrow \alpha_2 \cdot S_j.pingCount/t_{comp} + (1-\alpha_2) \cdot S_j.strength$
    $S_j.pingCount \Leftarrow 0$
    if $S_j.strength < \varepsilon_1$ **then**
      remove S_j from C_i
    end if
    if $S_j.id = id_{my}$ and $S_j.strength > \varepsilon_3$ **then**
      $id_{my} = newRandomID()$
    end if
  end for

is updated as a moving average of the number of *pings* received from that neighbor (typical 0, 1 or 2). However, since the channel has not spend all its time listening it may have missed some *pings*, this problem increase as the communication load increase. Hence, we compensate the *strength* based on the average percent listening time, t_{comp}, as shown in Algorithm 2. After updating the *strength*, the *pingCount* state variable is reset. If the *strength* of a neighbor channel falls below a threshold, ε_1, the neighbor is removed. In the implementation on the physical ATRON modules, the floating-point moving average is replaced with a fixed-point version to minimize computational overhead.

Several parameters has to be selected in Algorithm 2. The choice of α_1 should match the speed at which the load of the system can change. Likewise the choice of α_2 should match the speed at which the system self-reconfigures. For the ATRON communication load changes faster than the system should be able to detect new neighbors, that is why we select $\alpha_1 = 1/16$ and $\alpha_2 = 1/32$. Which means that a neighbor modules will be detected in less than a second, see Section VIII-A. Likewise, we select $\varepsilon_1 = 0.022$ which implies that one 'noise' ping will be removed after 10 updates or approximately $200ms$.

Locally unique IDs: In Algorithm 2, on a given channel i, if a neighbor has the same as the ID as the updating module and its *strength* is above a threshold ε_3 then the updating module will randomly select a new ID. Dependent on the number of different IDs used and average numbers of neighbors this method will very fast converge so that the modules locally but not globally have unique IDs. This strategy is a distributed one-hop version of the algorithm presented by Zhou et al. [23]. The communication system will adapt to the change of an ID the same as if it were a self-reconfiguration. The effects of changing ID on a module might result in that neighbor modules 'forget' the existence of that module, but only for a very short time and most likely not a all (dependent on the choice of ε_2 in Algorithm 3). For ATRON $\varepsilon_3 = 1/8$, which is a tradeoff between fast update to ensure local uniqueness and not being too sensitive to noise.

Neighbor Detection: The *strength* state variable allows the communication protocol to provide the application layer

Algorithm 3 Is neighbor module on channel C_i?

```
for all S_j ∈ C_i do
  if S_j.strength > ε_2 then
    return true
  end if
  return false
end for
```

with information about whether or not the module have a neighbor on a given channel, see Algorithm 3. The algorithm simply checks if the channel has any neighbor channels with a strength above a threshold ε_2. For ATRON we select $\varepsilon_2 = 3/8$ which is below 0.5 since this will reduce the probability of undesirable 'neighbor knowledge loss' by the neighbor modules if the local ID is changed. This is because that *strength* of new ID will rise above $3/8$ faster than *strength* of old ID will drop from 1 to $3/8$. Note that false-positive does not occur since crosstalk does not occur if the channel does not in fact have a true (physical) neighbor to reflect the signal from.

The use of ping bytes as a mean of detecting, removing and estimating the communication strength to a neighbor module allows the structure of modules to be self-reconfigured while detecting local changes in the topology of the communication network.

C. Cross-Talk Elimination

To the package, about to be send, the algorithm attach its own locally unique ID and the ID's of N of the channel's neighbors (N=2 for ATRON). These extra bytes send are the only communication overhead involved in cross-talk elimination, see Algorithm 4. The algorithm is designed so that it only accepts (and acknowledge) packages, which it is confident, is send from an immediate neighbor module. Packages from immediate neighbors may initially be rejected, but will eventually be accepted when the states of the neighbor channels have adjusted. This is less serious than wrongly accepting a crosstalk package, since this package then would never reach its indented destination and could cause serious confusion at the receiving end.

