Hindawi Publishing Corporation
Journal of Sensors
Volume 2008, Article ID 728415, 28 pages
doi:10.1155/2008/728415

Research Article

The Synchronized Peer-to-Peer Framework and Distributed Contention-Free Medium Access for Multihop Wireless Sensor Networks

Ahmad Khayyat and Ahmed Safwat

Laboratory for Advanced Wireless Networks, Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada K7L 3N6

Correspondence should be addressed to Ahmed Safwat, ahmed.safwat@queensu.ca

Received 26 September 2007; Revised 17 March 2008; Accepted 4 May 2008

Recommended by Athanasios Vasilakos

IEEE 802.15.4 is a low-power, low-rate MAC/PHY standard that meets most of the stringent requirements of singlehop wireless sensor networks. Sensor networks with nodal populations composed of thousands of devices have been envisioned in conjunction with environmental, vehicular, military applications, and many others. However, such large sensor network deployments necessitate multihop support as well as low power consumption. In the light of the standard's extremely limited joint support of the two aforementioned attributes, this paper presents two essential contributions. First, a framework is proposed to implement a new IEEE 802.15.4 operating mode, namely, the *synchronized peer-to-peer* mode. This mode is designed to enable the standard's low-power features in peer-to-peer multihop-ready topologies. The second contribution is a distributed GTS *(dGTS)* management scheme designed to function in the newly devised network mode. This protocol provides reliable contention-free access in peer-to-peer topologies in a completely distributed manner. Assuming optimal routing, our simulation experiments reveal perfect delivery ratios as long as the traffic load does not reach or surpass its saturation threshold. dGTS sustains at least twice the delivery ratio of contention-based access under suboptimal dynamic routing. Moreover, the dGTS scheme exhibits minimum power consumption by eliminating the retransmissions attributed to contention, which, in turn, reduces the number of transmissions to a minimum.

1. INTRODUCTION

Wireless sensor networks are characterized as low-power, low-cost networks consisting of radio nodes equipped with sensing devices. The low-cost requirement facilitates large deployments, compensating for the extremely limited transmission range attributed to the low transmission power. In the light of the large number of sensor nodes, the cost of replacing their power sources is relatively high. As a result, the low-power requirement became a key design objective [1] and a vehicle for lifetime extension; the lifetime of a sensor network is the time it takes the network to experience a partition [1].

The medium-access control (MAC) problem in wireless sensor networks is more involved compared to other networks. Besides the conventional objectives of MAC protocols, including minimizing collisions and maximizing channel utilization, a wireless sensor network MAC protocol must be power-aware. Sensor nodes have extremely limited power resources compared to their counterparts in other types of networks. Moreover, sensor nodes have limited computation and communication resources, thus requiring a simple yet efficient MAC protocol in terms of both its processing and communication capabilities.

A number of MAC protocols have been proposed by the research and development community for wireless sensor networks and a few bear some similarities to IEEE 802.15.4 [2], the reining standard for the PHY and MAC layers in low-rate wireless personal area networks (WPANs). IEEE 802.15.4 is suitable for sensor networks due to, among other factors, its low-power provisions, reasonable transmission range, and low-rate specifications, which increase the reliability of the RF links and hence reduce the probability of transmission failure.

The IEEE 802.15.4 standard defines two topologies in which a network operates: the star topology and the peer-to-peer topology. In a star network, a device can only communicate with a (single) central controller called the PAN coordinator. The PAN coordinator regulates medium access and facilitates low-power sleep periods primarily by providing synchronization services via periodically transmitting beacons. In a peer-to-peer network, a device can communicate with any other device in its neighborhood, which is referred to as its personal operating space (POS). This prevents any single device from controlling medium access in peer-to-peer networks. Consequently, either all the devices have to keep their receivers enabled constantly or synchronization mechanisms have to be employed. Keeping the receivers enabled constantly as well as using unslotted CSMA-CA is the simplest form of operating peer-to-peer networks. Besides being less energy-efficient, a peer-to-peer network does not support the features available for a star network, such as slotted access, controlled duty cycle, and, more importantly, contention-free access. Providing synchronization in peer-to-peer networks achieves significant energy savings by enabling the nodes to transition to low-power sleep mode periodically, providing slotted medium access (using IEEE 802.15.4 energy-aware slotted CSMA-CA), and supporting contention-free access.

In a wireless sensor network, multihop routing is required to transport a packet to its final destination. To support multihop routing in IEEE 802.15.4 networks, medium access must be regulated beyond the radio sphere of influence of a single node, which is not feasible in the star topology for it is inherent single-hop. While unsynchronized peer-to-peer networks (using unslotted CSMA-CA) are multihop-ready, providing multihop routing support in their synchronized counterparts depends on how synchronization is achieved, and is usually more involved.

This work investigates developing efficient multihop synchronized peer-to-peer IEEE 802.15.4-compatible networks. Such networks extend the physical space covered by the sensor network and its observatory reach. Additionally, we herein assert that such networks are more resource-efficient compared to unsynchronized peer-to-peer networks. Moreover, they provide much-needed mechanisms to accommodate time-sensitive traffic.

2. RELATED WORK

2.1. IEEE 802.15.4

In IEEE 802.15.4 networks, two modes of operation exist, namely, the *beacon-enabled* mode, in which nodes are synchronized and access is slotted, and the *nonbeacon-enabled* mode, in which nodes are not synchronized and access is unslotted. A coordinator is a device that provides synchronization services by transmitting beacons. The IEEE 802.15.4 standard identifies using beacons only as a means for providing synchronization and, hence, refers to synchronized networks as being operated in the beacon-enabled mode. We will use the more generic term *slotted mode* to refer to synchronized networks whether they achieve synchronization via beacons or any other means. On the other hand, we will use the term *unslotted mode* to refer to unsynchronized networks, reflecting the fact that access is unslotted.

The superframe structure and the slotted CSMA-CA contention-access mechanism are two important features of the standard.

2.1.1. The superframe structure

Superframes are employed in the beacon-enabled mode. They are responsible for facilitating the power-saving features of the beacon-enabled mode, slotted contention-based access, and contention-free access. These features as well as the superframe structure are retained in the more generic slotted mode.

As depicted in Figure 1, a superframe starts with the transmission of a beacon and is *superframe duration* (*SD*) symbols long, while the times corresponding to the beginning of two consecutive superframes are *Beacon Interval* (*BI*) symbols apart. The channel symbol rate is 62.5 k symbols/second in the 2.4 GHz band, corresponding to a bit rate of 250 kbps. *SD* and *BI* are specified using the integers *superframe order* (*SO*) and *beacon order* (*BO*), respectively, according to the formulas

$$\begin{aligned} SD &= aBaseSuperframeDuration \times 2^{SO}, \\ BI &= aBaseSuperframeDuration \times 2^{BO}, \\ &0 \le SO \le BO \le 14, \end{aligned} \tag{1}$$

If $SO = BO$, then the node is always active (i.e., 100% duty cycle). If, however, $SO < BO$, then the node is active only for the first *SD* symbols of every *BI*, and is in a low-power sleep mode (i.e., inactive) for the rest of the beacon interval. Note that the inactive period, if present, is at least as long as (or an integral multiple of) the active period (i.e., the superframe duration), according to the relation

$$Inactive\ Period = (2^{BO-SO} - 1)\, SD. \tag{2}$$

The superframe is divided into 16 equal-length time slots. Therefore, *aBaseSuperframeDuration* = *aBaseSlotDuration* ×16, where *aBaseSlotDuration* is equal to 60 symbols. The superframe consists of two periods: the contention access period (CAP) and the contention-free period (CFP). By default, the CAP occupies all the time slots in the superframe and nodes contend for channel access using a variant of slotted CSMA-CA. Furthermore, a node can request the allocation of a guaranteed time slot (GTS), consisting of one or more superframe slots, during which it is granted contention-free access and can transmit (without CSMA-CA) as soon as the GTS commences. GTSs are allocated from the end of the superframe and the time slots used for GTSs constitute the CFP. The PAN coordinator manages GTS allocation and deallocation.

2.1.2. The CSMA-CA algorithm

IEEE 802.15.4 defines two low-power variants of the CSMA-CA random-access mechanism, one for slotted access and

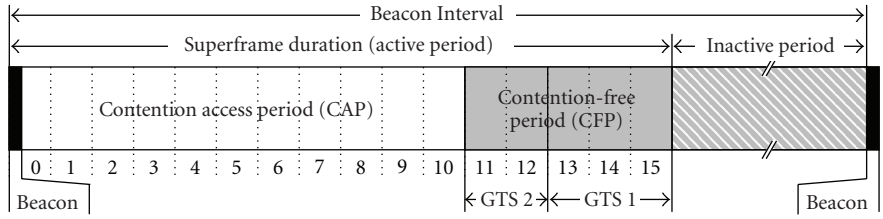

Figure 1: The superframe structure.

the other for unslotted access. They are used for transmitting data and MAC command frames during the CAP. ACK frames are transmitted immediately after receiving a frame that requires an acknowledgment, without using CSMA-CA. Similarly, in the beacon-enabled mode, beacon frames are transmitted without using CSMA-CA.

