A Route-Aware MAC for Wireless Multihop Networks with a Convergecast Traffic Pattern

Fabrice Theoleyre

A Route-Aware MAC for Wireless Multihop Networks with a Convergecast Traffic Pattern

Fabrice Theoleyre
CNRS - LSIIT
University of Strasbourg
Boulevard Sebastien Brant
F-67412 Illkirch Cedex, France
Email: theoleyre@unistra.fr

Abstract—The MAC layer for multihop wireless networks has drawn considerable research attention in the last few years. We focus here on the wireless multihop networks with a convergecast traffic pattern: the whole traffic is destined to a sink/gateway. We propose first to select a k-tree core, i.e. a sub-tree of the shortest paths to the sink containing exactly k-leaves. In particular, these k-tree core nodes are chosen among the nodes that must forward most traffic. We design C-MAC, an optimized MAC layer for this kind of topology. C-MAC is derived from the CSMA-CA like approaches and consists in giving a larger priority to the k-tree core nodes. Moreover, a proper coordination among the k-tree core nodes permits to limit collisions among them. Simulation results show that organizing the transmissions in C-MAC permits to achieve a much larger throughput than the original IEEE 802.11-like protocol it is based on. This simple solution can be adapted to most CSMA-CA like protocols, and is particularly relevant for WSN or WMN in which traffic is mostly destined to the sink/gateway.

MAC layer; convergecast traffic; k-tree core; CSMA-CA; route-aware MAC

1 Introduction

Multihop wireless networks are currently very popular and have been receiving a large attention of the research community. More specifically, many of these networks present a convergecast traffic pattern: all the packets are either destined or sent by the same entity.

Wireless Mesh Networks permit to offer a wireless connectivity while connecting only a subset of the Access Points to the wired Internet. Often, all the traffic passes through these gateways. Besides, Wireless Sensor Networks permit to instrument distributively an environment for house automation, smart buildings, sustainable development, etc. These networks are also often convergecast by nature: measurements are collected distributively by the sink.

The IEEE 802.11 technology has gained wide popularity when deployed in the infrastructure mode with an Access Point providing untethered connectivity to nomadic devices. The CSMA-CA mechanism has largely proven its efficiency to distribute the bandwidth among the stations in this kind of networks. With such a probabilistic approach, neither the number of packets nor the number of stations has to be fixed a priori. Moreover, the exponential backoff mechanisms allows the network to cope with a fluctuating number of contenders: the transmission probability will be self-adjusted. Indeed, if a collision occurs, the contention window is automatically doubled so that the traffic pressure is reduced.

However, there is a lot of evidence that the 802.11 MAC layer is not suitable for multihop wireless networks: the performance of packet forwarding over multiple hops quickly degrades with the number of hops due to channel contention and spatial problems such as hidden, exposed, masked, and blocked nodes [1], [2], [3]. In particular, the chain topology is very common in multihop: packets have to be forwarded toward the destination. However, collisions among the different forwarders can occur, degrading the end-to-end performances. MAC layers that adopt a similar approach in WSN (e.g. IEEE 802.15.4) suffer from the same drawbacks when the load approaches the network capacity. Besides, the nodes close to the gateway/sink have often more packets to forward, and a bottleneck quickly appears around it [4].

On the opposite side, TDMA scheme permits a fine scheduling and limits bandwidth wastage. However, it requires a very tight synchronization and does not cope with variations: TDMA slots are assigned in advance, even when the traffic load changes frequently. Besides, to compute an acceptable scheduling while enabling radio reuse requires a complete knowledge of the radio environment: the scheduler must know the sets of links that interfere with each other. Consequently, one can adopt e.g. a centralized scheduler with SINR constraints [5] or a distributed slot assignment scheme with an a priori knowledge of interferences [6]. However, to measure the level of interferences among radio links is a complicated task in multihop networks, which requires to measure atomically the interferences [7].

We propose here to adopt an innovative solution, mixing the assets of both approaches. In C-MAC (Convergecast-MAC), most nodes execute the classical CSMA-CA algorithm to transmit their packets. Oppositely, we *organize* the transmissions of the most-constrained nodes. Instead of

adopting a TDMA scheme which would require a global synchronization, we propose distributed MAC reservations, gateway-oriented: control frames that reserve the medium propagate from the gateway to the borders of the network, through the most constrained nodes.

This approach has the following promising assets:

- C-MAC extends the NAV of IEEE 802.11 by just introducing a new type of control frame. It is interoperable with the classical IEEE 802.11;
- it operates without any synchronization requirement;
- we jointly optimize routing and MAC by constructing a structure to collect the traffic: a *k-tree core*;
- we present a multichannel variant to multiplex transmissions across different channels. This feature limits greatly the number of collisions.

2 Preliminaries

We focus in this article on multihop wireless networks with a convergecast traffic pattern: one single node, i.e. the *gateway*, is either the source or the destination of each packet transmission. If the network comprises several gateways, we will consider individually each gateway and all its associated nodes, i.e. we will neglect the impact of nodes with frequent gateway changes. Moreover, we will focus here on the MAC layer: the optimal gateway selection, the encapsulation scheme, etc. are out of the scope of this article.

2.1 IEEE 802.11 Behavior

IEEE 802.11 is widely deployed to provide a wireless access in hotspots. The CSMA-CA technique it implements is efficient to share the bandwidth among an unknown number of stations. Thus, we will shortly describe the basic functions of CSMA-CA protocol, and more particularly of IEEE 802.11 [8], on which we based the protocol presented here.

In IEEE 802.11, each transmitter chooses a random backoff comprised between 0 and the contention window value (CW)[1]. If the medium is idle for more than DIFS time, it starts to decrement the backoff. As soon as the medium is detected busy, the backoff is paused and it will be redecremented when the medium becomes idle (still after the DIFS time). When the backoff is null, the node starts to transmit its `data` frame. The receiver decodes the `data` frame, waits for SIFS and sends an acknowledgement. If the source receives an `ack`, it considers the frame was correctly received. Else, it estimates a collision occurred: it doubles the contention window so that the backoff will probably be larger for the next retransmission. This exponential backoff helps to cope with a variable number of transmitters. Finally, when a node has correctly transmitted a frame, it reinitializes the contention window to the minimum value. This approach helps to converge to an accurate Contention Window value.

IEEE 802.11 is robust to collisions and performs well in cellular networks. [9] studied the optimal contention window value to almost avoid all collisions. However, IEEE 802.11 performs quite poorly in multihop [1]. Thus, we aim here at optimizing the IEEE 802.11 approach in networks with a convergecast traffic pattern.

1. More precisely, backoff $\in [0..2^{CW} - 1]$

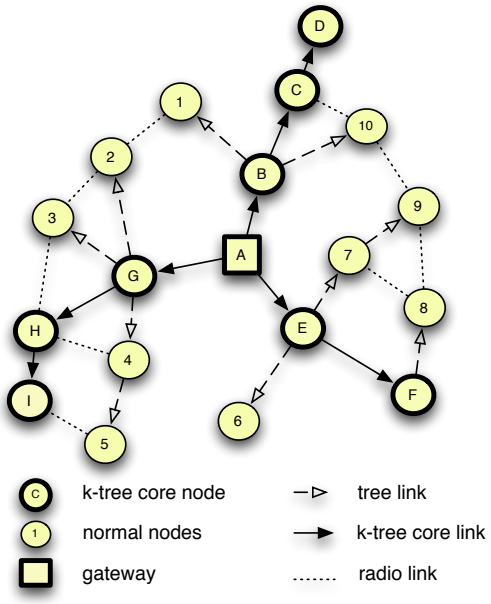


Fig. 1. Tree structure of C-MAC (here a 3-tree core)

3 C-MAC General Description

We are convinced that jointly optimizing the routing and MAC layers seems a promising way for convergecast wireless networks. Since all the traffic is destined or generated by the gateway, we can *self-organize* the transmissions to avoid the classical bottleneck around the gateway.

C-MAC uses a reservation-oriented mechanism: a node becomes *privileged* during a small duration. We will distribute this privilege mode only to most constrained nodes so that no pair of nodes privileged simultaneously are interfering with each other.

We focus on convergecast networks, in which the routes form a tree rooted at the gateway. Thus, C-MAC uses this tree to distribute the *privilege modes*: it sends a *token*, which is forwarded hop by hop toward the leaves. When a node receives the token, it becomes privileged during a short time.

We consider now the example illustrated in figure 1. Each node chooses as parent the next hop toward the gateway, which forms a spanning tree, rooted at the gateway. Besides, we selected in this example 3 different branches, starting from the gateway. Each branch is linear: a node has only one child, except the gateway. Only the nodes of these branches can become privileged after receiving the token. When it has finished to transmit its `data` frames, the node will then forward the token to its child, toward the leaves.

If the gateway sufficiently inter-spaces the tokens it generates, we can avoid interferences among *privileged nodes*. Indeed, the previous token will have been forwarded sufficiently far to avoid collisions.

4 Tree Construction

C-MAC aims at enabling the most constrained nodes to gain a privileged access to the radio medium while limiting interferences. Thus, a token can be forwarded only to one

node, i.e. along one single branch. Moreover, only the nodes part of the tree will be able to gain a special access to the medium. Consequently, we would like to obtain a tree with the following characteristics:

- to limit interferences among and inside the branches, they should be as linear as possible (i.e. we should avoid zigzag);
- we should bound the number of branches in order to limit the time a branch has to wait before receiving the next token;
- we want to minimize the distance of *normal* nodes to the tree, i.e. the packets should be forwarded mostly by the nodes that can become privileged.

The k-tree core structure was originally introduced in [10] and is particularly convenient for our purpose. Intuitively, the k-tree core proposes to extract from a tree the k leaves that minimize the average distance from a node to the closest k-tree node.

