EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Cross-layer medium access control protocol with quality-of-service guarantees for wireless sensor networks

Joel Ruiz[1,2*], Jose R Gallardo[1,2], Dimitrios Makrakis[2], Luis Villasenor-Gonzalez[1] and Hussein T Mouftah[2]

Abstract

There is increasing demand for wireless sensor networks (WSN) to be able to carry real-time information. However, current WSN technologies are not yet capable of offering quality-of-service (QoS) guarantees, which are required to support these types of applications. Achieving QoS is especially challenging in WSNs due to their multi-hop nature and their processing-power, memory, and energy constraints. In this article, we propose a cross-layer architecture in which the medium access control (MAC) and routing protocols collaborate to organize nodes into clusters and to achieve a coordinated time-shared access to the transmission medium. The resulting protocol is called QUAlity-of-service-capable clusTer-based Time-shared ROuting (QUATTRO)-assisted MAC protocol. Our performance evaluation results show that the protocol overhead observed in terms of configuration time, transmitted control messages, and consumed energy is very reasonable and that not only QoS is achieved but also great energy savings by eliminating collisions and considerably reducing idle listening.

Keywords: cross-layer protocols, medium access control, quality-of-service, routing, wireless sensor networks

1. Introduction

A wireless sensor network (WSN) is a self-configurable multi-hop local area network (WLAN) through which terminals, equipped with some type of sensor, transmit the measured parameters to a predetermined set of information receptacles, called sinks. The main difference between a traditional WLAN and a WSN is that, in general, the latter needs to be very energy-efficient since its nodes are powered with non-rechargeable batteries. Nodes in a WSN are also limited in processing power and memory. In addition, the routing mechanisms of a WSN must be dynamic since static routing would prematurely exhaust the energy of those nodes that participate in the packet-relaying process, in addition to the fact that some nodes can be turned off periodically to save energy or permanently as their batteries die.

There is currently great interest, both technological and economic, in the development of more efficient solutions for WSNs. These greatly-needed solutions, aimed at using the available network resources in the best possible way, include communications protocols to allow orderly and timely access to the transmission/reception medium (medium access control, MAC) as well as neighborhood discovery and intelligent routing, due to the inherent multi-hop nature of this type of networks.

Shortly after publication of the original IEEE 802.11 standard for wireless LANs, its commercial success fueled the development of improved technologies that allowed such networks to go from the initial 2 to 54 Mbps in the 802.11a/g standards, to 600 Mbps in the more recent 802.11n standard and there are even current efforts in the IEEE 802.11ac and 802.11ad study groups to generate technologies in the gigabit-per-second range. A similar evolution can be expected for WSN once their use in commercial applications intensifies. As the WSN technology evolves toward more processing power, more memory, and higher transmission rates, the implementation of systems able to sense and transmit real-time information, such as audio and video, come closer to being feasible. In fact the term wireless video sensor network has been coined (e.g., [1-3]),

* Correspondence: jruiz@cicese.edu.mx
[1]Electronics and Telecommunications Department, CICESE Research Center, Ensenada, BC 22860, Mexico
Full list of author information is available at the end of the article

reflecting the relevance that video transmission in WSN has gained. Examples of potential applications include intrusion detection, object identification and tracking, suspicious behavior classification, etc. These applications need for the system to be able to offer quality-of-service (QoS) guarantees, not yet available in existing technologies. This is a research topic that has recently received a great deal of attention (see e.g. [4-15]), as explained in Section 2.

In addition, researchers have recently realized that the traditional separation of protocols in layers, through the clear definition of the services that each layer must provide and the isolation of each layer's operation, is not the best way to go. In other words, the exchange of information among layers renders a far better performance than working in isolation. This discovery has given rise to the notion of cross-layer protocol design.

With all this in mind, the goal of this study is then to propose a cross-layer architecture in which the routing and the MAC protocols collaborate to achieve an energy-efficient and QoS-aware medium access mechanism for WSN. The tasks performed by the routing protocol include path discovery, resource reservation, cluster formation, and gathering of information as to interference among clusters. The MAC protocol uses that information to create activity-window schedules for clusters to time-share the transmission medium, and uses a collision-free protocol for communication within each cluster. The resulting protocol is called QUAlity-of-service-capable clusTer-based Time-shared ROuting (QUATTRO)-assisted MAC protocol.

The proposed mechanism saves energy by avoiding collisions and by allowing nodes to safely turn off their transceivers outside their activity windows without the risk of losing any relevant frame transmission. It also ensures QoS using a bandwidth-dependence-aware resource-reservation procedure. The concept of bandwidth dependence, introduced in [16], means that a node will affect and be affected by the transmissions of its one- and two-hop neighboring nodes, regardless of whether or not they belong to common established routes.

It is important to note that this protocol is suitable for fixed nodes only. Assuming node mobility would slow down convergence and highly increase the need for reconfiguration, which in turn would reduce the energy efficiency of the protocol, as explained in Section 5.

The rest of the article is organized as follows: Section 2 compares previously published proposals dealing with the provision of QoS guarantees in a WSN environment. Section 3 describes our proposal in detail. Section 4 describes a typical scenario in which our protocol would be highly useful and describes the different types of guarantees that can be offered, depending on the nature of the traffic carried by the network. Section 5 presents simulation results to evaluate the overhead incurred by the protocol and its ability to actually offer QoS guarantees and save energy at the same time. Section 6 summarizes the conclusions that can be drawn from this study.

2. Related work

Since one of the main goals in WSN protocol design is energy efficiency, many protocols work with low duty cycles by including periods in which the transceiver is put to sleep. SMAC was one of the first protocols designed with this in mind and works by coordinating sleep-wake times for neighboring nodes in combination with a contention-based transmission scheme [17]. Because of its pioneering role, SMAC has become a protocol of reference in WSN. Other examples of protocols that also use low duty cycles and coordinated activity times are T-MAC [18], RMAC [19], and DW-MAC [20]. A different approach that has also been explored is asynchronous transmission, in which nodes, instead of agreeing on recurring activity periods, use messages to indicate when transmissions can happen. Examples of protocols using asynchronous duty cycling are B-MAC [21], XMAC [22], and Wise-MAC [23]. These protocols succeed in achieving high energy efficiency. However, since they use contention-based transmissions, only achieve high throughput under low traffic conditions.

In addition to addressing energy efficiency, there have been recently a remarkably large number of papers dealing with the topic of QoS in WSN, which is a sign of the importance of solving this problem given its widespread applicability. As a sample, we discuss the benefits and shortcomings of only a few of these proposals.

There are several proposals that consider TDMA as a way to avoid collisions (e.g. [4-6]). TDMA is known to be costly, both in overhead and computation complexity, in addition to requiring a fine synchronization of nodes. Unfortunately, these articles do not describe in detail how nodes can exchange the necessary information to specify the TDMA schedule, which makes it difficult to estimate the overhead (both in control messages and in time) incurred by the protocol. PEDAMACS [6], for instance, assumes that the sink has unlimited transmission power that allows it to communicate directly with all of the nodes in the WSN, which is a condition that will seldom be satisfied in real systems.

Multipath routing, which consists of giving a node the possibility to use any of several paths to send a message to a particular destination at any given time, is also contemplated as a way for time-constrained messages to avoid congested routes (e.g. [7-10]). The authors of [7] propose for the sink to classify routes according to delay, reliability, and energy consumption so that nodes can send time-constrained data through the fastest route, error-constrained data through the most reliable route, and non-constrained data through least-energy routes. This is a good method for service differentiation, but it is difficult to offer QoS guarantees since, for instance, the fastest route may not be fast enough under certain conditions. The authors of [8] propose an adaptive technique based on the ant-colony algorithm in which routes are discovered when needed (reactive algorithm) by sending a probe message. When this message reaches the desired destination through multiple routes, messages are sent back using the reverse paths. These messages mark the nodes visited according to the quality of the route (available bandwidth, expected delay, and loss rate). Then, neighboring nodes share their quality marks so that each node is now able to select the best option when it needs to forward a frame. There is a scalability problem in this case since this algorithm has to be executed for each source-destination pair. In addition, similar to what was mentioned in the previous case, selecting the best local option does not guarantee an end-to-end acceptable performance. In [9], on the other hand, the emphasis is on reliability, thus the protocol is not applicable for real-time data delivery. Finally, MMSPEED [10] performs local estimations of the reliability and delay that packets will experience over the different paths available. Packets can then choose the best combination of service options depending on their requirements. Since both delay and reliability depend on the traffic pattern and traffic-forwarding decisions are made based on delay and reliability, this creates a feedback loop that may generate a great variability in the traffic dynamics, and therefore a demand for fast recalculation of the estimates. In addition, nodes are required to constantly keep track of deadlines and elapsed times. The algorithm therefore requires high computational resources from the nodes.

Other articles propose the introduction of priorities as a way to reduce the latency of time-constrained messages (e.g. [11,12]). Establishing priorities for different types of traffic is again only a service-differentiation mechanism, which means that it is not possible to offer QoS guarantees since all we know is that the service for high-priority traffic will be better, but we do not know if it will be good enough.

Another alternative is the use of congestion control techniques as a reactive way to detect poor performance and to take actions to alleviate it (e.g. [13-15]). These algorithms rely in general on the continuous assessment of congestion levels to decide when to act. Acting implies dropping low-priority traffic and/or notifying upstream traffic sources to reduce their transmission rates. In the former case, there is the implicit assumption that traffic tolerates these losses, which is not always justified. In the latter case, there is the implicit assumption that traffic sources are able to reduce their traffic generation rate, which again is in general not true (e.g. for streaming applications). In addition to that, high overhead may be incurred. Moreover, there is always the risk of losing important data when congestion conditions deteriorate more rapidly than what the system can manage.

Our approach is based on reserving from the very beginning enough resources to guarantee the required QoS performance. A preliminary version of this study was published in [24].

3. Protocol specification

Our proposed protocol includes different components designed to carry out the following functions:

- Route discovery and weight assignment.
- Route selection and reservation.
- Cluster formation.
- Collection of cluster interference information.
- Assignment of activity windows to the clusters.
- Normal operation.
- Reconfiguration.

As far as engineering decisions, we selected the mechanisms that compose this proposal based on the following criteria. The route discovery and weight assignment algorithm was borrowed from an existing protocol, as explained in the following section, because it is very well suited for WSN, in which information has to converge into a node with special responsibilities (the sink) and because the relevant protocol assigns weights to the different discovered paths. This notion of path weight, based on the nodes' anticipated traffic load and remaining energy, proved very helpful for our route selection and reservation phase.

Now the concept of cluster, as will be described more clearly in Section 3.2, was selected because the end result of the route discovery phase is the creation of a route tree, rooted at the sink. In this tree, when several branches converge together into a common node, it means that this common node will be in charge of forwarding traffic on behalf of all of the nodes included in those branches. So, we decided to allow the common node to be the head of a cluster and to have the next node in each branch to also be part of the same cluster; this way we can allow these nodes to communicate by scheduling them to be active at the same time as members of the same cluster.

Regarding the assignment of activity windows to the clusters, described in Section 3.4, we chose a staggered approach according to the depth of the cluster heads (CHs), so that information can move forward in every activity window to a node that has not had yet a chance to transmit in the current cycle. The end result is that all the information that is in the nodes' buffers at the beginning of a cycle will be able to reach the sink by the end of the same cycle, thus reducing delay.

We also selected polling as the method to send traffic from cluster members to the CH during the normal operation of the protocol (after the initial configuration) due to the fact that it is a simple protocol that incurs low and deterministic overhead. Moreover, it is perfectly suited for the master-slave relationship that exists between the CH and the other cluster members.

The idea of having periodic reconfigurations has two goals. One of them is to make sure that routing information remains up to date, and the other is to distribute more evenly the energy consumption among nodes. This idea is also borrowed from the LEACH protocol [25].

These functions and the proposed methods to carry them out will be described in the following sections. The names of the different messages used within these methods are composed of a prefix that identifies the protocol phase to which they correspond and a suffix that indicates the specific role played by the message. The prefixes used are R (route discovery), W (weight assignment), RS (reservation), CI (cluster interference information), AW (activity window assignment), and RC (reconfiguration).

3.1. Route discovery and weight assignment

The first steps of our proposed protocol are the discovery of a set of non-overlapping routes from each sensing node to the currently active information sink and the assignment of a weight value to each one of these routes. This procedure is taken from [26], which in turn is a generalization of [27], and consists of four phases. Briefly explained, the algorithm works by having the sink periodically broadcast a route-update message, also called RPRI, whose goal is to allow each sensing node to discover its *primary* route to the sink. The format of this message is shown in Figure 1. Each node, when receiving the update for the first time, marks the node from which it received the message as its *parent* and rebroadcasts the message. This continues recursively until every node in the network has received a copy of the message and has identified its parent, which represents the next hop in the primary route to the sink.

The second phase of the relevant routing protocol consists of the transmission of messages of type RALT, used by each sensing node to share with its neighbors the set of *alternative* routes discovered, either by hearing the route-update message from a node different from its parent or by receiving a RALT message from one of its neighbors. See Figure 1 for a list of fields included in these messages. When the second phase ends, the sensing nodes have discovered all the available minimum-cost disjoint routes to the sink. In our case, the cost is given by the number of hops in the route, but other metrics such as the received power, estimated BER, or level of node conglomeration (anticipated collision rate) could be used as well.

In phase three, each sensing node will send a WPRB (probe) message through each of the routes discovered, including the primary and the alternative ones. When a sensing node receives one of these messages, it will increase a counter of the number of routes that will potentially go through it (*num_routes*) and will forward the message. The message will eventually reach the sink, which will maintain a table of the paths contained in the received probe messages.

Finally, in phase four, the sink will send back a WRSP (probe response) message through each of the paths stored in its table. Figure 2 displays the format of WPRB and WRSP messages. When a sensing node receives a WRSP message, it will forward the message after tagging it with the proper values in the load and energy bottleneck fields as follows. If the value already

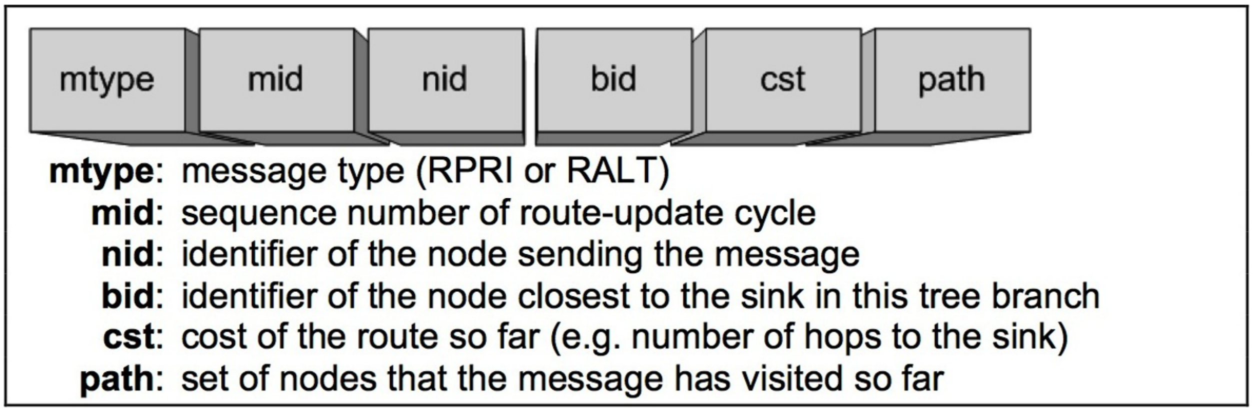

mtype
mid
nid
bid
cst
path
mtype: message type (RPRI or RALT)
mid: sequence number of route-update cycle
nid: identifier of the node sending the message
bid: identifier of the node closest to the sink in this tree branch
cst: cost of the route so far (e.g. number of hops to the sink)
path: set of nodes that the message has visited so far

Figure 1 Route-discovery message formats.

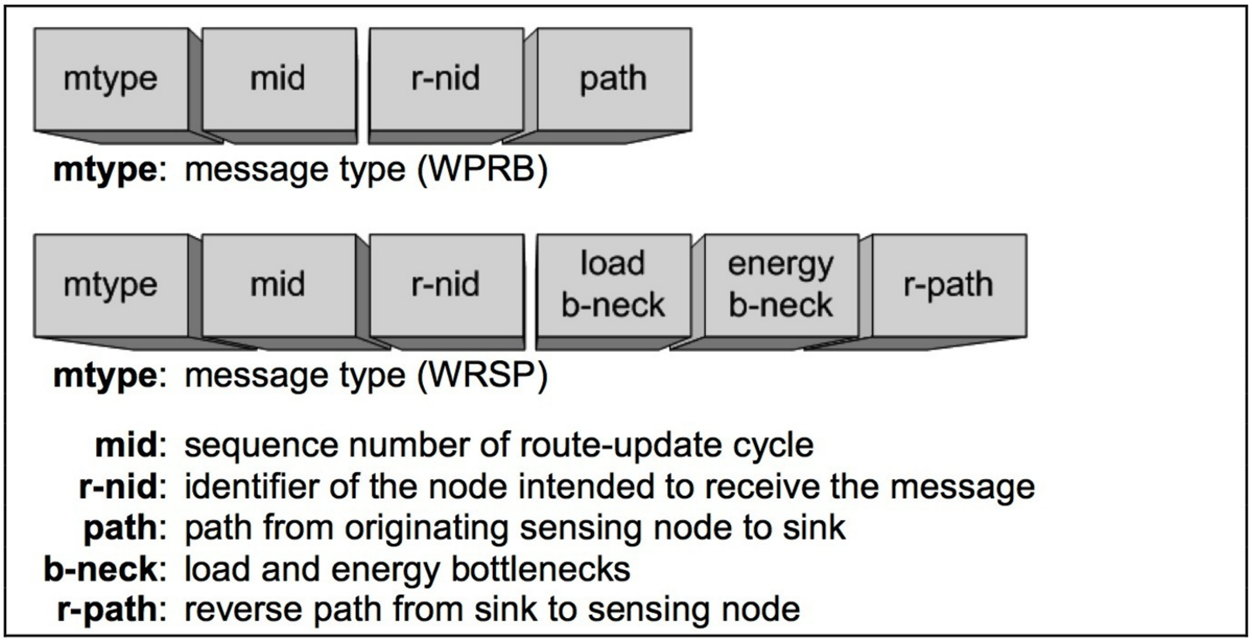

mtype
mid
r-nid
path
mtype: message type (WPRB)
mtype
mid
r-nid
load b-neck
energy b-neck
r-path
mtype: message type (WRSP)
mid: sequence number of route-update cycle
r-nid: identifier of the node intended to receive the message
path: path from originating sensing node to sink
b-neck: load and energy bottlenecks
r-path: reverse path from sink to sensing node

Figure 2 Probe and probe response message formats.

stored in the *load bottleneck* field is less than its expected load (*num_routes*), the node will set it to *num_routes*. Similarly, if the value stored in the *energy bottleneck* field is greater than its remaining energy reserve, it will set it to its available energy. The message will eventually reach the sensing node at the end of the reverse path. When all of its probe messages have been responded or a timer has expired, a sensing node will assign each of its N routes the weight p_i given by the following expression:

$$p_i = \frac{\varepsilon_i}{\lambda_i \cdot (h_i)^{\beta}}, \text{for } i = 1, 2, \ldots, N \tag{1}$$

where ε_i is energy bottleneck of route i received in related probe response message (or zero if none received), λ_i is load bottleneck of route i received in related probe response message (or 1 if none received), h_i number of hops in route i, $\beta \in (0, 1)$ is the factor defining the desired impact of the number of hops on the weight.