The IEEE 802.15.4 CSMA-CA algorithms achieve energy savings by keeping the node idle during the backoff procedure. Instead of continuously probing the medium, a node only performs the clear channel assessment (CCA) procedure at the end of the backoff duration. A backoff period is the time unit used by the CSMA-CA algorithms, and is equal to 20 symbols. Consequently, there are 3×2^{SO} backoff periods in each superframe slot. Backoff period boundaries must be aligned with the superframe slot boundaries. The CSMA-CA algorithms are, otherwise, very similar to the basic CSMA-CA schemes used elsewhere: a random backoff timer is started at the beginning of the backoff procedure and transmission is attempted when the timer expires.

There are differences between the IEEE 802.15.4 CSMA-CA algorithms used in the slotted and the unslotted modes. First, in the slotted mode, transmission starts at a backoff period boundary and so does the CCA procedure. Secondly, the CCA procedure must succeed two consecutive times, or else a new backoff procedure is initiated (unless the maximum backoff attempts are exhausted). In the unslotted mode, however, a single successful CCA is required to attempt transmission.

2.2. Synchronization in peer-to-peer networks

To provide synchronization in peer-to-peer networks, an extension to the standard was developed in [3, 4] to utilize beacon frames (the synchronization mechanism used in star networks) in peer-to-peer networks. While there is a single coordinator in a star network responsible for transmitting beacon frames, potentially many exist in a peer-to-peer network. A beacon-enabled peer-to-peer network can be implemented by allowing devices associated with the PAN coordinator to act as coordinators and transmit their own beacon frames to enable other devices located outside the wireless range of the PAN coordinator to join the network. This mechanism facilitates quasidistributed medium-access control among coordinator devices and is hierarchically scalable, but it introduces a few complications. Problems primarily arise from the fact that a node's operation is not governed by a single beacon. Mostly, a node must respect two beacons, and, thus, two superframe structures, simultaneously: the one it receives from the coordinator with which it is associated, defining its parent's schedule, and the one it transmits to nodes associated with it, defining its own schedule. As a result, the node has to ensure that it is idle in both schedules when it is time to transmit a beacon in one or receive a beacon in the other. In addition, superframe slot boundaries and backoff period boundaries are unaligned in both superframes, reducing the efficiency of slotted access. Moreover, contention-free access can cause a reduction in utilization since each schedule must be idle during a GTS in the other. Other neighboring coordinators will transmit beacons that are different from both beacons (and will interfere with possibly the two corresponding schedules). For example, in a multihop simulation scenario consisting of 21 nodes distributed in a grid, more than three quarters of the MAC commands and data generated by the simulation scenario are dropped as a result of receiving irrelevant beacon frames.

Another approach to realize beacon-enabled peer-to-peer networks is proposed in [5]. In this approach, beaconing coordinators are allowed to adopt different BIs. To ensure that beacons from coordinators within each other's radio range do not collide, they transmit their beacons sequentially within a common beacon transmission period, T_{Bf}. T_{Bf} is large enough to contain a number of subslots during which beacons can be transmitted. Subslots within T_{Bf} are assigned to beaconing coordinators such that neighboring coordinators transmit their beacons in different subslots. This approach resolves the issue of collisions between beacons. However, a coordinator A with shorter BI transmits beacons more frequently, and, hence, will transmit a few beacons while a neighboring coordinator B with larger BI is in the middle of its superframe. This takes place unless $SD_B \leq BI_A$ (or, more generally, $\max(SD) < \min(BI)$ for every pair of beaconing coordinators that are within radio range of each other), which is a tight restriction. Therefore, beacon data collisions are still a potential problem. Scalability is also an issue. The minimum number of beacon transmissions that fit within one T_{Bf} is dictated by the beaconing coordinator density. Therefore, in a network densely populated with beaconing coordinators in a given neighborhood, T_{Bf} must grow large enough, negatively impacting the network efficiency. Furthermore, allowing different coordinators to adopt different BIs prevents the use of CFPs because a GTS allocated at one coordinator may correspond to a portion of the CAP at a neighboring coordinator.

The ZigBee specifications [6] supplement the IEEE 802.15.4 MAC/PHY with a networking stack, providing yet another partial solution to the problem. The networking stack supports three topologies, namely, the star, tree, and mesh topologies, in line with the IEEE 802.15.4 topologies. The tree topology is a special case of the peer-to-peer topology. In tree networks, routers (i.e., the ZigBee equivalent of IEEE 802.15.4 coordinator devices) move data and control messages through the network using a hierarchical routing strategy. In this hierarchical routing strategy, the parent-child links are used to route packets along the tree; if the destination is a descendant of the device, the device routes the packet to the appropriate child, whereas if it is not a descendant, the device routes the packet to its parent. Tree networks may operate in beacon-enabled mode, but mesh networks may not. According to the ZigBee specifications, beacon scheduling is employed when implementing a beacon-enabled tree topology to prevent the beacon frames of one device from colliding with either the beacon frames or data transmissions of its neighboring devices. To facilitate this, the beacon order is made much larger than the superframe order giving the routing nodes the opportunity to sleep and conserve power. Since the inactive period is an integral multiple of the active period, the beacon interval is effectively divided into BI/SD non-overlapping time slots, each of which can be chosen by a device as its superframe upon joining the network. A beaconing device uses its own superframe for communicating with its children, and uses its parent's superframe to communicate with its parent. Similarly, a device's children use their own superframes to communicate with their respective children. To avoid keeping a device receiving for more than one superframe in the beacon interval, the parent uses the indirect transmission mechanism of the IEEE 802.15.4 MAC to transmit to its children, while the children transmit to their parent directly during the parent's CAP. In an indirect transmission, the parent indicates in its beacon all pending packets and their respective recipients, and each recipient is responsible for extracting its packet from its parent during the parent's superframe. Every beaconing device transmits its beacon at the beginning of its superframe. The beacon transmission time, which is equivalent to the superframe start time, is determined relative to the beacon transmission time of its parent. This time offset is included in the beacon payload of every device in a multihop beaconing network. Therefore, a device receiving a beacon frame can find out the beacon transmission time of both the neighbor transmitting the beacon and the neighbor's parent. This allows the nodes to avoid interfering with a neighbor, whether a parent or a child.

The previous model exhibits a few limitations. As the density of the beaconing coordinators in a given neighborhood increases, the duty cycle decreases. Additionally, neighbors that are not involved in a parent-child relationship, such as sibling nodes, whether coordinators or not, cannot communicate directly. Instead, they have to go through their closest common ancestor. Finally, there is no accommodation for contention-free access (using GTSs).

3. SYNCHRONIZED PEER-TO-PEER IEEE 802.15.4 FRAMEWORK

In this section, a framework is proposed to enable peer-to-peer IEEE 802.15.4 networks to have access to features that are originally feasible only in star networks. The first objective of the framework is to provide synchronization services suitable for the peer-to-peer topology. This enables the framework to achieve its ultimate objective of providing contention-free access in multihop peer-to-peer networks. Contention-free access is provided via a distributed, reliable, and efficient GTS management protocol, presented in Section 4.

In order to support contention-free access in multihop networks, GTS management has to be localized such that a slot allocated for GTS transmission in one neighborhood is still available in other neighborhoods, provided that there are no common nodes. By neighborhood here, we are referring to the set of nodes within the radio range of the nodes involved in the GTS transmission. According to the standard, a GTS is allocated at a coordinator for any of its children and the GTS transmission takes place to/from the coordinator, resulting in no direct GTS transmission between sibling nodes. This enlarges the neighborhood affected by the GTS transmission beyond the one that would be affected if only the GTS source and destination were involved in the transmission, that is, if GTS transmission is direct and does not go through the nodes' parent coordinator. Therefore, the proposed framework further localizes GTS management by involving only the source and the destination of a GTS in its allocation/deallocation (i.e., GTS management) procedures, enabling further slot reuse whether in GTS transmission or via contention-based access. Furthermore, the standard's GTS specifications are relevant to the star topology in particular and do not address the peer-to-peer topology. For example, it requires that GTSs be managed by the PAN coordinator only, which is appropriate for star networks, but dictates centralized GTS management, making it less practical for multihop peer-to-peer networks.

Our desired behavior mandates that GTS management be completely distributed, and special roles in GTS management be eliminated such that every node is capable of managing its own GTSs exclusively along with its partner. Thus, we introduce the *dGTS management protocol* to provide contention-free access in synchronized peer-to-peer IEEE 802.15.4 networks.

Since GTSs are allocated superframe slots, slots must be identical in nodes wishing to carry out a GTS transmission, with the GTS starting and ending at exactly the same times at both nodes. This implies having identical superframes at both nodes. Hence, they have to share the same BO (to force their superframes to start at the same time) and the same SO (to force their superframes as well as their slots to last for the same duration). However, distributed GTS management implies a flat peer-to-peer topology since any two neighboring nodes may conduct a GTS transmission and for which the intermediary role of a coordinator is no longer needed. As a result, BO and SO have to be common throughout the network. We consider this requirement as

part of the synchronization requirement, namely, to have a common superframe structure throughout the network.

The proposed framework consists of two components: the synchronization component, whose objective is to employ a single common superframe structure in a distributed manner, and the dGTS protocol component, whose objective is to manage GTSs in a distributed manner as well.

Synchronization is required to align the superframes in all nodes across the network. It basically serves the standard's beacon frames purpose of announcing *BO* and *SO*, and unifies the clock among nodes. The challenge is to make all nodes simultaneously recognize global events, including the beginning and end of superframes. For the purposes of our framework, synchronization has to be provided by means of a distributed mechanism.