More formally, $l(T)$ denotes the number of leaves in the tree T and $d(A, B)$ denotes the distance in hops between nodes A and B. We also define the distance from one node N to a set $\mathcal{S}$ as the distance from N to the closest node in $\mathcal{S}$:

$$d(N, \mathcal{S}) = min_{M \in \mathcal{S}} \left(d(N, M) \right) \quad (1)$$

If KT is a set of k-tree core nodes associated to the tree T, the following definition holds:

$$KT \subset T \quad (2)$$

$$l(KT) = k \quad (3)$$

$$Objective: \ min \sum_{N \in T} d(N, KT) \quad (4)$$

[10] proposes a centralized algorithm to select a k-tree core from any tree topology. The nodes report to the root of the tree the most convenient leaves. Intuitively, we must select the best k branches to form the k-tree core. To achieve this objective, each node computes a metric representing the suitability to select its branch (from it to the best leaf). This metric is computed from the leaves to the root. When the root obtained all the metrics from its children, the best branch can be selected. Eventually, this process can be repeated k times although some optimizations exist to reduce the convergence delay.

We will now explain how we construct such a k-tree core for C-MAC.

4.1 Approach

In existing approaches, the k-tree core is selected when the graph is already a tree. We adopt consequently the following heuristic:

1) We construct first the tree of shortest paths towards the gateway. The k-tree core aims at minimizing the distance to the tree. Thus, a node chooses preferentially as parent the node with the largest subtree size if several candidates are available. Intuitively, if this node is chosen as a k-tree node, it will save more hops for *normal* nodes;
2) We select then the k leaves that minimize the average distance to the k-tree core. Besides, we force the root to be part of the final k-tree core.

We will now detail the construction process.

4.2 Tree Construction

Algorithm 1: Self-stabilizing Tree construction

```
1  /* My parent                                   */
2  myParent ←∅ ;
3  /* The subtree size of my parent               */
4  myParentStSize ←0 ;
5  /* My sequence number                          */
6  mySeqnum ←0 ;
7  /* My distance to the root/gateway             */
8  myDist ←MAX;
9  while hello received do
10     /* Extract info from hello                 */
11     h_id, h_stSize, h_seqnum, h_dist
       ←extractInfo(hello) ;
12     /* To detect tree disconnections           */
13     if myParent = h_id then
14         mySeqnum ←h_seqnum;
15     /* Tree is disconnected: seqnum did
          not change for Δ_seqnum seconds          */
16     if noChange(mySeqnum, Δ_seqnum) then
17         myParent ←∅;
18         myDist ←MAX;
19     /* To keep shortest routes to the
          gateway                                  */
20     if (h_dist + 1 < myDist) or
21     (h_dist ≤ myDist and h_stSize > myParentStSize)
       then
22         myParent ←h_id;
23         myDist ←h_dist + 1;
24         myParentStSize ←h_stSize;
```

We first construct a self-stabilizing tree as described in algorithm 1. All control information is piggybacked in `hello` packets. In particular, the gateway sends its `hellos` with a strictly increasing sequence number. A node propagates the largest sequence number received from its parent. In this way, we are able to detect tree disconnections when the sequence number does not increase for a sufficiently long time (Δ_{seqnum} seconds). In this case, a node just reinitializes its parent and its distance to the gateway.

Besides, a node changes its parent when a neighbor announces a smaller distance to the gateway. The node just switches its parent and updates both its sequence number and distance to the gateway.

Several neighbors may announce the same distance to the gateway. In this case, a node chooses as parent the node which announces the largest subtree size. This will optimize later the k-tree core.

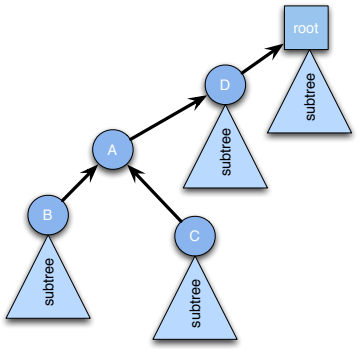


Fig. 2. Savings to find a k-tree core

Each node updates its subtree-size when it receives an `hello` from one of its children: a node simply sums up the cardinalities of the subtrees of its children.

The reader can note that the tree will be correctly maintained, even if one radio link or one node disappears: each node will monitor the tree connectivity and it will choose another parent if a starvation in the sequence number is observed.

The worst case corresponds to a linear topology: during one `hello packet` period, only one node is added to the tree. We denote by D the network diameter, and by n the number of nodes. The time complexity of algorithm 1 is $O(D) = O(n)$.

4.3 K-tree Core Selection

We elect the k-tree core as highlighted in algorithm 2. Similarly to [10], we use the concept of *savings*. The k-tree-core must be chosen to minimize the average distance to the closest node of the core. $saving(leaf, root)$ represents the number of hops *saved* if the path $(leaf, root)$ is a core compared to the average distance to the $root$. A larger saving means that we should choose the path $(leaf, root)$: it will minimize the number of transmissions exterior to the core.

Let's consider the example illustrated in figure 2 to explain intuitively how to compute savings. When A has to compute its saving value, it extracts the savings of its children B and C. Let's imagine that B has the largest saving. A will update its own saving to the sum of $saving(B, root)$ and the size of the subtree rooted at A. Intuitively, if A is chosen in the core, all its descendants will *save* one transmission through the link (A, D).

The reader can consequently remark that this saving value as introduced in [10] can be computed distributively.

In parallel, each node reports in its `hellos` the k best savings to select the best k leaves. More precisely, the savings are computed as follows (algo 2):

1) A node N orders all the savings announced by its children;
2) The largest saving value is updated taking into account the fact that N will become part of the k-tree core;

Algorithm 2: k-Tree Core savings computation

```
1  /* The list of my savings                    */
2  mySavingList ←∅ ;
3  while hello is received do
4      /* Extract info from hello               */
5      childId, savingList ←extractInfo(hello);
6      /* Replace the savings of this child
          in my neighborhood table              */
7      if IsChild(childId) then
8          flushSavings(childId);
9          for saving ∈ savingList do
10             mySavingList ←mySavingList ∪ {childId,
               saving };
11     /* Extract the savings to export:
          order them, and update the best
          one with what I would save by
          myself                                */
12     orderBySavings(mySavingList);
13     savingExport ←mySavingList[0] + mySubTreeSize;
14     for i ∈ [0..k − 1] do
15         savingExport ←savingExport ∪ mySavingList
           [i];
```

3) N piggybacks in its `hellos` this new value accompanied with the $k-1$ other largest values, forwarded as is.

The savings are correctly computed, according to lemma 3.3 [11].

The root (i.e. the gateway) has to be part of the k-tree core since it will forward all the traffic in up/download. Thus, it initiates the creation of k branches by selecting the k largest savings values among its children. The root associates to each branch a `branch id` and the associated `neighbor id`. This information is piggybacked in `hellos`: each node verifies if it has been selected by its parent. If this is the case, it extracts the corresponding `branch ids`. Then, it chooses the i^{th} largest saving value for the i^{th} branch for which it has been selected. The corresponding children are announced and they prolongate the branches in the same way.

Savings are reported from the leaves to the root while elected k-tree nodes are reported in the inverse direction. In the worst case, the tree depth (denoted T) is equal to the number of nodes (n). Thus, the time complexity for computing savings and electing k-tree nodes is in $O(T) = O(n)$.

Figure 3 illustrates a k-tree core with 3 branches. Thus, each node reports the 3 largest savings. The leaves (e.g. nodes B or F) report a unique saving of 1 (themselves). On the contrary, G has several children. It reports the largest saving (from H) incremented y its subtree size ($7 + 6$) and the 2 other savings from K and M without modification. Finally, the root A is able to select the 3 largest savings to create the branches. In particular, node G is chosen twice since it announces 2 of the largest savings. The k-tree core

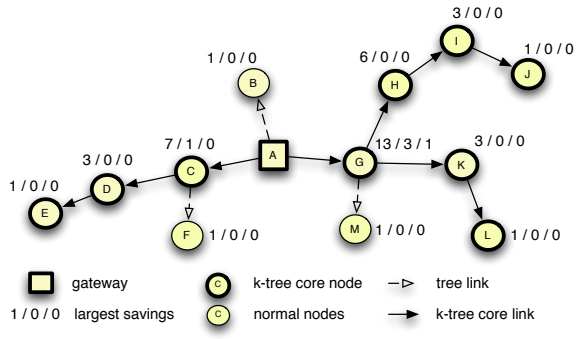


Fig. 3. K-tree core selection and savings computation (3 branches)

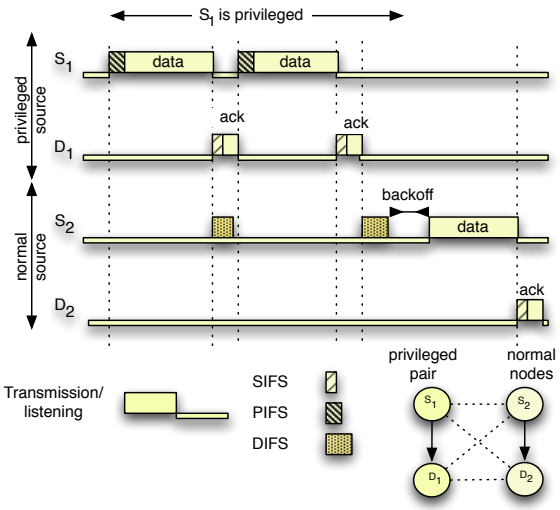


Fig. 4. MAC access for privileged nodes

is then propagated, each node selecting its children. In particular, G has been selected for 2 different branches and has to select the two largest savings (i.e. corresponding to nodes H and K).

