A LINK ALLOCATION PROTOCOL FOR MOBILE MULTI-HOP RADIO NETWORKS

by

I. Chlamtac and A. Lerner

Technical Report #363

January 1985
ABSTRACT

We present a distributed algorithm for assignment of transmission rights in a mobile multi-hop radio network. The proposed algorithm assigns a timeslot to each link forming a short cyclic TDMA. The resulting assignment allows collision free spatial reuse of the radio channel. The algorithm supports continuous slot assignment adaptation to topological changes while maintaining the collisionfree property. The adaptation process, which is an integral part of the link assignment algorithm, requires the participation of only the immediate neighbors of the change causing node. The adaptation to topological changes is accomplished in linear time and implemented in the order of their occurrence in the network.
1. INTRODUCTION

Mobile radio networks that employ packet switching when sharing a common communication channel are receiving increasing attention. Communication systems such as PRNET [1], JTIDS [2], or BID [3] are only some of the better known instances of such networks.

The design of communication protocols in broadcast radio based networks must overcome at least two difficult problems. One, the partial connectivity of the multi-hop network, which combined with the broadcast property of the radio channel, makes efficient channel sharing extremely difficult. The other, user mobility, due to which the network connectivity changes dynamically thus requiring a constant and dynamic adaptation capability from the protocol.

The radio network protocol architecture must provide the control needed for efficient and reliable network operation in face of these difficulties. At the channel access protocol level, several centralized and distributed algorithms for multi-hop networks have been recently proposed. For limited number of hops, i.e., small radius networks, both random access [4,5] and reservation protocols have been constructed [6,7]. For large radio networks with changing topologies most of the existing solutions employ variants of time division multiple access techniques, using centralized or distributed control strategies [2,8,17]. Among these the fully or partially (hierarchically) distributed architectures were proposed as providing a potentially higher network survivability and being therefore attractive, especially for mobile radio networks.

Related works have demonstrated that time division based allocations of transmission rights in these systems are difficult and their optimization leads to algorithms with prohibitively high complexity [9,10]. Some practical solutions have thus been recently introduced. The linked cluster algorithm presents a hierarchical approach in which nodes are organized in clusters consisting of ordi-
nary nodes, gateways, and cluster head nodes [11]. In this algorithm a link allocation is executed for each direction on each link, s.t. in a sense the broadcast based radio medium is "translated" into a point to point logical assignment of transmission rights for every pair of nodes which are in line of sight and within range. The algorithm employs time and frequency partitioning so that the allocation of transmission rights can be achieved with reduced probability of packet collisions.

The spatial reuse TDMA approach is a simpler variation of the classic TDMA scheme whereby nodes whose transmissions will not collide are assigned the same time slots so that channel reuse in time or frequency is obtained [12,17].

Both of the above approaches deal with node mobility by disseminating the topology change information in the network until reorganization is achieved.

In this paper we propose a new distributed channel access algorithm based on allocating link transmission rights. The transmission allocation is conflict free and reuse of the communication channel is permitted subject to absence of collisions. The proposed algorithm provides dynamic link assignment for mobile networks in which, in case of a topological change, only nodes in line of sight (LOS) of that change are required to exchange topology update information.

The proposed algorithm differs from existing distributed network algorithms in several ways which are significant for the radio environment. The distributed control mechanism underlying this algorithm does not assume that control messages arrive collision free [11,18,12]. Furthermore, in preceding algorithms it was assumed that the order of control message arrivals on every link corresponds to the order of their transmission. Since in a radio environment the reliability of the channel may be low, due to packet collisions, it is advantageous to assume that control messages can collide and be retransmitted after random delays. Relaxing, in the proposed protocol, the requirement for correspondence in the
transmission/reception order permits the use of unrestricted retransmission mechanism and the use of flow control mechanisms with sliding window greater than one [15]. Since the transmission time of control messages can be expected to be typically small compared to the propagation delay, larger window size can greatly improve performance.

Secondly, the number of nodes in a radio network can be large and therefore the probability of concurrent topological changes is not negligible. Therefore, it is also advantageous to relax the requirement restricting each node to a single coexecution of the protocol. In other words, as opposed to previous works [15,12], in the proposed algorithm a new protocol execution can be triggered at a node before the protocol activities associated with the previous triggering of the protocol have been completed. By allowing topological updates to take place concurrently and continuously, we can also abandon the requirement for periodic reorganization of slot assignment of the total network [11]. In this way topological changes can be accommodated without periodically suspending the data message transmission in the network. As an additional benefit from the continuous update process, it is guaranteed that at no time collisions will occur as the transmission rights assignment remains always current.

In the following sections we define the algorithm, demonstrate its operation and prove that in addition to the above given properties, the algorithm is linear in execution time and in number of messages generated owing to each update. We also prove that the TDMA cycle generated by the algorithm is not trivially bounded and that the algorithm remains deadlock free in face of any combination of topological changes.
2. THE LINK ALLOCATION PROBLEM IN MOBILE RADIO NETWORKS - PROBLEM DEFINITION AND PROPOSED SOLUTION

The link allocation problem can be stated as follows:

Given a mobile radio network in which

1) all links are bidirectional,

2) a node cannot receive correctly more than one message at a time (otherwise collision results) and,

3) a node cannot send and receive at the same time,

find a source-destination transmission rights assignment, ensuring one collision free transmission for both directions on each link, s.t., a) the cycle length, defined as the total number of distinct timeslots needed, is minimized or, 2) the number of simultaneous transmissions in every slot is maximized.

It was proven that this problem, even for a given static multihop radio network, is NPH [10]. As a result we look for a bidirectional source-destination transmission rights assignment, or link allocation, based on a polynomial heuristic algorithm. The heuristic algorithm should preserve the collision free property in a mobile radio network and yield a short, bounded cycle length.

To construct such algorithm we are guided by two main considerations:

a) the minimization of the set of conditions each node must consider, in a distributed algorithm, in order to execute a collision free link allocation and,

b) the minimization of the radius of nodes, given by the maximum number of hops each node must consult, in the link allocation process.

We therefore, proceed as follows. We first define a set of conditions which enables each node to locally execute a collision free link allocation and we prove that these conditions are both sufficient and necessary. Secondly, we introduce a "one-way" link allocation algorithm and prove that in this way, every node in the
link allocation process is required to exchange information only with its immediate neighbors, i.e., nodes one hop away, in order to satisfy the above conditions.

2.1 Necessary and Sufficient Conditions for Collision Free Link Allocation

To derive this set of conditions we first define the following model. We use the indirected graph \( G(V,E) \) where a node \( v \) in \( V \) represents a node in the network and an edge \( e \) in \( E \), \( e: v_1 -- v_2 \) signifies that nodes \( v_1 \) and \( v_2 \) are in line of sight (LOS) and within range of each other. We shall say that nodes \( v_1 \) and \( v_2 \) are neighbors (being one hop away). The rank of the graph \( G \), \( \text{rank}(G) \) is the maximum number of neighbors a node can have.