Figure 3 graphically describes how this phase of the protocol works.

3.2. Route selection and reservation

The second step is the reservation of resources over one of the existing routes between each sensing node and the currently active sink. Every sensing node will become a traffic source during the normal operation of the system, and the selected route has to be capable of satisfying the QoS requirements of the traffic that will be generated. Routes are reserved in a link-by-link fashion. When a link reservation attempt is successful, both nodes become part of a cluster and the node that agrees to forward the requesting node's traffic becomes the CH. The cluster will grow as more nodes select the same CH to be the next node *en route* to the sink. As we can see, in this proposal a cluster is defined as a set of one-hop neighboring nodes that need to communicate directly, and the goal is to coordinate the sleeping and waking patterns of all the nodes in a cluster. The cluster ID will be the same as that of the CH. We assume that there is no data traffic flowing from the sink to the sensing nodes, as is typical in WSNs; there may be control traffic flowing in this direction (from sink to sensing node), but it will be in general less intense and non-real-time, thus no reservation is necessary for this traffic.

Notice that a node that agrees to forward other nodes' traffic will in fact be a member of two clusters, one in which it is the head (in charge of collecting data from its members) and another one in which it is just another member (its function in this case is to forward to the new CH the data collected previously plus that generated locally).

Before the actual reservation process begins, the sink will broadcast a RSINT (reservation intention) message, indicating that it is time to start the reservation phase. When one-hop neighbors hear this message, they will in turn send an RSINT message to the sink. As shown in Figure 4, RSINT messages contain the identifiers of the

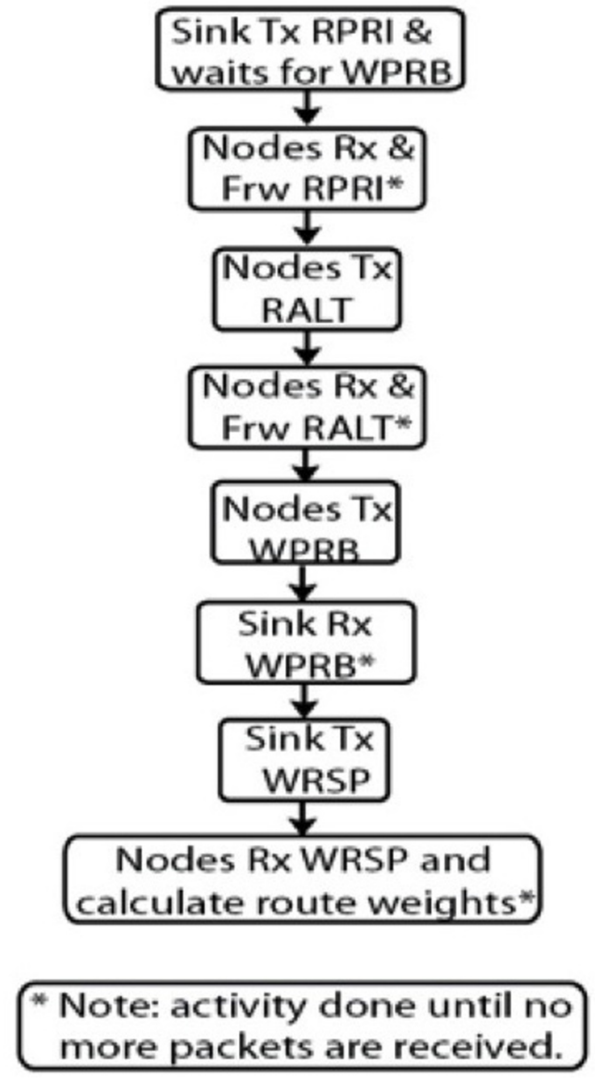

Sink Tx RPRI & waits for WPRB
Nodes Rx & Frw RPRI*
Nodes Tx RALT
Nodes Rx & Frw RALT*
Nodes Tx WPRB
Sink Rx WPRB*
Sink Tx WRSP
Nodes Rx WRSP and calculate route weights*
* Note: activity done until no more packets are received.

Figure 3 Route discovery and weight-assignment process.

sender and of the intended receiver. The meaning of the RSINT message is that the transmitter has selected the receiver as the potential next node in its path to the sink and the actual reservation request will be transmitted shortly. When a node different from the sink and its one-hop neighbors hears an RSINT message transmitted in its neighborhood, it will store the ID of the transmitter and start/restart a timer. On expiry, the node will randomly select one of its routes, with probabilities proportional to their weights, and will send an RSINT message to the next node in such a route. Notice that route weights are used as a guideline only, to increase the probability that a reservation attempt is successful, but bandwidth availability will be verified. The previous selection will exclude those routes for which the next node has not yet transmitted its RSINT message; the goal here is to avoid cyclic routes. The node will start at this point a new timer that will be restarted every time a new RSINT message is overheard. On expiry, this timer will indicate that the *intention* phase has ended and it is time to start the actual reservation of resources.

We argue that guaranteeing a certain amount of bandwidth is enough for QoS assurance, i.e., throughput, delay, jitter, and losses due to buffer overflow will be bounded as a consequence, as explained in [28]. This will be discussed in more detail later.

The bandwidth management procedure will be as follows. All nodes start with an available bandwidth equal to the wireless channel's achievable effective data rate R, which depends on the underlying physical layer and medium access policy. For instance, in a polling-based system such as the one assumed in this work, the achievable throughput is about 85% of the channel gross bit rate [28]. In addition, R can be reduced further to account for the expected channel errors, which are present in any transmission medium but are especially common in wireless environments.

The following inequality has to be satisfied at all times:

$$B_{\text{avail}} = R - \left(2\, B_{\text{committed}} + B_{\text{own}} + B_{\text{overheard}}\right) \geq 0 \quad (2)$$

In the previous equation B_{avail} is the bandwidth still available for new reservations, $B_{\text{committed}}$ is the total bandwidth that the relevant node has committed to forward on behalf of other nodes, B_{own} is the traffic that the node itself will generate, and $B_{\text{overheard}}$ is the amount of bandwidth committed in reservations that the node has overheard. $B_{\text{committed}}$ is counted twice since the same channel will be used by the node to receive and forward the traffic.

Let us denote by B_{req} the amount of bandwidth to be reserved by a node, which in general will be equal to $B_{\text{committed}}$ + B_{own}. The formats of the reservation-associated messages are shown in Figure 4. If the reservation-requesting node is only one-hop away from the sink, it will need no more verifications than Equation 2 before sending a RSRQ (reservation request) message to the sink asking if it is able to reserve the necessary bandwidth, and will wait for a response. If, on the other hand, the node is more than one-hop away from the sink, it will only send the reservation request to the next node in the selected route if its own available bandwidth is at least equal to B_{req}, in anticipation of the reservation that the next node in the path will have to make to further forward the traffic. All the nodes that can overhear the message will also check if their available bandwidth is at least equal to B_{req}; if not, they will send back a response message indicating that the new reservation

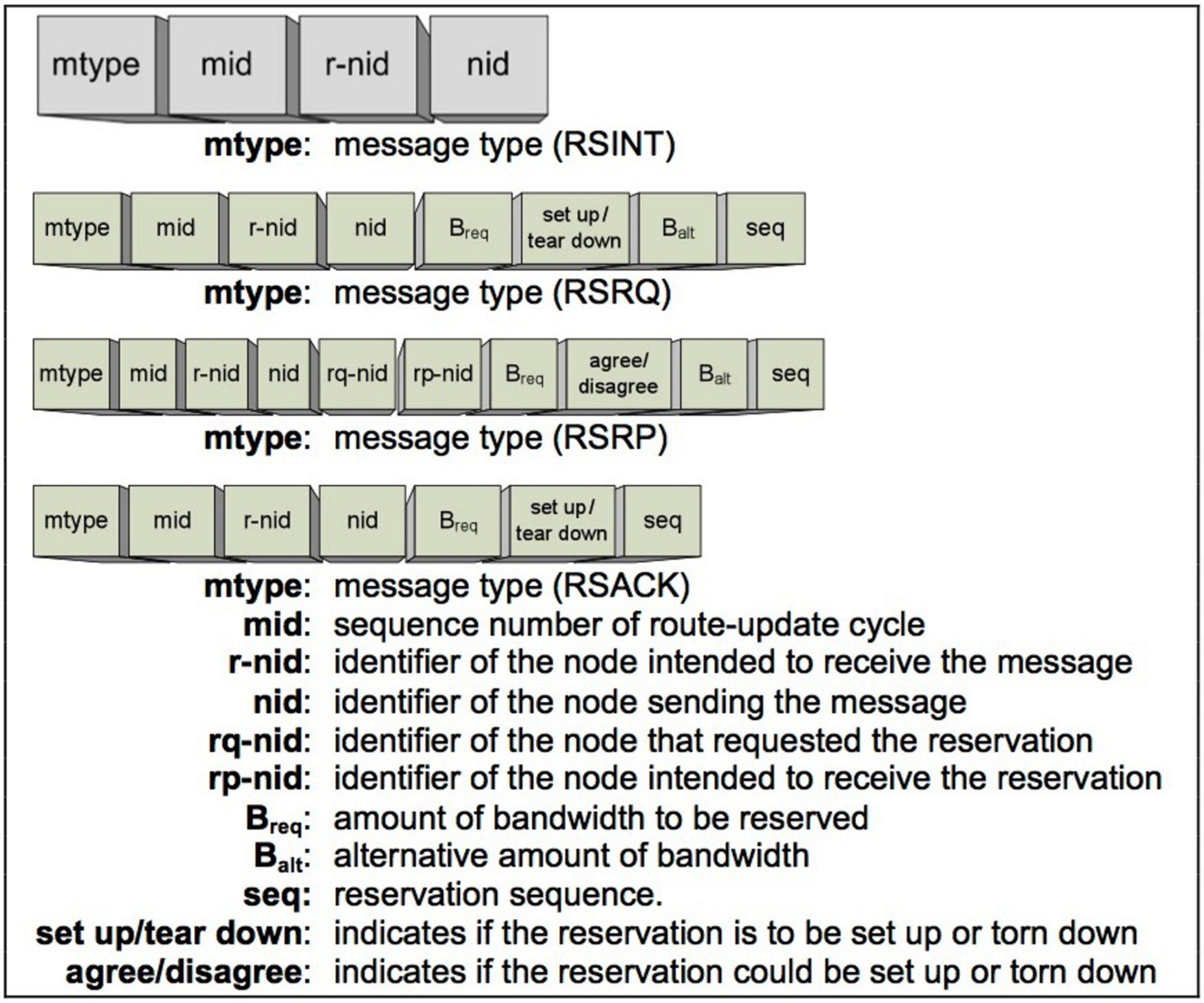

mtype
mid
r-nid
nid
mtype: message type (RSINT)
mtype
mid
r-nid
nid
Breq
set up / tear down
Balt
seq
mtype: message type (RSRQ)
mtype
mid
r-nid
nid
rq-nid
rp-nid
Breq
agree/ disagree
Balt
seq
mtype: message type (RSRP)
mtype
mid
r-nid
nid
Breq
set up / tear down
seq
mtype: message type (RSACK)
mid: sequence number of route-update cycle
r-nid: identifier of the node intended to receive the message
nid: identifier of the node sending the message
rq-nid: identifier of the node that requested the reservation
rp-nid: identifier of the node intended to receive the reservation
Breq: amount of bandwidth to be reserved
Balt: alternative amount of bandwidth
seq: reservation sequence.
set up/tear down: indicates if the reservation is to be set up or torn down
agree/disagree: indicates if the reservation could be set up or torn down

Figure 4 Format of reservation-associated messages.

is not possible. In turn, the node for which the reservation request message was addressed will check if its own available bandwidth is at least one, two or three times B_{req}, depending on whether it is the sink or a node one-hop or at least two-hops away from the sink, respectively, again anticipating the reservations that it itself and the next node in the path will have to make to further forward the traffic; if everything goes well, it will send a positive response as expected and will start over to reserve the next link in the route. If a node that had not heard the original reservation request hears the response, it will also verify its available bandwidth and, if it finds that the new reservation represents a problem, it will send a response message to indicate it. When a node sends a negative response, it will include in the B_{alt} field the amount of bandwidth it has available as a way to help make an alternative reservation.

If a reservation fails, the requesting node will try the same reservation request on another route, selected again based on their weights. If all routes fail, it will reduce the amount of bandwidth needed by canceling the agreements previously made with some of its cluster members, if any, and try again on the route with the most available bandwidth. When a node has to select another route because a reservation attempt failed, it will send again an *intention* message before the actual reservation request. If the next node already sent its own reservation request, it will forward the intention message to the next node in the relevant route. The idea here is that, even if some reservations take longer than expected, the sink should be aware that it has to wait a little longer before passing to the next step in the setup process, which would be the collection of interference information.

The actual reservation procedure will be started at the sensor nodes when a timer expires indicating that a sufficiently long time has passed without overhearing new RSINT messages, as explained above. Those nodes that did not receive *intention* messages (so-called leaf nodes) will send their RSRQ message immediately and will set B_{req} to B_{own} since they do not need to forward any data on behalf of other nodes. If, on the other hand, a node did receive one or more *intention* messages, which turned it into a potential CH, it will start a new timer to know how long it should wait for the RSRQ messages indicated by the *intention* messages previously received. In other words, it will not start the reservation of its own resources until it has received all of the expected RSRQ messages or the relevant timer expires. When that happens, it will send a RSRQ message to the next node in its selected route asking for enough resources to send its own traffic plus that of its cluster members; that is, it will set the field B_{req} to $B_{\text{committed}} + B_{\text{own}}$.

Figure 5 shows a graphic description of the reservation process. In summary, the following set of control messages should be included to reserve bandwidth: RSRQ, RSRP, and RSACK messages. The RSRQ message can be used to set up or tear down a reservation, as mentioned above. Similary, the RSRP message can be used to send a positive or negative response. Notice that not only the intended receiver of an RSRQ message can send a response, but the nodes that overhear the message will also check if their available bandwidth is enough for this new connection and, if not, they will send back a response message indicating that the reservation is not possible. Lastly, the RSACK message can be used by the node that requested to set up or tear down a reservation to acknowledge receipt of the relevant response. The messages exchanged to establish a new reservation have the goal of reaching the whole set of potentially interfering neighbors at each hop. It is sometimes possible that, even if two nodes cannot communicate, they can still cause interference to each other. To take this into account, reservation-related messages can be transmitted with a higher power than all other messages, including those used in the other setup stages and even data frames transmitted during the normal operation of the system. This way, even those nodes that are too far away to be able to correctly receive data frames will be aware of the potential interference that can be caused by a reservation and will take that into consideration.

In addition to listening to these messages to keep track of the bandwidth usage in their neighborhood, nodes will also extract from RSRP and RSACK messages information as to the creation of new clusters that can potentially interfere with them, which is needed in the next protocol step.

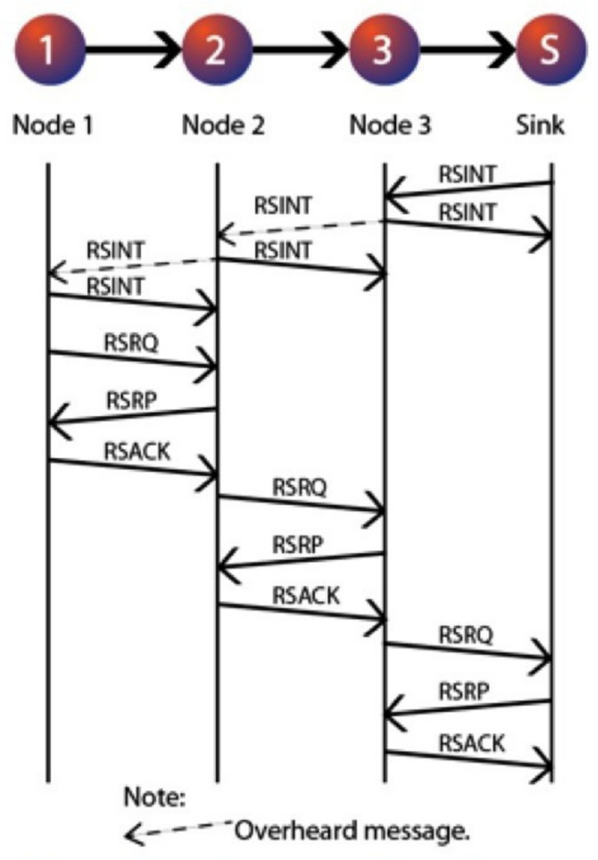
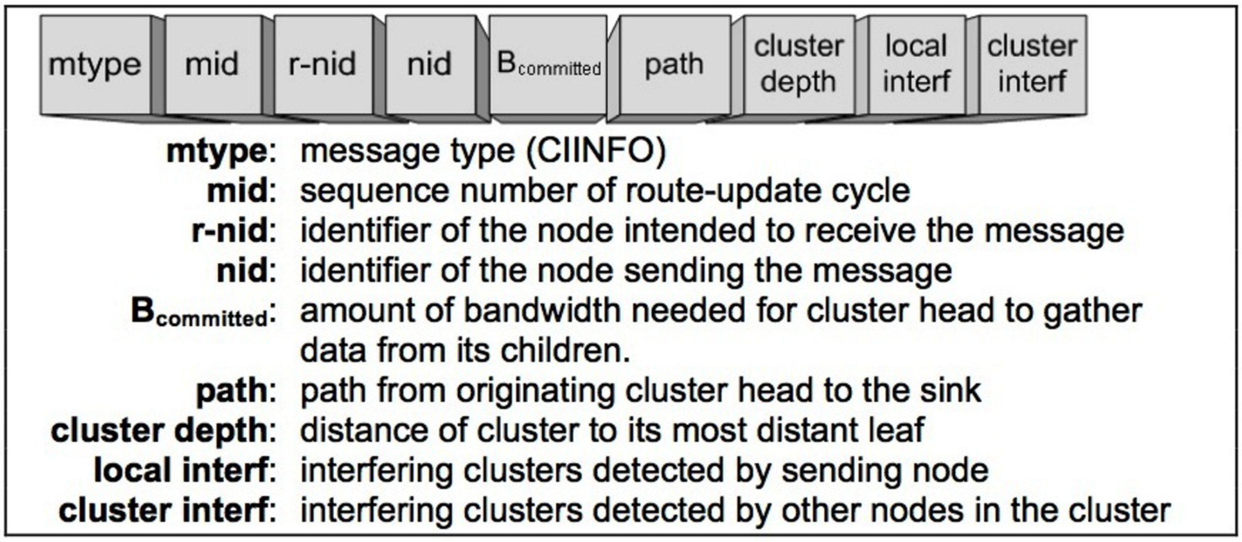


Figure 5 Route reservation phase.