There are several approaches to achieve global synchronization [7–13]. A popular classification distinguishes absolute synchronization from relative synchronization. Absolute synchronization refers to synchronization mechanisms where nodes end-up having their local clocks reflecting the same numerical clock value at any given moment. For example, a packet transmitted at the boundary of a slot at the source is received after a propagation delay from that boundary at the destination according to the destination's clock, which matches the source's clock. On the other hand, relative synchronization refers to synchronization mechanisms where nodes end-up having their local clocks reflecting the same numerical clock value at any given event as it is observed at each node. For instance, a packet transmitted on the boundary of a slot at the source is received on the slot boundary at the destination according to the destination's clock. In the latter form of synchronization, the source and the destination clocks are not synchronized to a global clock, but rather to events. Synchronization protocols based on single messages often provide relative synchronization as a single message does not reveal the propagation delay. Some single-message protocols, however, can achieve better accuracy by correlating multiple synchronization messages from multiple nodes. Although absolute synchronization seems more natural, relative synchronization has the advantage of better slot time utilization, since the propagation delay does not consume any portion of the slot. Instead, the propagation delay effect is implicitly neutralized and the slot starts just when the transmission arrives. However, multihop networks impose additional challenges that render using relative synchronization nontrivial. For example, if a node receives a GTS transmission from a node other than that with which it last synchronized, the packet might arrive slightly before the GTS begins at the receiver and, hence, gets dropped.

4. dGTS PROTOCOL

The major contribution of this work is the IEEE 802.15.4 MAC extension it defines to provide support for distributed contention-free access (via GTSs) in multihop peer-to-peer networks. It requires the network to be synchronized independently and to operate with common *BO* and *SO* parameters. We assume perfect synchronization by means of an independent synchronization protocol that uses the CAP for message exchange.

Table 1: Revised IEEE 802.15.4 modes.

Topology	Slotted mode	Unslotted mode
Star	Beacon-enabled	Nonbeacon-enabled
ZigBee tree	Beacon-enabled	Nonbeacon-enabled
Peer-to-peer	Synchronized peer-to-peer	Nonbeacon-enabled

The protocol requires devising a new network mode. The new mode provides slotted access without using the standard's beacon mechanism. Additionally, unlike the standard's peer-to-peer mode, the network must be synchronized. Therefore, to implement the dGTS protocol in a way compatible with the standard, we introduce a new mode, namely, the *synchronized peer-to-peer* mode, which is a variant of the generic slotted mode. Procedures and commands defined by the dGTS protocol are not accessed unless the node is operating in this mode, and received dGTS-related MAC commands are dropped at the MAC sublayer if the receiver is not operating in this mode. Having a flat peer-to-peer topology dictates that commands related to association, disassociation, and so forth are irrelevant when operating in this mode. Table 1 presents the revised network modes. The generic slotted mode indicates the use of slotted CSMA-CA for contention access during the CAP, while the unslotted mode indicates the use of unslotted CSMA-CA. The slotted mode decouples the fact that access is slotted from the means by which synchronization is implemented. The beacon-enabled and the synchronized peer-to-peer submodes specify that missing detail, where the beacon-enabled mode uses beacons as specified in the standard, and the synchronized peer-to-peer mode uses an independent synchronization protocol. Although the ZigBee tree topology is a special case of the generic peer-to-peer topology, the fact that it supports the standard's beacon-enabled mode promotes it to be considered as an individual topology. Nevertheless, it has some restrictions which makes the generic peer-to-peer topology still attractive for some applications. The ZigBee tree performs routing only along parent-child links, and requires a low duty cycle, such that, in any given neighborhood, all the coordinators' superframes fit in a single beacon interval without overlapping. Hence, the dGTS protocol is advantageous as it enables all the standard's features on virtually any topology.

In order to properly support the proposed protocol, some structures describing the GTS had to be modified. We refer to GTSs and modified GTS-related structures, while the dGTS protocol is active, by *dGTS* rather than *GTS* (e.g., *dGTS characteristics*) to distinguish them from those in the standard.

4.1. General description

The dGTS protocol allows any node to initiate a dGTS transaction (i.e., dGTS allocation or deallocation) at any time during the CAP. Initially, the CAP occupies the whole superframe, and shrinks as dGTSs are allocated. The CAP

ends as the first dGTS in the superframe starts. A dGTS is allocated at two nodes: the source, which initiates the transaction and will be transmitting data during the dGTS after it is allocated, and the destination. The source and the destination of a dGTS are partners in the context of that dGTS. To reliably allow any node to manage its own dGTSs, all of its transactions have to be announced to its neighbors to prevent them from interfering with its dGTSs. Therefore, all dGTS commands are transmitted as broadcast messages. Broadcasting the dGTS commands provides the neighbors with the information they need to avoid interference and resolves the hidden terminal problem. However, a dGTS command does have a single intended destination that cannot be addressed in the MAC header anymore. To work around this, we introduce a destination field in the payload of all dGTS commands. This means that every node will have to process the payload to find out whether the command is destined to it. This is not an additional overhead since it needs to process the payload anyway to monitor its neighbors' transactions.

The hidden terminal problem affecting the standard's CSMA-CA algorithms is a potential problem with no effective solution. Solutions involving request-to-send (RTS) and clear-to-send (CTS) messages used in the IEEE 802.11 DCF are not appropriate for small packets. Given the small frame size limit imposed by the IEEE 802.15.4 standard, they will provide no meaningful protection mechanism and will rather increase the overhead. However, GTSs are intended for long-term regular traffic, making the use of the RTS-CTS or similar protection mechanisms useful. This is achieved by forwarding dGTS allocation, deallocation, and related dGTS commands when they are received so that neighbors of both parties are aware of the transaction.

The dGTS protocol commands are always transmitted during the CAP per the standard's slotted CSMA-CA access rules. Furthermore, the dGTS protocol is fully acknowledged. Every command intended for a particular destination, and not for the sole purpose of broadcasting, has to be acknowledged by an ACK frame as specified in the standard. This is important to ensure reliability and to recognize topology changes without long waiting times since the timeout duration for ACK frames is relatively short.

An allocated dGTS is periodically checked for activity by its source and destination. If a dGTS is found idle long enough, it is automatically deallocated using the deallocation procedure outlined in Section 4.5.4. To identify idle dGTSs, the same rules defined in the original standard recognize expired GTSs are followed. The only difference is that each node checks its own dGTSs rather than having the coordinator check its children's GTSs. However, using the same rules to identify idle dGTSs at both the source and the destination could cause neither to respond to the other's deallocation command, despite the CSMA-CA randomization, for each would be busy transmitting its command. Figure 2(a) illustrates this potential problem. This could lead to unnecessary retransmissions and ultimately a failed transaction. To avoid this situation, the expiration rules are slightly modified for the destination so that it takes the destination slightly longer to declare the expiration of a dGTS. This gives the source the priority to start the deallocation process as shown in Figure 2(b), but still allows the destination to perform it in case the source is no longer around or its command was never successfully received at the destination.

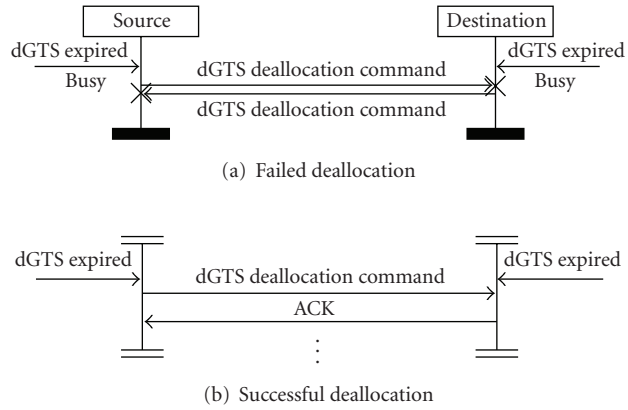

Figure 2: Idle dGTS deallocation at the source and the destination.

During the CAP, nodes keep their transceivers enabled, in the receive mode; the *macRxOnWhenIdle* MAC attribute is set to TRUE. Since a transmission should not be started unless its transaction, including completing the transmission, receiving the corresponding ACK frame (if required), and appropriate interframe spacing (IFS), fits into the remaining portion of the CAP, the transceiver should be idle by the end of the CAP. However, it is still possible for a neighbor to be receiving at the end of the CAP because, essentially, the CAP length is not necessarily the same for all nodes, as illustrated in Figure 5 (see Section 4.4.5 for an explanation). Therefore, a node with a longer CAP may start transmitting to a neighbor with a shorter CAP just before the neighbor's CAP ends, causing its transceiver not to be idle at the end of its CAP. Moreover, a node may be transmitting at the end of the CAP if it had to acknowledge a transmission from a neighbor with a longer CAP. As a result, to protect the integrity of dGTSs, the transceiver is forcibly disabled at the CAP end boundary, interrupting any ongoing communication, except for ACK frames already being transmitted. Acknowledgment frames in transmission avoid retransmitting correctly received frames, and have relatively small size anyways. During the CFP, the transceiver remains disabled except during dGTSs for which the node is either the source or the destination, in which case the transceiver is switched to the appropriate mode for the duration of the dGTS only, and then disabled again at the end of the dGTS. While in a dGTS, a node drops all frames received from all nodes but its dGTS partner. The transceiver is enabled again at the beginning of the CAP.

Throughout the rest of this paper, *dGTS source* refers to the node initiating the dGTS allocation procedure, which is also the node that will transmit data in that dGTS after it is allocated. Similarly, *dGTS destination* refers to the node that receives data during the dGTS after it is allocated.