**Definition 2.1.1:** A node is a Sending Node in a given timeslot if it is chosen by the link allocation algorithm to send a message during that slot.

**Definition 2.1.2:** A node is a Receiving Node in a given timeslot if it is chosen as a destination of a Sending Node.

**Definition 2.1.3:** A collision in a Receiving Node \( v \) occurs if in a given timeslot

1) More than one neighbor of \( v \) are chosen as Sending Nodes, or

2) Node \( v \) is chosen as a Sending Node.

**Theorem 2.1.4:** The following conditions are necessary and sufficient to ensure collision free transmissions:

In every slot:

- c.1) A node will not be a Sending Node and a Receiving Node at the same time, and
- c.2) Let \( v \) be a Sending Node, then none of its destination's neighbors must be a Sending Node.

**Proof of Theorem 2.1.1:** We first prove that conditions c.1 and c.2 are sufficient.
Assuming the conditions hold, we show that there can be no collision. By definition 2.1.3, a collision can occur only in a Receiving Node. Let \( v \) be a Receiving Node we show that collision states will not occur.

1. Exactly one neighbor of \( v \) is a Sending Node:  
At least one node transmits from definition 2.1.2. No more than one node transmits from condition c.2.

2. Node \( v \) is not a Sending Node (from condition c.1). Therefore the conditions are sufficient.

We now prove that conditions c.1 and c.2 are necessary. From definition 2.1.3, if a node is both a Sending and a Receiving Node in the same slot, collision results. Therefore, condition c.1 is necessary. If c.2 does not hold then more than one neighbor of a destination node is a Sending Node causing collision by definition 2.1.3, therefore c.2 is necessary.

Q.E.D.

Definition 2.1.5: We define a consistent link allocation algorithm to be an algorithm that assigns timeslots for links so that following the assignment the transmissions are collision-free.

We use the following notations for denoting the possible actions of a node in a given slot: S for a Sending Node, R for a Receiving Node, NS for a node prohibited by the allocation algorithm from being a Sending Node, NR for a node prohibited from being a Receiving Node, P for a passive node (takes no active action and has no restrictions).

Theorem 2.1.2: If an algorithm is a consistent link allocation algorithm, then in each slot, every node must be in one of the following states:

S.1: S+NR
S.2: R+NS
S.3: NR+NS
S.4: NS-R  
S.5: NR-S  
S.6: P  

Proof of Theorem 2.1.2: The consistency property ensures that in each timeslot, each Sending Node is in state S.1 (S+NR) and each Receiving Node is in state S.2 (R+NS). The other possible states consider nodes which are not Sending Nodes nor Receiving Nodes: each one of these nodes can be 1) a neighbor of one or more Sending Nodes (state S.5 - NR-S) or, 2) a neighbor of one or more Receiving Nodes (state S.4 - NS-R) or, 3) a neighbor of one or more Sending Nodes and one or more Receiving Nodes (state S.3 - NR+NS), or 4) the node and any of its neighbors are not Sending or Receiving (state S.6).  

Q.E.D.

2.2 Minimizing the Radius of Nodes Participating in the Link Allocation Process

To simplify the process which dynamically adapts the link allocation to topological changes, we wish to limit the number of nodes involved in an allocation updating process. Specifically, we optimally seek an algorithm by which only the node that has actively changed location, will be required to execute the allocation process and that in this process the node exchange information only with its immediate neighbors. As a result an allocating node must be responsible for executing the allocation on each link on both directions, i.e. for sending and receiving between itself and every other node "connected to" the links. It can be shown however that if during the allocation process, we request the allocating node to assign a slot for transmission (when acting as a Sending Node) and for reception of messages (when acting as a Receiving Node) the radius of nodes that must be involved in the concurrent allocation process must exceed one hop. Observe, for example, a network in which nodes i and l (three hops away) have
changed locations, leading to the network configuration shown in figure 1. Assume that node \( i \) has decided to allocate a time slot to link \( i \rightarrow j \) for sending messages to node \( j \). Given no information is exchanged between nodes which are more than one hop away node \( l \) can decide to allocate the same slot to link \( k \rightarrow l \) for reception of messages from node \( k \). In this way a collision will occur in node \( j \).

To obtain an allocation updating process in which only the position changing node remains responsible for executing the allocation update and in which messages will be exchanged only between itself and its immediate neighbors we introduce a "one-way allocation". In this approach all network nodes involved in allocation always assign slots for a single direction, i.e., sending or receiving, only. The complementing direction is obtained by subdividing the allocated slot and reversing the communication direction in one of the created subslots consistently in the network.

It therefore remains to show that the proposed approach preserves the consistency of the algorithm. In other words, we must prove that after reversing the direction on each link we again obtain a consistent allocation.

Definition 2.2.1: A SENDING SET is a set of sending nodes in a certain slot constructed by a consistent algorithm.

Definition 2.2.2: Let \( U \) be a sending set. We construct a new set of nodes \( U' \) in the following way:
1) \( U' = \{ \} \).
2) For every \( u \) in \( U \), let \( u' \) be the corresponding destination of \( u \), \( U' = U' + u' \).

Theorem 2.2.1: \( U' \) is a sending set.
In other words: We claim that the set of all destinations of a sending set forms a sending set.

Proof of Theorem 2.2.1: Let A1 be a consistent algorithm which constructs the
sending set $U$. Let A2 be the following algorithm:

1) Execute A1.

2) Change node states as follows:

<table>
<thead>
<tr>
<th>Node States after execution of A1</th>
<th>Node States after execution of A2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.1 (S+NR)</td>
<td>S.2 (R+NS)</td>
</tr>
<tr>
<td>S.2 (R+NS)</td>
<td>S.1 (S+NR)</td>
</tr>
<tr>
<td>S.3 (NS+NR)</td>
<td>S.3 (NS+NR)</td>
</tr>
<tr>
<td>S.4 (NS-R)</td>
<td>S.5 (NR-S)</td>
</tr>
<tr>
<td>S.5 (NR-S)</td>
<td>S.4 (NS-R)</td>
</tr>
<tr>
<td>S.6 (P)</td>
<td>S.6 (P)</td>
</tr>
</tbody>
</table>

A2 constructs $U'$, s.t. $U'$ consists of exactly those nodes which were receiving nodes in A1. We now show that A2 is consistent (equivalent to $U'$ being a sending set by Definition 2.2.1). Let $v$ be a Receiving Node (state S.2) in a certain slot. Then:

1) Only one of node’s $v$ neighbors is a Sending Node: Suppose node $v_1$ is a sending neighbor having node $v$ as a destination. (Following algorithm A1 it is obvious that such Sending Node exists.) Suppose there were more than one sending neighbors, so there is at least one more neighbor which is sending (node $v_2$). According to algorithm A1, node $v$ is sending to node $v_1$ in the same slot in which node $v_2$ is a Receiving Node. That leads to a collision in node $v_2$ which contradicts the assumption that algorithm A1 is consistent.

