Pruning Local Schedules for Efficient Swarm Communication

Felix Schill, Uwe Zimmer

Australian National University
Research School for Information Science and Engineering
and the Faculty for Engineering and Information Technology
Autonomous Underwater Robotics Research Group
Canberra, ACT 0200, Australia
felix.schill@rsise.anu.edu.au | uwe.zimmer@ieee.org

***Abstract* – Reliable wireless communication underwater is a precondition for swarming technologies. This paper discusses a time division multiple access (TDMA) algorithm suitable for dynamic multi-hop wireless networks, which offers quick all-to-all information exchange (Omnicast), dense local schedules and predictable latencies. The algorithm is based on an earlier algorithm published by the authors in [7]. This paper presents an improved and simplified algorithm to calculate the local schedules, and uses a new mapping function for logical time slots to actual time slots, which balances sending frequencies between nodes. An extension of this algorithm is then presented which employs a technique to reduce the average degree of the connection graph as seen by the scheduling algorithm. It is explained how this reduction of degree can be achieved without causing communication collisions. The results of experiments performed in a real time simulation show the performance of the algorithm, and the performance gain achieved by local reduction of the degree.**

I. INTRODUCTION

The importance of distributed sensing and swarming vehicles has been increasingly recognized by the underwater community and ocean sciences. The Serafina submersible robot (figure 1) has been developed as a platform for swarming and distributed sensing applications. The small size (50 cm), low weight (5 kg) and low cost allows easy deployment of swarms of dozens of submersibles with affordable effort. One of the most important problems that had to be solved apart from the miniaturisation is the communication between members of the swarm. For reasons of scalability to large swarms and also to keep power consumptions low, a digital longwave radio transceiver has been developed, which can transmit data over up to 10 m range with up to 8192 bit/sec. The transmitter uses differential binary phase shift keying on a fixed carrier frequency [8]. The use of electromagnetic channels instead of acoustics has now been shown to be beneficial in [1]. This is especially so due to the exponential attenuation of radio waves in water, which keeps interference very localised.

As opposed to the majority of other wireless networks, in which communication is sporadic and usually point-to-point, swarms require a communication system optimised for continuous communication between all members, and with quick local and global information exchange. The authors published the Omnicast problem in a previous paper [6] as well as a theoretical analysis based on a graph theoretical network model.

figure 1: Autonomous underwater platforms: Serafina

Due to the severe bandwidth limitations and real-time requirements in underwater swarms, time division multiple access (TDMA) is a good choice. TDMA scheduling algorithms are known in the literature [2, 4, 5], which are mostly tailored for sensor networks or applications with sporadic communication. A distributed algorithm adjusted to Omnicast communication in swarms is presented in [7].

Experiments with the phase-modulated longwave radio modules revealed that the graph-theoretical network model commonly assumed in the literature is too conservative [8]. In fact, if two or more transmitters send within the range of a receiver then the receiver will only observe a collision if the closest nodes have very similar incoming signal strength. Otherwise the receiver will reliably receive the message with the highest signal strength. This paper shows how this fact can be used to speed up both local and global information exchange by virtually decreasing the local degree of the connection graph as seen by the scheduling algorithm. This technique assumes that the signal strength can be measured by the receiver.

II. DISTRIBUTED DYNAMICAL OMNICAST ROUTING

The Distributed Dynamical Omnicast Routing (DDOR) algorithm has been published previously. For details on the algorithm refer to [7]. This paper describes a new, improved version of the DDOR algorithm, which is slightly simplified. The second part of this paper presents a variant of the algorithm in section III, which implements a form of spatial reuse of the channel by pruning schedules.

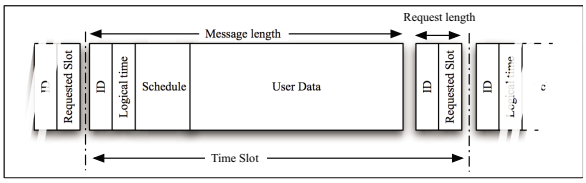

figure 2: Format of the data packet and the time slot

A. The Basics

DDOR is a distributed algorithm for TDMA scheduling in wireless multi-hop networks. It is computationally inexpensive and can be implemented on a small microcontroller. It is designed for continuous traffic and quick local and global information dissemination.