3.3. Collection of cluster interference information

Information relative to the potential interference among neighboring clusters has to be collected to enable the scheduling of waking times of the different clusters without damaging overlaps. To do that, at the end of the route reservation phase, the sink will broadcast a CISTART (start of cluster-interference information collection) message that will be flooded over the network so that all nodes know that this new phase has started. These messages only have two fields, *mtype* set to CISTART and *mid* set to the current route update cycle number.

Nodes that are not CHs, referred to as leaf nodes since they are at the end of a routing branch, will be the ones starting the collection of cluster interference information. Each of these nodes will send a CIINFO message to its respective CH containing, in the *local interf* field, a list of the interfering clusters (not individual nodes) it detected, and a zero in the *cluster depth* field. In these initial messages, the fields $B_{\text{committed}}$, *path* and *cluster interf*, shown and described in Figure 6, will be empty indicating the recipient that the sender is not a CH and that the message does not have to be forwarded. When a CH receives



mtype:	message type (CIINFO)
mid:	sequence number of route-update cycle
r-nid:	identifier of the node intended to receive the message
nid:	identifier of the node sending the message
$B_{committed}$:	amount of bandwidth needed for cluster head to gather data from its children.
path:	path from originating cluster head to the sink
cluster depth:	distance of cluster to its most distant leaf
local interf:	interfering clusters detected by sending node
cluster interf:	interfering clusters detected by other nodes in the cluster

Figure 6 Format of messages to collect interference information.

from one of its cluster members a CIINFO message in which the *path* field is either empty or the originating node (first node in the path) is equal to the *nid* field, meaning that the message was generated by one of its children, it will update the list of interfering clusters detected by the members of its own cluster based on the *local interf* field just received, and it will update its own depth as the largest depth of its cluster members plus one; it will then forward the message towards the sink as long as the *path* field is not empty, and will add its own ID to the path. If, on the other hand, the received CIINFO message is such that the *path* field is not empty and the originating node ID is different from the *nid* field, meaning that the message was not generated by one of its children, then the node will simply forward the message towards the sink after adding its own ID to the path. When a CH has received a CIINFO message generated by each of its cluster members or a timer expires, it will in turn generate a new CIINFO message in which it will set the *path* field to its own ID, the *cluster depth* to its own calculated depth, it will include in the *local interf* field a list of the interfering clusters it detected directly and in the *cluster interf* field a list of the interfering clusters detected by its cluster members, except for those already included in the *local interf* field, and it will set the $B_{\text{committed}}$ field to the amount of bandwidth needed to collect the data that the node has committed to forward on behalf of its children.

As the sink receives the CIINFO messages, it will create a temporary array called *temp_sched* in which clusters will be sorted by their depth. Each array element will contain the cluster ID, its depth, the path to reach it, the bandwidth required by its members and a list of interfering clusters resulting from merging the *local interf* and *cluster interf* fields.

Figure 7 shows a summary of the process needed to collect cluster interference information.

3.4. Assignment of activity windows to the clusters

At the end of the process described in the previous section, the sink will have enough information to assign sufficiently long activity windows to the different clusters in such a way that they do not interfere with each other. The assignment of activity windows to the clusters is the last stage of what will be referred to as the

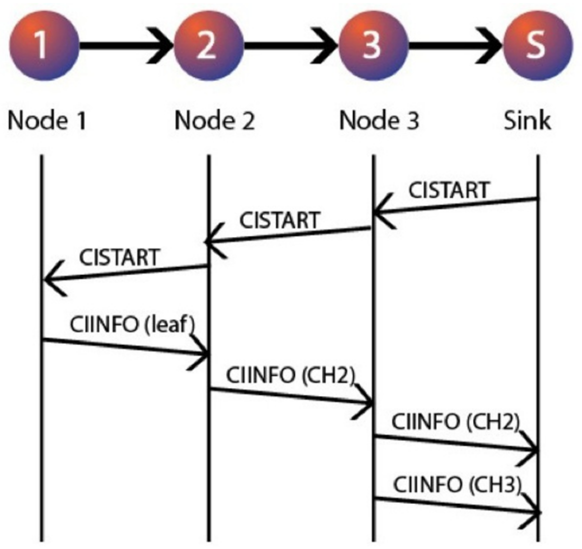


Figure 7 Collection of cluster interference information phase.

setup process. It is important to mention that the setup process will be repeated periodically, with a relatively low frequency (possibly in the order of hours or days) that depends on the size and density of the network, on the residual energy of nodes, on the traffic intensity, and/or other factors, with the goal of reassigning the responsibility of frame forwarding, hence redistributing the energy consumption to extend the system lifetime.

Activity windows will repeat periodically, giving the members of each cluster enough time to forward to the CH all of the information accumulated during the time they were inactive. The time interval that includes one activity window for every cluster in the system is known as a cycle, as shown in Figures 8 and 9. Depending on the availability of bandwidth and on the application's delay tolerance, a cycle may include idle periods in which no clusters are active to reduce the duty cycle and save more energy. The setup phase plus the sequence of cycles that precede the following setup phase is called a superframe. One or more non-interfering clusters may be active during each activity window, as exemplified in Figure 6. The algorithm accommodates the activity windows according to the depth of the CHs, so that information can move forward in every activity window to a node that has not had yet a chance to transmit in the current cycle. With this approach, all the information that is in the nodes' buffers at the beginning of a cycle will be guaranteed to reach the sink by the end of the same cycle. This is the main reason why we maintain that guaranteeing enough bandwidth to empty all buffers during each cycle is enough to make sure that all other QoS metrics (throughput, delay, jitter, and losses due to buffer overflow) will also be bounded as a consequence.

The process to assign activity windows to the clusters starts when the sink receives interference information messages from all of the CHs in the network or a timer expires. It will then create a two-dimensional structure called *cluster_sched* in which elements in the same column are such that they have the same depth and do not interfere with one another, and for that reason they can be scheduled to be active simultaneously; similarly, the clusters in the leftmost column will be scheduled first in each cycle and those in the rightmost column will be scheduled last. To achieve this, the sink removes one by one the clusters from the temporary array *temp_sched* it created before. If there are no columns in *cluster_sched* containing clusters of the same depth as the one just removed from *temp_sched*, the sink will insert it into the next empty column. If, on the other hand, there is already at least one column in *cluster_sched* containing clusters of the same depth as the cluster just removed from *temp_sched*, the sink will try to add this new cluster into one of such columns verifying first if there is no potential interference; if the cluster can be accommodated in more than one column, it will be included in one in which the difference between its required bandwidth and the maximum required bandwidth of clusters already in the column is either negative (if any) or as small as possible (if all of them are positive); if the cluster cannot be accommodated in any of the columns already in use due to potential interference, it will be inserted into the next empty column.

Once all the clusters have been moved into *cluster_sched*, the sink has to calculate the duration of the M resulting activity windows, each represented by a non-empty column of the relevant data structure. Let us first calculate the activity time each cluster needs during every cycle:

$$T_{\text{clust}}(j) = \frac{B_{\text{committed}}(j)}{R} \cdot T_{\text{cycle}} \tag{3}$$

where R is the wireless channel's achievable effective data rate, $B_{\text{committed}}(j)$ is the bandwidth needed by CH j to collect data from its children, and T_{cycle} is the duration of each cycle. T_{cycle} should be small enough so that the maximum delay that a frame will experience from the time it is generated to the time it is received by the sink is within acceptable values, and large enough to allow as many frame transmissions as needed by the members of all clusters. Notice that the maximum delay that a frame can experience is upper-bounded by twice the duration of a cycle. The reason for that is the fact that a frame can be generated after the activity window of the node's cluster has already passed within the current cycle, which means that it will reach the sink by the end of the next cycle. From here, the maximum

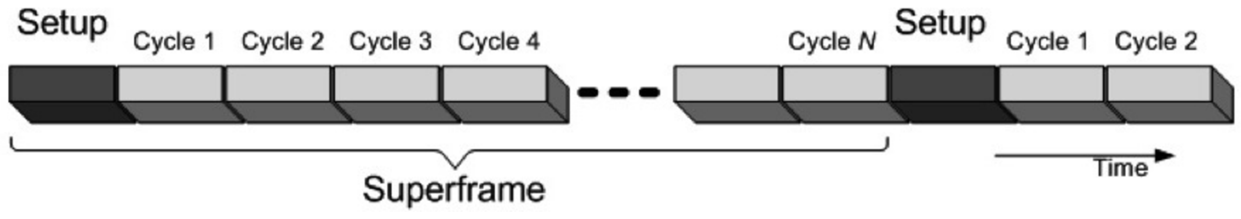


Figure 8 Superframe structure.

delay tolerated by the application being served should be less than 2 T_{cycle}.

We can now calculate the duration of the ith activity window as:

$$T_{\text{win}}(i) = \max_{j \in S_i} T_{\text{clust}}(j),\ i \in \{1, 2, \ldots, M\} \tag{4}$$

where S_i is the set of clusters to be scheduled within the ith activity window. Window sizes should be increased by a small amount to account for traffic that will flow in the opposite direction. Notice that, if:

$$\sum_{i=1}^{M} T_{\text{win}}(i) > T_{\text{cycle}} \tag{5}$$

it means that there is not enough bandwidth in the network to schedule non-overlapping activity windows. A possible solution in this case is to try to start a new cycle before the end of the previous one, but maintaining the condition of avoiding destructive interference among clusters. To do this, the sink will examine the consequences of overlapping consecutive cycles by an amount of time T_{overlap} equal to the duration of the first activity window. If there is no potential interference, the sink will analyze the possibility of overlapping consecutive cycles by an amount of time equal to the duration of the two earliest activity windows, and so on. Notice that clusters that are closer to the sink will need more bandwidth, so that it is possible that the last activity windows will be much larger than the first ones; this has to be taken into account when analyzing the possibility of allowing overlaps. When this procedure is finished, if the following inequality still holds:

$$\sum_{i=1}^{M} T_{\text{win}}(i) - T_{\text{overlap}} > T_{\text{cycle}} \tag{6}$$

it means that the system does not have enough bandwidth to satisfy the QoS requirements of its nodes. When this happens, a topology-control mechanism may optionally be run [29] to decide if a reduced number of nodes (hence a smaller amount of traffic) would be enough to satisfy coverage and connectivity requirements. Topology-control mechanisms are not required for the proper operation of QUATTRO and are therefore out of the scope of this article.

Notice that the reservation procedure is in charge of a first assessment as to whether the system has enough resources to satisfy QoS requirements. However, allocating different activity windows to the clusters means that, when a given node is transmitting, all members of every neighboring cluster have to be silent, and not only those that can experience a collision with the relevant transmitting node. In other words, allocating activity windows is more resource-demanding than local bandwidth reservation, meaning that the latter may be successful and still the former can fail.

When the sink has finished these calculations, it has to inform each CH of the activity window to which it has been associated and of its length. To that end, the sink will send an activity-window notification (AWN) message to each CH, as shown in Figure 10. The AWN

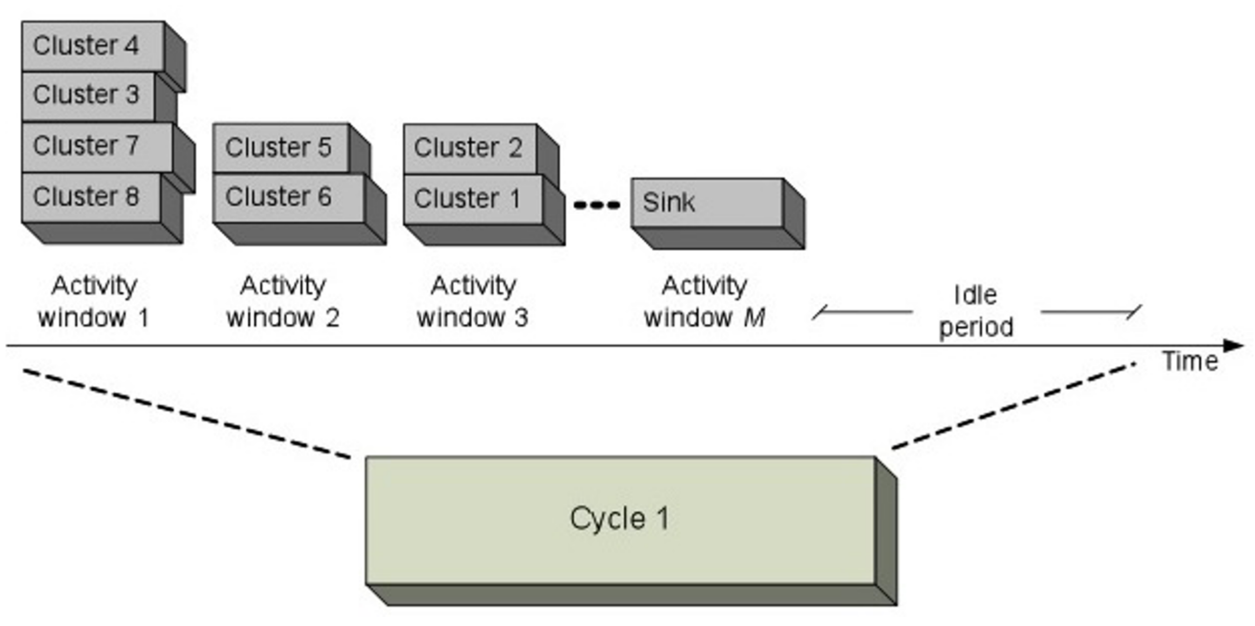


Figure 9 Cycle structure.

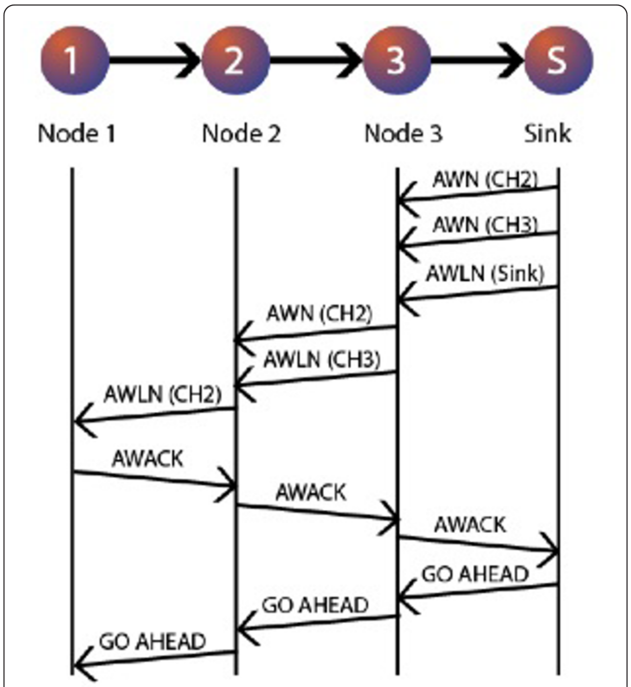


Figure 10 AWN phase.

message, shown in Figure 11, contains all the information that has to be exchanged in the AWN process. Upon reception, the CH will in turn broadcast an activity-window local notification (AWLN) message to inform its cluster members.

The leaf nodes will acknowledge receipt of the AWLN message by sending an AWACK message. A CH will wait to receive AWACK messages from all of its cluster members to generate its own AWACK message. This process continues until the sink eventually receives the corresponding acknowledgments from its own cluster members.

The system is now ready to start working in a QoS-aware, collision-free, safe-sleep fashion. To make that stage start, the sink sends out a GOAHEAD message that is flooded throughout the network to make every node aware of it. This message contains the time at which the first cycle will begin, as shown in Figure 11, which is what nodes still need to know in order to calculate their sleep and wake times. The duty cycle is included so that nodes can know what fraction of time during each cycle the whole system is idle; this information will be useful when the network has to be reconfigured, as explained in Section 3.7.

Up to this point, communication among nodes takes place using a contention-based MAC protocol, such as CSMA/CA in its IEEE 802.15.4 or 802.11 versions. If IEEE 802.11 (DCF) is used, it would be advisable to force nodes to backoff before every transmission, even when the channel is initially idle, to reduce collisions.

3.5. Normal operation phase

During the normal operation phase, the only active nodes will be those corresponding to the clusters whose activity window includes the current time. Access to the channel is not through contention anymore, but cluster members will wait to receive a poll from the CH indicating that the receiving node is allowed to send. The polls may allow the transmission of one frame at a time, or all frames corresponding to a single upper-layer packet (MSDU), or as many frames as possible during a pre-specified amount of time (TXOP), proportional to the amount of bandwidth requested during the reservation phase. In our simulations, we adopted the TXOP approach.

A node that receives a poll will respond with a *Data* frame if it has information to send, indicating respectively with the *More_Frag* and *More_Data* bits if there are more frames still to send corresponding to the packet being transmitted and if there are more packets to forward in addition to the one currently in service. If the node does not have information to send, it will respond with a *Null* frame. Either way, the CH can detect when a node has finished sending the information it stored in its buffer and stop polling it during the current cycle. The member node can go to sleep at this time to further save energy. If all the member nodes go to sleep before the end of the activity window, the CH can follow suit. This approach is designed to save energy by avoiding collisions and idle listening as much as possible.

If a node fails repeatedly to respond to polls, an error message will be generated by the CH to alert the sink.

3.6. Network-wide synchronization

Synchronization is very important in time-shared access protocols, such as this one, to keep activity windows from overlapping because of clock drifts. To achieve synchronization, a mechanism such as the one described in [30,31] can be used from the beginning of the initial setup process. As soon as a node identifies its parent node through the process described in Section 3.1, it can start exchanging synchronization messages with its parent and adopting its clock. This will eventually cause for all nodes to be synchronized to the sink's clock. After a successful reservation procedure, the node can start exchanging synchronization messages with its selected CH and again adopting its clock. This process can continue during the data exchange within the normal operation phase to avoid considerable drifts. *Poll*, *Data* and *Null* frames can carry time stamps to achieve this task.