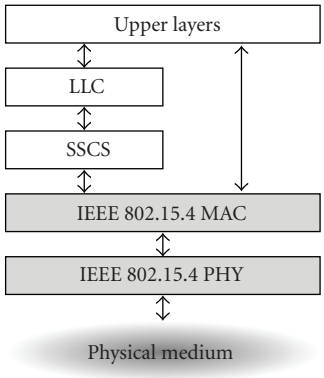

Figure 3: IEEE 802.15.4 node architecture.

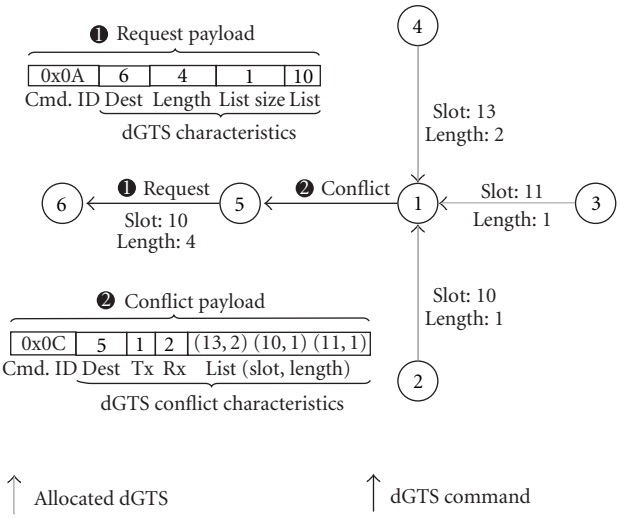

Figure 4: dGTS characteristics and dGTS conflict characteristics example.

4.2. Primitives

The IEEE 802.15.4 MAC protocol defines special primitives for higher layers to utilize the features it offers. Since these primitives are not standard 802.2 link layer control (LLC) primitives, other means to access them are needed, hence the need for the service specific convergence sublayer (SSCS). As shown in Figure 3, IEEE 802.15.4 MAC services can be accessed by IEEE 802.15.4-aware applications, or through the standard LLC using the SSCS sublayer. The SSCS sublayer provides higher layers with the primitives required to access the special features of the IEEE 802.15.4 MAC.

The dGTS protocol specifies the following additional service primitives defining the dGTS-related interface between the MAC sublayer and higher layers (referred to hereafter as SSCS).

4.2.1. MLME-dGTS.request

This primitive allows a device to send a request to a neighbor to allocate a new (or deallocate an existing) dGTS for which this node is the source and the neighbor is the destination. It has one parameter specifying the requested dGTS characteristics (see Section 4.4.1). When this primitive is received, the MAC sublayer constructs and transmits a dGTS request MAC command to the designated destination (see Sections 4.3.1 and 4.5.1).

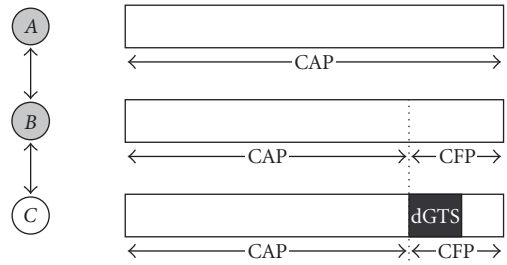

Figure 5: Example of neighbors with different CAP lengths. *A* and *B* are neighbors. *B* and *C* are neighbors. *A* and *C* are *not* neighbors.

4.2.2. MLME-dGTS.confirm

This primitive reports the results of a request primitive to allocate a new (or deallocate an existing) dGTS to the SSCS sublayer. It passes two parameters: the status of the request (dGTS granted, dGTS rejected, channel access failure, etc.) and the dGTS characteristics (only if the request was granted).

4.2.3. MLME-dGTS.indication

This primitive indicates to the SSCS sublayer that a dGTS request MAC command has been received at the MAC sublayer. It has a single parameter specifying the dGTS characteristics of the received request. Upon receiving this primitive, the SSCS sublayer decides how to respond to the request (possibly by consulting higher layers) and then conveys its decision to the MAC sublayer via the response primitive.

4.2.4. MLME-dGTS.response

This primitive is used by the SSCS sublayer to initiate a response to an indication primitive. It has one parameter reflecting the SSCS sublayer's decision about the dGTS request command in the form of a dGTS characteristics structure. When this primitive is received at the MAC sublayer, it constructs and transmits a dGTS response MAC command to the designated destination (see Sections 4.3.2 and 4.5.2).

Figure 6 shows how these four protocol primitives are used together with some commands to allocate a dGTS.

4.3. Commands

MAC commands of the dGTS protocol are IEEE 802.15.4 MAC commands. The destination address field is a common payload field in all dGTS commands since the MAC header

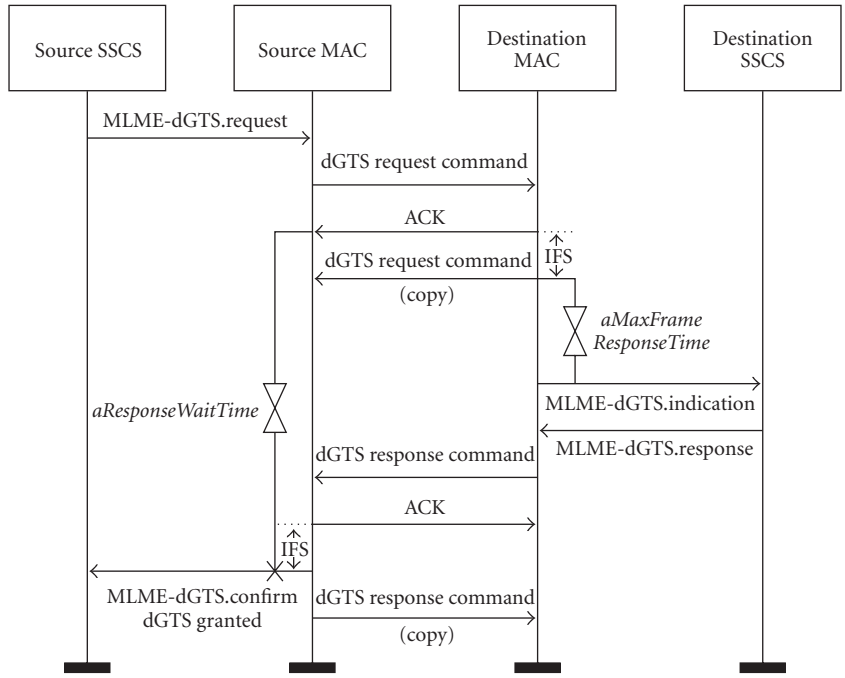

Figure 6: Successful dGTS allocation message sequence diagram.

destination address field is set to the broadcast address. Due to the lack of association, the dGTS protocol assumes a flat topology in which there is no notion of local domains or local address spaces, that is, no short addresses are assumed. Therefore, the payload destination address field is a 64-bit extended address. An ACK frame is expected from the node whose address is in the payload destination field. ACK frames are not used in association with commands intended for broadcast only and the destination field holds the source's address.

All of the dGTS protocol's MAC commands are transmitted using slotted CSMA-CA during the CAP. dGTS MAC commands are identified by their command frame identifiers (command ID). The original standard defines commands with IDs from 0×01 to 0×09. The following MAC commands, whose formats are also shown in Table 2, are used by the dGTS protocol.

4.3.1. dGTS request

This command initiates a dGTS transaction: dGTS allocation or deallocation. It holds the dGTS characteristics (see Section 4.4.1) of the requested dGTS. Upon receiving this command, the destination forwards a broadcast copy of the command to notify its neighbors about the slots being considered for allocation/deallocation to avoid potential multiple allocations. See Section 4.5.2 for details.

This command includes five fields in its payload, as shown in Table 2(a). They are (1) *command ID* (8 bits): the command frame identifier equal to 0x0A by which this command is recognized; (2) *destination address* (64 bits); (3) *dGTS length* (4 bits): the number of superframe slots the dGTS comprises; (4) *list size* (4 bits): the number of alternative superframe slots the dGTS may start at; and (5) *starting slot list* (*list size* $\times$ 4 bits): a list of alternative superframe slot indexes, the cardinality of which is equal to (4) above. The indexes mark the slots at which the dGTS may start. The command provides a list of starting slots instead of a single one to allow the destination of some choice in case some slots are unavailable at the destination (see Figure 8).

If the command is a deallocation request (Table 2(b)), then the *list size* field is set to zero and there is a single starting slot in the *starting slot list* field indicating the starting slot of the dGTS to be deallocated. The deallocation request also has a flag indicating whether nodes other than the intended destination should consider this request or not. This flag is always active (i.e., all nodes should consider the request), except when a pending dGTS allocation request is aborted. In that case, a deallocation request indicates the abortion of the ongoing dGTS request rather than an actual deallocation of an allocated dGTS (see Section 4.5.3 and Figure 9(a)). Since both the source and the destination of a dGTS can transmit a deallocation request command, the deallocation request has another flag indicating whether the node transmitting the command used to transmit or to receive data during that dGTS. The remaining two bits of the flags field are reserved for future use.

4.3.2. dGTS response

This command is used by the destination to reply to a previously received dGTS request command requesting dGTS allocation. If the destination has granted the request, this command holds the dGTS characteristics of the allocated dGTS at the destination. If the destination has rejected the request, this command holds dGTS characteristics structure indicating that no dGTS has been allocated.