2) Node $v$ is not a Sending Node in that slot: There is no state $S+R$ in algorithm A1 nor in the translated algorithm A2. (A1 is consistent.) Therefore, A2 is a consistent algorithm and $U'$ is a sending set.

Q.E.D.

3. THE DISTRIBUTED LINK ALLOCATION ALGORITHM (DLAA)

The DLAA executes a one way assignment of timeslots for links creating a collision-free TDMA cycle. The algorithm is based on the necessary and sufficient conditions for collision free transmissions given in Section 2.

Assumptions:
A1. The network graph is always connected.
A2. Every node has a unique number.
A3. The network nodes are synchronized up to one way propagation delay.
A4. Each node is aware of changes in its set of neighbors due to its own movement and of loss of contact with neighbors due to other reasons. (This information is obtained through the physical layer interface.)
A5. No two nodes join the network at exactly the same time. (It is assumed that appropriate time resolution can always be achieved.)

The following description is consistent with the formal algorithm description given in Section 4 and uses the same keywords.

For the execution of the DLAA each node keeps the local information describing its own TDMA cycle and that of its neighbors. Each node $i$ records its own version of the TDMA cycle in LOCAL-TDMA which contains its state per each cycle slot and state description. The state description contains the following information which is necessary for valid state transitions:

For S.1: Receiving node (message destination) id.
For S.2: Sending node (message source) id.
For S.3: id's of all $v$'s neighbors in states S.1 and S.2.
For S.4: id's of all $v$'s neighbors in state S.2.
For S.5: id's of all $v$'s neighbors in state S.1.
For S.6: None.

Similarly, the TDMA cycles for every neighbor $j$ of $i$ are maintained in NEIGHBORS-STATES($i$).

Every node has a STATUS field which designates the node as NEW or OLD. Node $i$ will acquire a NEW status when 1) node $i$ wishes to join the network or 2) node $i$ recovers from failure or 3) node $i$ changes physical location and as a consequence the set of its neighbors has changed. We assume a node learns of its NEW status
from a higher layer protocol (event NEW-STATUS).

Information between local structures of different nodes can be exchanged by the use of control messages. For simplicity of presentation we assume that control messages do not collide. (A correct protocol operation that does not make this assumption, is given in Section 6.)

We next describe the operation of the DLAA algorithm. A new node \( i \) (STATUS=NEW) sends a WAKEUP message to each one of its neighbors requesting their PERMISSION to execute a link allocation (the assignment of timeslots in which node \( i \) will act as a Sending Node) for all its UNALLOCATED links. A link between \( i \) and its neighbor \( j \) is considered unallocated if \( j \) has not executed the link allocation prior to \( i \). In addition, the WAKEUP message is also used to get synchronized with the existing network TDMA-cycle. We denote the cycle beginning by CYCLE_SYNC.

When the WAKEUP message is received at an OLD (/NEW) neighboring node \( j \), an entry is added to \( j \)'s WAITING-FOR-PERMISSION local queue, and a WELCOME (/WAKEUP) message, consisting of \( j \)'s LOCAL_TDMA and CYCLE_SYNC, (or \( j \)'s priority) is sent back to \( i \). The WAITING-FOR-PERMISSION queue is sorted by the dynamic priority of the requesting nodes, determined by the TIME-STAMP which is the awakening time of node \( i \).

Node \( j \) will send a PERMISSION message to the node at the top of the queue when it learns that the PREVIOUSLY-PERMITTED node has completed its allocation process. At this point, the PREVIOUSLY-PERMITTED node's entry is removed from the WAITING-FOR-PERMISSION queue. Eventually node \( i \)'s WAKEUP entry reaches the top of the queue and \( i \) receives a PERMISSION message from \( j \). When node \( i \) receives allocation PERMISSIONs from all its neighbors (i.e.,PERMISSION-GRANTING=NEIGHBORS) it can proceed with the link allocation process. The DLAA uses a greedy link allocation algorithm obtained by finding the earliest available
slot from the beginning of the cycle. A slot is available for allocation from node \( i \) to node \( j \) if in that slot:

1) Node \( i \) and \( j \) are not Sending Nodes nor Receiving Nodes (states S.1 or S.2),
2) Node \( i \) is not prohibited from being a Sending Node (states S.3 or S.4) and
3) Node \( j \) is not prohibited from being a Receiving Node (states S.3 or S.5).

Following the slot allocation on all previous UNALLOCATED links, node \( i \) sends its new LOCAL-TDMA version in an OWNVERSION message, thus informing all its neighbors of the completion of its allocation process. Following the sending of the OWNVERSION message, node \( i \)'s STATUS is changed to OLD. At this point, node \( i \) can start collision free transmission and reception of data messages.

**Example:** We shall demonstrate the allocation process for the sample network given in figure 2. For simplicity of presentation, we divide the algorithm execution into a series of steps with each step describing the actions related to a group of control messages of the same type which can be sent and serviced concurrently.

We use the following notations:

PERMISSION \((id_1, id_2, id_1, \text{LOCAL}_\text{TDMA}, t)\) - PERMISSION message sent from node \( id_1 \) to node \( id_2 \), with \( t \) the \( id_1 \)'s version of the CYCLE-SYNC.

WAKEUP \((id, pr)\) - WAKEUP message sent by node \( id \) with temporal priority=pr.

WELCOME \((id_1, id_2, id_1, \text{LOCAL}_\text{TDMA})\) - WELCOME message

OWNVERSION \((id, id, \text{LOCAL}_\text{TDMA})\) - OWNVERSION message.

OWNVERSION+PERMISSION \((id_1, id_2, id_1, \text{LOCAL}_\text{TDMA}, t)\) - a "combined" message.

WELCOME+PERMISSION \((id_1, id_2, id_1, \text{LOCAL}_\text{TDMA}, t)\) - the "WELCOME" part can be sent as a broadcast.

We show the effect of the messages, passed in each step, on the following local information stored at each node: STATUS (NEW or OLD), WAITING-FOR-PERMISSION queue, PERMISSION-GRANTING list, LOCAL-TDMA and NEIGHBORS-STATES. For the description of LOCAL-TDMA and NEIGHBORS-STATES we use the notation: state
Due to space limitations in the following execution scenario, we give full state descriptions only at selected steps. All remaining descriptions can be easily deduced from the given detailed description of messages sent in each step.

**STEP 1: Messages sent:**
- WAKEUP(1,10); WAKEUP(2,9); WAKEUP(3,8);
- WAKEUP(4,7); WAKEUP(5,6); WAKEUP(6,5);
- WAKEUP(7,4); WAKEUP(8,3); WAKEUP(9,2);
- WAKEUP(10,1);

The **STATUS** of all nodes is NEW. PERMITTED lists, LOCAL-TDMA and NEIGHBORS-STATES are empty. The only local information effected by Step 1 is the WAITING-FOR-PERMISSION queue to which all WAKEUP-messages received by neighboring nodes are added according to their priority.