Sending a package: When sending a package the algorithm first ensures that it has a neighbor on that channel (by comparing the neighbors strength with $\varepsilon_2 = 3/8$). To the package the algorithm will attach its module ID as well a number, N = 2, of known neighbor IDs. The IDs attached is the potential immediate neighbor channels on that channel, the neighbors are selected based on their *strength*. These extra bytes are the only communication overhead involved, see Algorithm 4. For ATRON the three IDs are encoded as two bytes, giving an overhead of two bytes per package.

Receiving a package: Crosstalk elimination is performed as shown in Algorithm 5. If the two communicating channels do not agree that they are neighbors or if the communication strength is too low ($strength < \varepsilon_2$) the package is not accepted (first and second condition in Algorithm 5). Further, the channels must have *no common neighbors* (third condition in Algorithm 5). This is due to the fact that a module (M1) and its crosstalk neighbor (M3) will also have a common neighbor in its immediate physical neighbor (M2), see Figure 2. The intuition is given in Figure 3. The algorithm assumes that the amount of crosstalk received from a module two hops away (M4) is low (in fact we have recoded no such crosstalk, but consider it a theoretical possibility). This simple algorithm rejects any received crosstalk packages. The dynamics of the *strength* ensures that no crosstalk occur, even when a module is briefly reset, change its ID or self-reconfigures. If a physical neighbor module is turned off or broken, crosstalk can happen since it may reflect the IR signal between non-neighbor modules.

Algorithm 4 Send package, P, through channel C_i

Ensure: $\exists S \in C_i : S.strength > \varepsilon_2$

```
add id_my to P
for k = 0 to N do
  select id with max(S_id.strength), id ∉ P ∧ S_id ∈ C_i
  add id to P
end for
send P on hardware
if P is not acknowledge or timeout then
  schedule resend of P
end if
```

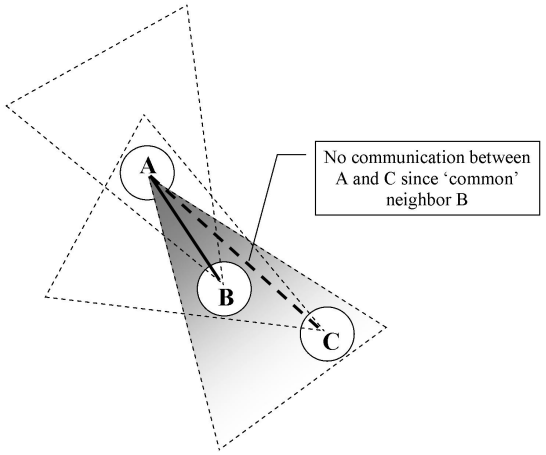


Fig. 3. Intuition of crosstalk elimination. Crosstalk is eliminated by comparing the neighbors of the sending channel with the neighbors of the receiving channel. If the channels have a common neighbor, the communication is crosstalk and ignored. This strategy does not work in general for any robot, but does work in general for the ATRON because the modules sit in a lattice. The figure shows three robots (A, B and C) with directional communication channels (triangles). Robot A and B has no common neighbor, why they can communicate. Robot A and C do have the common neighbor B which is why they cannot communicate.

VIII. Experiments

The proposed algorithms were first tested in a simulation of ATRON modules containing dozens of modules. Here, we saw that the reactive assignment of local IDs quickly stabilized, neighbors were detected, and that crosstalk was eliminated. Here, we present experimental validation on the physical ATRON modules and on a simple model of two

Algorithm 5 Receive package, P, through channel C_i

Require: id_{rec} and ids_{rec} read from P
  if $S_{id_{rec}} \notin C_i$ or $S_{id_{rec}}.strength < \varepsilon_2$ **then**
    return // package is not accepted
  end if
  if $id_{my} \notin ids_{rec}$ **then**
    return // package is not accepted
  end if
  if $(ids_{rec} \cup S.id \in C_i) \neq \emptyset$ **then**
    return // package is not accepted
  end if
  acknowledge and process message // package accepted

communication channels.