Communication nodes maintain a logical clock $L(t)$ which increments at the beginning of each time slot. The logical clocks are synchronised during the startup of the network by monitoring incoming messages. It can be assumed that the logical clocks $L(t)$ are equal and in sync up to sufficient accuracy on all participating nodes.

To allow for dense packing of schedules on one hand, but reconfigurability on the other hand, time slots are divided into packet slots, separated by short 2-byte request slots. Figure 2 shows the packet and time slot structure. Each packet contains the unique identifier of the sender, the sender's logical clock, and the current local schedule of the sender. A request packet contains the sender's ID and the number of the requested schedule slot.

The following data structures are maintained by each node:

- The visible neighbourhood N. This is a database of all nodes that are within range of the node. The node database is maintained according to received messages. It is assumed that a node which message has been received is within range. Nodes are removed from the list and the schedule if no messages from them were received over a certain amount of time. With every node in the list the node's local schedule is stored as received in the last message of this node.
- The local schedule s_i. Nodes try to establish local collision-free schedules. The schedule of each node consists of a number of schedule slots. A slot can be either empty, blocked, or used by a node present in the visible neighbourhood. The local schedule is recalculated with every received message from the most recent schedules received from all nodes in visible neighbourhood.

A node can be in one of three states: *Listening*, *Requesting*, and *Running*. The default is *Listening*. The node remains in the *Listening* state for a random number of time slots, while updating its local schedule and visible neighbourhood list from received messages. If the visible neighbourhood list is still empty after the time out (no messages received), the node adds itself to the first slot of its local schedule and transmits a message containing this schedule. Otherwise the node sends out a request for the first empty slot in the current schedule, and returns to *Listening* for a random time.

B. The DDOR algorithm

All nodes keep track of their neighbourhood (all nodes from which they recently received a message). Nodes also save the most recent schedules which they received from their neighbours. A neighboured node is marked as *established* if a complete message with schedule has been received from this node, and as *not established* if only a request has been received up to now (respectively if the last message from this node was a request).

If a messages from a neighboured node cannot be received for an extended period of time (at least two schedule rounds), the node is removed from the database of neighboured nodes. Removal of a node from the database also invokes the removal of that node from all locally stored schedules (the locally stored copies of schedules received from other neighboured nodes).

The algorithm consists of two tasks, the transmitter task and the receiver task. The two tasks communicate indirectly through the node database, which has to be implemented as a multi-tasking save entity (i.e. a protected object in Ada).

Receiver task

The receiver task takes care of incoming messages. It only becomes active if a message is received, and can therefore also be implemented as an interrupt handler. This is especially useful for implementations on microcontrollers, on which a distributed programming language might not be available. This is the description of the receiver task in pseudo code:

```
loop:
  Wait_For_New_Message;
  Message := Retrieve_Message;
  Synchronise_Clock (Message.Logical_Clock);

  if type of Message is 'Request' then
    if Message.Sender is in Node_Database,
      remove Message.Sender from Node_Database;
    end if;
    Clear requested slot from all schedules in Node_Database;
    Create new entry for Node_Database:
      Node.ID          := Message.Sender;
      Node.Schedule    := Empty_Schedule;
      Mark requested slot in Node.Schedule
                              as used by Message.Sender
      Node.Established  := false;
    Store new Entry 'Node' in Node_Database.
  else if type of Message is "Message" then
    update Node_Database with
      (Message.Sender, Message.Schedule, Established := True;)
  end if;
end loop;
```