Table 2: Payload format of dGTS commands.

(a) dGTS request command payload: allocation

Command ID	Destination address	dGTS length	List size	Starting slot list
8 bits	64 bits	4 bits	4 bits	List Size × 4 bits
0x0A		1–15	1–15	1–15 each

(b) dGTS request command payload: deallocation

Command ID	Destination address	dGTS length	List size	Flags	Starting slot
8 bits	64 bits	4 bits	4 bits	4 bits	4 bits
0x0A		1–15	0	0–3	1–15

(c) dGTS response command payload

Command ID	Destination address	dGTS length	List size	Starting slot
8 bits	64 bits	4 bits	4 bits	4 bits
0x0B		1–15	0/1	1–15

(d) dGTS conflict command payload

Command ID	Destination address	Tx Count	Rx Count	dGTS list
8 bits	64 bits	4 bits	4 bits	(Tx + Rx) × (4 + 4) bits
0x0C		0–15	0–15	1–15, 1–15

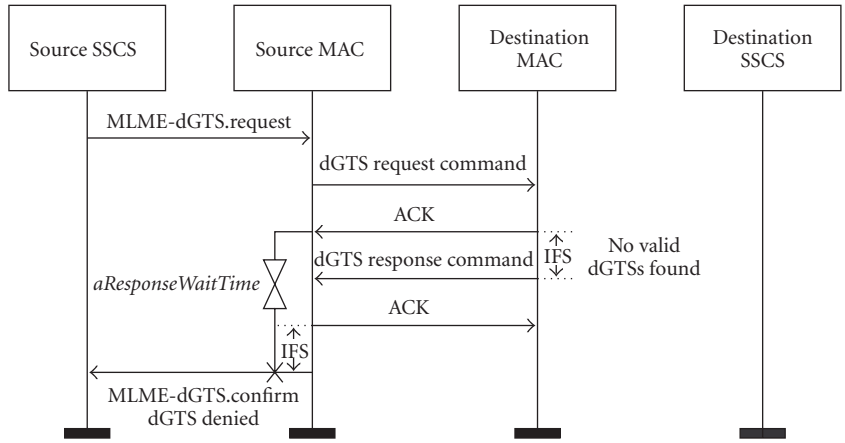

Figure 7: Denied dGTS allocation message sequence diagram.

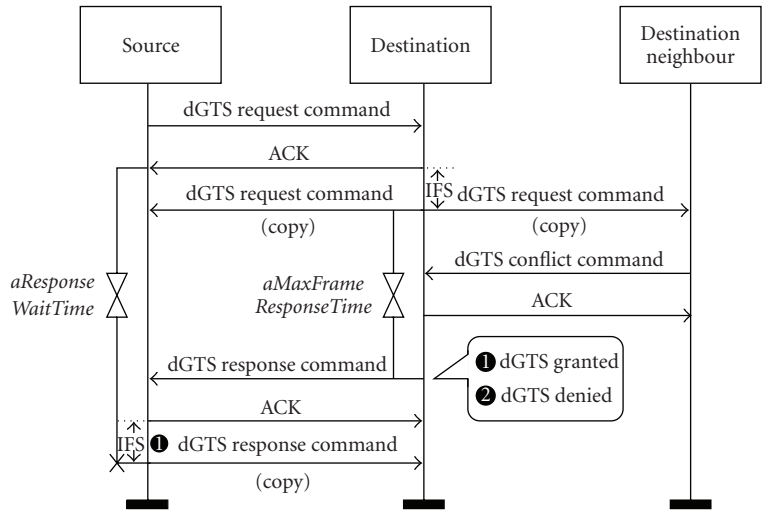

Figure 8: dGTS conflict resolution at the destination.

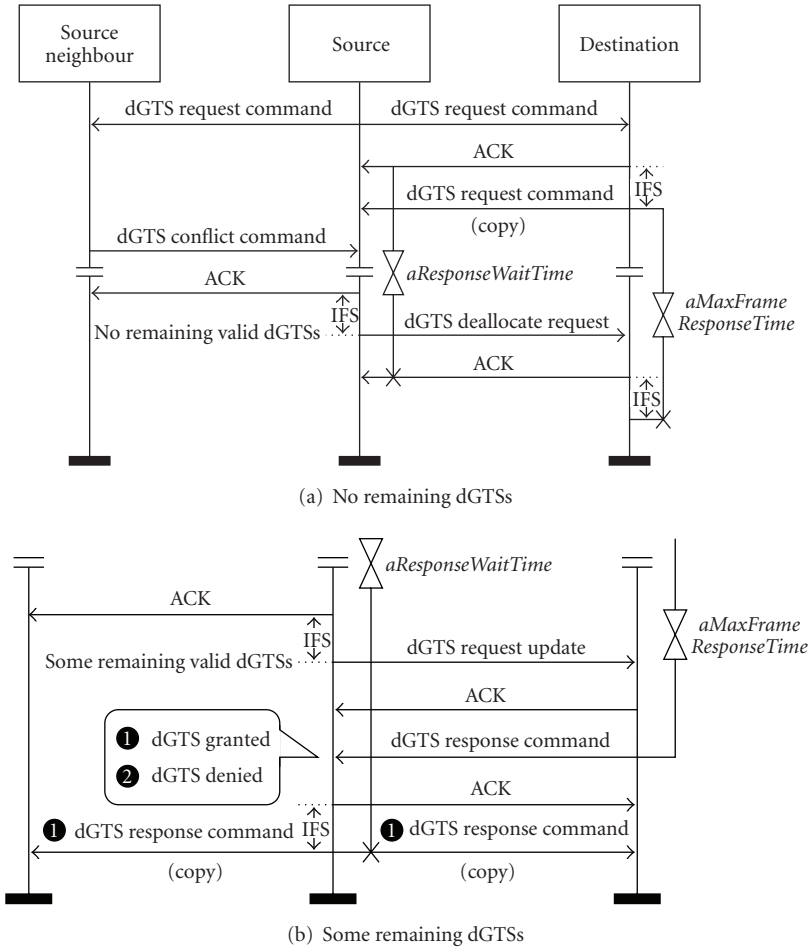

Figure 9: dGTS conflict resolution at the source.

Upon receiving this command, the dGTS source forwards a broadcast copy of the command to notify its neighbors about the allocated slot(s), so that the neighbors avoid using them for CAP and CFP. See Section 4.5.2 for details.

As shown in Table 2(c), this command includes the same five fields of the dGTS request command, except that the *list size* field can only be 0, indicating request rejection, or 1, indicating the request was granted. Also, the *starting slot* field is not a list anymore and has a single starting slot only. Lastly, the *command ID* field is set to 0x0B for this command.

4.3.3. dGTS conflict

This command indicates to its receiver that it is attempting allocation of a dGTS that conflicts with an allocated dGTS at the command sender. It is used by a neighbor of either the source or the destination of a dGTS if it receives a dGTS request or a dGTS response command involving a superframe slot (or more) that is already allocated at that neighbor. The neighbor notifies the source or the destination by means of this command about all of its allocated dGTSs that conflict with their dGTS request/response command.

This command includes five fields in its payload, as shown in Table 2(d). They are (1) *command ID* (8 bits): the command frame identifier equal to 0x0C by which this command is recognized; (2) *destination address* (64 bits); (3) *tx count* (4 bits): the number of dGTSs included in the command for which the node transmitting the command is the source; (4) *rx count* (4 bits): the number of dGTSs included in the command for which the node transmitting the command is the destination; and (5) *dGTS list* ((*tx count* + *rx count*) × 8 bits, 4-bit starting slot index and 4-bit dGTS length): the list of dGTSs, starting with the *transmit dGTSs* and then the *receive dGTSs*, with every dGTS specified by its starting slot index in the superframe and its length. A transmit dGTS is a dGTS for which the node is the source, while a receive dGTS is a dGTS for which the node is the destination.

4.4. Data structures

The dGTS protocol defines and uses the following data structures to provide its services.

4.4.1. dGTS characteristics

This structure holds sufficient information to identify a dGTS. It allows specifying the dGTS so that it may start at any

slot of a list of alternative starting slots. This allows a dGTS request command to efficiently provide a prioritized list of alternative dGTSs of the same length for the destination to choose from. The first slot in the list has the highest priority (the most preferred), while the last slot in the list has the lowest priority (the least preferred). Exchanging dGTS specification is always performed using this structure, specifically in dGTS request and dGTS response commands. The last four fields in these commands' payload constitute this structure. Figure 4 illustrates this fact and presents an instance of the structure for a node (node 5) wishing to allocate a 4-slot dGTS to another node (node 6), providing a single starting slot only. Moreover, exchanging dGTS information between the MAC sublayer and the SSCS sublayer uses this structure as a parameter to all protocol primitives (see Section 4.2).

The dGTS characteristics structure consists of four fields. These are (1) *partner address* (64 bits): the address of the node involved in the dGTS represented by this instance of the structure, besides the one handling it; (2) *dGTS length* (4 bits): the number of superframe slots forming the dGTS; (3) *list size* (4 bits): the number of alternative starting slots provided in this instance of the structure; and (4) *starting slot list* (*list size times* 4 bits): the list of the alternative superframe slots at which the dGTS may start. Note that the *partner address* field holds the address of the dGTS destination in dGTS request commands, MLME-dGTS.request primitives, and MLME-dGTS.confirm primitives, while it holds the address of the dGTS source in dGTS response commands, MLME-dGTS.indication primitives, and MLME-dGTS.response primitives.