A. Neighbor Detection

Here, we validate the proposed neighbor detection strategy with communication load compensation. As explained in Section VII-B and Algorithm 1 and 2. Every module emits a *ping* every 20*ms* which is received by neighbor modules if they are listening. Neighbor detection is achieved by using two moving averages in combination, one for load compensation and one estimating the for communication $strength$.

In Figure 4(a) and 4(b) results obtained from simulation of a model of two communication channels are shown. A receiving channel will receive a percentage of the *pings* sent by another channel (percentage given by the communicating load). The model assumes a perfect load estimation and noise free communication between the two channels. The parameters of the system are otherwise the same as for the physical ATRON modules. As can be seen from Figure 4(a) an uncompensated version of the neighbor detection system fails to detect the neighbor under high loads. This problem is reduced with the compensated version, see Figure 4(b).

The theoretical average time to detect a neighbor is always 600*ms* independent on the communication load (for the given parameters). The time variation is, however, dependent on the load (percent time not spend listening). For 25% load the expected time to detect a neighbor with one standard deviation range from 540*ms* to 660*ms* while it at 75% load range from 460*ms* to 840*ms*. On the physical ATRON modules, the time for one module to detect another has also been measured under various realistic loads. In 20 trials under a load of approximately 5% we found that the average time to detect a neighbor were 484*ms* with a max time of 780*ms* and min time of 372*ms*. The reason that the physical system is somewhat faster than the theoretical model is due to the dynamics of the communication protocol which is likely to shift to another task (than listening) just after receiving a *ping*.

B. Crosstalk Elimination

In the following experiment we compare three communication protocols for their ability to eliminate crosstalk in the ATRON system. The first is a basic protocol which makes no attempt to remove crosstalk, as explained in Section VI. The second protocol uses reactive local unique ids, neighbor detection based on pings but do not uses the 'neighbor' information to eliminate crosstalk (first condition in Algorithm 5). Third protocol is the full implementation as proposed in this paper, it is the same as the second protocol but also make use of the neighbor information to eliminate crosstalk (second and third condition in Algorithm 5).

The setup is shown in Figure 2, M1 sends out 100 packages on a given channel with payload of 1 byte, total package size is 7 bytes inclusive header (3 bytes) and CRC check (2 bytes). For each protocol 20 trials were performed, five trials on each of four permutations of modules. The baud rate were set low (9600*bps*) to increase the probability of collisions and thereby noise (to stress test the protocol). In Table I we report the number of packages received by M2, M3 and M4. We observe that neither the basic or strength based protocols works very well, both accept a lot of crosstalk. The full implementation performs much better, no crosstalk were observed and just 5 percent of the packages are received twice due to a collision between an acknowledge byte and a ping. In none of the trials do M4 receive any packages, but it does add to the IR noise and communication load to the system since it is pinging the other modules. In summary, the proposed algorithm eliminates the crosstalk between ATRON modules.