Transmitter

The transmitter task is more complex. It is a periodic task, which becomes active at every beginning of a time slot or a request slot.

```
loop:
  Wait_For_Next_Time_Slot;
  Local_Schedule     := Recalculate_Local_Schedule (Own_Slot);
  Current_Time_Slot := Calculate_Active_Time_Slot;
  if This_Node is in Local_Schedule, then
    State  := Run;
  else if not State = Listen then
    State  := Listen;
    Set Listen_Time to random number of time slots
  end if;
  case State is
    Run:
      if Local_Schedule (Current_Time_Slot) = This_Node then
        if better slot with lower index available then
          with Probability of 33%:
            Prepare_Request for better slot;
            State := Request;
        end if;
        with Probability of 95%:
          Transmit_Message (Local_Schedule);
      end if;
    Listen:
      if Node_Database is empty, then initiate:
        Own_Slot := first empty schedule slot;
        Local_Schedule (Own_Slot) := This_Node;
        Transmit_Message (Local_Schedule);
      else
        if Listen_Time = 0, then
          Prepare Request for first empty slot in local schedule;
          State := Request;
          Set Listen_Time to random number of time slots
        else
          Decrement Listen_Time
        end if;
  end case;
  Wait_For_Next_Request_Slot;
  if State = Request then
    Clear this node from local Node_Database
                                and all locally stored schedules;
    Send out Request for prepared slot;
    Set Own_Slot to Requested slot
  end if;
end loop;
```

The two crucial functions called in the transmitter task are *Recalculate_Local_Schedule* and *Calculate_Active_Time_Slot*. The nodes in the local node database are denoted $n_j \in N$; each node in the local database has a locally stored schedule s_j (the most recently received schedule from each neighbour) with schedule slots $s_{i,j}$. Schedules of nodes from which no schedule has been received yet are empty, i.e. all slots of that schedule are marked as empty slots. The calculation of the local schedule is described here by the function $s_i(o)$:

$$s_i(o) = \begin{cases} b\text{: } \exists n_j \in N\text{:}(s_{i,j} \neq e \wedge s_{i,j} \neq b \wedge s_{i,j} \notin N) \\ j\text{: } j = min(A_i) \\ o\text{: } i = o \wedge \text{Confirm} \\ e\text{: otherwise} \end{cases} \quad (1)$$

with

$$A_i = \{k\text{:}(n_k \in N \wedge s_{i,k} = k)\} \quad (2)$$

$$\text{Confirm} = \exists n_j \in N\text{:}(s_{i,j} = o) \wedge \forall n_j \in N\text{:}(s_{i,j} = o \vee s_{i,j} = e) \quad (3)$$

It can be seen from the definition of $s_i(o)$ that collisions between competing nodes are resolved by favouring the node with the lowest index. This is an invariant which can be equally computed by all affected nodes. The "own" slot (the slot that a particular node requested last) is only assigned if at least one node in the 1-hop neighbourhood confirms this, and if the slot is otherwise unused.

The active time slot is calculated recursively from the current logical time t, and the current local schedule $s = s_1 \ldots s_l$. The schedule length l is assumed to be a power of 2.

$$\alpha\text{:}(N, S, N) \rightarrow N \quad (4)$$

$$(L(t), s_t, l) \rightarrow \alpha(L(t), s_t, l) \quad (5)$$

$$= \begin{cases} L(t) \bmod l + 1 & : s_{1+L(t) \bmod l} \neq e \\ \alpha(L(t), s_t, l/2) & : s_{1+L(t) \bmod l} = e \\ 1 & : l \leq 1 \end{cases}$$

The initial call of the recursive function uses the maximal schedule length (must be a power of 2) as a parameter. The described mapping function has the advantage that it does not return empty time slots if the first slot of the schedule is filled, thus increasing utilisation especially in the case of sparse schedules. Alternatively other time slot mapping functions can be used.

III. PRUNED DISTRIBUTED OMNICAST ROUTING (PDOR)

The algorithm described in the last section assumes a graph topological collision model (a node receives a message if and only if exactly one of its neighbours transmits). However, some real radio links (notably frequency modulated or phase modulated channels) have different characteristics [8]. Typically a node receives the message from a transmitting node, if the ratio of signal strength of that node over noise and interfering messages in the local vicinity is greater than a certain threshold. Due to the strong attenuation of radio waves over distance (especially for omnidirectional links), there is only a small region where a collision occurs (the receiving node cannot decode any of the received messages).