4.4 Properties

We can remark that the algorithm is deterministic but relies on unreliable transmissions. For instance, a node may have two neighbors, closer to the root and with the same subtree cardinality. In that case, unreliability for `hello` transmissions will conduct the node to choose pseudo-randomly one of these nodes as parent.

5 C-MAC Mechanisms

We propose here to use the tree constructed previously to regulate the transmissions in the network: we focus on the k-tree core nodes since they will carry, by construction, most of the traffic. In particular, we give them a privileged access to the medium. A reservation mechanism, propagated by the gateway, limits interferences among these constrained nodes.

5.1 Normal Nodes

Any node which is not part of the k-tree core executes the normal CSMA-CA algorithms, as in IEEE 802.11. Thus, it has to wait a DIFS before decrementing its backoff. It may either use a `RTS-CTS` to limit the hidden terminal problem if the frames are long or directly transmit them.

We adopt also the exponential backoff algorithm of IEEE 802.11 to cope with different densities. The contention window is doubled upon a collision to reduce the collision's probability.

5.2 Medium Access for Privileged Nodes

The most-constrained nodes become iteratively privileged: they have the largest priority to transmit their frames. Since we control the number of nodes privileged simultaneously, we can grant an immediate medium access without contention to these nodes.

To be compatible with IEEE 802.11, we re-use the PIFS value, dedicated originally to the Point-Coordination-Function of IEEE 802.11 when the Access Point polls the stations. Thus, a privileged nodes has only to wait for PIFS after it detects the medium is free. Since all the other nodes have to wait for DIFS, the privileged node will always win for the contention. Besides, the privileged node will not break a transmission since PIFS is superior to SIFS: it will wait the current exchange among another pair of nodes is finished.

Let's focus on the example described in figure 4. We assume that two nodes S_1 and S_2 are backlogged (i.e. they have a large buffer of `data` to transmit). When S_1 becomes privileged, it just has to wait PIFS before gaining the medium access. It will eventually send several data frames before finishing to be privileged, waiting every time PIFS. When S_1 is not privileged anymore, it is authorized to transmit any `data` frame. Thus, S_2 will be able to sense an idle medium during DIFS and its backoff: it will then transmit its `data` frame. Such a mechanism is efficient because we choose the most-constrained nodes to be privileged. Other nodes have less traffic to forward and we do not need to regulate their transmissions: the IEEE 802.11 mechanism works well.

The reader can note that C-MAC can profitably acknowledge data in bursts when a node is privileged. D_1 would send an `ack` only after receiving the second data frame. However, we did not implement this option for the performance evaluation to have a fair comparison with IEEE 802.11.

5.3 Privilege Forwarding

C-MAC regulates the medium access by focusing only on most-constrained nodes. These constrained nodes can become privileged to have an exclusive access to the medium. However, we must limit interferences among nodes becoming privileged simultaneously. Besides, we argue a TDMA-like approach presents too strong limitations compared to CSMA-CA approaches. Firstly, it requires a fine grained synchronization and a guard time has to be reserved to avoid collisions among the different slots. Besides, the

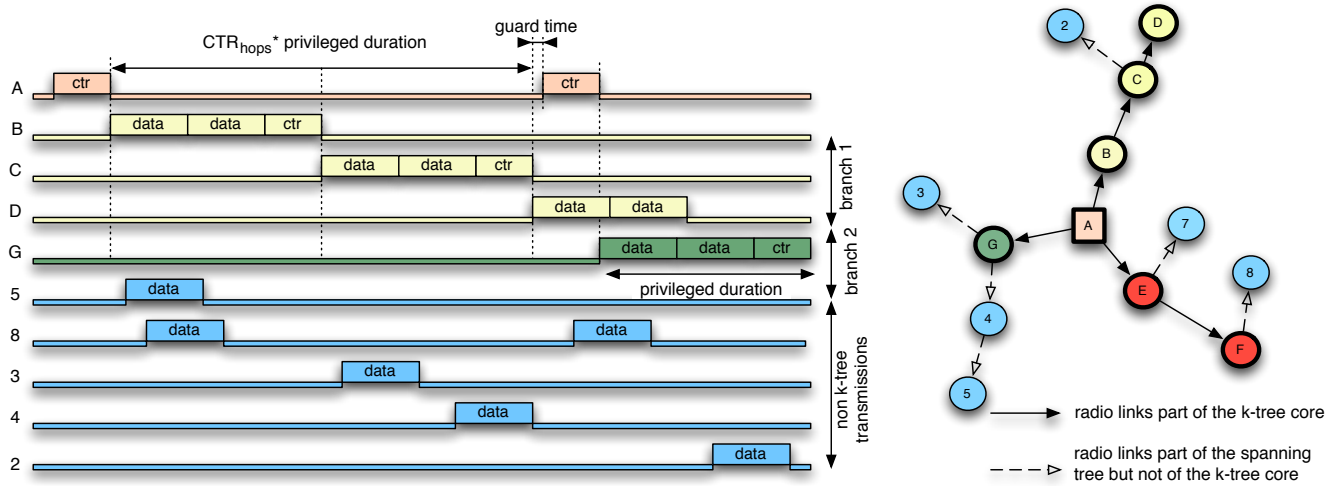


Fig. 5. CTR forwarding and data transmissions in upload (nodes → root) with $CTR_{hops} = 2$

scheduling should be derived from the contention graph, which is usually not trivial to compute. A TDMA-like approach presents also a low flexibility: the scheduling has to be entirely recomputed when the traffic or the conditions vary. Finally, we aim at keeping the compatibility with the classical IEEE 802.11 approach.

Thus, we will use the convergecast nature of the network: the gateway is the root and can distribute the bandwidth to its neighbors, and more specifically to its neighboring branches. Thus, we chose to use a new control frame we call `Clear-To-Receive` (`CTR`). This `CTR` acts as a token to become privileged and should *reserve* the medium for the newly privileged node. Consequently, the `CTR` acts like the `Clear To Send` of IEEE 802.11: destination has only to wait for PIFS before transmitting its data frame. Using PIFS instead of SIFS permits to limit collisions with ongoing transmissions while having a larger priority than other nodes (using DIFS). When the gateway sends a `CTR`, it will gain access to the medium and will not release it when its neighbor become privileged: the medium is not idle for a duration equal or longer than DIFS.

Besides, this `CTR` acts as token and is forwarded along the branches of the k-tree core. A node has to forward the `CTR` when it remained privileged for a duration longer than T_{slot}. Thus, each `CTR` constitutes a kind of wave. Each wave propagates along a branch and dies when it reaches the leaves of the k-tree core. If the gateway inter-spaces sufficiently the `CTR` it generates for each branch, it can avoid the interferences. Thus, a gateway generates a new `CTR` for another branch when the other one is estimated CTR_{hop} apart (i.e. it has to wait $CTR_{hop} * T_{slot}$).

Let's focus on the example in figure 5. We reported the subset of nodes in fig. 1 that interests us in this example. We assume that the gateway generates a new `CTR` for a different branch every 2 slots. In this case, the `CTR` is forwarded along the first branch, and the sink generates a new `CTR` after C ends its privileged slot. We did not represent the `ack` for a sake of clarity.

Two k-tree core nodes can be privileged simultaneously if they are sufficiently far from each other (more than 2 hops). For instance, D and G can transmit their data frames to their parent without collision.

We represented possible concurrent transmissions from non k-tree nodes on the lower part of the figure. A privileged node maintains the reservations made by the `CTR`: all the frames are interspaced by either PIFS or SIFS. Thus, a *normal* transmission can take place if no other privileged transmission already reserved the medium. For instance, the node 5 can send a data frame to its parent 4 during the privileged slot of B without collision: no signal is sensed since B is assumed to be out of interference range.

Transmissions are unreliable although a packet loss is very prejudicial for `CTR`: the whole *privileged slot* is conditioned by the correct reception of this packet. Thus, a `CTR` must be acknowledged by:

- a `data packet` from the destination: the next hop has data packets and starts its transmission because it has received the `CTR`;
- a `CTR`: if the next hop has no data to transmit, it forwards without delay the `CTR` to its own child. Thus, this implicitly acknowledges the previous transmission.

Collisions may occur between different k-tree nodes if CTR_{hops} is too small or if two interfering k-tree nodes are privileged simultaneously. These repeated collisions would also create `hello` packet losses, triggering tree and k-tree core reconfigurations. Meanwhile, a k-tree node implements a classical retransmission strategy: it tries to retransmit its packets `long retry` times, as in IEEE 802.11. During the first `short retry` retransmissions, the k-tree node transmits directly after PIFS, and for the last retransmissions, it chooses a random backoff after having waited PIFS: we would solve contention among k-tree core nodes. If this is not sufficient, the node drops the packet (either a `CTR` or a `data packet`) and consider that the privileged slot is lost.

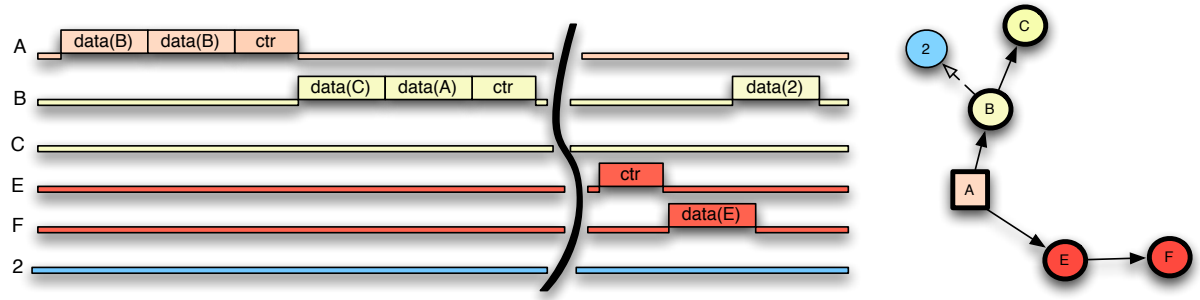


Fig. 6. CTR forwarding and data transmissions in download (nodes ← root)

Thus, one of the two competing k-tree nodes will *win* and *keep* the privilege token/slot. We limit unfairness because the retransmission mechanism is based on a pseudo-random strategy.