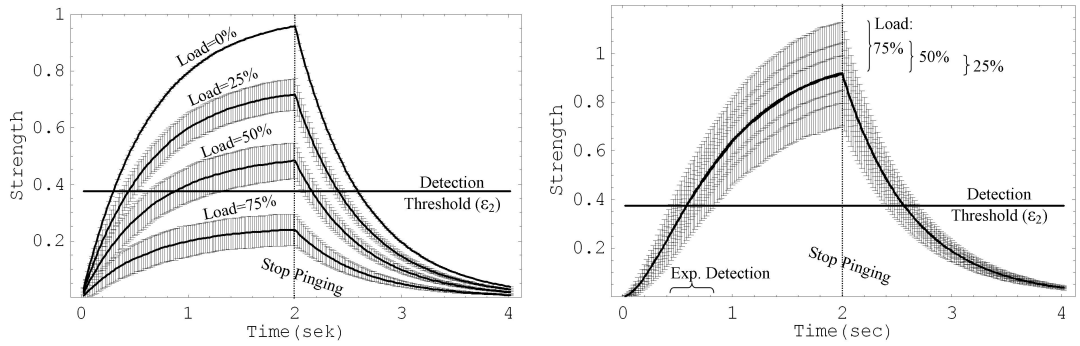
C. Self-reconfiguration

The communication system has been applied in the context of self-reconfiguration of several ATRON modules. For this purpose the baud rate was increased to 38.4*kbps* which reduces the chance of communication collision and therefore resend of packages. At this baud rate we have found that it takes on average approximately 30*ms* for a one byte package to be send from a controller program on one module to a controller program on a neighbor module (the package passes through two RS485 as well as the IR channel in this case). For a 5-byte package, the time has increased to 35*ms*. The protocol has been tested with a payload of up to 40 bytes. Further, the communication protocol has been tested in a self-reconfiguration scenario with four modules, where it successfully detected appearing neighbor modules dozens of times and eliminated crosstalk hundreds of times.

IX. Future Work

Relying on the current hardware platform, several improvements could be made to the ATRON communication protocol. For example the throughput of the system could be improved and noise reduced by replacing the current randomized strategy with a self-organizing synchronization mechanism, such as those used in the context of sensor networks [6], [12].

Further, the proposed strategy for neighbor detection and crosstalk elimination may have applications beyond self-reconfigurable robots. Especially in the context of mobile multi-robot systems with directional communication channels, the proposed crosstalk elimination algorithm may be



(a) Uncompensated (b) Compensated

Fig. 4. Theoretical *strength* of a neighbor channel as a function of the load on the listening channel (load is percent time not spend listening). (a) No load compensation, note that under high loads neighbors will not be detected. (b) Load compensation is used (as in Algorithm 2) here neighbors will be detected in spite of high loads. The time to detect a neighbor is 600*ms*. Error bars indicate one standard deviation.

	Basic (no check) Mean (Std. Dev.)	Strength Check Mean (Std. Dev.)	Full Check Mean (Std. Dev.)
Module 2	103%(32%)	62%(16%)	105%(3%)
Module 3	120%(9%)	63%(16%)	0%(0%)
Module 4	0%(0%)	0%(0%)	0%(0%)

TABLE I

CROSSTALK CHARACTERISTICS OF THREE VERSIONS OF THE COMMUNICATION PROTOCOL. PERCENTAGE OF PACKAGES RECEIVED IS REPORTED. RESENT PACKAGES, DUE TO COLLISIONS, RESULTS IN HIGHER RECEIVE RATES THAN 100%. NOTICE THAT THE FULL IMPLEMENTATION COMPLETELY ELIMINATES CROSSTALK.

used as one of several means to ensure nearest neighbor communication. In such a scenario the degree to which the algorithm can eliminate crosstalk is still an open question, it is, however, a function of many parameters related to the physical aspects of the communication channels.

X. CONCLUSION

This paper proposed a distributed and self-organizing communication strategy for dealing with self-reconfiguration and crosstalk, which is based on locally unique ID which are reactively assigned, ping messages send with a fixed time interval and the exchange of known neighbor IDs when sending packages. The communication system was validated on the physical ATRON modules. Result showed that the communication system were able to detect/forget neighbor modules in less than a second independent on communication load and were able to eliminate crosstalk between modules completely. In conclusion the system is simple to implement and sufficient for ensuring reliable communication between neighbor modules and may be general enough to be used on other self-reconfigurable robots or mobile robots.

ACKNOWLEDGMENT

This work is partly funded by the EU FET project HYDRA, the Self-assembling Robotic Artefacts project sponsored by the Danish Technical Science Council, and the Morphing Production Line project financed by the Danish Agency for Science, Technology and Innovation. C. Ryberg, D. Kyrping, E. Østergaard, J. Hallam, J. Mikkelsen, K. Støy, K. Kassow, L. Dalgaard, L. Paramonov, R. Beck have contributed to the ATRON.