A problem of the DDOR algorithm described in the previous section is the increasing schedule length for networks with

high connectivity (high graph degree). Longer schedules have a negative impact on the performance. Roundtrip times locally and globally grow. An additional problem is that longer schedules take up large parts of the transmitted messages, wasting valuable bandwidth. If the geometric/radiometric collision model is applied, it is possible to make use of the fact that collisions actually occur less frequent than the graph topological model suggests. In principal it is possible to pack schedules more densely by re-issuing slots taken by distant nodes to closer nodes. Messages sent by closer nodes will 'overwrite' the messages by the more distant nodes sending in the same time slot. The perceived connectivity is quantitatively similar to a network of the same geometric layout, but with reduced transmission ranges. This technique is often referred to as *spatial reuse* [9].

The following algorithm implements spatial reuse on top of the DDOR algorithm. It requires the following assumptions:

- Symmetric links: If (in undisturbed conditions) node A can receive messages from node B, then node B can also receive messages from node A.
- Monotonically decaying signal strength over distance.
- A node receives whichever message send out in its local neighbourhood which is received with the strongest signal (a small 'collision zone' is acceptable, where no message is received if the n strongest messages have comparable signal strength. The collision zone is assumed to be small compared to the maximum range, or the distance between nodes).
- The signal strength can be measured (directly or indirectly).

The modified algorithm is largely identical to the original DDOR algorithm. The two differences are in the collision resolution in the scheduling function (1), and in the slot request mechanism during the *listen* state. The new scheduling function is defined here:

$$s_i(o) = \begin{cases} b\colon \exists n_j \in N' \colon (s_{i,j} \neq e \wedge s_{i,j} \neq b \wedge s_{i,j} \notin N') \\ j\colon\ j \in A_i \wedge \forall k \in A_i \setminus j \colon (\sigma(j) > \sigma(k)) \\ o\colon \exists n_j \in N' \colon s_{i,j} = o \\ e\colon \text{otherwise} \end{cases} \quad (6)$$

with

$$N' = \{(n_j \in N) \colon \exists k \colon (s_k(o) = j)\} \quad (7)$$

$$A_i = \{k \colon n_k \in N \wedge s_{i,k} = k\} \quad (8)$$

and whereas $\sigma(j)$ denotes the signal strength of the last message received from node $n_j \in N$.

The main differences are that collisions are now resolved based on signal strength instead of node index. Since every node has different signal strengths for their neighbours, this decision is not identical any more on different nodes. In DDOR, the decision who occupies a slot is coherent within a 2-hop neighbourhood around that node. In PDOR, the decision is localised. Nodes geometrically close to a node n_k will assign the slot to n_k, while other nodes within range or within a 2-hop neighbourhood may assign the same slot to other nodes that are received stronger. This effectively shrinks the virtual neighbourhood of nodes in dense areas.