5.4 CTR self-adaptation

The reader can note that the gateway can regulate the bandwidth assigned to each branch. Indeed, it can fix a different T_{slot} value for each branch, reserving *de facto* a different bandwidth for each of them. However, we let an adaptive algorithm fixing dynamically these values to a future work.

In the same way, the gateway could be able to test different values for the CTR_{hops} parameter. After having measured the achievable throughput, the gateway could be able to adjust dynamically the value. In particular, when the CTR_{hops} is too small, many collisions would occur between k-tree nodes during their *privileged time*. These collisions will negatively impact the throughput, and CTR_{hops} would be re-increased. The AIAD (Additive-Increase Additive-Decrease) method could be particularly interesting to find dynamically the optimal value.

5.5 Upload/Download Transmissions

In a convergecast network, the traffic can follow two directions: in upload (from the nodes to the root) and in download (from the root to the nodes). C-MAC can deal with both cases.

Obviously, non-k tree nodes follow the classical IEEE 802.11 approach. Thus, they are able to transmit to their parent and children in a similar manner, without any specificity.

For k-tree nodes, C-MAC operates in the following manner:

upload: this case is illustrated in fig. 5. Just after having received the CTR from its parent, a node transmits all its buffered packets to its parent;

download: this case is illustrated in fig. 6. When node B receives a CTR, it can transmit its packets to C. If necessary, B can use an RTS/CTS to avoid the hidden terminal problem: neighbors of node C may not be aware of the existence of a privileged slot. This reservation is then active for the whole burst. In our case, node B mixes download (to C) and upload (to A) directions in the same privileged slot. When a k-tree node has to forward a packet to a *normal* node, it uses the non privileged mode: these transmissions are not part of the k-tree core. For instance, B will send a data packet to the node 2 using the normal CSMA-CA mode after having chosen a random backoff (with the usual DIFS).

To avoid unfairness, a k-tree node has to share equally the bandwidth between the up and download directions: a node should forward roughly as much traffic as it receives.

5.6 Forwarding Delay

For non-k tree nodes, packets are forwarded according to the classical CSMA-CA mechanism. Thus, the delay remains unchanged compared to IEEE 802.11.

On the contrary, k-tree nodes can only forward packets after having received a CTR. To avoid any synchronization requirement, each k-tree node forwards its CTR to its child. In other words, we create a kind of precedence along the tree. Thus, the forwarding delay is different according to the direction in the k-tree:

download: the *privileged slots* are directly consecutive. Thus, a node has to wait at most T_{slot} (i.e. the end of its slot) before forwarding the data packets.
For instance, the node B receives its packets in the privileged slot of A and forwards them to C just after that (fig. 6);

upload: we face an inverse situation. A node receives the data packets from one child just after having forwarded to it the CTR. Thus, it has to wait for the next CTR from its parent. Thus, it has to wait exactly $CTR_{hops} * privileged\_duration * nb\_branches$.

This buffering delay in upload also exists when a k-tree node receives a data packet from its children not present in the k-tree core. In particular, each k-tree node will buffer all the data packets received between two CTR (i.e. $CTR_{hops} * privileged\_duration * nb\_branches$ seconds). As a side effect, C-MAC is consequently efficient to aggregate the data packets from its children.

The reader can also remark that the the data packets of non-k tree and k-tree nodes can be aggregated in the same manner. When a node receives packets from one k-tree child, it waits for the next CTR before forwarding them. This gives the occasion to receive the other packets from its other non-k tree children. In figure 5, data packets from 2 and D can be aggregated for the next transmissions of C.

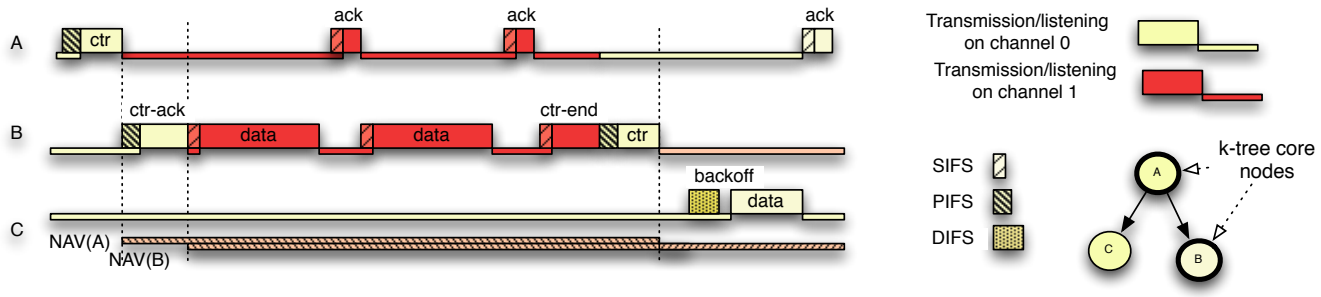


Fig. 7. C-MAC behavior in multichannel

5.7 Multichannel Extension

Although C-MAC was not conceived for this specific purpose, it can be very easily adapted to work in a multichannel environment. Indeed, all the transmissions use by default a common channel (e.g. channel 0) while transmissions of privileged nodes can use different orthogonal channels. If strictly more than 2 channels are available, different channels are used for different branches: `CTR` can be generated safely more frequently. The gateway mentions the channel id used for the privileged transmissions directly in the `CTR`.

More precisely, the `CTR` are transmitted over the common channel. Let's focus on the example described in figure 7. The node A forwards the token by sending a `CTR`. It will automatically switch to the privileged channel to receive frames. The node B first acknowledges the `CTR` by a `CTR-ACK` and then switch to the reserved channel for sending its `data` frames. The data frame exchange takes place on this interference-free channel. Thus, the node B can use safely SIFS for all its transmissions since it is alone to transmit. Besides, each node maintains a Network Allocation Vector (NAV) per channel as highlighted in figure 7. In this example, the node C is aware of the ongoing transmission and will send its data frame to A only when the reservation elapsed, i.e. it extracted the T_{slot} value from the `CTR`.

Since the `CTR` can suffer from collisions or bad transmissions, the source of the `CTR` may retransmit it after switching back to the common channel. The `CTR` retransmission process is identical to the single-channel case. This mechanism maximizes the probability that the token is correctly forwarded along the branches.

The reader can note that the performances of C-MAC will not improve for more than $CTR_{hop} + 1$ channels. Indeed, the gateway may act as a bottleneck since it is blocked for the whole T_{slot} duration after sending a `CTR` (i.e. it cannot transmit simultaneously two `CTR` on two different channels). Moreover, a new `CTR` can be safely generated on the same channel when the other one is CTR_{hop} apart. With the additional common channel, this gives the aforementioned limit. However, we could cope with this particular situation by having multi-radio gateways. This would relax this constraint.

6 Integration

We will now describe how C-MAC can efficiently cohabit with the other protocols present in the multihop network.

6.1 MAC Layer Independency

Although we used IEEE 802.11 to explain the C-MAC approach, the philosophy of C-MAC can be implemented with many other CSMA-CA approaches. Exhaustivity is impossible, but we will here describe how C-MAC can cohabit with various solutions in Wireless Sensor and Mesh Networks.

Busy Tones can be used to limit the hidden terminal problem [12]. They permit to notify the transmissions in the vicinity. C-MAC can be easily adapted to use for instance busy tones in reception and/or emission.

In WSN, we can also add a preamble [13] before each transmission to wait for the destination to wake up. For k-tree node, a preamble is required to transmit the `CTR` or a `data packet` to one child. Then, other packets can be transmitted without preamble since the destination will wait for the end of the *privileged slot*.

C-MAC would also be useful when only short preambles are used, such as in Single Channel Polling (SCP) [14]. To reduce power consumption, nodes agree on a schedule to wake-up synchronously: shorter preambles are sufficient to cope with clock drifts. However, the nodes may have buffered packets when they slept. This would lead to a traffic storm when all nodes wake-up. C-MAC could be implemented when all the nodes are awake to regulate the transmissions, during the active period.

6.2 Energy Consumption

C-MAC was not designed to save energy but rather to optimize the network capacity. Thus, it consumes similar energy as other CSMA-CA protocols. However, C-MAC can be easily adapted to be integrated with other solutions saving energy in WSN.

As highlighted previously, C-MAC can be adapted to work with various other MAC layers. In particular, the preamble sampling techniques permit to save energy efficiently.

In the same way, topology control solutions permit to choose the most *accurate* neighbors. By choosing to use the most economical radio links, the network can save energy

TABLE 1
Simulation parameters – default values

Bit rate	11Mbps
Packet reception threshold	-86dBm
Transmit power	5mW
Frequency	5Ghz (5180→5805)
RTS/CTS	inactive
Packet size	128 bytes
Privileged duration	5ms
CTR inter-spacing	3 hops
Number of branches for the k-tree core	5

globally. Thus, C-MAC can perfectly be executed after a topology control algorithm has been executed. In this way, C-MAC would use *efficient* radio links.

Finally, C-MAC could also use a metric based on *energy* to choose the radio links in the tree. In that way, C-MAC would use only most thrifty transmissions.