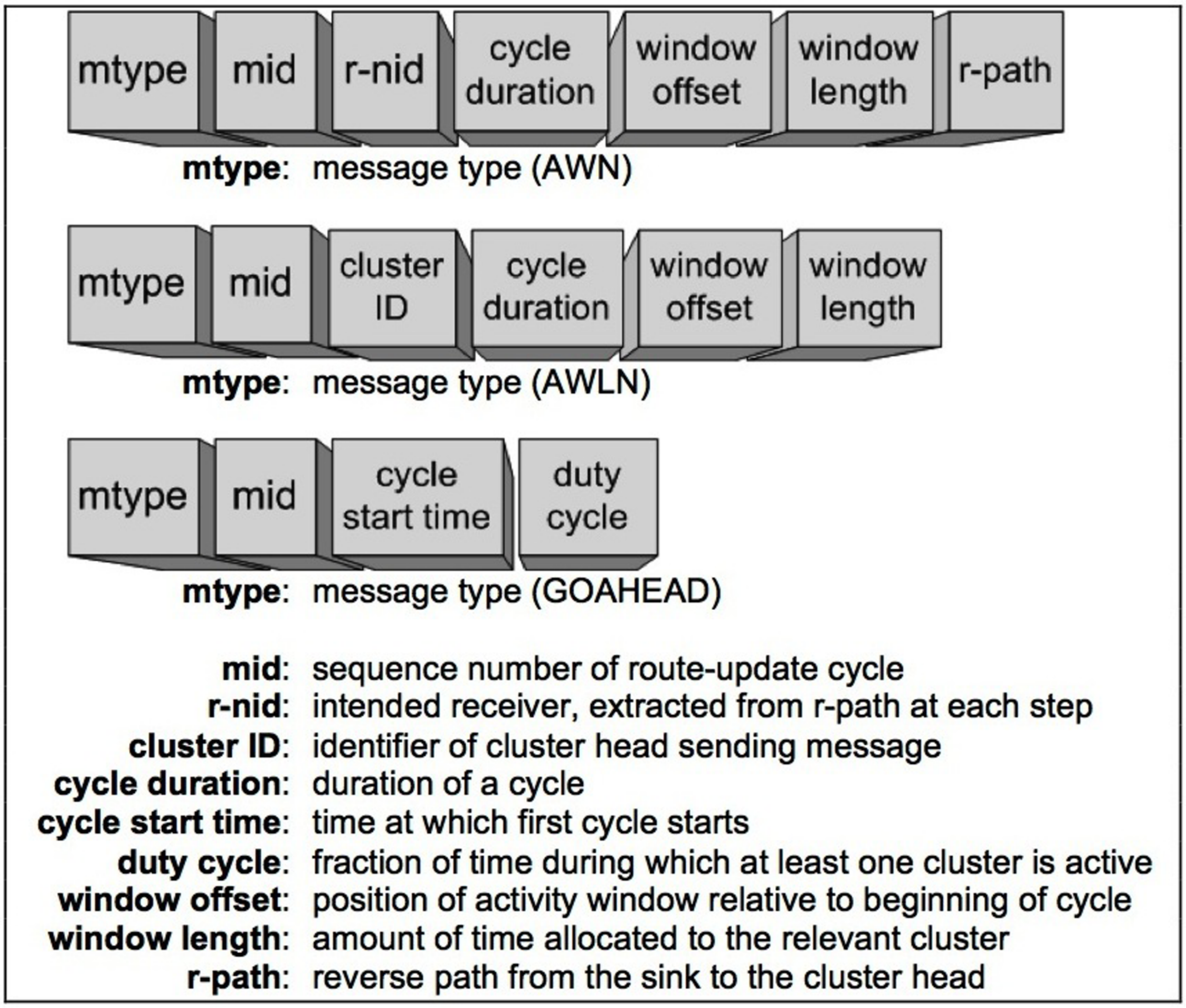

mtype
mid
r-nid
cycle duration
window offset
window length
r-path
mtype: message type (AWN)
mtype
mid
cluster ID
cycle duration
window offset
window length
mtype: message type (AWLN)
mtype
mid
cycle start time
duty cycle
mtype: message type (GOAHEAD)
mid: sequence number of route-update cycle
r-nid: intended receiver, extracted from r-path at each step
cluster ID: identifier of cluster head sending message
cycle duration: duration of a cycle
cycle start time: time at which first cycle starts
duty cycle: fraction of time during which at least one cluster is active
window offset: position of activity window relative to beginning of cycle
window length: amount of time allocated to the relevant cluster
r-path: reverse path from the sink to the cluster head

Figure 11 Format of AWN messages.

Notice that even though this is a time-shared access technique, there is no need for nodes to be very precisely synchronized as in TDMA-based systems. The reason for this is the fact that, even during the polling-based data transmission, this remains to be a random-access technique in the sense that every transmission includes a preamble that indicates when a frame is about to begin. Dividing time into activity windows is only used as a reference to specify, with relatively low accuracy, when attempts to access the medium are allowed to be made.

3.7. Reconfiguration

Being a CH can be energy consuming because they have to be awake during two activity windows and because, in addition to sending their own collected data, they have to forward those of their cluster members. That is why nodes will alternate taking the role as CHs by rerunning periodically the setup process. The redistribution of energy consumption relies on the fact that nodes select a route based, among other aspects, on the available energy of the weakest node in that path, as explained in Sections 3.1 and 3.2. If there is only one sink in the system, the redistribution of energy consumption may not be as effective as we would like since nodes closer to the sink will always carry more traffic than those farther away. Hence, having several sinks and alternating their activity is advisable.

In preparation for the beginning of the new setup procedure, after some time of working in the normal operation phase, the current sink will broadcast a RCWK (wake up) message that will be repeatedly rebroadcast by CHs, at the beginning of each subsequent cycle, with the goal of eventually reaching the sink that has to be active during the next superframe. When the sought-after sink receives the RCWK message, it will immediately respond with a RCWRP (wake-up response) message to confirm that it is ready to play its role. The CHs that hear the RCWRP message will forward it until it reaches the currently-active sink; they will also stop rebroadcasting the RCWK message previously received. It is clear that sleeping sinks will have to turn on their receivers periodically to listen for wake-up messages. It is also recommended for them to use overheard synchronization-related messages to remain synchronized with the currently active sink. If no answer to the RCWK is received during a certain number of cycles, the current sink will try to wake up another sink or, if none responds, it itself will remain active during one more superframe.

The next step is for the currently active sink to broadcast an reconfiguration notification (RCN) message indicating that the setup phase has to start again. Receiving an RCN message indicates a CH to stop rebroadcasting the RCWK message if it has not stopped yet. If there is only one sink in the whole network, the RCWK and RCWRP messages will not be used and the sink will proceed immediately to send the RCN message.

Each CH will rebroadcast the RCN message during its corresponding activity window to make its members aware of the imminent reconfiguration phase. Those members that are at the same time heads of other clusters will retransmit the message to their respective cluster members, and so on until all nodes in the system are reached. The RCN message will have a time-to-setup field, as shown in Figure 12, which indicates the number of cycles still pending until the beginning of the setup process. This field is initially set by the sink to a value equal to its depth. Each CH will reduce this value by 1 before retransmitting it, so that setup messages will start to be transmitted one cycle after the farthest leaf receives this message; by this time every node in the system is aware and ready to start the setup phase, either by remaining awake during the time in which the system would otherwise be idle or by stopping temporarily the data transfer phase, depending on whether the *reconfig_type* field has the value *normal* or *exclusive*. In other words, to avoid having to stop the data transmission during the setup process, in a normal reconfiguration nodes can use the portion of each cycle in which all clusters would normally be sleeping, assuming that the duty cycle is sufficiently less than 1. If this is the case, nodes will continue to send information to the current sink until such an instant in which it is time to send the AWACK message to acknowledge receipt of the activity window they will use to send data to the new sink. If the duty cycle is too close to 1, on the other hand, an exclusive reconfiguration will be needed in which nodes will have to stop data transmission when the time-to-setup variable counts down to zero so that they can be ready to execute the new setup process.

The reconfiguration procedure can also be initiated if a severe problem is detected in the system, such as the loss of a CH, which would isolate a set of nodes from the sink. If this happens, an error message will be generated by the node detecting the problem and forwarded until it reaches the sink, as mentioned in Section 3.5. Depending on the extent of the damage (e.g. number of nodes that have failed) the sink can decide if a complete setup procedure is needed, or if a fast reconfiguration might be enough. This will again be indicated in the *reconfig_type* field of the RCN message transmitted by the sink. In a fast reconfiguration, nodes do not go through the route discovery and weight assignment phases, described in Section 3.1, but use the information collected during the most recent setup process and go straight into the reservation phase described in Section 3.2. In fact, nodes that have not been disconnected can send the RSRQ message to the same node currently acting as its CH, which will have a high probability of accomplishing a successful reservation, thus saving time and energy. Regardless of whether the reconfiguration is complete or fast, it can be carried out using the unused portion of time (normal) or stopping data transmission to use the whole bandwidth (exclusive).

A node will know that it has become isolated from the rest of the network if polls are not received from its CH during several consecutive cycles. When that happens, if the node is itself a CH, it will stop polling its cluster members to indicate them that their data cannot be forwarded all the way up to the sink. To be reconnected to the rest of the network, isolated nodes will remain awake (during the time that the system would otherwise be idle, if the duty cycle is sufficiently less than 1, or constantly if not) waiting for the RCN message that will start a new reconfiguration process, allowing them to select a new CH.

Reconfiguration is the last ingredient of our proposal. Figure 13 shows the whole protocol at a glance, including all of its phases and the events that trigger the beginning and end of each of them.

4. Applicability of our proposal

The following is a typical scenario in which our protocol would be highly useful. Assume a security surveillance system either for home or for office, commercial or government buildings. Assume that sensor nodes are equipped with at least two types of sensors, one to detect the presence of intruders (motion sensors) and another to collect video once suspicious activity has been identified. Data collected by the motion sensors may not be continuously transmitted to the sink, but

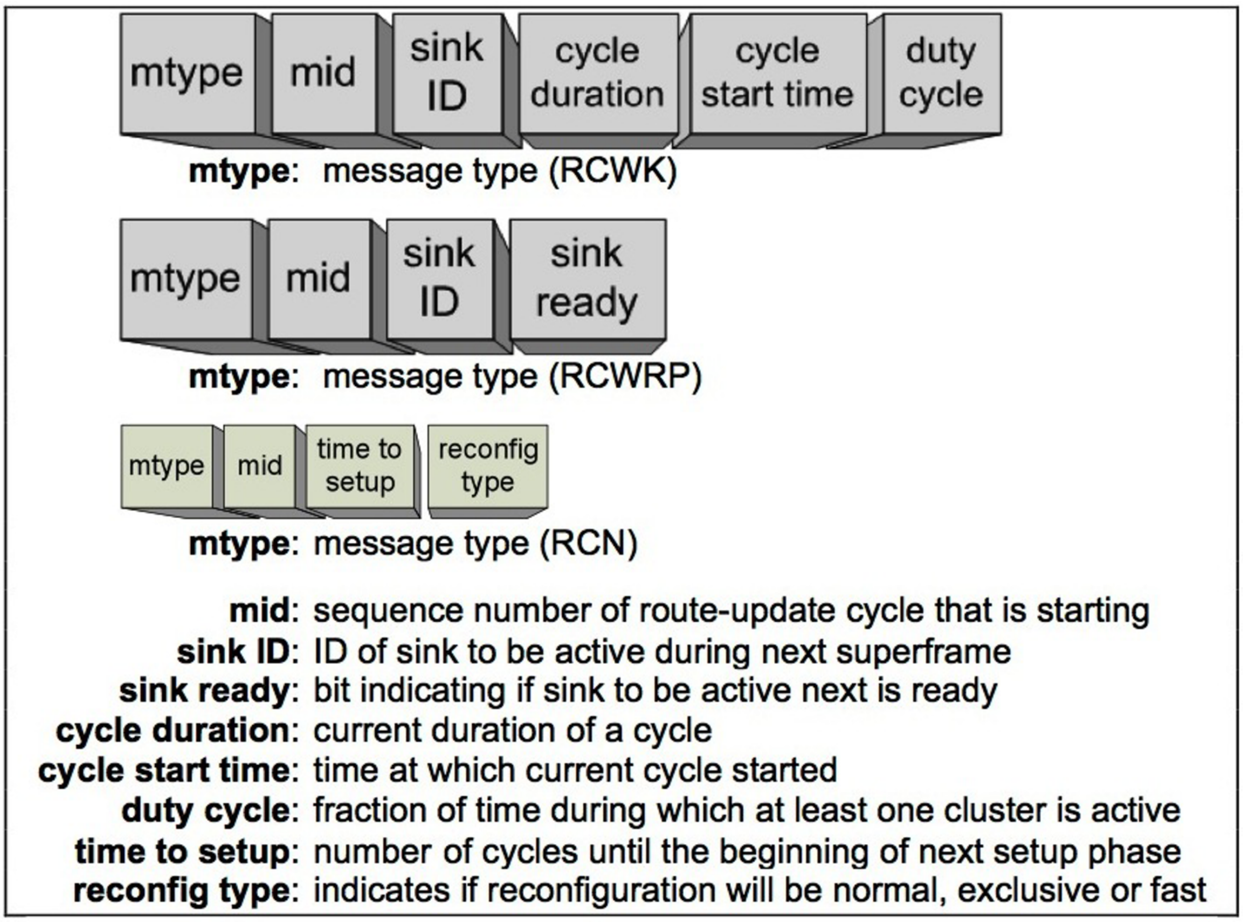

mtype
mid
sink ID
cycle duration
cycle start time
duty cycle
mtype: message type (RCWK)
mtype
mid
sink ID
sink ready
mtype: message type (RCWRP)
mtype
mid
time to setup
reconfig type
mtype: message type (RCN)
mid: sequence number of route-update cycle that is starting
sink ID: ID of sink to be active during next superframe
sink ready: bit indicating if sink to be active next is ready
cycle duration: current duration of a cycle
cycle start time: time at which current cycle started
duty cycle: fraction of time during which at least one cluster is active
time to setup: number of cycles until the beginning of next setup phase
reconfig type: indicates if reconfiguration will be normal, exclusive or fast

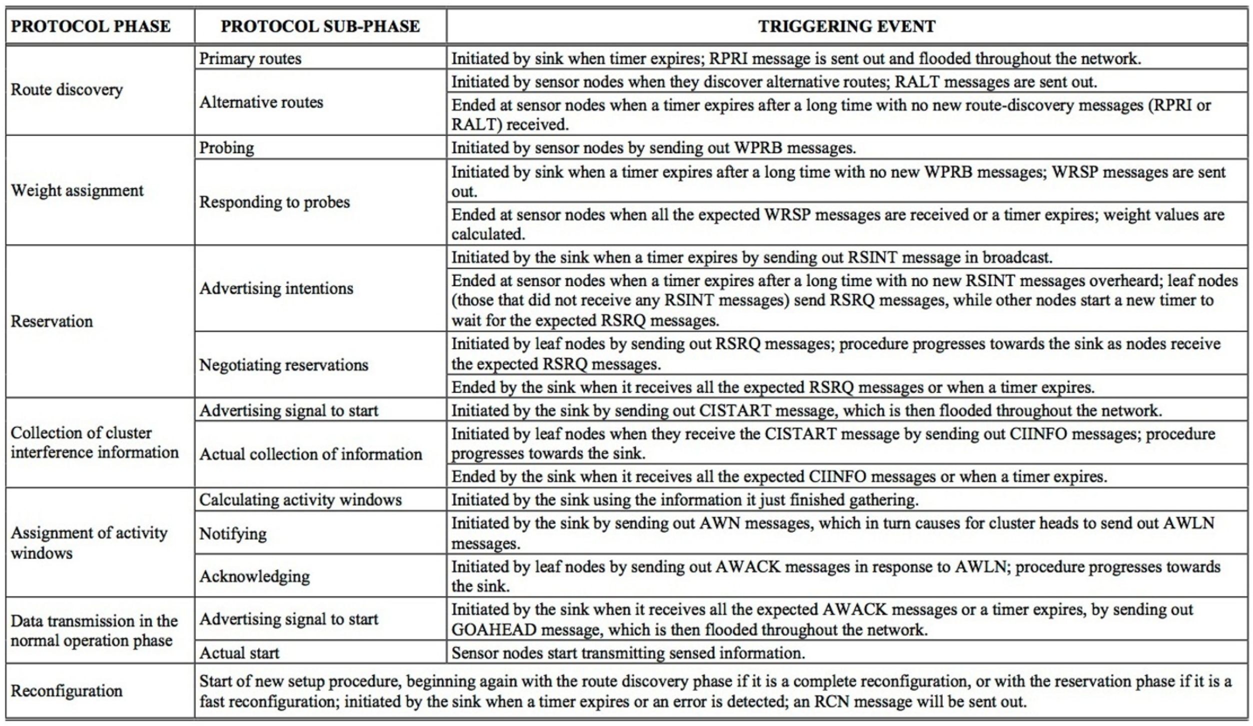
Figure 12 Format of reconfiguration messages.

can be used locally by sensor nodes to know when to alert the sink and when to activate the video sensors. Video information, on the other hand, will be transmitted continuously to the sink once a video sensor is activated. If there are intruders in the building, suspicious activity will eventually be detected by several motion sensors located within a certain region, and the corresponding video streams will have to start flowing concurrently toward the sink. Even if not all video sensors become active at the same time, the fact that a set of close neighbors become active concurrently makes reservation of enough resources, as proposed in this study, to be necessary and pertinent.

A similar scenario would be a surveillance system for the battlefield. Sensor nodes may be equipped to detect the presence of enemy forces and, once identified, to start collecting video so that military personnel analyzing the data collected at the sink in real time can estimate the number of rival troops that they are faced with and identify the type of equipment they have.

Our proposed protocol can work with different types of traffics. If the maximum amount of data that can be generated by a node during a cycle can be known beforehand, such as in constant-bit-rate or leaky-bucket-constrained streams [28], then deterministic (hard) QoS guarantees can be offered. If, on the other hand, traffic presents a higher variability so that reserving enough resources to process the maximum amount of data that can be generated during a cycle is not practical, then stochastic (soft) guarantees can still be offered, in the sense that QoS requirements can be violated but with a quantifiable low probability. It is out of the scope of this study to design a method to calculate the necessary amount of bandwidth that has to be reserved so that, when the offered traffic varies according to a certain random behavior, the probability of violation of QoS guarantees is upped-bounded by a desired value.