Although this structure is capable of holding multiple starting slots, it is still representing one dGTS, but with multiple possibilities for the slot at which it may start. This is why it has a single *dGTS length* field. Also, there is no need to specify the direction of the dGTS represented by dGTS characteristics structure, as this information is usually available from the context. The direction of a dGTS can be either *transmit* or *receive*. For instance, in a dGTS allocation request command, it is implicitly known that the dGTS characteristics represented by the command is a transmit dGTS with respect to the source, since dGTS allocation commands are transmitted by the node that will transmit the data during the dGTS once it is allocated. Conversely, a node receiving an allocation request command considers the underlying dGTS a receive dGTS. This is also the case for dGTS response commands, as the node transmitting the dGTS response command considers the involved dGTS a receive dGTS while the node receiving the command considers it a transmit dGTS. However, this context is unclear in deallocation request commands since both the source and the destination of a dGTS may transmit a deallocation request command. Hence, the dGTS deallocation request command has a *direction* bit in its *flags* field.

4.4.2. dGTS conflict characteristics

This structure holds information regarding a number of dGTSs. Unlike the dGTS characteristics structure, dGTSs represented in this structure do not necessarily share a common dGTS length, and they can be of any direction: transmit dGTSs, receive dGTSs, or a mixture of both. This structure is used to exchange dGTS characteristics for the conflict command. The dGTS characteristics structure is not suitable for conflict commands as it is intended to represent a single dGTS, while a conflict command may hold multiple dGTSs that conflict with a received request or response command.

Since this structure potentially holds a number of dGTSs, it indicates the number of dGTSs it represents in two counters: one for transmit dGTSs and one for receive dGTSs. It also lists the dGTSs, starting with all the transmit ones and followed by the receive ones. Each dGTS is described by two pieces of information: its starting slot and its length. Obviously, the fields of this structure are the same as the fields of the payload of the conflict command (Table 2(d)), excluding the *command ID* field.

Figure 4 shows an instance of this structure where a node (node 1) needs to notify a neighbor (node 5) that its request involves slots already used by the node with other neighbors. In this example, one transmit dGTS and two receive dGTSs (of different lengths) conflict with the request sent by the neighbor (node 5) and are included in the conflict command. Such information cannot be conveyed via the dGTS characteristics structure and requires the more flexible dGTS conflict characteristics structure.

4.4.3. dGTS table

This structure holds the node's information regarding available and allocated slots. It consists of two tables: the *own table* and the *neighbor table*. Information about dGTSs for which the node is either the source or the destination is stored in the own table, while information about dGTSs for which the node is neither the source nor the destination is stored in the neighbor table. In other words, the neighbor table holds the node's information about its neighbors' dGTSs, which it has to keep track of to avoid interfering with them. A major difference between the two tables is that the neighbor table allows dGTS overlap. This is because two neighbors of a node may not be neighbors of each other and hence they can use the same slot. The node needs to keep track of both allocations to be able to identify when the slot is going to be available again after both dGTSs are deallocated. In the own table, a node can use a slot in at most one dGTS. In addition, the own table should support flagging dGTSs for deallocation to avoid using them before they are actually deallocated, as the deallocation procedure is sequential (it deallocates dGTSs one by one) and not instant (see Section 4.5.4 for details).

The dGTS table structure is not communicated in any of the protocol procedures, so it is not part of the protocol. It is an internal nodal structure that is essential to the protocol operation, though. Therefore, it can be implemented in any form as long as it provides the needed information.

The dGTS table is utilized when constructing allocation request and response commands. These commands should not involve any dGTS that conflicts with any dGTS already in

the dGTS table. A conflict is recognized whenever there is a common slot in a dGTS in the table and a prospective dGTS. (See Section 4.5.3 for details on how conflicts are handled.) The table structure should disallow conflicts to be stored in the table. In addition, the dGTS table is consulted whenever an allocation request command or a response command is received, and is updated whenever a deallocation request command or a conflict command is received. It is worth noting that received allocation request commands do not change the table, but rather alerts the node about potential slots for allocation that are available according to the source's table.

4.4.4. dGTS queue

Whenever a data packet (or any higher-layer packet) arrives at the MAC sublayer from the LLC queue, the packet is transmitted and the queue is blocked until the packet is completely handled. If, however, the packet happens to be an application-layer data packet to be transmitted during a dGTS, this model no longer works. This is due to the fact that such a packet cannot be handled immediately. In fact, it has to wait until a suitable dGTS starts. In the meantime, according to this model, the MAC sublayer will be unable to retrieve any further packets from the LLC queue and the rest of the superframe will be wasted until a dGTS suitable for the received packet is reached. To work around this situation, a MAC-level first-come, first-serve (FCFS) queue is introduced for holding data packets that are to be transmitted during a dGTS. This queue, namely, the *dGTS queue*, holds all such packets irrespective of their possibly different destinations/dGTSs. Immediately prior to the beginning of a transmit dGTS, the node checks its dGTS queue for the first packet whose destination is the destination of the current dGTS, and that fits within the dGTS time together with its ACK frame (if one is required). After handling that packet, the dGTS queue is repeatedly searched again for an appropriate packet until there is none or the remaining time of the dGTS does not suffice to complete the transaction for any packet intended for transport to the dGTS destination.

Equivalently, this queue can be implemented in the LLC layer, in which case the LLC layer has to be able to identify dGTS packets upon reception from higher layers and append them to the dGTS queue instead of the standard LLC queue.

Although this queue does not necessarily belong to the MAC sublayer, it requires a special interface if it is to be implemented in the LLC sublayer, for example. In particular, it should support the search-by-destination function described above. Also, it should support retrieving packets without dequeuing them. This is required to check whether the transaction fits in the remaining time of the dGTS or not. If the transaction does not fit in the remaining time of the dGTS, the packet is kept in the queue and a subsequent packet is sought until one is found or the end of the queue is reached. If the transaction does fit within the dGTS boundaries, the packet is dequeued.

As is the case for the dGTS table structure, the dGTS queue is required to provide the aforementioned interface only. It is not part of the communication protocol and, hence, any implementation providing this interface is acceptable. The maximum queue size is a protocol/implementation parameter that can be tweaked for better performance. For example, a very large queue reduces packet dropping and improves delivery ratios for bursty traffic, but results in longer delays (depending on the size of the average queue occupancy).

4.4.5. Retransmission queue

The synchronization component of the framework guarantees a common superframe structure throughout the network. However, it has no control over the subdivision of the superframe into CAP and CFP periods. This subdivision is merely a result of which dGTSs are allocated at each node (and its neighbors) and, hence, can be different for every node in the network. An undesirable result of this system is that although the CAP always starts at the same point in time for all nodes, some nodes may have shorter CAPs than their neighbors. Figure 5 demonstrates an example of neighbors with different CAP lengths. Consequently, there exists a period for every two neighbors (in Figure 5) with different CAPs in which one is still in its CAP and the other is in its CFP. During that period, any CAP transmission (from the node with the longer CAP destined to the node with the shorter CAP, i.e., during the latter's CFP) will fail since it will be ignored by the intended receiver. Note that transmissions during that period will always be from the node with the longer CAP. Also, note that such a transmission will never interfere with the receiver's own transmit/receive dGTSs, for if the receiver were in the CFP due to an own dGTS, the node would be in its CFP too to avoid interfering with its neighbor's dGTS.

This behavior negatively impacts the efficiency of the protocol. To recover from such failures, a MAC-level retransmission queue is introduced. Whenever ACK frames are not received during the CAP and all retransmission attempts (referred to hereafter as *simple retransmissions*) are exhausted, the packet is queued into the *retransmission queue* instead of being dropped, and the status of the transmission is withheld from the higher layers. This queue is checked at the beginning of every superframe, and a packet is dequeued and attempted for transmission, or *secondary retransmissions*. This mechanism is applicable only to packets that are transmitted during the CAP and that require acknowledgment.

The retransmission queue is not part of the protocol either although it boosts its performance. It has to meet a simple interface allowing the MAC sublayer to retrieve packets from the queue, but not to dequeue them implicitly. Instead, packets should be explicitly dequeued upon the protocol's request. This is basically needed in case the retrieved packet causes task overflow given the current status. For example, if the retrieved packet happens to be a dGTS request command and there is a dGTS allocation procedure in progress, the request command is not dequeued and the queue is queried for another packet, since carrying out two dGTS allocations simultaneously is not permitted. If the packet can be handled, then it is dequeued.

4.5. Protocol procedures

In this section, we describe how the protocol performs the individual tasks that collectively achieve the ultimate objective of contention-free medium access in a flat peer-to-peer topology. A task or procedure involves one or more MAC commands and is performed by a single node, but usually requires some interaction with other nodes. While another node's interaction may be required, it is part of another task performed at that other node.

4.5.1. Requesting dGTS allocation

We first define superframe slot and dGTS properties pertinent to dGTS allocation. A superframe slot is said to be *available* if it is *not* allocated in either of the own or neighbor dGTS tables. A *valid* dGTS is one that starts and ends within the superframe, does not start at the first superframe slot, and is made up of available superframe slots. A dGTS is *suitable* for a given packet if it is valid, allocated to the destination of that packet, and the packet transmission transaction fits in the dGTS duration.