7 Performance Evaluation

We have simulated C-MAC in OPNET with the parameters presented in Table 1.

We have compared the performance of our proposal with the standard IEEE 802.11 DCF, one representative CSMA-CA protocol. We conjecture that the mechanisms of C-MAC could be adapted to any CSMA-CA approach, when the traffic to forward is large (e.g. when nodes wake up simultaneously to transmit data frames after a long sleeping duration or after having detected an event).

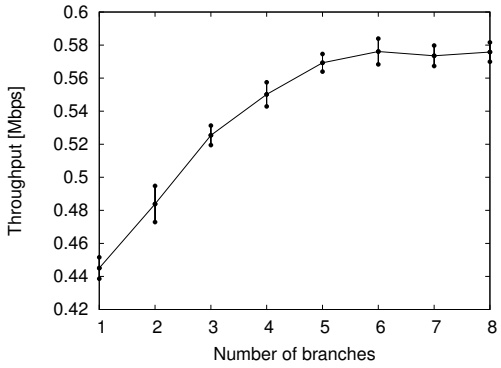
We did not compare C-MAC with a TDMA-like approach since such a solution would require a global synchronization. Besides, an efficient conflict-free scheduling algorithm must be implemented to organize the transmissions. This approach would require to know a priori the conflict-graph which is practically a difficult problem. However, we plan to experiment in the future C-MAC and compare it in realistic conditions to a TDMA-like approach, adopting for instance the approach described in [15].

We first evaluate the impact of the parameter's values in C-MAC in a grid network of 7x7 nodes, the sink being located at the center of the simulation area. Then, we focused on random circular topologies of nodes to generalize these results. Data traffic consists of several constant-bitrate (CBR) flows, their rate being represented in the figures below as the offered load in packet per second (pps). Each node transmits a CBR to the central sink, to represent a convergecast traffic pattern. We focus on the upload direction since the download direction offers less constraints in fairness and delay with C-MAC.

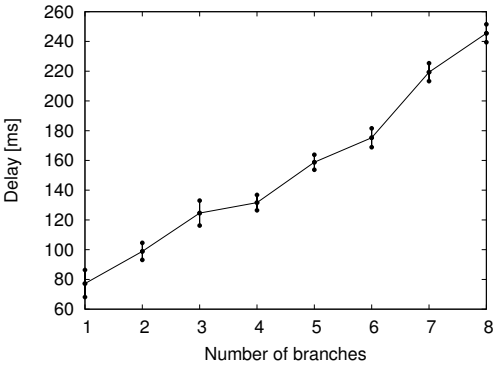
We have averaged the results presented below over several different simulation runs and have plotted the 95% confidence intervals. We have run simulations with and without the RTS/CTS option and obtained results that are not significantly different, so we have decided not to represent them in the figures.

We have evaluated the performance of three MAC layers according to three metrics:

1) *End-to-end delay*: the delay between packet generation and its reception by the final destination;



(a) Throughput vs number of branches



(b) Delay vs number of branches

Fig. 8. Regular grid of 49 nodes, 13 pps – *impact of the number of branches in the k-tree core*

2) *Aggregated throughput*: the volume of all received data in the network per unit time in Mbps;
3) *Jain index* defined as:

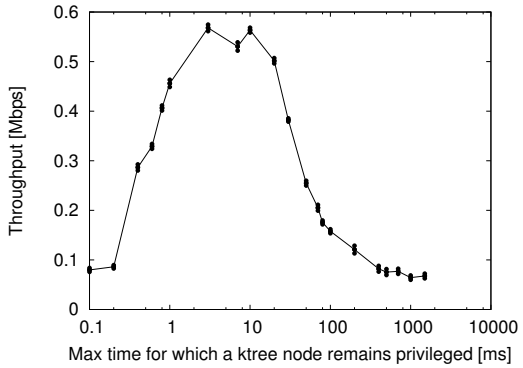
$$\frac{(\sum_{i=1}^{n} X_i)^2}{n \sum_{i=1}^{n} X_i^2}, \tag{5}$$

where X_i is the throughput obtained by flow i, measures throughput fairness of different flows in the network. A low Jain index means poor fairness;
4) *Overhead*: the ratio of the volume of control and data packets transmitted per node and the volume of `data` packets received by the sink. The volume is measured either in number of packets or in bits.

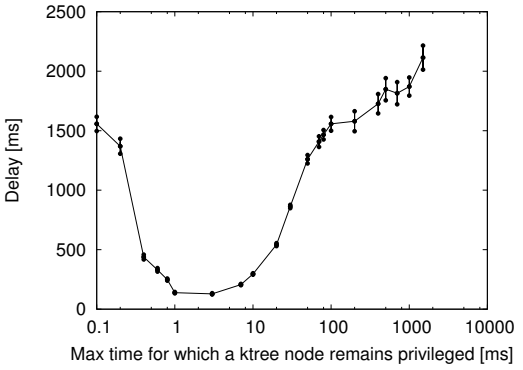
7.1 Tuning C-MAC Parameters

First, we measured the influence of the number of branches in the k-tree core (i.e. the k value) for a grid of 49 nodes. We chose an offered load of 13 pps so that the network operates in saturation: all the generated packets cannot be delivered to the sink. We are consequently able to study the impact of the C-MAC parameters in saturated mode.

Figure 8(a) illustrates the throughput for different k values. When the k-tree has one single branch (i.e. it is a *core*), the throughput is minimal: C-MAC does not achieve to balance the load in the network, and some data packets are dropped. On the contrary, more branches permit to forward more packets while avoiding collisions. However, it also increases the delay (fig. 8(b)). Indeed, the sink generates a

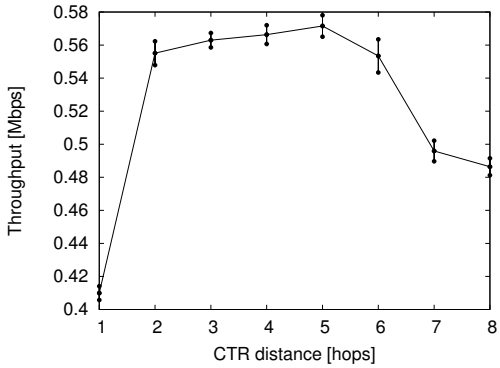


(a) Throughput vs privileged duration

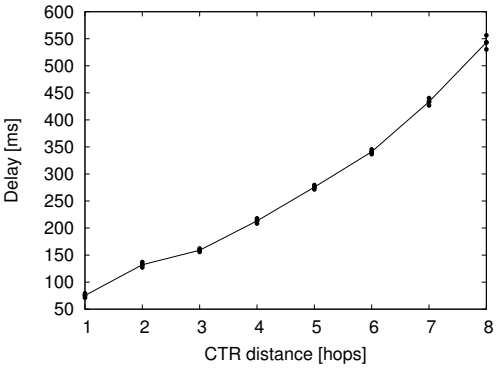


(b) Delay vs privileged duration

Fig. 9. Regular grid of 49 nodes, 13 pps – *impact of the privileged duration (i.e. time before a CTR is forwarded)*



(a) Throughput vs CTR interspacing



(b) Delay vs CTR interspacing

Fig. 10. Regular grid of 49 nodes, 13 pps – *impact of the CTR interspacing (hops that separate two consecutive CTR generated by the sink)*

new CTR iteratively for each branch. When a branch has just forwarded a CTR, it has to wait for the sink to send a new CTR for all the other branches. On average, a k-tree core node has to wait longer before having again the right to be privileged. We consider that 5 branches constitutes a good trade-off for the delay and throughput.

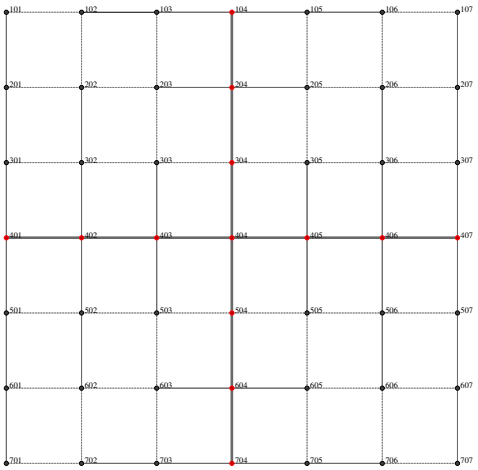
Then, we measured the influence of the *privileged duration*, i.e. the time between a k-tree core node receives a CTR and it has to forward it (fig. 9). A k-tree core node has the right to send its data frame to its parent only during this privileged duration. If this duration is too small, the overhead for forwarding the CTR becomes large. If this duration is long, the end-to-end delay increases. The throughput reaches a maximum when the privileged duration is around 5ms (fig. 9(a)). In the same way, the end-to-end delay reaches a minimum for the same values (fig. 9(b)). Moreover, C-MAC is relatively insensitive to a small variation for this duration (the graphs are in log-scale). We verified also that 5ms constitutes the best privileged duration for other non-grid networks.

Finally, we measured the influence of the CTR inter-spacing, the time separating two consecutive CTR generated by the sink (fig 10). The CTR inter-spacing is represented in hops (the sink has to wait $CTR_{hops} * T_{slot}$ before generating a new CTR). If the sink generates too frequently new CTR, they will surely collide with each other: data packets will be retransmitted, having a negative impact on the throughput (fig 10(a)). On the contrary, if CTR are more than 6 hops apart, bandwidth is wasted, and the throughput decreases. Besides, we should maintain CTR_{hops} as low as possible since it has a negative impact on the delay (fig 10(b)): it increases the buffering delay, as described in section 5.6. In consequence, we consider that a CTR interspacing of 3 hops constitutes a good trade-off.