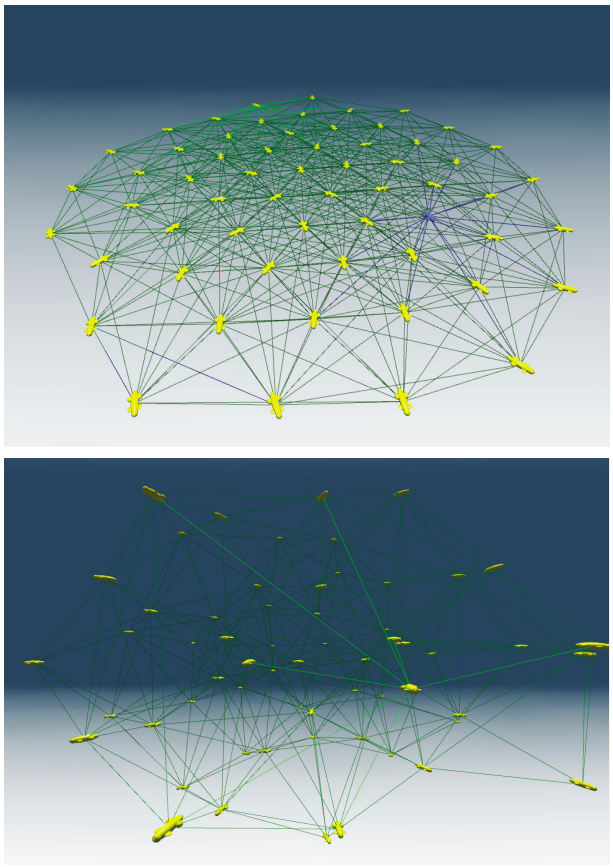

figure 3: Swarming simulation showing a 2D and a 3D configuration

Derived modifications take into account that only nodes that appear in the final local schedule are used to mark blocked slots. Also, a slot is only marked 'o' (for own use) if at least one neighbour that appears in the local schedule confirms this slot.

The second difference is in the request mechanism. As before, nodes apply for empty slots within the schedule. As an extension, if there are no available empty slots, a node may apply for a blocked slot (with a preference for blocked slots at the end of the schedule). If no blocked slots are available, a node requests the slot occupied by the node with the lowest locally measured signal strength.

IV. DISCUSSION

A theoretical analysis of the omnicast problem [6] revealed an upper bound of $2n - 2$ for networks with n nodes. Other upper bounds have been presented in [2, 3]. An upper bound for the DDOR algorithm can be given as $O(\gamma \cdot d/2)$, where γ is the maximum size of any 2-hop neighbourhood ($\gamma = \Delta G^2$):

The diameter of a 2-hop neighbourhood is at most 4. In a connected graph it is possible to find a chain of overlapping 2-hop neighbourhood subgraphs that minimally cover the

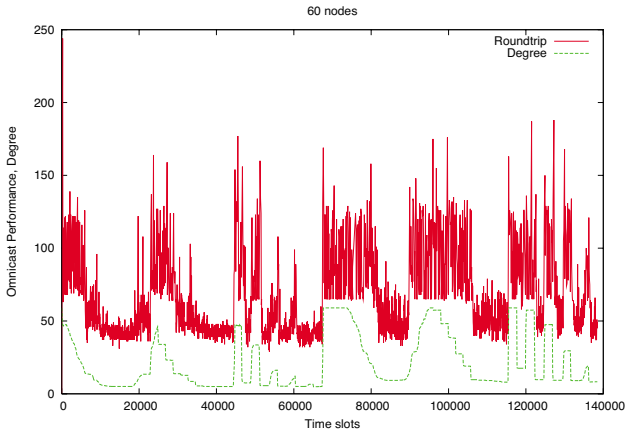

figure 4: A complete simulation run of DDOR with 60 Nodes.

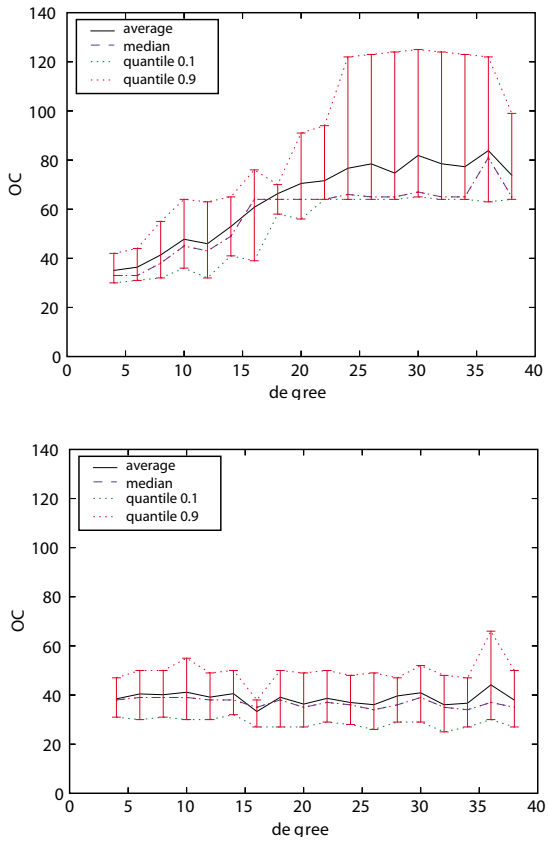

figure 5: Plot of Omnicast performance over graph degree, with 40 nodes, for DDOR (upper: schedule length 64 slots) and PDOR (lower: schedule length 16 slots).