**STEP 2: Messages sent:**
- WELCOME+PERMISSION(1,10, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(2,10, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(3,8, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(4,7, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(6,7, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(5,9, LOCAL-TDMA, CYCLE-SYNC);
- WELCOME+PERMISSION(8,9, LOCAL-TDMA, CYCLE-SYNC);

The only local information effected by Step 2 is the list of PERMISSION GRANTING that contains the list of nodes from which a PERMISSION message was received.

**STEP 3: {Allocation process is executed by nodes 9 and 10} - messages sent:**
- PERMISSION+OWNVERSION(9,8, LOCAL-TDMA, CYCLE-SYNC);
- PERMISSION+OWNVERSION(10,2, LOCAL-TDMA, CYCLE-SYNC);

The situation by the end of Step 3 is summarized in table 1.

**STEP 4: Messages sent:**
- PERMISSION(5,8, LOCAL_TDMA, CYCLE_SSYNC);
- PERMISSION(2,5, LOCAL_TDMA, CYCLE_SSYNC);
- PERMISSION(1,2, LOCAL_TDMA, CYCLE_SSYNC);
STEP 5: {Node 8 executes link allocation procedure} - messages sent:
PERMISSION+OWNVERSION(8,5, LOCAL_TDMA, CYCLE_SYNC);

STEP 6: Messages sent:
PERMISSION(3,7, LOCAL_TDMA, CYCLE_SYNC);
PERMISSION(9,5, LOCAL_TDMA, CYCLE_SYNC);

STEP 7: {Nodes 5 and 7 execute link allocation} - messages sent:
PERMISSION+OWNVERSION(5,2 LOCAL_TDMA, CYCLE_SYNC);
PERMISSION+OWNVERSION(7,6 LOCAL_TDMA, CYCLE_SYNC);

The updated local information at each node by the end of step 7 is shown in table 2

STEP 8: Messages sent:
PERMISSION(8,3, LOCAL_TDMA, CYCLE_SYNC);
PERMISSION(4,6, LOCAL_TDMA, CYCLE_SYNC);

STEP 9: {Nodes 6 and 2 execute link allocation} - messages sent:
PERMISSION+OWNVERSION(6,4, LOCAL_TDMA, CYCLE_SYNC);
PERMISSION+OWNVERSION(2,1, LOCAL_TDMA, CYCLE_SYNC);

STEP 10: Messages sent:
PERMISSION(10,1, LOCAL_TDMA, CYCLE_SYNC);
PERMISSION(7,4, LOCAL_TDMA, CYCLE_SYNC);

STEP 11: {Nodes 1 and 4 execute link allocation which is actually a null allocation as all their links are already allocated} - messages sent:
OWNVERSION(1, LOCAL_TDMA);
OWNVERSION(4, LOCAL_TDMA);

STEP 12: Messages sent:
PERMISSION(7,3, LOCAL_TDMA, CYCLE_SYNC);

STEP 13: {Node 3 executes link allocation} - messages sent:
OWNVERSION(3, LOCAL_TDMA);
The Final Allocation:

<table>
<thead>
<tr>
<th>Slot 1</th>
<th>Slot 2</th>
<th>Slot 3</th>
<th>Slot 4</th>
<th>Slot 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 10 -&gt; 2</td>
<td>10 -&gt; 1</td>
<td>8 -&gt; 5</td>
<td>5 -&gt; 2</td>
<td>2 -&gt; 1</td>
</tr>
<tr>
<td>7 -&gt; 3</td>
<td>7 -&gt; 4</td>
<td>7 -&gt; 6</td>
<td>6 -&gt; 3</td>
<td></td>
</tr>
<tr>
<td>9 -&gt; 5</td>
<td>9 -&gt; 5</td>
<td></td>
<td>6 -&gt; 4</td>
<td></td>
</tr>
</tbody>
</table>

At this point, each slot is subdivided into two slots: The first subslot serves the direction indicated in the allocation table while the second subslot serves the opposite direction. The link allocation for each direction on every link is given in figure 2b.

4. PSEUDO CODE OF THE DLAA

We present here a code to be run in node $i$

VAR
NEIGHBORS {a list of all neighbors}
PERMISSION_GRANTED {a list of neighbors from which a PERMISSION was already received}
LOCAL TDMA {a table of own state and state description in each slot}
UNALLOCATED {a list of unallocated links}
NEIGHBORS STATES {a table of neighbors states and their states description}
WAITING FOR PERMISSION {queue of neighbors waiting for PERMISSION}
PREVIOUSLY PERMITTED {the last node to which a PERMISSION was sent, null if none}
TIME_STAMP {the dynamic priority of WAKEUP requests, for reasons of simplicity we shall assume that no two nodes become NEW at exactly the same time}
STATUS: (NEW, OLD)
UPDATE RECEIVED LIST {a list of neighbors with respect to which the node is already updated, i.e., the node has received their LOCAL TDMA and CYCLE_SYNC {the beginning of the next TDMA cycle}}.

Event Type

NEW STATUS(neighbors) {the node's status becomes NEW}
LOSS OF CONTACT(j) {OLD node $i$ looses contact with node $j$}
PERMISSION(id$_1$, id$_2$, ids, LOCAL TDMA, CYCLE_SYNC) {PERMISSION message (sent by node id$_1$ to node id$_2$) received at $i$. CYCLE SYNC carries the starting time of the next TDMA cycle included for synchronization purposes}
OWNVERSION(id, LOCAL TDMA) {OWNVERSION message (sent by node id}
received at all neighbors}
WAKEUP(id,pr) {a WAKEUP message sent by node id, with id’s TIME_STAMP passed in pr, received by all neighbors}
WELCOME(id1,id2,id1.LOCAL_TDMA,CYCLE_SYNC) {a WELCOME message is received from OLD neighboring id1}

{Event Description}

ON NEW-STATUS(NEIGHBORS) do {at node i}
begin
 initialization of data structures
UNALLOCATED:=NEIGHBORS; \{list of neighbors passed to DLAA as parameters\}
PERMISSION_GRANTING:=\{\}
UPDATE_RECEIVED_LIST:=\{\}
PREVIOUSLY_PERMITTED:=null;
TIME_STAMP:=actual time;
STATUS:=NEW;
WAITING_FOR_PERMISSION:=(i, TIME_STAMP);
LOCAL_TDMA, NEIGHBORS STATES:=empty;
send WAKEUP(i,TIME_STAMP) \{procedure send executes transmission of a control message\}
end