REFERENCES

[1] D. Brandt, D. J. Christensen, and H. H. Lund. ATRON robots: Versatility from self-reconfigurable modules. In *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 2254–2260, Harbin, China, August 2007.

[2] Z. Butler, K. Kotay, . Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[3] A. Castano, W.-M. Shen, and P. Will. Conro: Towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, 2000.

[4] D. J. Christensen. Evolution of shape-changing and self-repairing control for the ATRON self-reconfigurable robot. In *Proceedings of the IEEE Int. Conference on Robotics and Automation(ICRA)*, May 2006.

[5] D J. Christensen. Experiments on fault-tolerant self-reconfiguration and emergent self-repair. In *Proceedings of Symposium on Artificial Life part of the IEEE Symposium Series on Computational Intelligence*, Honolulu, Hawaii, April 2007.

[6] J. Degesys, I. Rose, A. Patel, and R. Nagpal. DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 11–20, Massachusetts, USA, April 2007.

[7] M. W. Jørgensen, E. H. Østergaard, and H. H. Lund. Modular ATRON: Modules for a self-reconfigurable robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2068–2073, 2004.

[8] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. Distributed adaptive locomotion by a modular robotic system, M-TRAN II. *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2004)*, pages 2370–2377, 2004.

[9] K. Kotay, D. Rus, M. Vona, , and C. McGray. The selfreconfiguring robotic molecule: Design and control algorithms. In *Robotics: The Algorithmic Perspective*. AK Peters, 1998.

[10] J. Kubica, A. Casal, and T. Hogg. Complex behaviors from local rules in modular self-reconfigurable robots. In *Proceedings of IEEE*

International Conference on Robotics and Automation (ICRA), pages 360–367, Seoul, Korea, May 2001.

[11] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2454–2459, 2003.

[12] D. Lucarelli and I-J. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*, pages 62–68, New York, NY, USA, 2004. ACM Press.

[13] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-d self-reconfigurable structure. In *Proceedings, IEEE Int. Conf. on Robotics & Automation (ICRA'98)*, pages 432–439, Leuven, Belgium, 1998.

[14] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.

[15] E. H. Østergaard. *Distributed Control of the ATRON Self-Reconfigurable Robot*. PhD thesis, Maersk McKinney Moller Institute for Production Technology, University of Southern Denmark, Odense, Denmark, November 2004.

[16] E. H. Østergaard, D. J. Christensen, P. E. Hotz, T. Taylor, P. Ottery, and H. H. Lund. Hydra: From cellular biology to shape-changing artefacts. In *Proceesings of International Conference on Artificial Neural Networks (ICANN)*, pages 275–281, 2005.

[17] W.-M. Shen, M. Krivokon, M. Rubenstein, C. H. Chiu, J. E., and J. B. Venkatesh. Multimode locomotion via superbot reconfigurable robots. *Autonomous Robots*, 20(2):165–177, 2006.

[18] W.-M. Shen, B. Salemi, and P. Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, 18:700–712, 2002.

[19] K. Støy, W.-M. Shen, and P. Will. Using role based control to produce locomotion in chain-type self-reconfigurable robots. *IEEE Transactions on Mechatronics*, 7(4):410–417, 2002.

[20] M. Yim. New locomotion gaits. In *Proceedings, International Conference on Robotics & Automation (ICRA'94)*, pages 2508 –2514, San Diego, California, USA, 1994.

[21] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of IEEE International Conference on Robotics & Automation (ICRA)*, pages 514–520, San Francisco, CA, USA, 2000.

[22] E. Yoshida, T. Arai, M. Yamamoto, J. Ota, and D. Kurabayashi. Evaluating the efficiency of local and global communication in distributed mobile robotic systems. In *IEEE International Conference on Intelligent Robots and Systems (IROS96)*, page 16611666, Osaka, Japan, November 1996.

[23] H. Zhou, M. W. Mutka, and L. Ni. Reactive id assignment for sensor networks. *International Journal of Wireless Information Networks*, 13(4):317–328, 2006.