diameter of the graph. This requires $d/4$ subgraphs. DDOR only considers nodes in a local schedule that are contained in the 2-hop neighbourhood of a node. It follows that executing a schedule once completely takes $O(\gamma)$ time slots. In the worst case, after a local schedule has been executed twice, all nodes within this 2-hop neighbourhood have locally solved omnicast, including the nodes that overlap with neighbouring subgraphs. This process has to be repeated at most $d(G)$ times to spread all information along the diameter. Since the diameter is the longest shortest path across the graph, this implies that omnicast can be solved in at most $O(\gamma \cdot d/2)$ time steps. This assumes that the maximum schedule length of DDOR can accommodate at least γ nodes. If that is the case, γ can be substituted by the maximum schedule length l.

Both γ and l are linked to the graph degree Δ. In the general case, an upper bound for γ is Δ^2. However, in network graphs that are derived from a two- or three-dimensional embedding, the size of a 2-hop neighbourhood is typically lower. The schedule length l has to be chosen accordingly. It is obvious that short schedules are preferable. This implies that omnicast can be solved quicker in graphs of lower degree.

PDOR is able to deal with 2-hop neighbourhoods which are larger than the schedule length, by virtually reducing the neighbourhood according to signal strength. It is therefore advisable to choose the schedule length as short as possible, while still preserving good connectivity for the given swarm configuration. PDOR requires the schedule length to be long enough to accommodate all nodes in the closest proximity, so that these nodes maintain good connectivity with the rest of the graph. In the case of homogenous 3-D configurations, nodes typically have 12 neighbours in close proximity, or 6 neighbours for 2-d configurations. A schedule length of 16 slots is therefore sufficient for most homogenous swarm configurations. A short schedule length will enforce pruning of schedules, which effectively lowers the upper bound for omnicast. A further advantage of PDOR is that it continuously updates the schedules according to the currently received signal strength of a node's neighbours. If a node moves throughout the network, the schedules are continuously updated, and only rarely a node loses its slot and has to re-apply. The next section goes into more detail of the actual performance of both algorithms.

C. Performance in Simulations

A simulation has been implemented in Ada in order to verify the theoretical findings. While it is not a complete physics simulation, the measured collision behaviour of the tested longwave radio modules has been included as accurately as possible. The simulated submarines are created as independent, encapsulated tasks. Sending and receiving of messages is performed by interaction with the simulation environment at the lowest level. All higher level software parts (including the scheduling algorithm) are not aware of the simulation, and can be run identically on appropriate hardware.

In order to test the communication system in varying scenarios, the simulated submarines obey simple force-based swarming rules, which have access to the measured distance to all submarines within sensing / communication range once a second. The swarming rules allow to change the average distance between submarines, which results in changing

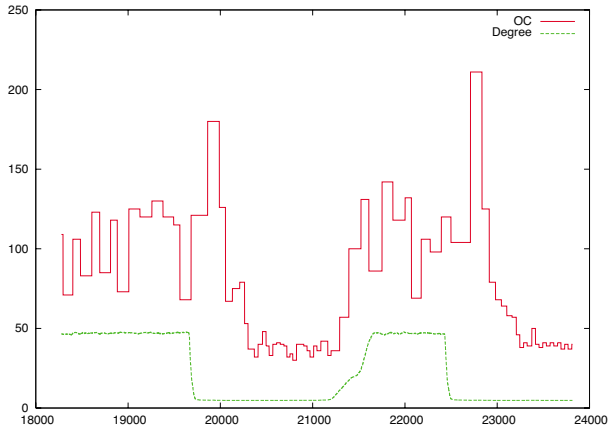

figure 6: Step response in a simulation with 60 Nodes (DDOR)

degrees of the network graph. During the simulation, the current graph degree is measured.