When a higher layer decides that a dGTS is needed, it instructs the MAC sublayer to initiate the allocation procedure using the SSCS interface's MLME-dGTS.request primitive, and passes along the required dGTS characteristics. Alternatively, an allocation procedure can also be automatically initiated by the MAC sublayer upon receiving a data packet to be transmitted in a dGTS while there are no allocated dGTSs suitable for the packet transmission, for instance. This may happen if a suitable dGTS had expired, if the packet is too large to fit in allocated dGTSs, or if this is the first packet in its flow and allocation is data-triggered (see Sections 4.5.5 and 4.6 for details).

When the MAC sublayer of the source node receives the MLME-dGTS.request primitive, it first makes sure that the requested dGTS is valid and the corresponding slots are available in its dGTS table. Otherwise, the procedure is terminated with an `INVALID PARAMETER` return code. After that, the source constructs and transmits a dGTS request command to the destination, the node to which it wishes to transmit during that dGTS, the destination. The command specifies the requested dGTS by specifying its length (the number of slots it spans) and its starting slot. Since available slots may be different from node to node, the source may specify a set of starting slots to increase its chances of matching available slots at the destination. The command is transmitted as a broadcast frame enabling the source's neighbors to make note of involved slots that are available according to the source. After consulting its own dGTS table, if a neighbor finds out that a slot is involved in the command and is allocated to itself, it follows the dGTS conflict procedure to notify the source about the conflict. The source then notifies the destination, as explained in the dGTS conflict procedures in Section 4.5.3. Neighbors never allocate any slots in their tables upon receiving dGTS request commands. Next, the source awaits an ACK frame from the destination within an acknowledgment timeout period, namely, *macAckWaitDuration* (54 symbols long), as per the standard. Retransmission procedures are identical to those in the standard too.

The source then waits for a relatively long period of time during which the destination makes its decision. The exact duration is a protocol parameter that can be manipulated, depending on network conditions, to achieve better performance. The period may span multiple superframes and does not have to fit within the superframe during which the allocation request was transmitted by the source. A potential value for this period is the standard's *aResponseWaitTime* (i.e., $32 \times aBaseSuperframeDuration$). The source expects to receive a dGTS response from the destination within this period. Otherwise, the MAC sublayer reports to the SSCS sublayer (and, in effect, to higher layers) that a response has not been received (via the `NO DATA` return code) and the procedure terminates as far as the MAC sublayer is concerned. Higher layers may choose to reattempt the allocation later. This ensures that a node that is no longer a neighbor is ignored, rather than needlessly attempting to reach it.

If a dGTS response command is received within its timeout, the command's *list size* field is checked. A value of zero indicates that the request has been rejected by the destination, while a value of one indicates that the request has been granted and the *starting slot* field holds the index of the first slot of the allocated dGTS (since the request could have had more than one starting slot). If the request is granted by the destination (as illustrated in Figure 6), the source checks the received dGTS characteristics against its dGTS table to make sure that the involved slots are still available (since a neighbor of the source only could have had a conflict and notified the source about it after receiving its dGTS request command). If the dGTS is found valid, it is allocated in the source's own dGTS table and the source forwards a copy of the dGTS response command it received to its neighbors to make them allocate the involved slots in their neighbor dGTS tables, successfully terminating the procedure. The forwarded dGTS response is not acknowledged since it is targeted to all of the source's neighbors. This is achieved by using the source's own address in the *destination address* field of the command. If the dGTS is found invalid though, the source deallocates the dGTS as per the deallocation procedure (see Section 4.5.4).

4.5.2. Responding to dGTS allocation request

When a node receives a dGTS request command requesting the allocation of a dGTS, the node first filters out all starting slots in the command that will cause a conflict with the node's dGTS table if they were to be erroneously used. If this step results in filtering out all the starting slots included in the dGTS request command, the node constructs and transmits a dGTS response command whose *list size* field is zero indicating a denied request (as illustrated in Figure 7), and the procedure terminates. Otherwise, the node forwards a modified copy of the dGTS request command to its neighbors in which (1) the *destination address* field is the node's own address to indicate that this is a forwarded copy, and (2) the *starting slot list* field contains the filtered list

rather than the original one. The node's neighbors handle this forwarded copy in exactly the same way the source's neighbors handle the dGTS request command; no actual allocation is performed, and the node is notified about conflicts.

The node then waits for a period of time shorter than that for which the source awaits the dGTS response command, providing sufficient room for retransmissions if necessary. A potential value for this period is the standard's *aMaxFrameResponseTime* (i.e., 1220 symbols long). The purpose of waiting for this period is to ensure that all conflicts have been reported (by the neighbors) by the end of the period. Afterwards, the node checks the list of starting slots against its dGTS table again in case any conflicts were received. If no starting slots are left, a dGTS response whose *list size* field is zero is constructed and transmitted, indicating that the request has been denied. Otherwise, if there are remaining valid starting slots, the new list is passed to the higher layer via the SSCS interface using the MLME-dGTS.indication primitive to choose one or more starting slots from the new list (if such a feature is desired).

The SSCS sublayer then calls the MAC sublayer's MLME-dGTS.response primitive and passes along the chosen starting slot(s). If the MAC sublayer receives more than one starting slot from the SSCS sublayer, it picks the first. The convention is that nodes construct their starting slot lists with the first entry designated as the most desired and the last as the least desired.

The node constructs and transmits a dGTS response command containing the single chosen starting slot together with the other dGTS characteristics to the source, and waits for an ACK frame, following the standard's retransmission rules. Upon receiving the ACK, the node updates its own dGTS table to reflect the allocation of this receive dGTS. If the dGTS response command is not acknowledged, the source is assumed to be no longer a neighbor and the allocation is ignored.

4.5.3. dGTS conflict handling

Whenever a node receives a dGTS request or a dGTS response command, it reacts differently depending on whether it is the intended destination whose address is in the *destination address* field of the payload or not. If it is the intended destination, it checks the command for conflicts against its dGTS table (both own table and neighbor table). A conflict is declared whenever two different dGTSs have at least one superframe slot in common. If a conflict is detected and the command is a dGTS request, the starting slots causing the conflict are removed from the list and the remaining starting slots only are considered when processing the command (see Section 4.5.2). If a conflict is detected and the command is a dGTS response, the response is considered invalid since it has a single starting slot and the deallocation procedure is conducted to deallocate the dGTS at the destination (see Section 4.5.1). Higher layers can decide whether to reattempt the allocation.

If the node receiving the command is not the intended destination, it checks the command for conflicts against its own dGTS table only. The neighbor dGTS table contains information about slots the node cannot use for its own dGTSs because they are being used by some of its neighbors. However, the neighbor dGTS table cannot be used to judge whether a neighbor can use a particular slot or not; a node cannot decide for any other node which slots are available and which are not, but can tell which slots cause conflicts with itself (by consulting only its own dGTS table). On the other hand, both tables are used to assess slot availability when a node is considering allocating slots for its own dGTSs.

If a dGTS conflict is detected, whether triggered by a dGTS request or response command, the node constructs and transmits a dGTS conflict command containing a list of all of its own dGTSs that have at least one slot in common with any of the dGTSs included in the command that triggered the conflict checking. Additionally, if the command triggering the conflict checking is a dGTS response command, and a conflict is detected subsequently, the dGTS in the command will not be stored in the dGTS neighbor table. The address of the node that transmitted the problematic dGTS request or dGTS response command is copied into the *destination address* field in the dGTS conflict command.

When a dGTS conflict command is received by a node, regardless of whether it is destined to it or not, each dGTS included in the command is checked individually as follows. First, if the dGTS is an own dGTS and the received command was transmitted by the partner node, the dGTS is ignored because it is already in the own dGTS table. Otherwise, the following steps apply. If the dGTS is not found in the neighbor dGTS table, the node inserts it there. Note that the neighbor dGTS table is capable of storing identical dGTSs by counting them. Nevertheless, if the dGTS in the dGTS conflict command is found in the neighbor dGTS table, the counter is not incremented and the dGTS is ignored. This is to avoid reallocating already allocated dGTSs whenever they are redeclared in a conflict command. The counter is incremented only upon receiving dGTS response commands where there is no chance the dGTS is already in the table. Finally, the own dGTS table is checked to see if it conflicts with the dGTS in case it just got allocated in the neighbor dGTS table. This can happen if the source of the conflict command has just joined the neighborhood or the former commands, through which the dGTS was allocated, were not correctly received at the node processing the conflict command. If a conflict is found, the neighbor's dGTS takes precedence and the own dGTS is deallocated using the deallocation procedure.

If the node happens to be in the middle of a dGTS transaction (i.e., requesting a dGTS or responding to an allocation request) as it receives the dGTS conflict command, a few extra checks are carried out. If the node is undergoing a dGTS allocation request procedure and a conflict command is received, the dGTS characteristics used in the dGTS request command are checked against the dGTSs in the conflict command. If a conflict is found, the node constructs and transmits a new dGTS request command with an updated *starting slot list* field holding only valid starting slots after

considering the conflict command: a request update. The destination then discards the original starting slot list and considers the updated one only when making its decision with respect to which starting slot(s) to use. If none are left, the node transmits a deallocation request indicating to the destination that the allocation procedure is aborted. This deallocation request should be ignored by other nodes, and thus has the corresponding *ignore flag* active. This is because there is no allocated dGTS for the request to deallocate. Note that the request update command is discarded if a response is received any time before the command is relayed to the physical layer. This is because the transaction would have already been terminated at the destination.