It might be a little premature to think of implementing this proposal in real devices, especially because video transmission requires high bandwidth capabilities



PROTOCOL PHASE	PROTOCOL SUB-PHASE	TRIGGERING EVENT
Route discovery	Primary routes	Initiated by sink when timer expires; RPRI message is sent out and flooded throughout the network.
	Alternative routes	Initiated by sensor nodes when they discover alternative routes; RALT messages are sent out.
		Ended at sensor nodes when a timer expires after a long time with no new route-discovery messages (RPRI or RALT) received.
Weight assignment	Probing	Initiated by sensor nodes by sending out WPRB messages.
	Responding to probes	Initiated by sink when a timer expires after a long time with no new WPRB messages; WRSP messages are sent out.
		Ended at sensor nodes when all the expected WRSP messages are received or a timer expires; weight values are calculated.
Reservation	Advertising intentions	Initiated by the sink when a timer expires by sending out RSINT message in broadcast.
		Ended at sensor nodes when a timer expires after a long time with no new RSINT messages overheard; leaf nodes (those that did not receive any RSINT messages) send RSRQ messages, while other nodes start a new timer to wait for the expected RSRQ messages.
	Negotiating reservations	Initiated by leaf nodes by sending out RSRQ messages; procedure progresses towards the sink as nodes receive the expected RSRQ messages.
		Ended by the sink when it receives all the expected RSRQ messages or when a timer expires.
Collection of cluster interference information	Advertising signal to start	Initiated by the sink by sending out CISTART message, which is then flooded throughout the network.
	Actual collection of information	Initiated by leaf nodes when they receive the CISTART message by sending out CIINFO messages; procedure progresses towards the sink.
		Ended by the sink when it receives all the expected CIINFO messages or when a timer expires.
Assignment of activity windows	Calculating activity windows	Initiated by the sink using the information it just finished gathering.
	Notifying	Initiated by the sink by sending out AWN messages, which in turn causes for cluster heads to send out AWLN messages.
	Acknowledging	Initiated by leaf nodes by sending out AWACK messages in response to AWLN; procedure progresses towards the sink.
Data transmission in the normal operation phase	Advertising signal to start	Initiated by the sink when it receives all the expected AWACK messages or a timer expires, by sending out GOAHEAD message, which is then flooded throughout the network.
	Actual start	Sensor nodes start transmitting sensed information.
Reconfiguration	Start of new setup procedure, beginning again with the route discovery phase if it is a complete reconfiguration, or with the reservation phase if it is a fast reconfiguration; initiated by the sink when a timer expires or an error is detected; an RCN message will be sent out.	

Figure 13 Protocol at a glance.

that, to the best of our knowledge, are not yet available in currently commercially available motes. The protocol itself, however, is not as demanding in terms of computer power as it may seem. The protocol operation is divided into well-defined states, as shown in Figure 13, and in each state only a small subset of messages and events are expected; similarly, the corresponding actions only involve the transmission of new messages and the starting or stopping of some timers. No complicated calculations are required from the nodes. Nonetheless, there is currently a fair amount of research in this direction because, as mentioned in the introduction, technology advances very rapidly, especially when it is driven by commercial success.

5. Performance evaluation

We first evaluate setup costs measured in terms of time and energy, which can be considered as the price to pay in overhead for the possibility to work not only with QoS guarantees, but also without collisions and with minimum idle listening, thus achieving high energy savings.

The second evaluation focuses on the performance of our proposed protocol during data transmissions in the normal operation phase. In this case, performance is measured in terms of achieved throughput, delay, and energy consumption. To put QUATTRO into perspective, we compare it with SMAC [17] which, as mentioned in Section 2, is one of the most important protocols of reference in WSN. Even though SMAC was not designed to provide QoS, the fact that its main objective is to save energy allows us to assess how energy-efficient our proposal is in addition to providing the other already mentioned benefits.

In the previous two analyses, we use random deployment of nodes in the selected area. When this is the case, it is common that some clusters do not interfere with others, which allows for some bandwidth and time gains by scheduling non-interfering clusters to be active simultaneously. To analyze the limit case in which there are no such bandwidth or time gains, we also include in this study the case in which nodes are located in a row, in the sense that each node has connectivity with at most two nodes, one that becomes its parent and the other, if any, that becomes its child. This scheme is more restrictive because of the fact that a CH has to wait to receive data from all of its children before starting its own transmission.

Finally, we evaluate reconfiguration costs, which allows us to analyze in detail the tradeoff between the time and energy invested and the benefits achieved in return when the different options are considered: normal or exclusive, and complete or fast reconfigurations. We also examine how far apart reconfiguration periods should be for the protocol to remain efficient.

We evaluated the performance of our proposal through simulations using OPNET [32]. Ten simulations are run for every case, each with a different seed for the random numbers, and the results shown correspond to the average values and the associated 95% confidence intervals. We assume an underlying IEEE 802.11-compatible physical layer running at 1 Mbps. This assumption is based again on the fact that high bandwidth is required from nodes. In our models, no frames are lost due to channel errors, but all losses are due to collisions. If the system is to be analyzed with channel errors, then this fact has to be taken into consideration when the wireless channel's effective data rate R is estimated, as mentioned in Section 3.2. It is worth emphasizing that, even though channel errors were not considered in our simulations, provisions are made to recover from them in the protocol.

The coverage area or transmission range of each node is assumed to be 10 m. Frames containing control messages are 100 bits long and data frames are 1 kbits long.

5.1. Setup cost evaluation

There are two key factors that affect setup time, which are node density and maximum number of hops from the sink to the farthest sensor node. We include two sets of simulations:

1. One in which the number of nodes is fixed to 100 and the total deployment area is varied, and
2. Another in which the deployment area is fixed to 25×25 m^2 and the number of nodes is varied.

The goal in both cases is to analyze the system under different node densities. The difference between them lies in the fact that, as node density is increased in the first set of simulations by reducing the deployment area, the maximum number of hops is eventually reduced to 1, which does not happen in the second set of simulations. The sink is always assumed to be in the center of the deployment area.

Regarding the first set of simulations, Figure 14 shows the time needed to discover routes and assign weights, as well as the time needed to successfully negotiate all reservations as a function of node density (nodes per coverage area–n/CA). The lowest and highest densities correspond to areas of 65×65 and 18×18 m^2, respectively. Similarly, Figure 15 shows the average number of control messages per source node needed to accomplish route discovery and weight assignment as well as reservation negotiation. We can see in both figures that graphs increase up to a certain point and then start to decrease as the density increases. That behavior is due to the fact, mentioned above, that when the total deployment area is reduced beyond a certain threshold, all nodes are one-hop apart from the sink, which greatly simplifies the setup process. The maximum number of hops in these cases is 5, corresponding to the smallest node density (7.4 n/CA).

Figures 16 and 17 show the results obtained in the second set of simulations. The lowest and highest densities correspond to 2 and 159 nodes, respectively. Nodes can be up to two hops away from the sink. In these cases, the setup time and the number of control messages keep increasing as the node density increases, as expected.

We can see in both scenarios that the time needed for the entire setup phase is less than a minute. We can also see that the price to pay in terms of extra transmissions is about 70 control messages in the worst case. We believe that the overhead incurred by our proposed protocol, both in time and control messages, is very reasonable given the great benefits obtained when the system enters the collision-free transmission phase.

The energy consumption is based on Atheros energy model for WLAN products [33]. The power consumption assumed is as follows: Transmit = 2 W, Receive = 0.9 W, and Listen = 0.8 W. WLAN devices are usually expected to have coverage areas much greater than the 10 m assumed in our simulations for a WSN. These results are presented for illustrative purposes only, but sensor nodes in a real system may consume far less energy.

Figure 18 shows the energy consumed during the setup phase in the second set of simulations, based on Atheros energy model for WLAN products [33].

5.2. Performance during data transmission in random deployment

To evaluate the performance of QUATTRO during data transmission, we reproduced the second set of simulations in Section 5.1, in which an increasing number of nodes are randomly deployed in a fixed-size area. In each simulation, sensor nodes spend 60 s transmitting data to the sink.

Each sensing node generates data traffic at a constant bit rate (CBR) of 4 kbps. Notice that handling CBR traffic can be considered the worst-case scenario if we are dealing with hard QoS guarantees. The reason for this is the fact that, if traffic were generated with variable rates, we would have to reserve enough bandwidth to transmit the maximum amount of data that may accumulate during a cycle, which means that the offered traffic would be on average less than what the network can handle. If the traffic is CBR, on the other hand, the amount of traffic generated by the nodes during each cycle is comparable to the maximum that the network can handle.

We use the following metrics to evaluate our proposal and compare it to SMAC:

• *Average throughput*, defined as the fraction of the frames generated by the sensor nodes that successfully arrive at the sink.

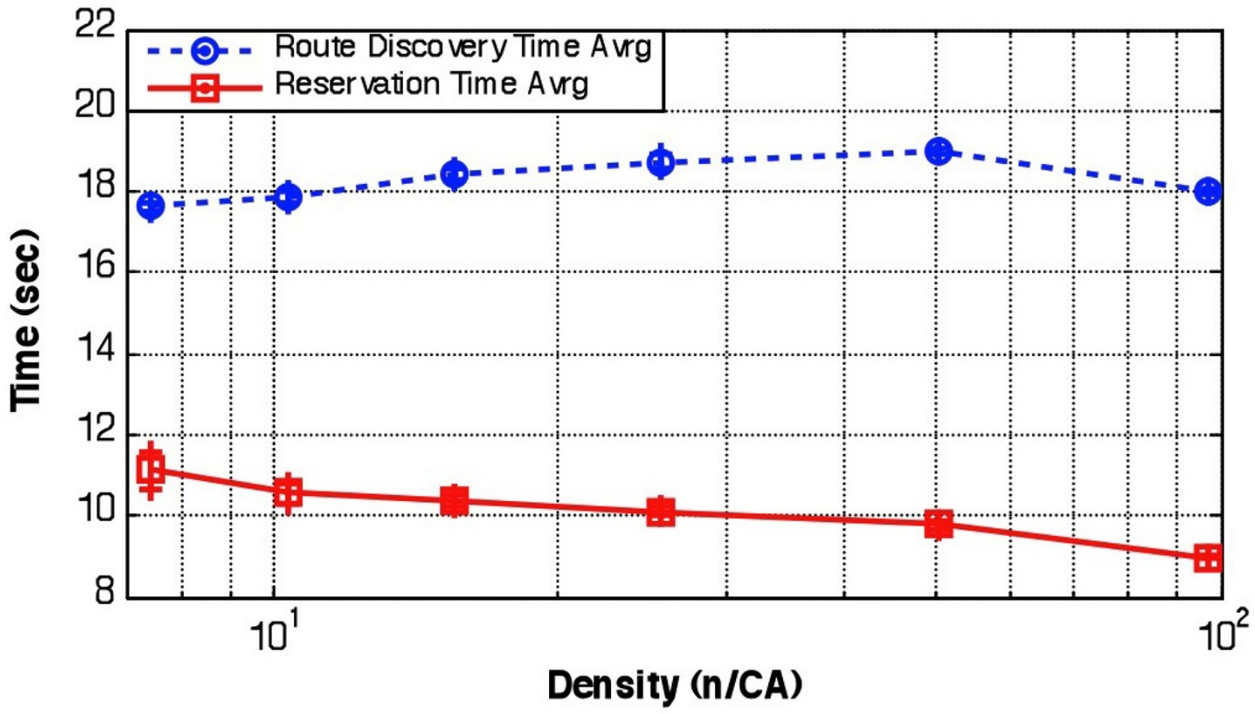


Figure 14 Setup time, 100 nodes in area of varying size.

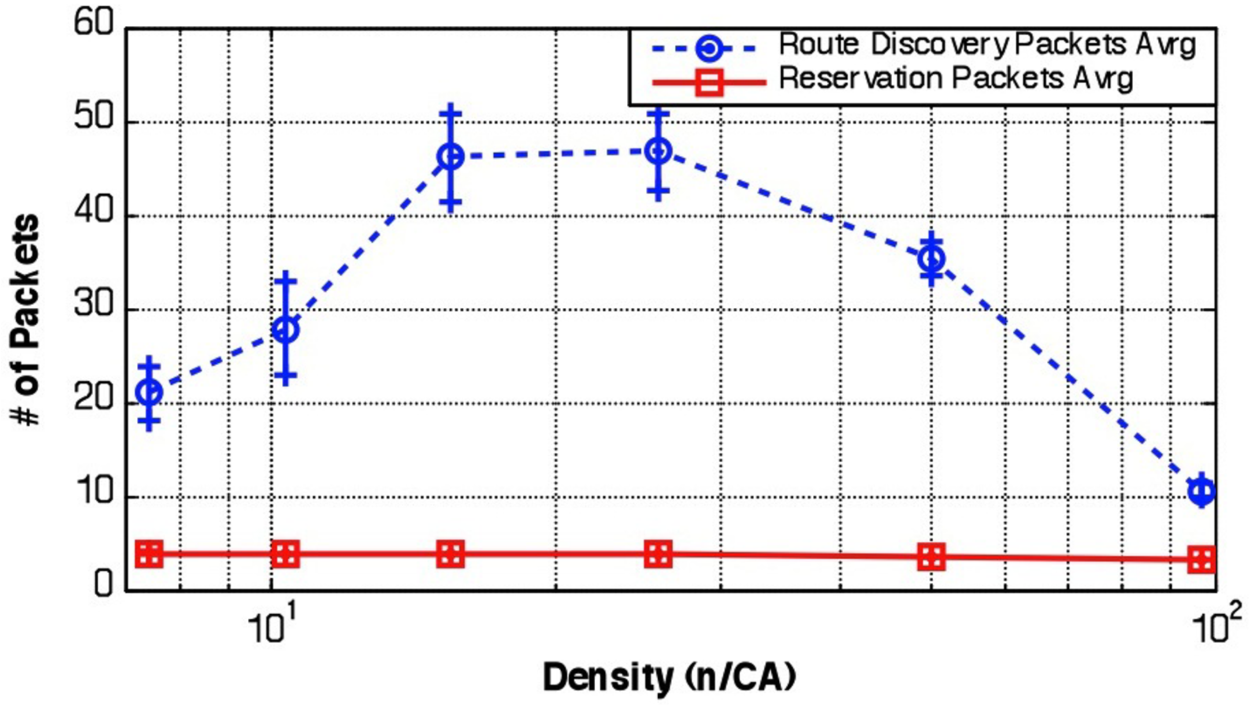


Figure 15 Setup packet count, 100 nodes in area of varying size.

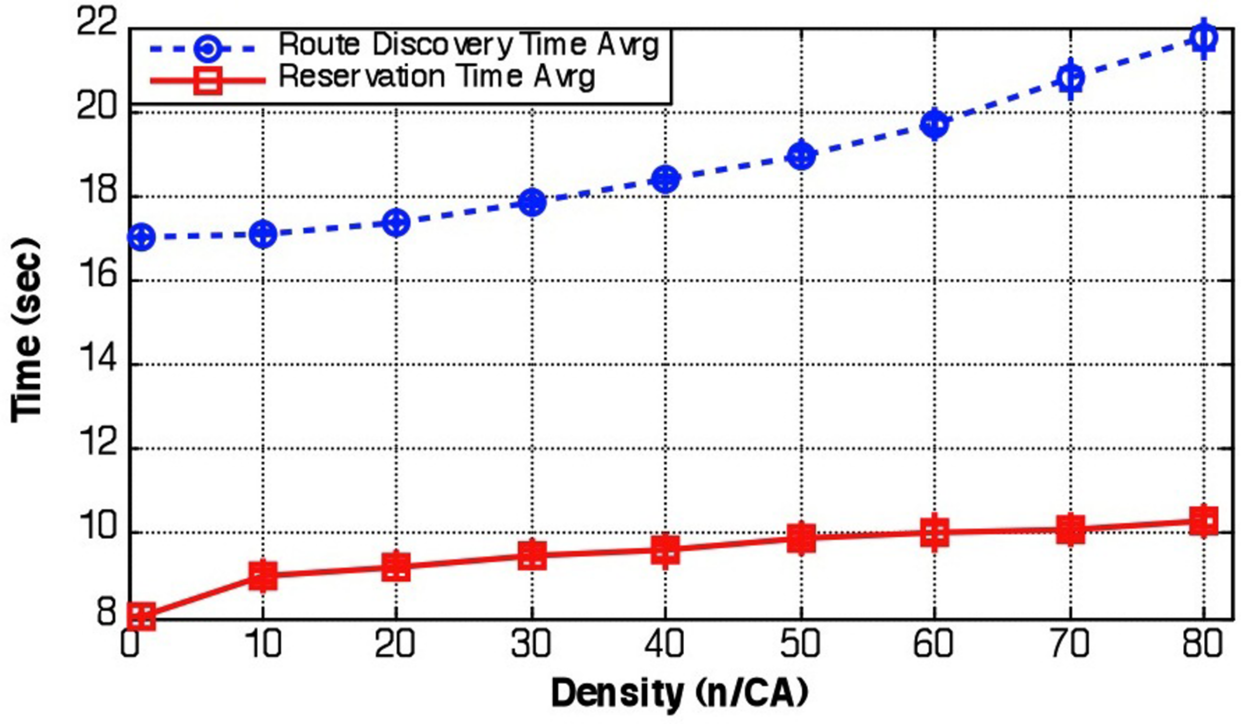


Figure 16 Setup time, varying number of nodes in fixed area.

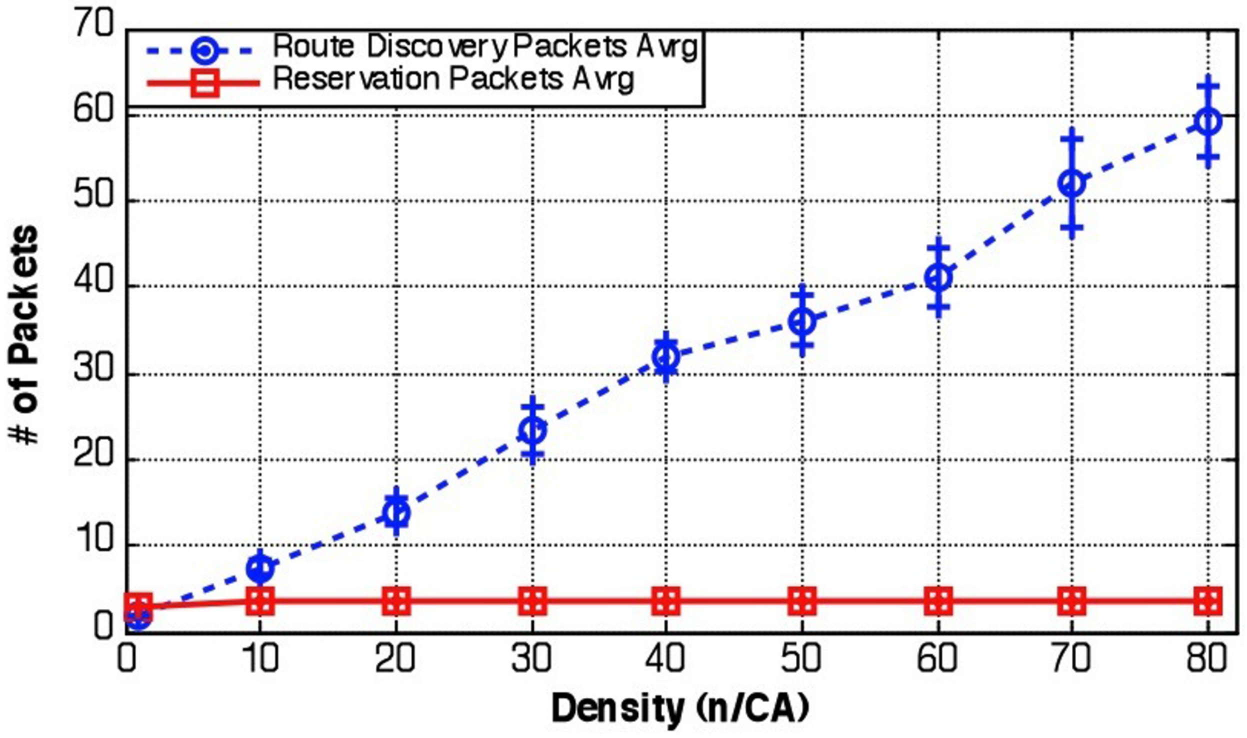


Figure 17 Setup packet count, varying number of nodes in fixed area.