ON LOSS_OF_CONTACT(i) do
begin
update UNALLOCATED, NEIGHBORS, UPDATE_RECEIVED_LIST, PERMISSION_GRANTING WAITING_FOR_PERMISSION, NEIGHBORS STATES, LOCAL_TDMA.
\{procedure update executes all changes relevant to j in the listed data structures\}
if PREVIOUSLY_PERMITTED=\{j then
begin \{a PERMISSION was sent to a node no longer in LOS\}
PREVIOUSLY_PERMITTED:=\{top (WAITING_FOR_PERMISSION)
\}top returns null when the queue is empty; notice a top operation conventionally does not remove the first element from queue
if PREVIOUSLY_PERMITTED not in \{null, i\} then
send PERMISSION (i, PREVIOUSLY_PERMITTED, LOCAL_TDMA, CYCLE_SYNC)
\{PERMISSION is sent only after a delay of at least a propagation-delay time from the time the WAKEUP message was received.
This delay is needed to make sure there are no higher priority requests on their way to node i\}
end; \{PREVIOUSLY_PERMITTED=\{\}
end;

ON PERMISSION (id1,id2,id1.LOCAL_TDMA,CYCLE_SYNC) do
begin
if id2=i then \{the message is directed to node i\}
begin
PERMISSION_GRANTING:=PERMISSION_GRANTING+id1
update LOCAL_TDMA, NEIGHBORS STATES, CYCLE_SYNC
UPD_RCVR_LIST:=UPD_RCVR_LIST+id1
if PERMISSION_GRANTING=NEIGHBORS then
begin \{start the allocation process\}
remove \{i\_TIME\_STAMP\} from WAITING\_FOR\_PERMISSION;
allocate \{procedure allocate executes DLAA allocation as
given in Sec.3 and adds the allocation information to
LOCAL\_TDMA\}\nPREVIOUSLY\_PERMITTED:=top(\{WAITING\_FOR\_PERMISSION\})
send OWNVERSION \{i,LOCAL\_TDMA\}
STATUS:=OLD
if PREVIOUSLY\_PERMITTED ≠ null then send PERMISSION
\{i,PREVIOUSLY\_PERMITTED,LOCAL\_TDMA,CYCLE\_SYNC\}
end
if UPDATE\_RECEIVE\_LIST=NEIGHBORS and STATUS=NEW and PREVIOUSLY-
PERMITTED=null then
begin
PREVIOUSLY\_PERMITTED:=top(WAITING\_FOR\_PERMISSION)
if PREVIOUSLY\_PERMITTED not in \{i,null\} \{set operation\}
then send PERMISSION \{i,PREVIOUSLY\_PERMITTED,
LOCAL\_TDMA,CYCLE\_SYNC\}
end
end

ON OWNVERSION \{id1,id1,LOCAL\_TDMA\} do
begin
update LOCAL\_TDMA, NEIGHBORS\_STATES
if id1=PREVIOUSLY\_PERMITTED then
begin \{the expected response to PERMISSION is received\}
remove id1 from WAITING\_FOR\_PERMISSION
UNALLOCATED:=UNALLOCATED-id1
end
PREVIOUSLY\_PERMITTED:=top \{WAITING\_FOR\_PERMISSION\}
if \{PREVIOUSLY\_PERMITTED not in \{null,i\}\} and
(UPDATE\_RECEIVER\_LIST=NEIGHBORS or STATUS=OLD)
then send PERMISSION \{i,PREVIOUSLY\_PERMITTED,
\{LOCAL\_TDMA,CYCLE\_SYNC\}
end

ON WAKEUP \{id,pr\} do
begin
If \{(id,pr) not in WAITING\_FOR\_PERMISSION\} then
begin \{first arrival of this WAKEUP message\}
enter \{(id,pr) to WAITING\_FOR\_PERMISSION \{procedure enter adds
element to sorted queue\}
if in NEIGHBORS then
update LOCAL\_TDMA,NEIGHBORS\_STATES, PERMISSION\_GRANTING;
NEIGHBORS:=NEIGHBORS+id;
if STATUS=NEW then
begin
UPDATE\_RECEIVED\_LIST:=UPDATE\_RECEIVED\_LIST+id
send WAKEUP \{i,TIME\_STAMP\}
if UPDATE\_RECEIVED\_LIST=NEIGHBORS and PREVIOUSLY-
PERMITTED=null
then begin PREVIOUSLY\_PERMITTED:=top(WAITING\_FOR-
PERMISSION)
If PREVIOUSLY_PERMITTED not in i,null then
    send PERMISSION (i, PREVIOUSLY_PERMITTED, LOCAL_TDMA, CYCLE_SYNC)
end

UNALLOCATED:=UNALLOCATED+i
end

if STATUS=OLD then if PREVIOUSLY=PERMITTED ≠ null then
    send WELCOME (i.id, LOCAL_TDMA, CYCLE_SYNC)
else begin
    send PERMISSION (i.top(WAITING_FOR_PERMISSION, LOCAL_TDMA, CYCLE_SYNC)
    PREVIOUSLY_PERMITTED:=TOP(WAITING_FOR_PERMISSION)
end

if PREVIOUSLY_PERMITTED=id and priority (PREVIOUSLY_PERMITTED) ≠
    then {a PERMISSION to be ignored}
begin
    PREVIOUSLY_PERMITTED:=top(WAITING_FOR_PERMISSION)
if PREVIOUSLY_PERMITTED not in (i,null) then
    send PERMISSION (i,PREVIOUSLY_PERMITTED, LOCAL_TDMA, CYCLE_SYNC)
end
end

ON WELCOME (id₁, id₂, id₁, LOCAL_TDMA, CYCLESYNC) do

begin
    if id₂=i then
    begin
        UPDATE_RECEIVED_LIST:=UPDATE_RECEIVED_LIST+id₁
        update LOCAL_TDMA, NEIGHBORS_STATES
        if UPDATE_RECEIVED_LIST=NEIGHBORS and PREVIOUSLY_PERMITTED=null
            then begin
                PREVIOUSLY_PERMITTED:=top(WAITING_FOR_PERMISSION)
                if PREVIOUSLY_PERMITTED ≠ i then
                    send PERMISSION (i,PREVIOUSLY_PERMITTED, LOCAL_TDMA, CYCLE_SYNC)
            end
        end
    end
end

5. THE DLAA ALGORITHM PROPERTIES

We use the following notations: n for the number of nodes (|V|) and r for the
maximum number of neighbors (rank (G)). We next prove that the presented
DLAA algorithm creates a collision free TDMA slot assignment with a cycle length
bounded by TIME,

TIME ≤ min{2(r exp 2)–2r+1, r×n / 2}. 
In every given cycle of length TIME every node transmits to all its neighbors, and all nodes remain always synchronized to the same cycle starting point. We prove that every execution instance of the DLAA algorithm, i.e., the handling of every topological change is deadlock free, is accomplished in linear time and requires $O(r)$ number of messages. We further show that the algorithm remains consistent and deadlock free under any combination and ordering of topological changes in the network, as long as the network remains connected.

**Lemma 5.1:**

A) Neighbors do not execute the link allocation process at the same time.

B) Nodes two hops away do not execute the link allocation process at the same time.

**Proof of Lemma 5.1:**

A) Let nodes $w$ and $v$ be neighboring nodes. Assume that node $w$ is executing link allocation then, 1) a PERMISSION sent from node $v$ has been received at node $w$ prior to the allocation execution, or 2) node $w$ is not yet aware of node $v$ being a neighboring node.