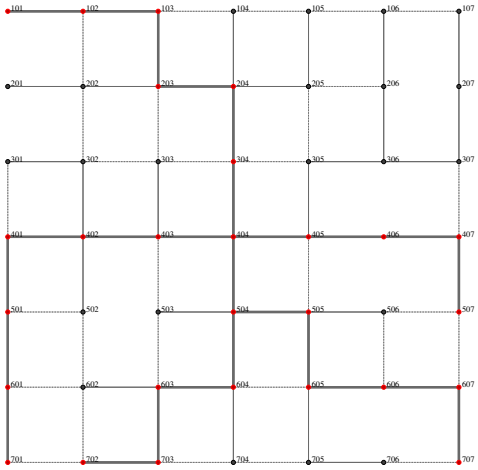
7.2 Grid Network

We first evaluated the robustness of the algorithm electing the k-tree core. In a grid network, the optimal assignment would use shortest paths toward the sink and balance the load among the different branches. Thus, we compared the performance of C-MAC with the distributed algorithm we presented in section 4 and with a centralized static k-tree core (we chose statically the k-tree core as represented in figure 11(a)).

We first reported the throughput of C-MAC and IEEE 802.11 in figure 12(a). IEEE 802.11 uses the tree to route packets toward the gateway. Since IEEE 802.11 is not route-aware, its performances are not impacted when it uses a static or a dynamic routing tree. Besides, we can remark that the throughput of C-MAC is the same for the centralized and our distributed k-tree core construction. Thus, our algorithm seems robust. Finally, we can remark that C-MAC uses the same routes as IEEE 802.11 but optimizes the throughput by reducing the number of collisions and by giving a privileged access to k-tree core nodes. C-MAC achieves a gain of almost 100% for the throughput.



(a) Static centralized assignment with a cross of 4 branches



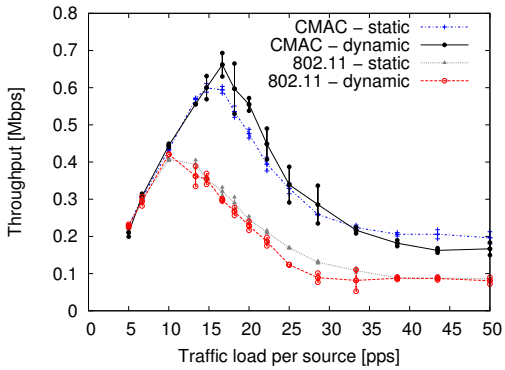
(b) Distributed algorithm with 5 branches

Fig. 11. k-tree core election in a grid of 49 nodes

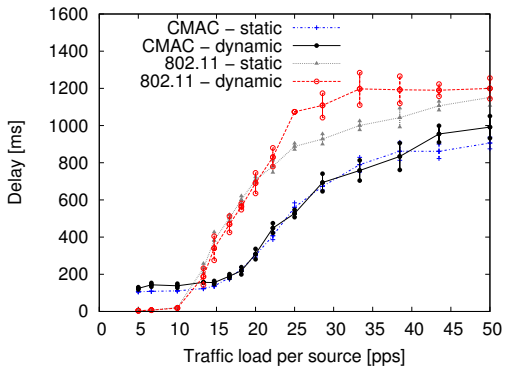
Then, we measured the delay (fig. 12(b)). When CBR flows are small, C-MAC achieves larger delays than IEEE 802.11: a node in the k-tree core that receives a packet has to wait for a CTR before being authorized to forward it. However, the collisions in IEEE 802.11 quickly increase the congestion when the network is more heavily loaded. Thus, C-MAC achieves quickly better delays than IEEE 802.11. We can also remark that the delays are similar whatever the k-tree core algorithm we used (either static or distributed).

We plotted the Jain index in figure 12(c). IEEE 802.11 quickly saturates and drops many packets, in priority from sources far from the sinks: the route length being larger, these packets have a larger chance to be dropped by a forwarder. On the other side, C-MAC achieves a good fairness for larger loads. Obviously, when the network saturates, the packets through shorter routes have a larger probability to be delivered, and the fairness index decreases.

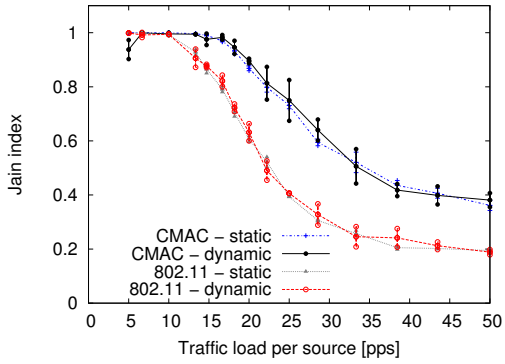
Finally, we measured the overhead in bits for both C-MAC and IEEE 802.11 (fig. 12(d)). We can remark that the overheads are similar for low traffic values: the CTR generated by C-MAC are negligible. In the same way, the additional bits inserted byC-MAC in the `hello` packets have a very limited impact on the global overhead. For a



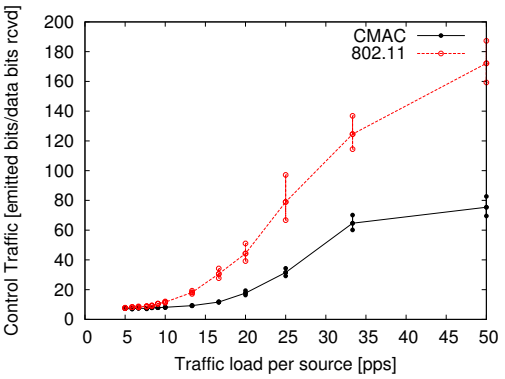
(a) Throughput vs offered load



(b) Delay vs offered load

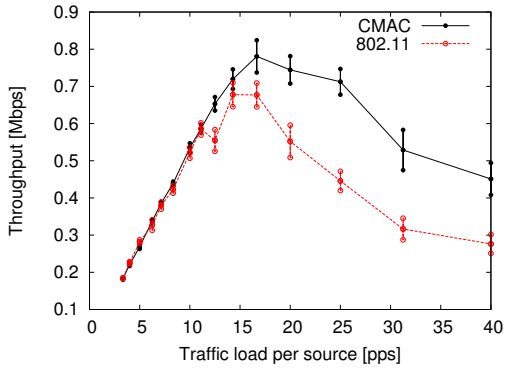


(c) Fairness vs offered load

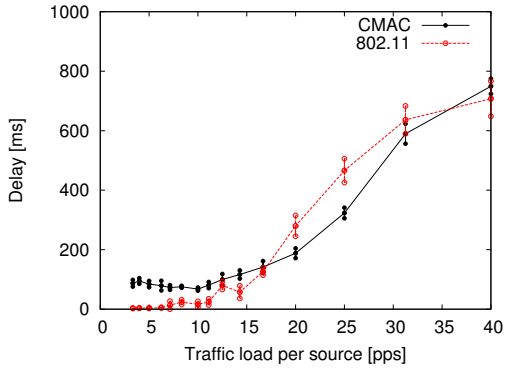


(d) Overhead in bits vs offered load

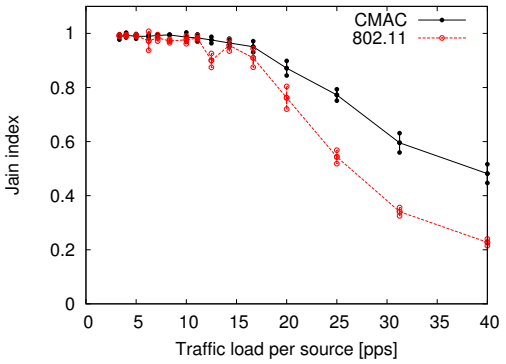
Fig. 12. Regular grid of 49 nodes with a convergecast traffic



(a) Throughput vs offered load

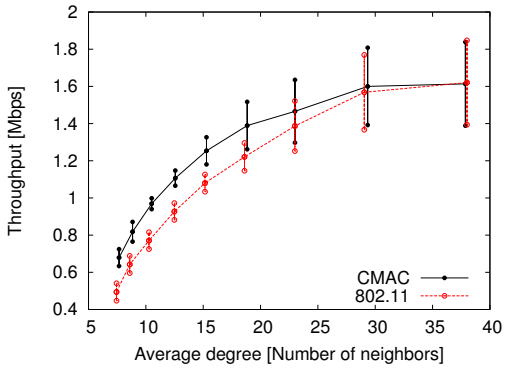


(b) Delay vs offered load

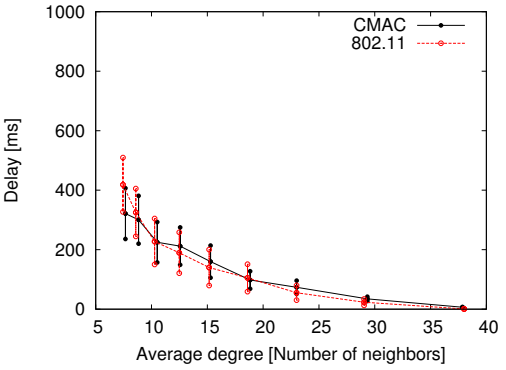


(c) Fairness vs offered load

Fig. 13. Circular Random Network of 60 nodes with a convergecast traffic and an average degree of 8



(a) Throughput vs density



(b) Delay vs density

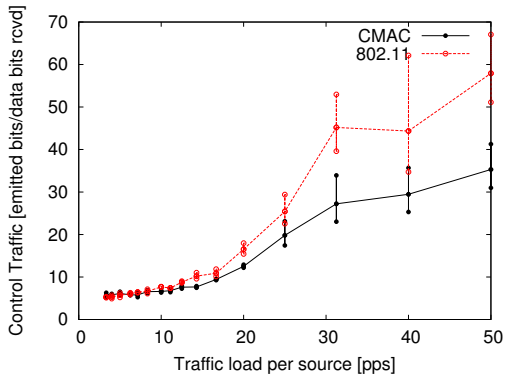
Fig. 14. Circular Random Network of 60 nodes – *CBR of 20pps*

large traffic, long flows suffer from collisions, decreasing the packet delivery ratio. Thus, both protocols generate more packets to deliver the same number of `data` packets to the sink. Mechanically, the global overhead increases.