The measurement of the omnicast performance is achieved by a distributed counting algorithm. Every node maintains a counter. All counter values for all nodes are passed on with every message. A node sets its own counter to the maximum of all other counters, and increments its counter if all counter values are greater or equal to its own. The average duration between counter increments roughly corresponds to the duration of a performed omnicast. This duration is referred to as *OC* in the following plots.

The measurement protocol starts up the simulation and, while monitoring degree, diameter and omnicast performance, changes the average distance between nodes dynamically in various pattens. The scheduling algorithm and the swarming behaviour are never stopped for the whole duration of the experiment. A sample run can be seen in figure 4. The plot shows the changing degree and diameter of the graph as measured over time, together with the measured duration between omnicast counter increments. The simulation run starts with a 2-D configuration, and then switches to a 3-D configuration and repeats the changes in density (also refer to the images in figure 3). The change to 3-D occurs at approximately $t = 68000$.

As expected, the collision-avoiding DDOR algorithm performs well for low graph degrees, but loses performance for high degrees (figure 5 and also figure 6). This is due to the fact that for high connectivity only one node can send per time slot, or otherwise messages would collide. The PDOR algorithm performs equally well over the full range of network densities, in both 2-D and 3-D. The average performance is around 40 time steps, which coincides with the number of nodes. There is very little variation in the performance, as the 10% and 90% quantiles indicate (80% of all data points are between these lines). It should be noted that the PDOR algorithm achieved this with a schedule length of only 16, while the DDOR algorithm required 64 slots to be able to fit all nodes into the schedule during periods of high connectivity. This means that the message size overhead in PDOR can be greatly reduced.

Dynamic tests

Both algorithms have been subjected to dynamic changes in the network topology and geometry. One test involves the step response of the omnicast performance following a sharp change in density.

The DDOR algorithm responds so quickly that the performance can be maintained during the adaptation of the schedules to the new topology (figure 6). In the case of the network thinning out, there is a short period of reduced performance. This is because the schedule of the dense network is still active after the transition (i.e. one node at a time sends), but the network has a worse connectivity now. After a brief period of time, the change is detected and the schedule adapted, leading to a much shorter roundtrip time. The first adaptation is then followed by further optimizations.

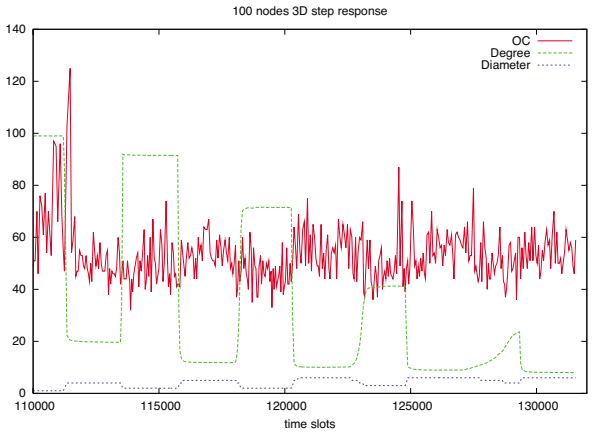

figure 7: Step response in a simulation with 100 Nodes in a 3D configuration (PDOR, schedule length 32 slots