1) If a PERMISSION message was received then node $v$ cannot allocate until:

a) reception of an OWNVERSION message sent by node $w$ notifying the completion of the allocation process at node $w$ or

b) loss of contact with node $w$ which is detected by both nodes

Theorem (LOSS_OF_CONTACT event) and causes the last PERMISSION to be ignored, or

c) node $w$ becomes a NEW node (without loss of contact). In this case, node $w$ ignores all PERMISSIONs sent in the past.

On reception of the WAKEUP message sent by the NEW node $w$. Node $v$ locally cancels the PERMISSION it has previously sent, or
d) node $v$ becomes a NEW node (without loss of contact). In this case, node $v$ sends a WAKEUP message and is blocked (i.e., will not send further messages) until it receives confirmations from all neighbors (via the WELCOME or WAKEUP messages), while node $w$ ignores the PERMISSION already received from node $v$ as soon as the WAKEUP message arrives.

2) If node $w$ is not aware of node $v$ being its neighbor then the WAKEUP message sent by node $v$ has not yet arrived at node $w$. Node $v$, as a NEW node is informed about the neighbors and can not proceed with link allocation prior to reception of PERMISSIONs from all its neighbors.

B) Let node $u$ and $v$ be two hops away sharing node $w$ as a neighbor. Node $w$ can send PERMISSION only to one of its neighbors at a time.

Assume that node $w$ has sent PERMISSION to node $u$. Then $w$ can not send any other PERMISSION until: a) reception of an OWNVERSION message from node $u$ indicating the completion of the allocation at node $u$, or b) loss of contact with node $u$, which is detected by both nodes $u$ and $w$ and causes the last PERMISSION to be ignored, or c) node $u$ becomes a NEW node and thus gives up the allocation rights previously given to it. Thus nodes sharing a neighbor cannot hold that node’s PERMISSION at the same time.

Q.E.D.

**Theorem 5.1:** The allocation created by the DLAA is collision free.

**Proof:** We show that the DLAA preserves the necessary and sufficient conditions.

In every slot:

1) A node cannot send and receive: Assume that in slot $m$ node $v$ sends to a neighboring node $u_1$ and receives from node $u_2$. Nodes $v$ and $u_2$ are neighbors and therefore, by Lemma 5.1 do not execute link allocation at the same
time. Thus the second node that allocates (node $v$ or node $u_2$) is already aware of the allocation executed by the other node (node $u_2$ or node $v$).

2) If a node is a sending node then none of its destination node’s neighbors is a sending node. Assume that in a given slot node $v$ is a sending node and node $u$ is the corresponding receiving node while another neighbor of node $u$ - node $w$ - is also a sending node. If nodes $v$ and $w$ were both neighbors of $u$ while executing the link allocation process, then the link allocation process was not executed concurrently. Assume node $v$ has executed link allocation before node $w$. Then, node $u$ sends PERMISSION to node $w$ after the reception of node’s $v$ OWNVERSION which prohibits $w$ from being a sending node in that slot and vice versa if node $w$ has executed link allocation before node $v$ then. If node $w$ was not a neighboring node of node $u$ while allocating then there are two possibilities:

1. Node $u$ has changed location and became NEW which causes cancelation of the previous allocation made by node $v$, or
2. Node $w$ has changed location and became NEW and therefore its previous allocation is canceled.

In both cases the consistency property is preserved.

Q.E.D.

Theorem 5.2: The length of the TDMA cycle (TIME) is less than or equal to

$$\min\{n \times \tau / 2, 2(r \exp 2) - 2\tau + 1\}.$$ 

Proof:

A) $TIME \leq 2(r \exp 2) - 2\tau + 1$. Let $v$ be a node executing link allocation and $w$ be a neighbor of $v$ on an unallocated link. The allocation is greedy and thus the earliest slot available is chosen. A slot is unavailable if:

1. Node $v$ is already sending or receiving - this can happen in at most $r-1$ slots.
2. Node $w$ is already sending or receiving - in at most $r-1$ slots.
3. Node \( v \) is prohibited from sending. Node \( v \) has at most \( r-1 \) neighbors that can receive from at most \( r-1 \) of their neighbors. Reception from node \( v \) is counted in case 1 above. Reception by \( w \) is counted in case 2 above. Case 3 can happen in at most \((r-1) \exp 2 \) slots.

4. Node \( w \) is prohibited from receiving. Node \( w \) has at most \( r-2 \) neighbors relevant to counting unavailable slots (node \( v \)'s transmissions are counted in case 1). Each of the \( r-2 \) neighbors can be sending to at most \( r-1 \) nodes.

(Transmissions to node \( w \) are counted in case 2); case 4 can occur in at most \((r-1) \exp 2 \) slots.

The maximum number of unavailable slots is therefore

\[
(r-1) + (r-1) + (r-1) \exp 2 + (r-1) \exp 2 = 2r \exp 2 - 2r
\]

so that among the first \( 2r \exp 2 - 2r+1 \) slots we will always find an available slot.

B) We now prove that \( TIME \leq (n \times r)/2 \). A slot can be unavailable only if there is a link activated in that slot. As the number of links is always less or equal \((n \times r)/2\), the number of unavailable slots is less than \((n \times r)/2\).

Q.E.D.

**Theorem 5.3:** The DLAA algorithm is deadlock free.

**Proof:** We prove that every node sending a WAKEUP message will eventually execute link allocation. Notice that link allocation is executed iff PERMISSIONs were previously received from all neighbors. A delay in PERMISSIONs arrival can occur when:

a) A PERMISSION was sent to a node no longer in LOS. This delay is bounded since on loss-of-contact a node checks whether the node PREVIOUSLY_PERMITTED point is still in LOS. In case it is not, the next PERMISSION is sent.

b) A node is prohibited from sending PERMISSION (a new node is prohibited from
sending PERMISSION till it gets a response from all its neighbors). The delay
is finite as a WAKEUP message demands an immediate response from all
nodes: from nodes, whose STATUS=OLD a WELCOME-message will be sent in
response, nodes whose STATUS=NEW send a WAKEUP-message in response.
c) A "chain problem" shown in figure 3 occurs. We prove that a situation leading
to this construction cannot occur.

Lemma 5.2: The chain problem occurs only when one or more WAKEUP messages
are received out of order (not in the order they were sent).

Proof of Lemma 5.2: Assume all WAKEUP messages were received on time. We use
the notation node i > node j if (priority node(i) > priority node (j)).

Node \( j_k \) sends PERMISSION to node \( i_k \)

\[ \Rightarrow node i_k > node i_{(k-1)}, k = 2, \ldots, n \Rightarrow i_n > node i_1 \]

but \( node i_1 > node i_n \) since node \( i_n \) was the preferred choice of node \( j_1 \). Contradic-
tions.

Therefore the chain problem can occur only when at least one node is not aware of
a WAKEUP message that was sent.

Q.E.D.