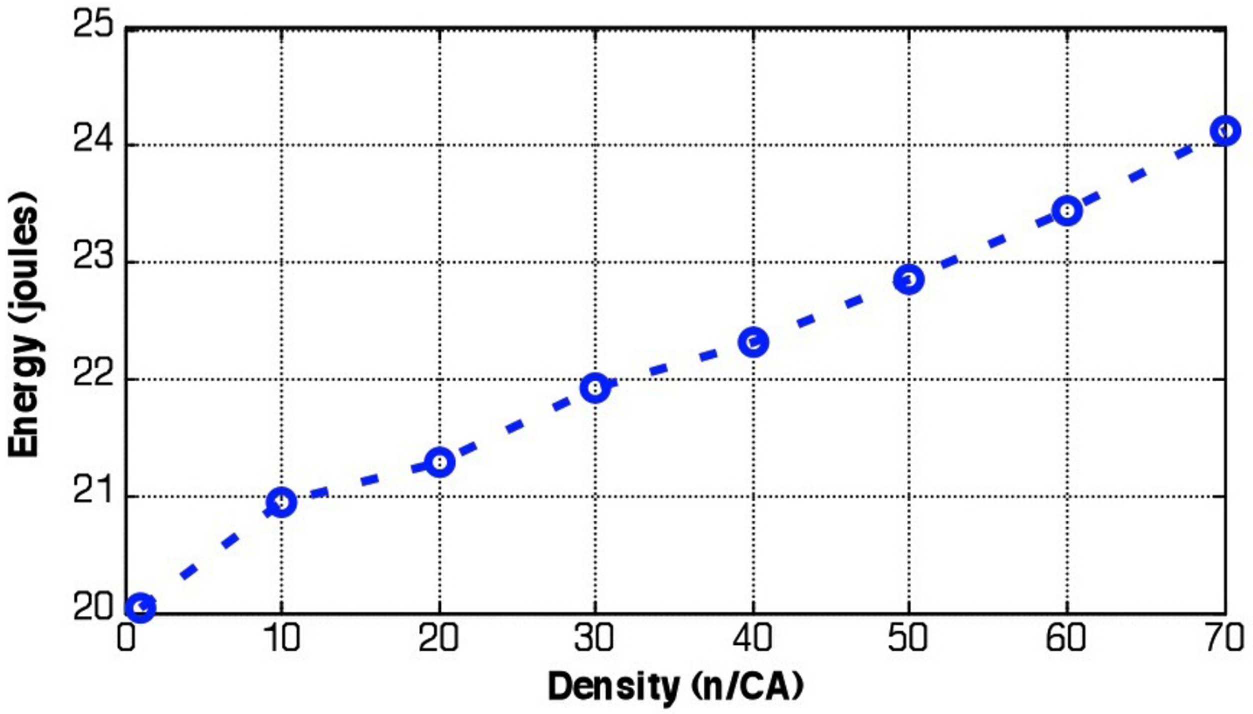


Figure 18 Energy consumed during configuration phase.

• *Average frame delay*, defined as the average time elapsed from the generation of a data frame at a sensor node to its arrival at the sink.

• *Average amount of energy consumed* to transmit data frames, per sensor node over the entire simulation time.

• *Fraction of time on*, defined as the percentage of time that sensor nodes remain awake on average, regardless of whether they are transmitting, receiving or listening idly.

We compare QUATTRO and SMAC using the same scenarios, routes, and data transmission rates. The cycle duration in both QUATTRO and SMAC is 0.25 s and the selected duty cycle for SMAC was 10%. Note that, according to the way SMAC was designed, a node will remain awake more than the designated duty cycle if there are data frames pending to be transmitted in which it itself is the transmitter or the intended receiver. We include results for SMAC using 7 and 255 as the maximum number of retransmission attempts.

Figure 19 shows the average throughput for all the three methods considered. We can see that QUATTRO achieves a 100% throughput in all cases, as expected, given its reserved resources and scheduled transmissions. SMAC, on the other hand, only achieves 100% throughput when the node density is low: 1/CA for the case of seven retransmission attempts and 20 n/CA or below for the case of 255 retransmission attempts. Note that the throughput achieved by SMAC decreases very rapidly for higher node densities.

Figure 20 displays results related to average frame delay. We can observe that QUATTRO maintains its delay under 0.23 s, whereas the delay experienced by frames in SMAC reaches 2 and 12 s, respectively, when 7 and 255 retransmission attempts are allowed.

Notice that, as explained in Section 3.4, we designed our protocol in such a way that the maximum delay tolerated by the application being served is equal to twice the duration of a cycle. Therefore, we are implicitly assuming that the maximum tolerated delay is 0.5 s. With the information provided by Figures 19 and 20 we can conclude that QUATTRO succeeds at delivering every frame generated by the sensor nodes within the required timeframe, while SMAC in most cases exceeds the maximum delay or even ultimately fails to deliver some of the frames (or most of them, in some cases) after attempting all of the allowed retransmissions.

Figure 21 depicts the average amount of energy spent per node to transmit data frames during the entire simulation. Notice that values corresponding to QUATTRO are much smaller than those associated to SMAC. The energy consumption in QUATTRO increases linearly with the node density, indicating that the additional traffic generated by the increasing number of nodes is efficiently processed by the network. The energy

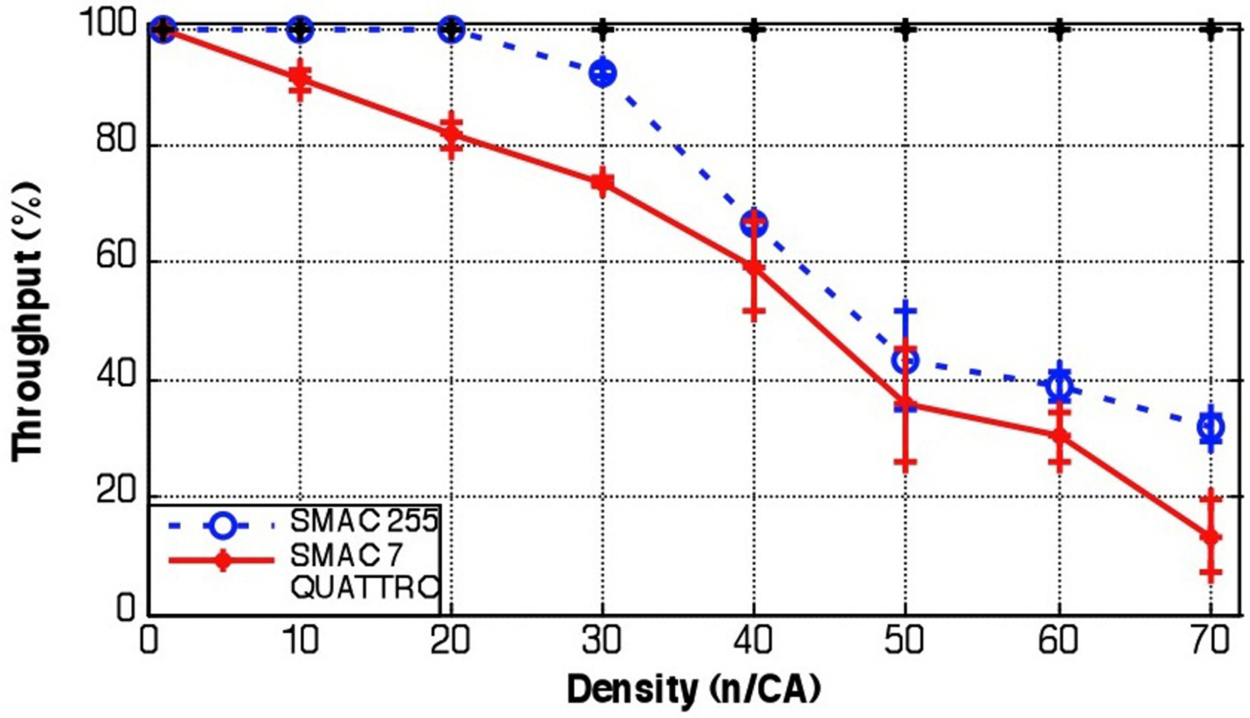


Figure 19 Average throughput, varying number of nodes in fixed area.

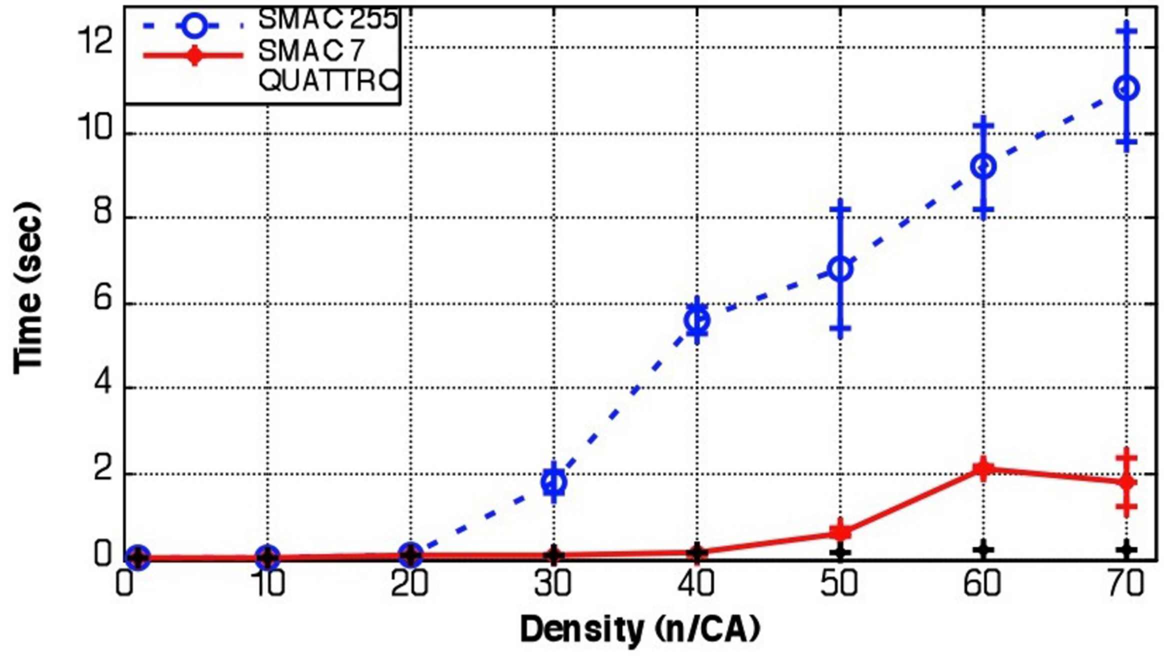


Figure 20 Average frame delay, varying number of nodes in fixed area.

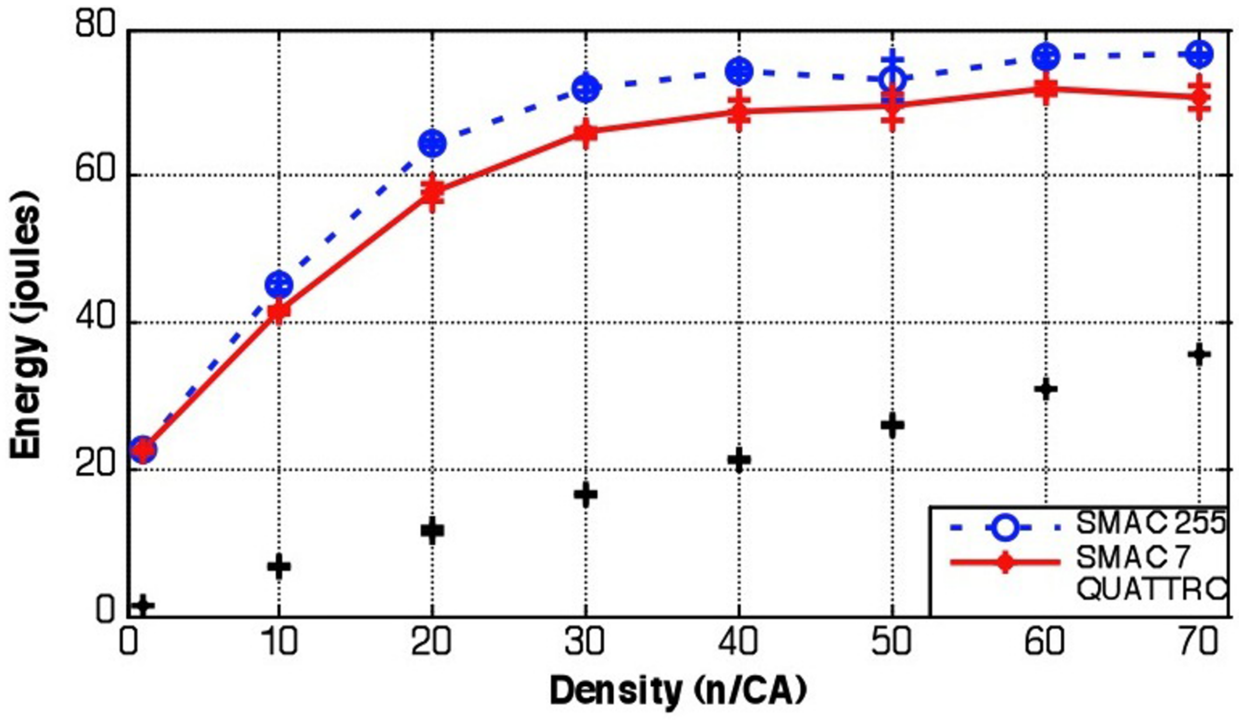


Figure 21 Average energy consumed, varying number of nodes in fixed area.

consumption in SMAC, on the other hand, seems to converge to a constant value. This is an indication that, even if more traffic is generated, the system is unable to process it because of the large number of retransmissions generated, which cause for the network to reach a saturation point.

The last set of results of this section, included in Figure 22, shows the average fraction of time that nodes stay awake. We can see in this figure that in SMAC nodes tend to stay on almost constantly for node densities of 20 n/CA and above. The reason for this behavior is the fact that it takes increasingly longer for nodes to successfully transmit the frames they have accumulated in their queues due to the ever larger number of collisions experienced. In contrast, the fraction of time that nodes stay awake in QUATTRO increases linearly, as needed to transmit the additional traffic, but to values much lower than those observed in SMAC.

From the results presented here we can conclude that our proposed protocol improves transmission efficiency, thus being able to provide QoS guarantees without having to pay a price in terms of increased energy consumption; on the contrary, energy efficiency is greatly improved as well.

5.3. Performance during data transmission when nodes are deployed in a row

We ran the simulations with topologies including from 2 to 18 sensor nodes plus the sink. We assumed again that the traffic generated by each node is CBR at a rate of 4 kbps. In each simulation, nodes are active generating traffic during 60 s. The metrics used to evaluate and compare both protocols here are the same as those described in the previous section.

Figure 23 shows the average throughput for the three methods considered. We can see that all the three approaches achieve a throughput of 100% in all cases. SMAC is able to reach the maximum throughput in this scenario because of the fact that competition is at most between two clusters at a time, which is not as bad as that experienced in scenarios examined in the previous section.

As for average frame delay, it can be seen in Figure 24 that values corresponding to QUATTRO are higher than those belonging to SMAC for scenarios with eight nodes or more. This behavior is due to the fact, mentioned above, that a CH has to wait to receive data from its children before starting its own transmission. Conversely, SMAC nodes start to transmit as soon as they have new data in their queues. Delay could be reduced in QUATTRO if we took advantage of the fact that the sequence of activity windows can restart before the end of the current cycle, as explained in Section 3.4, but it is not done in this case because bandwidth is more than enough to schedule all transmissions without overlaps. The delays experienced in QUATTRO, however, are still far lower than the maximum assumed tolerable delay

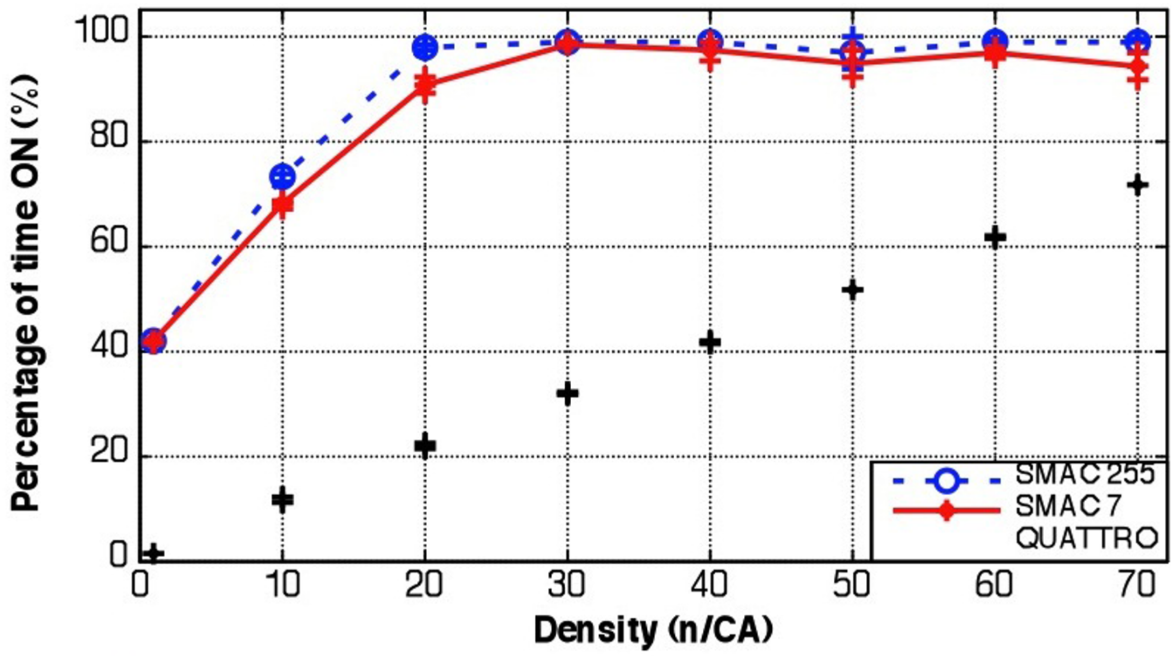


Figure 22 Fraction of time on, varying number of nodes in fixed area.