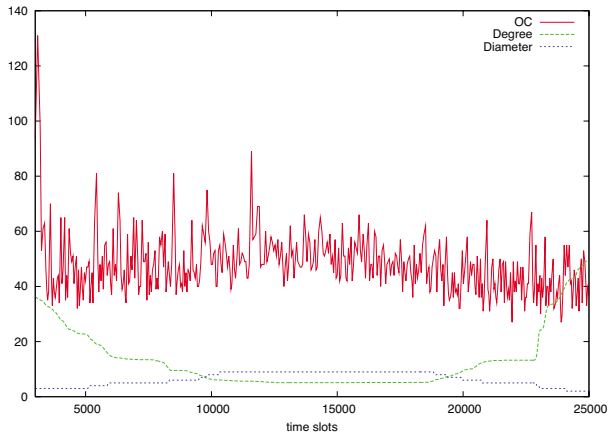

figure 8: Omnicast performance for 64 Nodes in a 2D configuration with slowly varying density (PDOR, schedule length 16 slots

In the opposite case of the network becoming denser, the change can be detected more quickly, because nodes now receive more messages from new neighbours, containing their schedules. Changes can be immediately made, and the performance immediately reaches the best possible performance for the new configuration. The variation in performance for fully connected networks can be explained by the stochastic omission of transmissions by nodes. This is necessary to detect messages that might be hidden by own transmissions.

Extended step response experiments have been carried out with the PDOR algorithm (figure 7) As can be seen, the performance of PDOR is largely unaffected by dramatic changes in the network topology. The average performance is around 50-60 time steps, which is below the number of nodes in the network. The result is not surprising, since the PDOR algorithm packs schedules more densely in the case of high network degree, which means that the schedules resemble the schedules achieved during periods of low degree. The geometry of the swarm mainly changes the scale, but only slowly the neighbourhood relations, so that the already established schedule only requires minor changes. A slight effect can be seen that the performance profits from lower graph diameters - the omnicast roundtrip time is slightly reduced for graphs of high degree and low diameter.

A final experiment was conducted, where the graph density is changed slowly. The results are shown in figure 8. Again, the performance of PDOR is mostly unaffected by the changes in the network. It is also again visible that the performance is slightly better for low diameters.

V. CONCLUSIONS

An efficient TDMA scheduling algorithm DDOR has been presented and discussed. By taking advantage of the collision behaviour typically found in phase and frequency modulated radio systems, a modification to the original algorithm (PDOR) has been furthermore presented and discussed. Both algorithms are able to rapidly adjust to changes in the network topology, and achieve a very good performance well within the theoretical upper bounds. While DDOR requires large schedules being sent with every message, and suffers lower performance for dense network, the modified PDOR algorithm maintains almost constant performance, and only requires short schedule lengths down to 16 slots.

REFERENCES

[1] Michael R. Frater, Michael J. Ryan, and Robin M. Dunbar. *Electromagnetic communications within swarms of autonomous underwater vehicles*. In WUWNet '06: Proceedings of the 1st ACM international workshop on Underwater networks, pages 64–70, New York, NY, USA, 2006. ACM Press.

[2] Leszek Gasieniec and Andrzej Lingas. *On adaptive deterministic gossiping in ad hoc radio networks*. In SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pages 689–690, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[3] S. L. Hakimi and E. F. Schmeichel. *Gossiping in radio networks*. Ars Combinatoria, 35-A:155–160, 1993.

[4] Ted Herman and Sébastien Tixeuil. *A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks*. In ALGOSENSORS, pages 45–58, 2004.

[5] Sandeep S. Kulkarni and Umamaheswaran *Arumugam. TDMA Service for Sensor Networks*. In ICDCS Workshops, pages 604–609, 2004.

[6] Felix Schill, Jochen Trumpf, and Uwe R. Zimmer. *Towards optimal TDMA scheduling for robotic swarm communication*. In Proceedings Towards Autonomous Robotic Systems, 2005.

[7] Felix Schill and Uwe R. Zimmer. *Distributed dynamical omnicast routing*. Complex Systems (intl. Journal), 16(4), 2006.

[8] Felix Schill and Uwe R. Zimmer. *Effective communication in schools of submersibles*. In Proceedings IEEE OCEANS '06, 2006.

[9] Bulent Tavli and Wendi B. Heinzelman. E*nergy and spatial reuse efficient network-wide real-time data broadcasting in mobile ad hoc networks*. IEEE Transactions on Mobile Computing, 5(10):1297–1312, 2006.