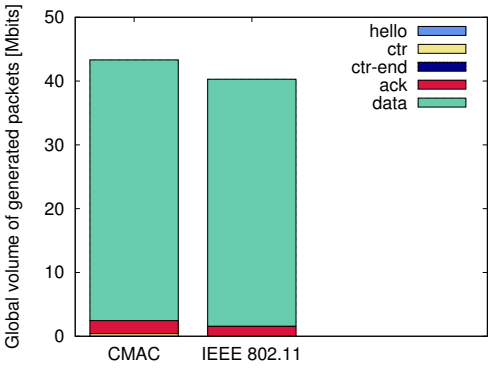
7.3 Random Convergecast Networks

We compared the performance of C-MAC and IEEE 802.11 in a random network (fig. 13): 60 nodes are randomly located in a circular area while maintaining an average degree of 8. Besides, the sink is located in the center of the simulation area to limit side effects. We keep on using the default values as mentioned in table 1. We can remark that C-MAC supports a larger throughput than IEEE 802.11. While some packets begin to be dropped with IEEE 802.11 when each source generates more than 12 pps, C-MAC achieves to transmit almost 17 pps without loss (the gain superior to 40%). When the load increases further, C-MAC keeps on delivering more packets than IEEE 802.11. When the network has only a small amount of packets to forward, IEEE 802.11 presents a lower delay: k-tree core nodes need to wait for CTR in C-MAC, increasing the delay. However, when the network begins to be congested, C-MAC achieves a smaller delay than IEEE 802.11: it succeeds to regulate the traffic and limits retransmissions and collisions. Finally, the reader can verify that C-MAC presents always a better fairness than IEEE 802.11, whatever the traffic conditions are.

We also measured the impact of the density (fig. 14): we adjust the simulation area to increase the average degree. Each source generates 20 pps so that congestion begins to appear in the network. We discard topologies that are disconnected. When the density is low, the throughput decreases: routes are longer and some nodes become disconnected because they only have unreliable links with their neighbors. C-MAC outperforms IEEE 802.11 for small and average densities. When the network is almost single hop, the throughput is maximum, and IEEE 802.11 and C-MAC perform in a similar manner: the k-tree core is very limited and almost all the nodes transmit their packets directly to the sink, without CTR. The gain over IEEE 802.11 is equal to 40% for smallest densities and to 16% for medium densities (av. degree of 15). Finally, C-MAC and IEEE 802.11 achieve a similar delay since we are in saturated mode (fig. 14(b)).

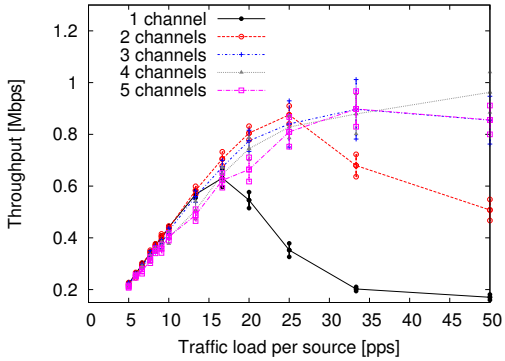


(a) Overhead in bits

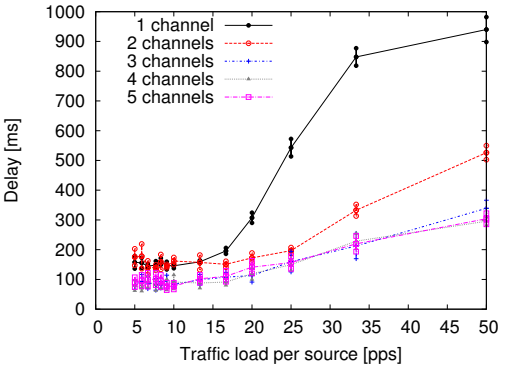


(b) Delay vs density

Fig. 15. Circular Random Network of 60 nodes – *CBR of 20pps*



(a) Throughput vs offered load



(b) Delay vs offered load

Fig. 16. Regular grid of 49 nodes with a convergecast traffic – *multichannel extension*

Finally, we measured the overhead for these topologies (fig. 15). Firstly, we can remark that C-MAC and IEEE 802.11 achieve more or less the same overhead for a low and a very large traffic (fig. 15(a)). We can remark that for a very low traffic, C-MAC generates slightly more control packets than IEEE 802.11: in this case the CTR overhead is not enough mutualized and its cost is not negligible. However, the overhead is not sensitive in this case since the network capacity is not reached. For CBR flows of about 30 to 50 pps, C-MAC is more efficient than IEEE 802.11 to deliver packets to the sink/root. This decreases the average number of data and control packets generated for each delivered data packet.

We also plotted the repartition of the overhead for CBR flows of 20 pps to have a more detailed view of the origin of control packets (fig. 15(b)). We can remark that `hello` packets and `CTR-END` are negligible. We have also a small amount of `CTR` packets for C-MAC (below the `ack`). Besides, `data` constitute the vast majority of transmissions, with the associated `acks`. We can conclude that C-MAC presents an acceptable overhead. We also verified we obtained the same type of result when we measured the number of packets without taking into account the packet sizes.

7.4 Multichannel Feature

We evaluated in figure 16 the multichannel feature of C-MAC. When more than 1 channel is available, the channel 0 is used by default while the other channels are used for the k-tree nodes that become privileged. When C-MAC operates with more than 2 channels, the sink will alternate the channels it uses for the different branches of the k-tree core. Thus, the sink can generate `CTR` more frequently, and thus increases the network capacity.

We measured first the throughput (fig. 16(a)). While the network saturates with one single channel, the multichannel version of C-MAC supports a larger load. The gain is the largest when we have two channels: a k-tree node that receives a `CTR` becomes privileged and uses an orthogonal channel, avoiding entirely the collisions with non k-tree core nodes. Since `CTR` seldom suffer from collisions in the control channel (channel 0), this multichannel mechanism performs well. The reader can remark that when the number of channels is almost the number of branches, the frequency of `CTR` cannot be increased, and the throughput saturates. Moreover, the gain is reduced because non k-tree core nodes may start to have difficulties to send their `data` packets if the receiver is a k-tree core node, deaf to transmissions on the channel 0. We can also remark that using different channels limits the number of retransmissions and transitively the end-to-end delay (fig. 16(b)). In conclusion, C-MAC can exploit very efficiently a collection of orthogonal channels, multiplexing the transmissions without any need of synchronization or multiradio nodes.

8 Related Work

The MAC layer has recently received a large attention for multihop wireless networks. IEEE 802.11 [8] is the predomi-

nant protocol for hotspots, i.e. single hop wireless networks for multimedia. The CSMA-CA mechanism is flexible and particularly efficient to cope with traffic variations. However, IEEE 802.11 has been proved to perform very poorly in multihop [1].

In wireless mesh networks, some researchers have proposed to assign different channels for multi-radio mesh nodes [16], [17], [18], [19]. [18] focuses on the routing metric to construct the tree rooted at the gateway, and then assigns a channel per link, taking into account the traffic load. [17] starts from the gateway, assigning the channels to the most constrained links. [19] extends it to cope with multicast. However, these approaches require multi-radio capable nodes and color the radio links in a centralized manner. [16] proposes that a subset of interfaces stay static while the other ones switch from one channel to another. However, it works only for multiradio nodes. To the best of our knowledge, no MAC protocol was proposed to cope with wireless mesh nodes with one single interface, typically adapted to the convergecast traffic pattern.

In wireless sensor networks, most approaches focused on energy savings, like [20]. In [21], the PSM mode of IEEE 802.11 is modified to work in WSN. In SMAC, each node wakes up periodically and publishes this periodicity in its `hellos` [22]. A node just has to know this scheduling to know when to send the frame. Single Channel Polling (SCP) proposes a global synchronization scheme in which all the nodes wake up simultaneously and sense once the channel [14]. However, contention may be large since a node wakes up only seldom, leading to collisions and energy wastage. [23] already jointly optimized the MAC and routing layer. However, it focused on very low traffic conditions. Since C-MAC could be adapted to work with CSMA-CA like approaches, we could extend these protocols to adapt for them the concept of privileges in C-MAC.

Other propositions adopted a TDMA-like approach in WSN. For instance, [24], [25] computes a distributed scheduling to assign timeslots to each node. [6] adapts the scheduling to interferences measured by the nodes. [26] proposes to adopt a multichannel approach, coloring each radio link to avoid interferences. However, it uses a dedicated separated low power radio for transmitting pulses.

WSN present often a convergecast traffic pattern: this particular property can be used to optimize certain functions. [27] optimizes the retransmissions when several bursts of traffic have to arrive to a sink. [28] extends DMAC [29], rearranging the slots attributed to each node. [30] proposes to balance the load for the convergecast tree, optimizing the routing and not the MAC layer as C-MAC does. [31] focuses also on controling the data flows in the routing layer to limit collisions.