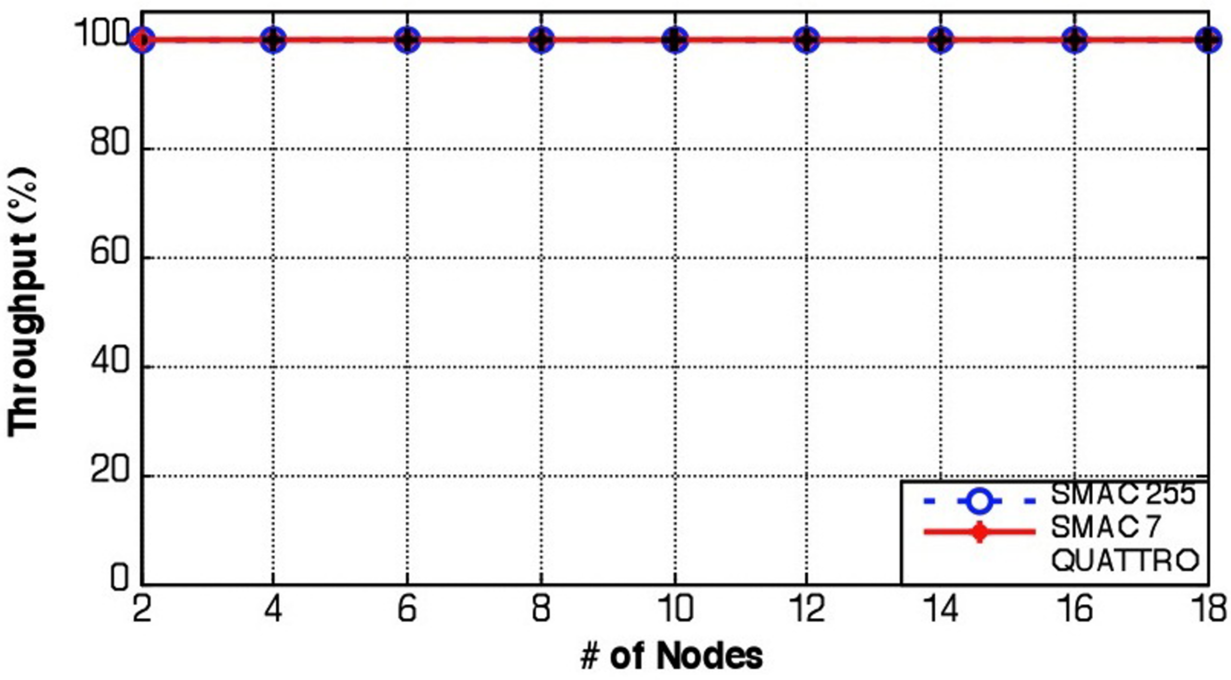


Figure 23 Average throughput, varying number of nodes in a row.

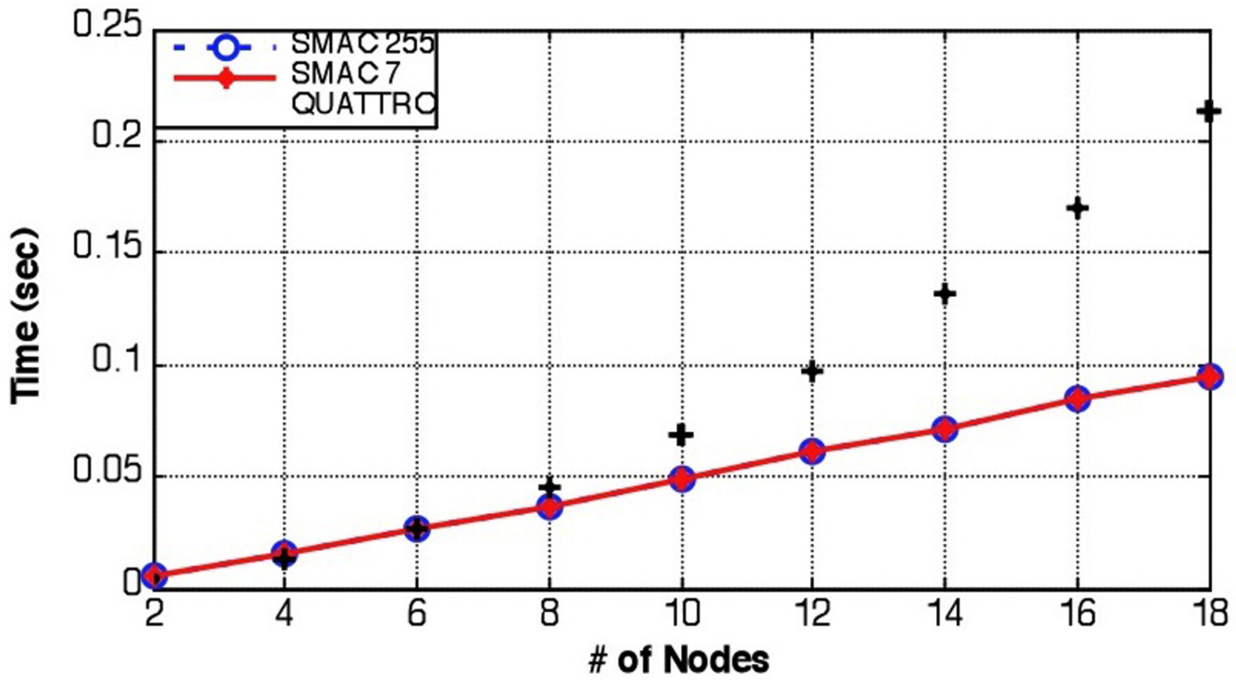


Figure 24 Average frame delay, varying number of nodes in a row.

which, as mentioned in the previous section, is equal to twice a cycle duration, or 0.5 s.

Figure 25 shows results related to energy consumption. We can see again in this scenario that the energy consumed by nodes in QUATTRO is considerably less than that consumed in SMAC. The explanation as to why this happens can be obtained in part from Figure 26, which compares the average fraction of time that nodes remain awake when using the different methods. It is clear from that figure that nodes using SMAC remain awake longer since nodes using QUATTRO have a better way to decide when it is safe to go to sleep without the risk of losing important data. Another reason why QUATTRO uses energy more efficiently, as mentioned above, is the absence of collisions and retransmissions.

5.4. Recovery of setup cost

We consider the energy spent during setup as an investment that will bring about, among others, the benefit of saving energy during the normal operation phase. In this section we analyze how long it takes for the energy invested during setup to start paying off.

The effective amount of energy spent to transfer each data bit from its origin to the sink as a function of time is calculated as follows:

$$E_b(t) = \frac{E_0 + E_{nb} \cdot N_b(t)}{N_b(t)} \tag{7}$$

In the above equation, E_0 is the average energy spent per node during the entire setup period, E_{nb} is the average energy spent to transfer each data bit from its origin to the sink during the normal operation phase of the protocol, and $N_\text{b}(t)$ is the number of data bits that have been received at the sink up to time t. It is clear that, for very large values of t, $E_b(t)$ will approach E_{nb}, meaning that the initial investment E_0 has been diluted to the point that it is insignificant.

For this analysis, we consider the set of simulations described in Section 5.2 for the specific case in which the node density is 70 n/CA.

Figure 27 shows the behavior of $E_b(t)$ and compares it to the energy consumption in SMAC. Notice that we are disregarding the energy spent by SMAC in its own configuration procedure and we are assuming that the average energy consumed by SMAC per data bit transmitted does not vary with time. From our simulations we obtained that $E_0 = 24.1$ J, $E_{nb} = 1.49 \times 10^{-4}$ J/bit and that the average energy spent by SMAC to transfer each data bit from its origin to the sink is $E_{nb,\ \text{SMAC}} = 3.19 \times 10^{-4}$ J/bit. It can be seen from Figure 27 that, after approximately 37.5 s or 150 cycles, the effective amount of energy spent to transfer each data bit from its origin to the sink in QUATTRO starts being less than that spent in SMAC. In other words, in terms of energy, our investment starts paying off 37.5 s after we begin transmitting data. We can see as well that $E_b(t)$ converges rapidly to its asymptotic value En_b; for instance, after 15

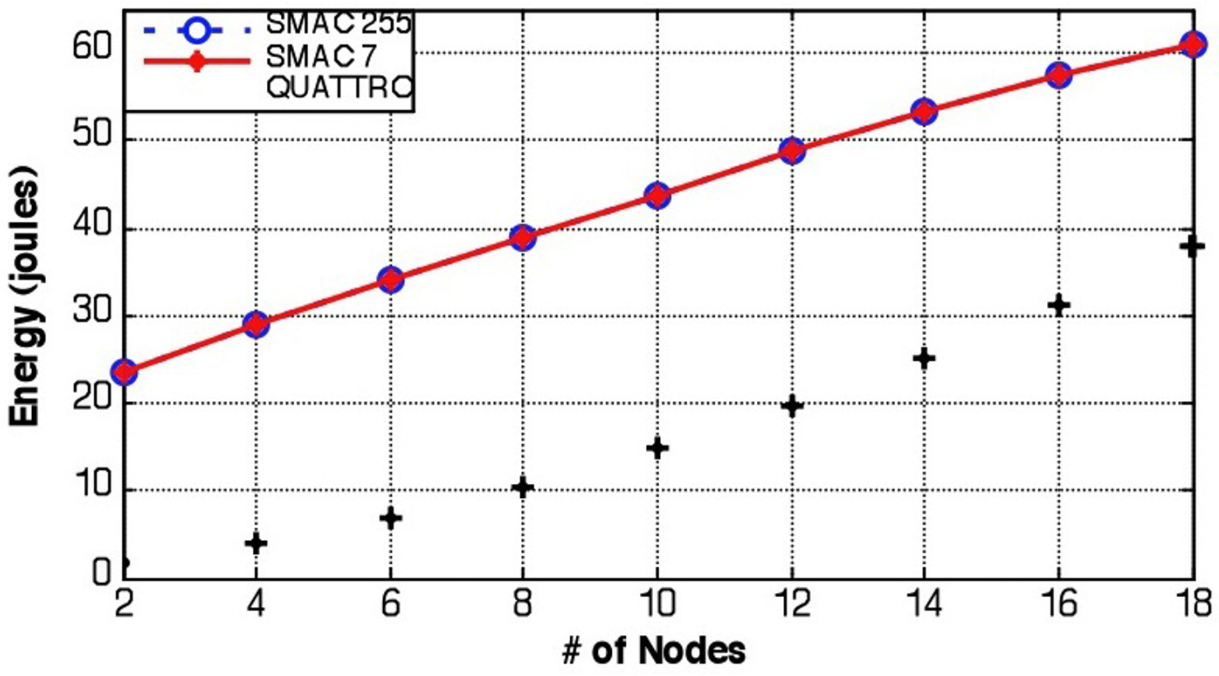


Figure 25 Average energy consumed, varying number of nodes in a row.

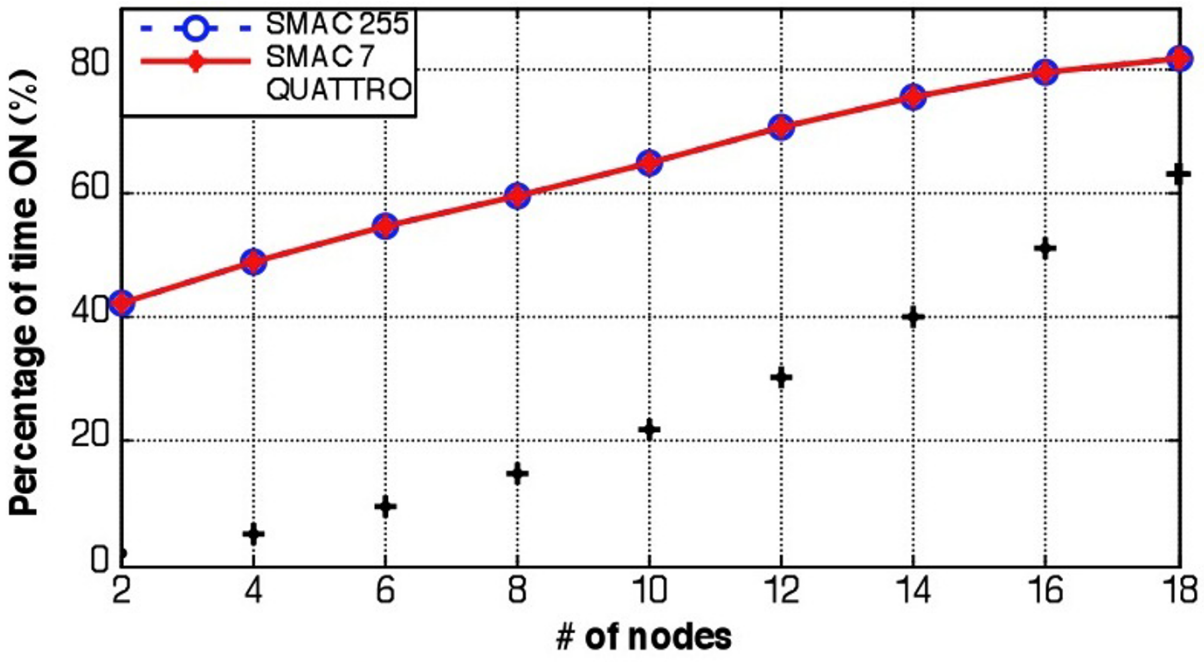


Figure 26 Fraction of time on, varying number of nodes in a row.

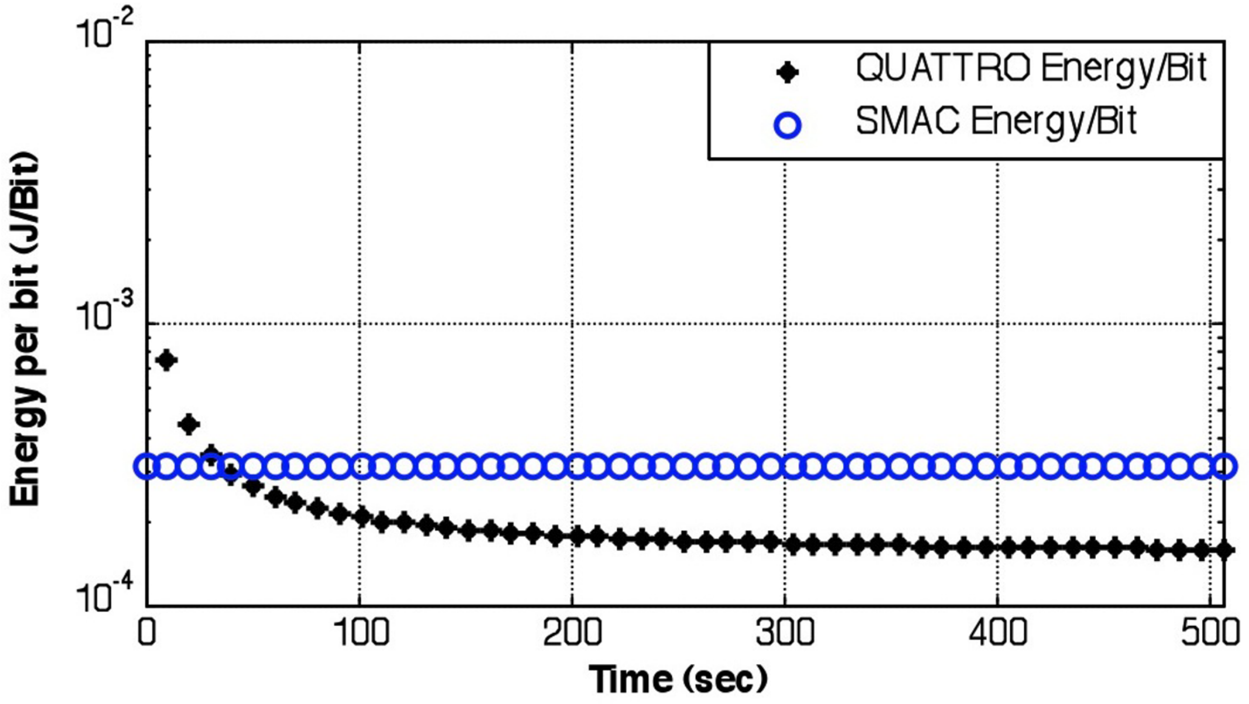


Figure 27 Effective energy spent per data bit, node density is 70 n/CA.

min (not shown in the figure), the difference is less than 5%, and after an hour the difference is approximately 1%.

5.5. Cost of the different reconfiguration options

As mentioned in Section 3.7, when a reconfiguration is needed, nodes can use the time in which the system would otherwise be idle to start exchanging control messages (*normal*) or they can instead stop temporarily the data transfer phase to use the entire time for control messages (*exclusive*). It was also explained that the reconfiguration procedure does not necessarily have to be *complete*, but it can be sped up using the routing information collected previously and by going straight into the reservation phase (*fast*). Given the expected time to carry it out and the amount of data that may be lost if reconfiguration is due to node failures, we analyze the conditions under which it is better to perform a complete or a fast reconfiguration, using only the otherwise unused time or stopping data to run an exclusive reconfiguration.

Figure 28 depicts the time needed to reconfigure the network using the three options:

- Fast reconfiguration using only idle time (labeled *fast*)
- Complete reconfiguration using only idle time (labeled *complete*)
- Complete reconfiguration using the entire time (labeled *exclusive*)

To recover from node failures, assuming that only a few nodes break down simultaneously (e.g. 1 or 2), we can see that it clearly makes sense to run fast reconfigurations when node densities are less than 50 n/CA since it is faster than running an exclusive complete reconfiguration and there is no need to stop the data transfer of nodes that are still connected.

It even makes sense to run a fast reconfiguration for higher node densities if the percentage of nodes that become disconnected is below a certain threshold. The criterion used to decide here is the amount of data that would be lost as a consequence of the time taken to complete the reconfiguration. In other words, if a small number of nodes are disconnected during the relatively long time needed to carry out a normal fast reconfiguration, we might lose less data than if we stop data transmission from all nodes during the shorter period of time needed to run an exclusive complete reconfiguration.