Even though WAKEUP messages can arrive at different times at different nodes the
priorities are consistent in all the network and proportional to the time-stamp
associated with the message. As a PERMISSION sent by a NEW node in the past is
ignored and the only possibility for late arrival is associated with a node that has
changed location (new node) the chain problem is unreachable. (Late arrival as a
result of collisions is discussed and solved in the next section.) This completes the
proof of Theorem 5.3.

Q.E.D.

Theorem 5.4: All OLD nodes in the network are synchronized to the same starting
point of the TDMA-Cycle (CYCLE-SYNC).
Proof of Theorem 5.4: A node "creates" a new CYCLE_SYNC point when none of its neighbors are OLD or NEW having a higher priority. At initialization time - as nodes are awaken serially, the first node in the network decides upon the CYCLE_SYNC point. From the connectivity assumption, all other nodes, upon joining the network have either an OLD neighbor or a NEW neighbor with a higher priority (which therefore was awaken earlier). Thus, they are not candidates for creation of a new cycle starting-point (CYCLE_SYNC).

Theorem 5.5: The topology change adaptation time (from the moment a WAKEUP message is sent to the moment the node can execute the link allocation process) is linear in the number of nodes in the network \(0(n)\).

Proof of Theorem 5.5: If there is only one NEW node in the network, then in constant time the node receives PERMISSION from all its neighbors and can proceed with link allocation. The worst case obtains when all nodes are NEW. The highest node (the first to be awaken) receives PERMISSION from all its neighbors in constant time, and is thus able to proceed with the link allocation. As soon as the highest node sends its OWNVERSION message its status changes to OLD and the next node can execute link allocation in \(O(1)\) time. The lowest node is thus permitted to execute allocation in \(O(n)\) time.

Theorem 5.6:

A) The number of messages passed when constructing the network is \(O(|E|)\).
B) The number of messages passed on a single topological change is \(O(r)\).

Proof of Theorem 5.6:

A) On every edge one message of each kind is sent on each direction: WAKEUP/WELCOME, PERMISSION, and OWNVERSION.
B) An update process demands that only the immediate neighbors send control messages. The number of edges involved is thus \(O(r)\) and from part A we obtain that the total number of messages is \(O(r)\).

Q.E.D
6. LOSS OF CONTROL MESSAGES-RECOVERY

The correct execution of the DLAA algorithm as presented in Section 4 depends on the correct delivery of its control messages. In general, such delivery can be guaranteed by the use of e.g., CDMA access, as proposed in several existing algorithm [7,13]. However, these techniques guarantee a collision-free delivery at additional cost in terms of bandwidth and/or time.

As an alternative, we show that the DLAA algorithm can be adjusted to operate in the presence of possible collisions of control messages. The control messages can thus be sent without special scheduling requirements and collisions can be handled by conventional selective acknowledgement and retransmission policies [15]. We shall therefore assume that a control message will eventually arrive in finite but undetermined time. It is important to note that we furthermore 1) do not assume that control messages arrive in the order in which they are sent and, 2) we do not require a stop and wait protocol for transmission of control messages [15].

In the following discussion we separate the handling of loss of PERMISSION, OWNVERSION and WELCOME messages - which need no special handling by the DLAA algorithm, from the handling of loss of WAKEUP messages that require a process of detection and correction of potential deadlocks.

6.1 Loss of PERMISSION, OWNVERSION and WELCOME Messages

We first prove that the algorithm continues to be correct when loss of PERMISSION, OWNVERSION and WELCOME messages occurs.

Theorem 6.1.1: Loss of PERMISSION messages does not cause deadlock or inconsistency.

Proof of Theorem 6.1.1: Assume that a PERMISSION message sent from node \( i \) to node \( j \) is lost. Node \( j \) cannot execute the link allocation prior to the reception of
a retransmission of that PERMISSION message. (An allocation process is executed only when PERMISSIONs are received from all neighboring nodes.) The lack of information which was included in the lost message has no effect on the relevant data needed by node \( j \) in order to send a PERMISSION message to a neighboring node - say node \( w \). The information needed by node \( w \) prior to its execution of the link allocation in order to preserve the consistency property consists of the slots where node \( j \) is a Sending Node (S+NR), Receiving Node (R+NS) or prohibited from receiving (NR). This information is transferred by OWNVERSION messages received at node \( j \) and thus an inconsistent allocation at node \( w \) will not occur. It is obvious that no deadlock will result from loss of a PERMISSION message since the message will be eventually received. The reception of the retransmitted message will enable the local process to continue.

Q.E.D.

**Theorem 6.1.2:** Loss of OWNVERSION messages does not cause deadlock or inconsistency.

**Proof of Theorem 6.1.2:** A node cannot send another PERMISSION nor can it execute the link allocation process as long as the previous OWNVERSION message has not arrived. Thus, no deadlock nor inconsistency will occur.

Q.E.D.

**Theorem 6.1.3:** Loss of a WELCOME message does not cause deadlock of inconsistency.

**Proof of Theorem 6.1.3:** A destination of a WELCOME message is a NEW node that cannot send any message (PERMISSION or OWNVERSION) prior to the reception of the WELCOME message. Given the assumption that each node is always aware of its neighboring nodes, the node will be in a waiting state until the WELCOME message eventually arrives.

Q.E.D.
6.2 Loss of WAKEUP Messages

We first demonstrate that the loss of WAKEUP messages in the DLAA algorithm as given in Section 4 can result in deadlock. Therefore, if WAKEUP control message delivery is not guaranteed the DLAA algorithm must possess a mechanism for detection and recovery from this situation. An example for a possible deadlock is shown in figure 6. The list of events that can lead to it is:

1) WAKEUP (2,1); {arrives only at node 3}
2) PERMISSION (3,2,LOCAL_TDMA,CYCLE_SYNC);
3) WAKEUP (4,2);
4) PERMISSION (1,4,LOCAL_TDMA,CYCLE_SYNC);

**deadlock**

In this case nodes 2 and 4 cannot execute the link allocation process since they both lack a PERMISSION that will never arrive.

The Detection Process

In order to detect a potential deadlock, i.e. a state which may lead to a deadlock we extend the algorithm presented in Section 4. The proposed extension enables each node to detect situations which may lead to a deadlock and initiate a recovery.

On WAKEUP(j,p) do
begin
if (PREVIOUSLY_PERMITTED != null) and (pr < time-stamp
PREVIOUSLY_PERMITTED)
{the node whose request has just arrived has a higher priority
than the PREVIOUSLY-PERMITTED node!}
then
call CORRECTION_PROCESS
end

CORRECTION_PROCESS::

We introduce a DELETE PERMISSION control message sent by node v to the
PREVIOUSLY_PERMITTED node, say w, when a potential deadlock is detected. When
node $w$ receives a DELETE_PERMISSION message, it checks whether the link allocation process can be executed (PERMISSION_GRANTING=NEIGHBORS). If so, then no actual deadlock has occurred and an OWNVERSION message is sent in response. Otherwise, node $w$ sends a PERMISSION_DELETED message and deletes node $v$ from its local PERMISSION_GRANTING list.