C-MAC uses a k-tree core structure as introduced originally in [10]. [32] provides a parallelized version of the algorithm. Then, [11], [33] extended it to provide a distributed version, adapted for routing in ad hoc networks. They use any spanning-tree before selecting the k-tree core while we use only shortest paths to the gateway and we try to create straight branches that forward most traffic. Indeed, our objective here is different since we focus on regulating the transmissions at the MAC layer. [34] proposed to upper bound the k-tree core diameter. [35] uses this k-tree core for multicast. To the best of our knowledge, the k-tree core structure was never used before for organizing the transmissions in the MAC layer although it is particularly adequate.

9 Conclusion & Perspectives

In this paper, we studied the wireless multihop networks with a convergecast traffic pattern. We proposed to organize the network into a k-tree core. This structure coupled with a specific MAC protocol helps to organize the transmissions and to reduce the collisions. By giving a different medium access in CSMA-CA to the nodes that carry most of the traffic, we reduce the number of collisions, which may occur only with nodes with less traffic. Besides, C-MAC does not need any synchronization mechanism, and is much more flexible than a classical TDMA scheme. Finally, C-MAC is entirely compatible with IEEE 802.11, i.e. it can cohabit transparently with non-C-MAC enabled nodes. We have also shown that C-MAC operates efficiently in multichannel environments and optimizes the throughput while avoiding the deafness problem. Simulation results demonstrated that C-MAC with a k-tree core outperforms IEEE 802.11.

One important future research consists in validating C-MAC in a realistic environment, but it requires to have access to the IEEE 802.11 firmware to allow a k-tree core node to use PIFS for its transmissions. Besides, it could be interesting to jointly optimize the routes and the k-tree core structure. Indeed, C-MAC optimizes the transmissions for a tree of shortest paths. However, shortest routes could not be the best solution for balancing the load to reduce contention. We expect also to compare C-MAC and a TDMA-like approach in realistic conditions. In particular, TDMA requires to know exactly the conflict graph and needs to compute a conflict-free scheduling. We plan to measure the impact of conflict graph imprecisions, and to verify that C-MAC achieves similar or better results without synchronization and without computing a complex and less flexible scheduling. Finally, we plan to adapt C-MAC to SCP in order to reduce the collisions when all the nodes wake-up simultaneously. This feature would be particularly accurate in Wireless Sensor Networks for event detection since the traffic is mostly in bursts in these cases.

10 Acknowledgements

The author would like to thank Catherine Rosenberg (University of Waterloo, Canada) for providing helpful advices and sharing its expertise in this research topic.

This work was partly supported by the French Government and the Competitive Clusters Minalogic and System@tic under the contract FUI SensCity.

References

[1] D. Dhoutaut C. Chaudet and I. Guérin Lassous. Performance issues with IEEE 802.11 in ad hoc networking. *IEEE Communications Magazine*, 43(7):110–116, July 2005.

[2] Saikat Ray, Jeffrey B. Carruthers, and David Starobinski. Evaluation of the masked node problem in ad hoc wireless LANs. *IEEE Transactions on Mobile Computing*, 4(5):430–442, Sept.-Oct. 2005.

[3] Aravind Iyer and Catherine Rosenberg. Understanding the key performance issues with mac protocols for multi-hop wireless networks: Research articles. *Wireless Communications and Mobile Computing*, 6(6):745–760, September 2006.

[4] A. Karnik, A. Iyer, and C. Rosenberg. Throughput-optimal configuration of fixed wireless networks. *IEEE/ACM Transactions on Networking*, 16(5):1161–1174, Oct. 2008.

[5] K. Papadaki and V. Friderikos. Robust scheduling in spatial reuse TDMA wireless networks. *IEEE Transactions on Wireless Communications*, 7(12):4767–4771, December 2008.

[6] Mario Macedo, Antonio Grilo, and Mario Nunes. Distributed latency-energy minimization and interference avoidance in TDMA wireless sensor networks. *Computer Networks*, 53(5):569 – 582, 2009.

[7] Jitendra Padhye, Sharad Agarwal, Venkata N. Padmanabhan, Lili Qiu, Ananth Rao, and Brian Zill. Estimation of link interference in static multi-hop wireless networks. In *Conference on Internet Measurement (IMC)*, pages 28–28, Berkeley, CA, October 2005. ACM SICOMM.

[8] IEEE 802.11, local and metropolitan area networks - specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications, IEEE standard, 1999.

[9] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless LANs. In *SIGCOMM*, pages 121–132, Philadelphia, Pennsylvania, USA, 2005. ACM.

[10] S. Peng, A.B. Stephens, and Y. Yesha. Algorithms for a core and k-tree core of a tree. *Journal of Algorithms*, 15(1):143–159, July 1993.

[11] Saurabh Srivastava and R. K. Ghosh. Cluster based routing using a k-tree core backbone for mobile ad hoc networks. In *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL'M)*, Atlanta, Georgia, USA, 2002. ACM.

[12] Z.J. Haas and Jing Deng. Dual busy tone multiple access (DBTMA)-a multiple access control scheme for ad hoc networks. *IEEE Transactions on Communications*, 50(6):975–985, June 2002.

[13] J.L. Hill and D.E. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[14] Wei Ye, Fabio Silva, and John Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *SenSys*, pages 321–334, Boulder, Colorado, USA, 2006. ACM.

[15] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. DRAND: Distributed randomized TDMA scheduling for wireless ad-hoc networks. *IEEE Transactions on Mobile Computing*, 8(10):1384–1396, Oct. 2009.

[16] Pradeep Kyasanur and Nitin H. Vaidya. Routing and link-layer protocols for multi-channel multi-interface ad hoc wireless networks. *SIGMOBILE Mobile Computer Communication Review*, 10(1):31–43, 2006.

[17] Krishna N. Ramachandran, Elizabeth M. Belding, Kevin C. Almeroth, and Milind M. Buddhikot. Interference-aware channel assignment in multi-radio wireless mesh networks. In *INFOCOM*, pages 1–12, Barcelona, Spain, April 2006. IEEE.

[18] A. Raniwala and T. Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *INFOCOM*, Miami, USA, March 2005. IEEE.

[19] Hoang Lan Nguyen and Uyen Trang Nguyen. Channel assignment for multicast in multi-channel multi-radio wireless mesh networks. *Wireless Communications and Mobile Computing, Special Issue on Next Generation Wireless Communications and Mobile Computing/Networking Technologies*, 9(4):557–571, April 2009.

[20] Muneeb Ali, Umar Saif, Adam Dunkels, Thiemo Voigt, Kay Romer, Koen Langendoen, Joseph Polastre, and ZArtash Afzal Uzmi. Medium access control issues in sensor networks. *ACM SIGCOMM Computer Communication Review*, 36(2):33–36, April 2006.

[21] Amre El-Hoiydi, Jean-Dominique Decotignie, and Jean Hernandez. Low power MAC protocols for infrastructure wireless sensor networks. In *European Wireless*, pages 563–569, Barcelona, Spain, February 2004.

[22] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM*, New-York, USA, June 2002. IEEE.

[23] Sunil Kulkarni, Aravind Iyer, and Catherine Rosenberg. An address-light, integrated MAC and routing protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 14(4):793–806, August 2006.

[24] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. DRAND: Distributed randomized TDMA scheduling for wireless ad-hoc networks. In *International symposium on Mobile ad hoc networking & computing (MOBIHOC)*, Florence, Italy, May 2006. ACM.

[25] Injong Rhee, Ajit Warrier, Mahesh Aia, Jeongki Min, and Mihail L. Sichitiu. Z-MAC: Hybrid MAC for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 16(3):511–524, June 2008.

[26] Kaushik R. Chowdhury, Nagesh Nandiraju, Pritam Chanda, Dharma P. Agrawal, and Qing-An Zeng. Channel allocation and medium access control for wireless sensor networks. *Ad Hoc Networks*, 7(2):307 – 321, March 2009.

[27] Hongwei Zhang, Anish Arora, Young ri Choi, and Mohamed G. Gouda. Reliable bursty convergecast in wireless sensor networks. *Computer Communications*, 30(13):2560 – 2576, February 2007.

[28] Thiemo Voigt, Fredrik Österlind, and Adam Dunkels. Improving sensor network robustness with multi-channel. In *ERCIM Workshop on e-Mobility*, Tampere, Finland, May 2008.

[29] Gang Lu, Bhaskar Krishnamachari, and Cauligi S. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Parallel and Distributed Processing Symposium, International (IPDPS)*, volume 13, Los Alamitos, CA, USA, 2004. IEEE.

[30] Tzung-Shi Chen, Hua-Wen Tsai, and Chih-Ping Chu. Adjustable convergecast tree protocol for wireless sensor networks. *Computer Communications*, 33(5):559–570, 2010.

[31] Qingfeng Huang and Ying Zhang. Radial coordination for convergecast in wireless sensor networks. In *International Conference on Local Computer Networks (LCN)*, pages 542–549, Tampa, Florida, USA, November 2004. IEEE.

[32] Biing-Feng Wang. Finding a k-tree core and a k-tree center of a tree network in parallel. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):186–191, February 1998.

[33] Saurabh Srivastava and R.K. Ghosh. Distributed algorithms for finding and maintaining a k-tree core in a dynamic network. *Information Processing Letters*, 88(4):187–194, November 2003.

[34] Biing-Feng Wang, Shietung Peng, Hong-Yi Yu, and Shan-Chyun Ku. Efficient algorithms for a constrained k-tree core problem in a tree network. *Journal of Algorithms*, 59(2):107–124, May 2006.

[35] Yamin Li, Shietung Peng, and Wanming Chu. K-tree trunk and a distributed algorithm for effective overlay multicast on mobile ad hoc networks. In *International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, pages 53–58, Sydney, Australia, May 2008. IEEE.