If several nodes (e.g. 3 or more) fail roughly at the same time, it may not be prudent to run a fast reconfiguration since the routing information stored at the sensor nodes may need to be updated to reflect the new topology. If that is the case, a complete reconfiguration is advisable, but the sink still has to decide if stopping data transmission from all nodes during the time needed to run an exclusive reconfiguration would cause more/less data losses than running a normal reconfiguration

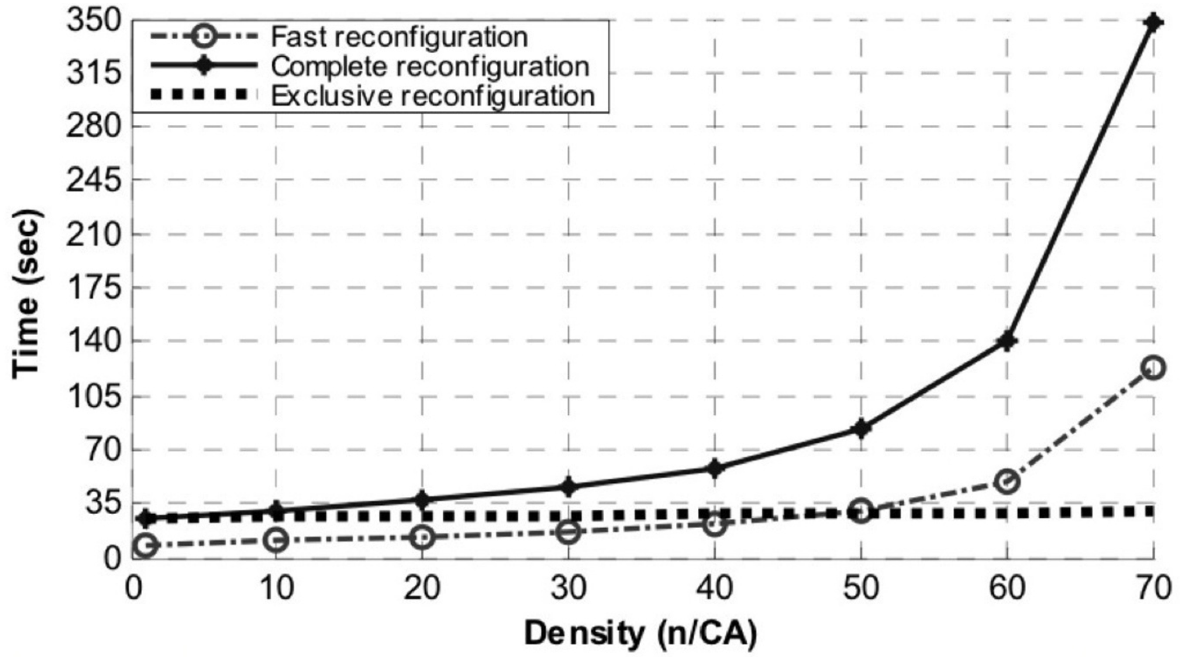


Figure 28 Reconfiguration time, varying number of nodes in fixed area.

losing the data generated only by the disconnected nodes during a longer time.

Based on the previous discussion and on the results shown in Figure 28, Figure 29 shows the thresholds that define when to choose a fast reconfiguration over a complete one given that a few nodes failed at the same time (labeled *fast reconfiguration threshold*) and when to choose a normal complete reconfiguration over an exclusive one (labeled *normal reconfiguration threshold*) given that the simultaneous failure of several nodes has been detected. To illustrate how to interpret the *fast reconfiguration threshold*, let us mention for instance that, for a node density of 50 n/CA, when the percentage of disconnected nodes is 93% or below it is better to opt for a fast reconfiguration instead of a complete one. Similarly, to illustrate how to interpret the *normal reconfiguration threshold*, we can say for instance that, for a node density of 60 n/CA, if the percentage of disconnected nodes is 20% or below it is better to go for a normal reconfiguration instead of an exclusive one.

When the reason for reconfiguration is simply that it is time to run the setup process again to redistribute energy consumption, as opposed to node failures, then it is advisable to select a *complete normal* reconfiguration. The reason to select the *normal* option is that it is better not to stop collecting data, and the reason to choose the *complete* option is the fact that there is no urgency to finish precisely because of the fact that the data generated by the sensor nodes is not being lost.

Figure 30 depicts with a solid line the amount of time QUATTRO takes to achieve reconfiguration (i.e. find routes from every sensing node to the sink and make the corresponding reservations), for different network densities, after 50% of nodes are disabled. Contrary to what was shown in Figure 28, where it was assumed that only one or two nodes fail at the same time, in this figure results correspond to a situation in which many nodes fail simultaneously and in which the execution of an *exclusive* reconfiguration process is required. The reason for that is the fact that the routing information needs to be updated before making reservations; at the same time, when many nodes fail, it is very likely that a large amount of information is being lost, which makes it necessary to use the entire bandwidth for control messages to speed-up the reconfiguration process. As we can see, in all cases recovery takes between 26 and 31 s.

When nodes fail, it is possible that some other nodes will become disconnected from the rest of the network. Figure 30 also shows with a doted line the percentage of nodes that need to be disabled on average for the network to become disconnected, as a function of node density. It is to be expected that networks with lower node densities may become disconnected with fewer disabled nodes since they do not provide as many alternative routes. However, even for the lower densities examined in this study, we can see that when 75% of nodes fail the network will still remain connected with a high probability.

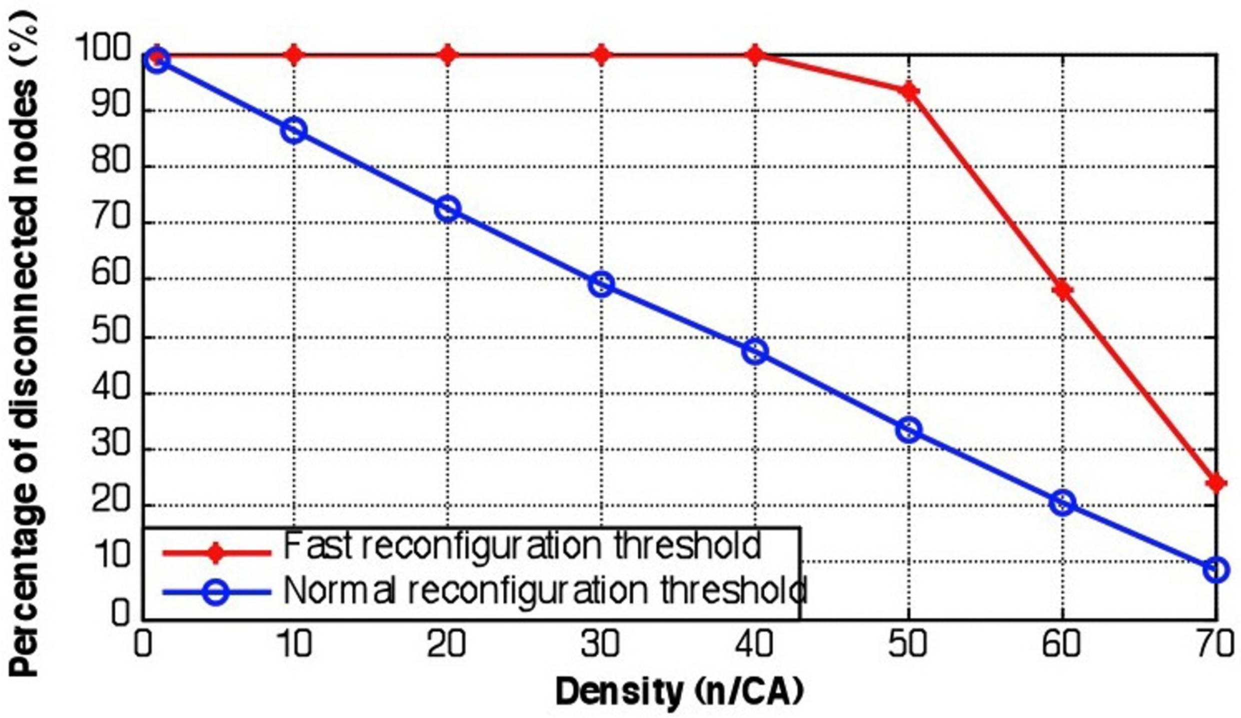


Figure 29 Reconfiguration decision thresholds, varying number of nodes in fixed area.

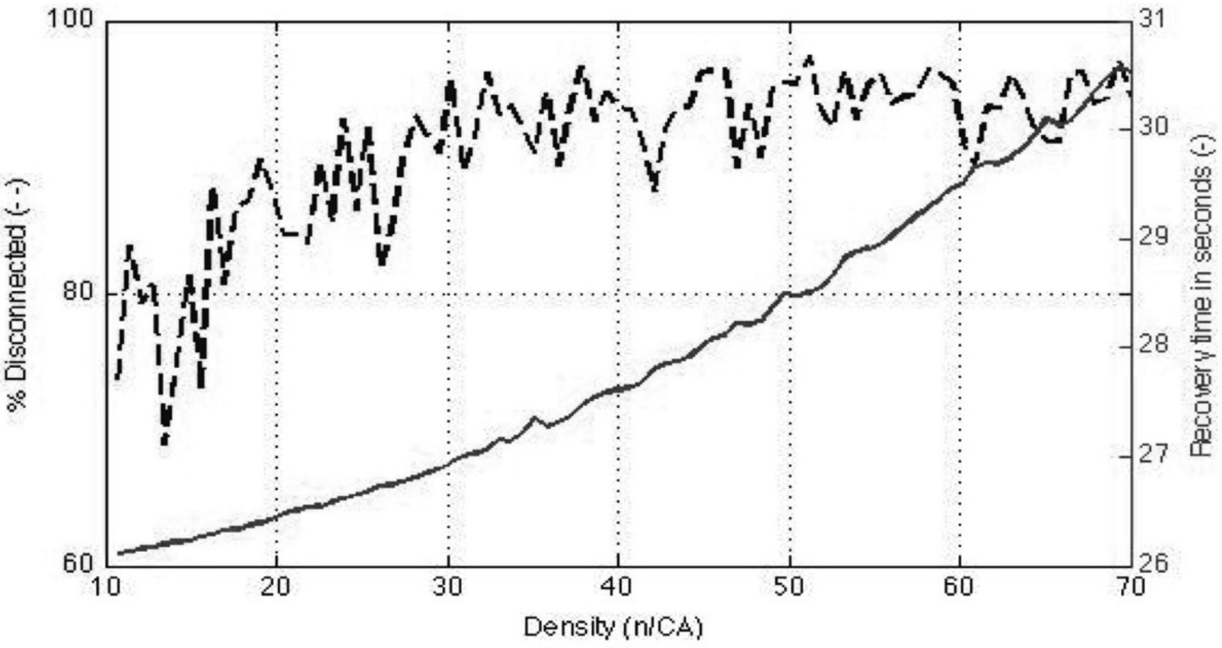


Figure 30 Recovery time and percentage of failures needed to create a disconnected network, as functions of node density.

6. Conclusions

We have proposed an architecture in which the MAC and routing protocols collaborate to discover routes and reserve resources, to organize nodes into clusters and to schedule the access to the transmission medium in a coordinated time-shared fashion. As a consequence, we are able to offer QoS guarantees in addition to achieving great energy savings by eliminating collisions and considerably reducing idle listening. We evaluate our proposal using simulations and our results show that the protocol overhead observed in terms of configuration time, transmitted control messages, and consumed energy is very reasonable. When data transmission begins, we can see that the energy invested during setup starts paying off immediately in terms of QoS and soon after in terms of energy efficiency.

In addition to that, we also evaluate the performance of our protocol QUATTRO during data transmission under different node densities and topologies. We conclude that the QoS guarantees promised during setup in terms of delay and throughput are satisfied in all cases. We compare the performance of QUATTRO to that of SMAC, which is one of the first protocols designed specifically for WSN and which, precisely because of its pioneering role, has become a protocol of reference in sensor networks. Our results show that QUATTRO outperforms SMAC in its ability to provide QoS guarantees and also because of the fact that it uses energy more efficiently.

In addition, we propose a fast reconfiguration alternative that can be used when only a few nodes fail at about the same time. Also, depending on the number of nodes that become disconnected when one or more CHs break down, we offer the option of stopping data transmission during reconfiguration or continuing to gather the data generated by the sensor nodes that remain connected. We analyze these different reconfiguration alternatives and provide guidelines that can be used for the sink to decide when to choose one option over the others.

Overall, we can conclude that QUATTRO possesses all of the characteristics required in a QoS-oriented protocol designed to be used in a WSN environment, in the sense that it is able to satisfy QoS guarantees and achieve great energy efficiency.

Acknowledgements
The authors would also like to thank the anonymous reviewers for their valuable contributions to improve and enrich the presentation of the material included in this article.

Author details
[1]Electronics and Telecommunications Department, CICESE Research Center, Ensenada, BC 22860, Mexico [2]Broadband Wireless and Internetworking Research Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, ON K1N 6N5, Canada

Competing interests
Joel Ruiz would like to thank the Mexican Science and Technology Council (CONACYT), the Center for Scientific Research and Higher Education of Ensenada (CICESE), and the University of Ottawa for their support during the development of this project.

Received: 28 February 2011 Accepted: 23 November 2011
Published: 23 November 2011

References

1. A Fallahi, E Hossain, A dynamic programming approach for QoS-aware power management in wireless video sensor networks. IEEE Trans Veh Technol. **58**(2), 843–854 (2009)
2. B-S Chow, A limited resources-based approach to coding for wireless video sensor networks. IEEE Sens J. **9**(9), 1118–1124 (2009)
3. E Giirses, L Yuan, R Boutaba, Distributed quality-lifetime maximization in wireless video sensor networks, in *IEEE International Conference on Communications (ICC '09),* Dresden, Germany, (14–18 June 2009)
4. A Cano-Tinoco, AO Fapojuwo, Performance analysis of a QoS-enabled routing protocol for wireless sensor networks, in *18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2007),* (3–7 September, 2007)
5. H Su, X Zhang, Battery-aware TDMA scheduling schemes for wireless sensor networks, in *IEEE Global Telecomm Conference (GLOBECOM 2008),* (November 30–December 4, 2008)
6. SC Ergen, P Varaiya, PEDAMACS: power efficient and delay aware medium access protocol for sensor networks. IEEE Trans Mobile Comput. **5**, 920–930 (2006)
7. A Boukerche, RB Araujo, L Villas, A novel QoS based routing protocol for wireless actor and sensor networks, in *IEEE Global Telecomm Conference (GLOBECOM 2007),* (November 26–30, 2007)
8. S Peng, SX Yang, S Gregori, F Tian, An adaptive QoS and energy-aware routing algorithm for wireless sensor networks, in *IEEE International Conference on Information and Automation (ICIA 2008),* (June 20-23, 2008)
9. Y Chen, N Nasser, Enabling QoS multipath routing protocol for wireless sensor networks, in *IEEE International Conference on Communications (ICC 2008),* (19-23 May 2008)
10. E Felemban, C-G Lee, E Ekici, MMSPEED: multipath multi-speed protocol for QoS guarantee of reliability and timeliness in wireless sensor networks. IEEE Trans Mobile Comput. **5**, 738–754 (2006)
11. MY Aalsalem, J Taheri, M Iftikhar, AY Zomaya, Providing QoS guarantees to multiple classes of traffic in wireless sensor networks, in *IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2008),* (March 31-April 4, 2008)
12. E Gelenbe, EC-H Ngai, Adaptive QoS routing for significant events in wireless sensor networks, in *5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2008),* (September 29-October 2, 2008)
13. C Wang, B Li, K Sohraby, M Daneshmand, Y Hu, Upstream congestion control in wireless sensor networks through cross-layer optimization. IEEE J Sel Areas Commun. **25**(4), 786–795 (2007)
14. MO Rahman, MM Monowar, CS Hong, A QoS adaptive congestion control in wireless sensor network, in *10th International Conference on Advanced Communication Technology (ICACT 2008),* (17-20 February 2008)
15. GN Shirazi, P Wang, X Dong, ZA Eu, C-K Tham, A QoS network architecture for multi-hop, multi-sink target tracking WSNs, in *11th IEEE Singapore International Conference on Communication Systems (ICCS 2008),* (19-21 November 2008)
16. Y Dong, D Makrakis, Effective bandwidth calculation for QoS routing in IEEE 802.11 DCF ad hoc networks. Ad Hoc Sensor Wireless Netw. **2**(3), 245–280 (2006)
17. W Ye, J Heidemann, D Estrin, Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Trans Netw. **12**(3), 493–506 (2004). doi:10.1109/TNET.2004.828953
18. T van Dam, K Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks, in *First International Conference on Embedded Networked Sensor Systems (SenSys 2003),* (November 2003)
19. S Du, AK Saha, DB Johnson, RMAC: a routing-enhanced duty-cycle MAC protocol for wireless sensor networks, in *26th Annual IEEE Conference on Computer Communications (INFOCOM 2007),* (May 2007)
20. Y Sun, S Du, O Gurewitz, DB Johnson, DW-MAC: a low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks, in

Ninth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2008), (May 2008)
21. J Polastre, J Hill, D Culler, Versatile low power media access for wireless sensor networks, in *Second International Conference On Embedded Networked Sensor Systems (SenSys 2004),* (November 2004)
22. M Buettner, GV Yee, E Anderson, R Han, X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks, in *4th International Conference on Embedded Networked Sensor Systems,* (2006)
23. A El-Hoiydi, J-D Decotignie, WiseMAC: an ultra low power MAC protocol for multi-hop wireless sensor networks, in *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS 2004),* (Lecture Notes in Computer Science, LNCS 3121), pp. 18–31 (July 2004)
24. J Ruiz, JR Gallardo, L Villasenor-Gonzalez, D Makrakis, HT Mouftah, QUATTRO: QoS-capable cross-layer MAC protocol for wireless sensor networks, in *IEEE Global Communications Conference (GLOBECOM 2009),* Honolulu, HI, (November 30-December 4, 2009)
25. WR Heinzelman, A Chandrakasan, H Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in *Hawaii International Conference on System Sciences,* Maui, Hawaii, (January 4–7 2000)
26. JR Gallardo, A Gonzalez, LA Villasenor-Gonzalez, J Sanchez, Multipath routing using generalized load sharing for wireless sensor networks, in *IASTED International Conferences on Wireless and Optical Communications,* Montreal, Canada, (May 30-June 1 2007)
27. W Lou, An efficient N-to-1 multipath routing protocol in wireless sensor networks, in *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005),* Washington, DC, (November 2005)
28. JR Gallardo, P Medina, W Zhuang, QoS mechanisms for the MAC protocol of IEEE 802.11 WLANs. Wireless Netw. **13**(3), 335–349 (2007). doi:10.1007/s11276-006-7529-7
29. B Krishnamachari, *Networking Wireless Sensors* (Cambridge University Press, Cambridge, 2005)
30. DL Mills, Internet time synchronization: the network time protocol. IEEE Trans. Commun. **39**(10), 1482–1493 (1991)
31. B Sundararaman, U Buy, AD Kshemkalyani, Clock synchronization for wireless sensor networks: a survey. Ad Hoc Netw. **3**, 281–323 (2005). doi:10.1016/j.adhoc.2005.01.002
32. OPNET Technologies, Inc., OPNET Modelerhttp://www.opnet.com/
33. Atheros, Power consumption and energy efficiency comparisons of WLAN Products, White paper http://www.qca.qualcomm.com/media/resource/resource_15_file2.pdf (May 2003)

doi:10.1186/1687-1499-2011-179
Cite this article as: Ruiz *et al.*: Cross-layer medium access control protocol with quality-of-service guarantees for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2011 **2011**:179.