In the following formal description we give only the additions required for the algorithm presented in Section 4.

CORRECTION_PROCESS: \{executed by node $v$\}

begin
  send DELETE_PERMISSION($v, w$)
  wait for:
    OWNVERSION ($w, w_{LOCAL_TDMA}$)
    do <execute as in Sec.4 'ON OWNVERSION' > end
  or
    PERMISSION_DELETED ($v, w$)
  do
    PREVIOUSLY_PERMITTED:=top (WAITING_FOR_PERMISSION)
    send PERMISSION ($v, PREVIOUSLY_PERMITTED, LOCAL_TDMA, CYCLE_SYNC$)
  end
end

On DELETE_PERMISSION ($id_1, id_2$) do

begin
  if $id_2=v$ then
    begin
      if (PERMISSION_GRANTING $\neq$ NEIGHBORS and STATUS=NEW) then {potential deadlock}
        begin
          PERMISSION_GRANTING:=PERMISSION_GRANTING id_1
          send PERMISSION_DELETED ($v, id_1$)
        end
    end
end

We next prove that the algorithm remains robust in the presence of the loss of DELETE_PERMISSION and PERMISSION_DELETED messages.

Theorem 6.2.1: Loss of DELETE_PERMISSION messages does not cause deadlock or algorithm inconsistency.

Proof of Theorem 6.2.1: A node sending a DELETE_PERMISSION message is involved
in a possible deadlock and thus cannot send any other messages prior to the reception of a response (a PERMISSION_DELETED or an OWNVERSION message). If a DELETE_PERMISSION message is lost and the alarm was a false alarm (no deadlock) an OWNVERSION message might arrive and be treated as a response. The sender of the OWNVERSION message becomes OLD and thus if it does receive the DELETE_PERMISSION message it will retransmit the OWNVERSION message. If this was not a false alarm then the process will be locally blocked until the reception of a retransmission of the DELETE_PERMISSION message. In both cases no deadlock or inconsistency results.

Q.E.D

Theorem 6.2.2: Loss of PERMISSION_DELETED messaged does not cause deadlock or inconsistency.

Proof of Theorem 6.2.2: A PERMISSION_DELETED message is sent from node \( v \) in response to DELETE_PERMISSION request sent by node \( w \) only after the deletion of node \( w \) from the PERMISSION_GRANTING list. Thus no allocation can be made by node \( v \). Node \( w \) is blocked until it receives a response from node \( v \). As node \( v \) cannot execute link allocation the only possible response is a PERMISSION_DELETED message. If a retransmission of a DELETE_PERMISSION is then received at node \( v \), node \( v \) resends a PERMISSION_DELETED message, thus inconsistency is prevented.

Q.E.D.

SUMMARY

This paper presented the Distributed Link Allocation Algorithm (DLAA). The algorithm forms a bounded, collision-free TDMA cycle which allows spatial reuse of the radio channel. It was proven that at all times the algorithm remains consistent and deadlock free under any combination and ordering of topological
changes. It was shown that each topology update is implemented in the TDMA allocation within linear time and according to the order of occurrence of the topological changes in the network. In this respect, the algorithm is fair to nodes, e.g., it allows new nodes to begin transmission in the order of their arrival to the network. The algorithm is completely distributed and requires that each node participating in link allocation need only the knowledge of its local information and information available by communication with its immediate neighbors.

Lastly the DLAA algorithm allows for continuous adaptation of the TDMA allocation to topological changes. Therefore, unlike in previous link allocation algorithms in which TDMA updates are executed at discrete intervals, in the proposed DLAA algorithm a node will not transmit "incorrectly" generating collisions in between consecutive updates and the data packet transmission process does not have to be periodically suspended for recomputation of the total network TDMA slot assignment. It is easy to see though that the adaptation process of the DLAA can also be executed at arbitrary discrete time intervals if so requested. Due to these properties we consider the DLAA algorithm to be well suited for mobile radio networks.
REFERENCES


An example demonstrating a conflict associated with two way allocation.

Arrows represent the intended allocation.
Figure 2a: An example of a fixed topology. Numbers in vertices correspond to the temporal node priority.
Figure 2.b: An example of a fixed topology. Numbers in vertices correspond to the temporal node priority. Numbers adjacent to edges represent the final timeslot assignment, with arrows showing the corresponding communication direction.
Figure 3 : Partial graph demonstrating the "chain problem", causing dead-lock.
Figure 4: An example leading to a possible dead-lock on loss of a WAKEUP-message.
<table>
<thead>
<tr>
<th>Slot 4</th>
<th>Slot 3</th>
<th>Slot 2</th>
<th>Slot 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEIGHBORS</td>
<td>LOCAL-ADMA</td>
<td>NEIGHBORS</td>
<td>LOCAL-ADMA</td>
</tr>
<tr>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
<td>STATES</td>
</tr>
<tr>
<td>7-NS</td>
<td>R(5)</td>
<td>NS(5)</td>
<td>R(10)</td>
</tr>
<tr>
<td>7-NS</td>
<td>R(8)</td>
<td>NS-NR(7,9)</td>
<td>NS-NR(7,9)</td>
</tr>
<tr>
<td>7-NS</td>
<td>R(8)</td>
<td>NS-NR(7)</td>
<td>NS-NR(7)</td>
</tr>
<tr>
<td>9-NS</td>
<td>R(9)</td>
<td>NS-NR(8)</td>
<td>NS-NR(8)</td>
</tr>
<tr>
<td>7-NS</td>
<td>R(7)</td>
<td>NS-NR(7)</td>
<td>NS-NR(7)</td>
</tr>
<tr>
<td>5-NS</td>
<td>S(6)</td>
<td>S(4)</td>
<td>S(6)</td>
</tr>
<tr>
<td>3-NS</td>
<td>S(5)</td>
<td>S(3)</td>
<td>S(5)</td>
</tr>
<tr>
<td>5-NS</td>
<td>S(5)</td>
<td>S(1)</td>
<td>S(2)</td>
</tr>
</tbody>
</table>

**TABLE 2**

<table>
<thead>
<tr>
<th>STATUS</th>
<th>NUMBER</th>
<th>STATUS</th>
<th>NUMBER</th>
<th>STATUS</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>1</td>
<td>NEW</td>
<td>2</td>
<td>NEW</td>
<td>1</td>
</tr>
<tr>
<td>OLD</td>
<td>3</td>
<td>OLD</td>
<td>4</td>
<td>OLD</td>
<td>5</td>
</tr>
<tr>
<td>NEW</td>
<td>6</td>
<td>NEW</td>
<td>7</td>
<td>NEW</td>
<td>8</td>
</tr>
<tr>
<td>OLD</td>
<td>9</td>
<td>OLD</td>
<td>10</td>
<td>OLD</td>
<td>-</td>
</tr>
</tbody>
</table>