If the node is undergoing a response procedure and a dGTS conflict command is received, it removes the starting slots in the request it is processing for which the requested dGTS conflicts with any dGTS in the conflict command. Upon the conclusion of the waiting period associated with receiving potential conflicts from neighbors, the node chooses a starting slot from the updated starting slot list.

4.5.4. dGTS deallocation

The dGTS deallocation procedure can be initiated by the source or the destination of an allocated dGTS. Whenever a node needs to deallocate a dGTS, whether due to instructions from higher layers, to resolve conflicts, or because a dGTS has expired according to the own dGTS table, it constructs and transmits a dGTS request command whose (1) *destination address* field is the address of the partner node for that dGTS according to the dGTS table; (2) *list size* field is zero; and (3) *starting slot list* field contains a single starting slot: the starting slot corresponding to the dGTS to be deallocated. The node then waits for the ACK frame. Upon receiving it, the dGTS is deallocated from the own dGTS table. If the command is not acknowledged after exhausting all the retransmissions attempts, the dGTS is deallocated and the partner node is assumed to be no longer a neighbor.

Receiving a dGTS deallocation request destined to the node triggers the node to forward a copy of the command, whose *destination address* field is the node's own address, to prevent further forwarding. The forwarded copy is not acknowledged and is used to notify the node's neighbors about the deallocation so that they can free the involved slots in their respective neighbor dGTS tables.

If a node receives a deallocation request that is not destined to it, it first checks the *ignore flag*. If it is active, the request is ignored. Otherwise, the dGTS is deallocated from the neighbor dGTS table. As the neighbor dGTS table supports allocating multiple identical dGTSs, the counter for the corresponding dGTS is first checked. The table has a counter for every possible dGTS, characterized by its starting slot and dGTS length pair. If the counter is greater than one, it is decremented by one. If it is equal to one, the dGTS is completely removed from the node's neighbor table.

A node may not be able to deallocate a dGTS immediately when it decides that it needs to be deallocated, since only a single deallocation transaction can take place at the same time. This can take place if there is another dGTS still undergoing the deallocation procedure, a conflict command with multiple conflicting dGTSs is received, multiple dGTSs expire within the same superframe, or a higher layer requested the deallocation of multiple dGTSs at once. To avoid using these dGTSs while they are waiting for the deallocation of one of them to finish, or while in the CFP, the dGTS table's deallocation flag feature is utilized. Consequently, all own dGTSs that are to be deallocated are flagged as such and treated as if they are not allocated, and the deallocation procedure starts for only one of them. Once it terminates, it checks the own dGTS table for further flagged dGTSs and continues deallocating them until there is none left or the end of the CAP is reached. Also, at the beginning of every superframe, dGTSs flagged for deallocation are retrieved and the deallocation resumes. Note that only own dGTSs can be flagged for deallocation since they are the only dGTSs that can be actively deallocated by the node. Neighbors' dGTSs are managed by the neighbors and are only kept in the table to avoid interfering with them.

4.5.5. Transmitting data

Data packets are packets generated by the application layer as well as any higher layer relative to the MAC sublayer. To avoid ambiguity, in the context of the dGTS protocol, data packets represent application layer packets only. This distinction is necessary since the dGTS protocol handles its *data* packets in a way different from the rest of the higher-layer packets: it may transmit them during the CFP. This is not applicable to all higher-layer packets because some are required to setup the dGTSs that constitute the CFP, and hence have to be transmitted during the CAP. Obvious examples of such higher-layer packets are routing packets and address resolution protocol (ARP) packets. Also, some application layer packets generated by IEEE 802.15.4-aware applications may be transmitted during the CAP.

As demonstrated in Figure 10, whenever a higher-layer packet is received by the MAC sublayer, the MAC sublayer decides whether the packet is to be transmitted during the CAP or the CFP, depending on its type (or possibly using a flag in the packet set by IEEE 802.15.4-aware applications). If it decides that the packet is to be transmitted during the CAP, the slotted CSMA-CA algorithm is initiated, and upon the completion of the packet handling (including simple retransmissions), whether with success or failure, the higher layer (i.e., LLC) is notified and the next packet in the LLC queue is retrieved. The LLC queue is blocked while the current packet is being handled. If, however, the MAC sublayer decides that the packet is to be transmitted within a dGTS (i.e., during the CFP), the dGTS queue is checked for available space. If the dGTS queue is found full, the packet is dropped, for lack of dGTS queue space. If there is room in the queue, the own dGTS table is searched for (1) a *transmit* dGTS such that (2) the destination of the dGTS matches the data packet's destination, and (3) the packet transmission transaction (packet transmission, ACK reception, and appropriate IFS) fits in the dGTS duration. If no such dGTS is found, a dGTS allocation procedure is initiated with the appropriate dGTS characteristics and

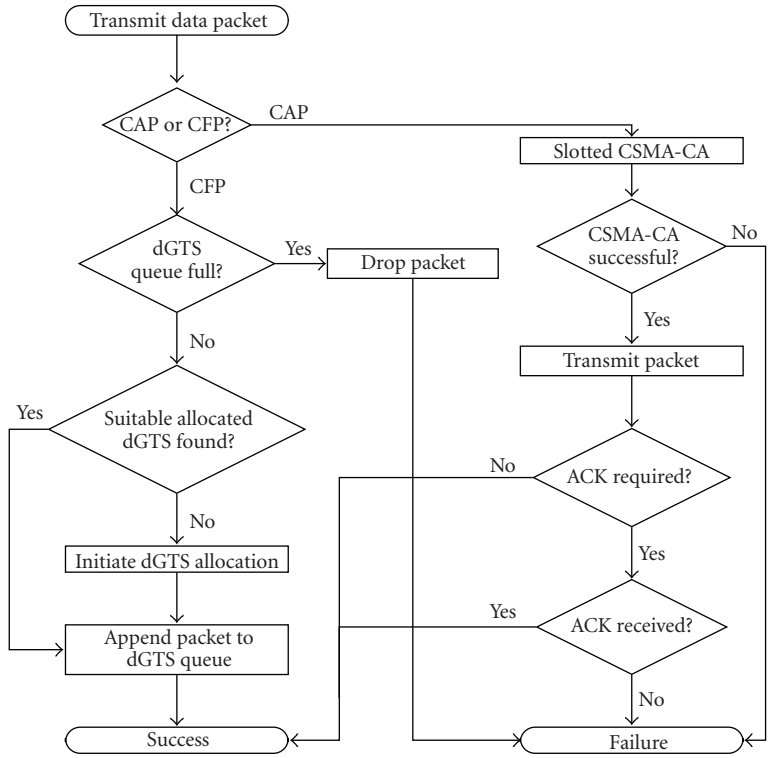

Figure 10: Transmitting data packets.

the packet is appended to the queue. If a suitable dGTS is found, the packet is appended to the queue too. In all cases, packet handling is considered complete as far as the LLC queue is concerned although the packet has not been transmitted yet. Hence, the MAC sublayer reports completion of the packet handling and the next packet, if any, in the LLC queue is retrieved.

Figure 11 outlines an outgoing packet's journey across the MAC sublayer, and shows the roles of various queues encountered by the packet.

At the beginning of every active transmit dGTS, the transceiver is set to transmit mode and the dGTS queue is searched for the first data packet whose destination matches the destination of the current dGTS, and whose transaction fits in the dGTS duration. An active transmit dGTS is a dGTS that is *not* flagged for deallocation and during which the node *transmits* one or more data packets. If none is found, nothing is done. If a packet for which the dGTS is suitable is found, it is transmitted immediately (i.e., without CSMA-CA). Unlike protocol commands, ACK frames are optional for data packets. If ACK frames are used for data packets, the node waits for one for at most *macAckWaitDuration*, as is the case for MAC commands. If an ACK is received, or if one is not required, the packet is removed from the dGTS queue. Otherwise, it is kept in the queue until it is transmitted successfully. In any case, these steps are repeated until there are either no packets left in the dGTS queue or until none of the packets left in the dGTS queue are suitable for transmission in the remaining time of the dGTS. If the next slot does not belong to an own dGTS, the transceiver is shutdown at the end of the current dGTS, unless the current dGTS ends at the next superframe boundary.

4.5.6. Secondary retransmission

Due to the fact that the CAPs can be of different lengths in neighboring nodes (see Figure 5), a node may attempt transmission to a neighbor that is no longer in the CAP, and hence may never receive an ACK. As discussed in Section 4.4.5, we introduce the *retransmission queue* as a recovery mechanism from this problem. This mechanism exploits the fact that although CAPs may end at different times in neighboring nodes, they always start simultaneously. Therefore, if an ACK is not received for a CAP transmission and having exhausted all retransmissions, the packet is queued in the retransmission queue and its transmission is reattempted at the beginning of the next superframe, as outlined in Figure 11. However, this is not the only transmission attempted at the beginning of the superframe. In particular, there could also be a pending CSMA-CA transmission whose backoff timer was started in a previous CAP, a packet (or more, of different types) waiting for CSMA-CA to finish transmitting a previous packet, and/or one or more own dGTS(s) flagged for deallocation (but not yet deallocated) awaiting a deallocation request to be constructed and transmitted at the beginning of a superframe. These four types of packets are checked for at the beginning of every superframe, and one of them is attempted. Priorities are assigned to these four types based on how critical they are for the protocol performance. As per our performance

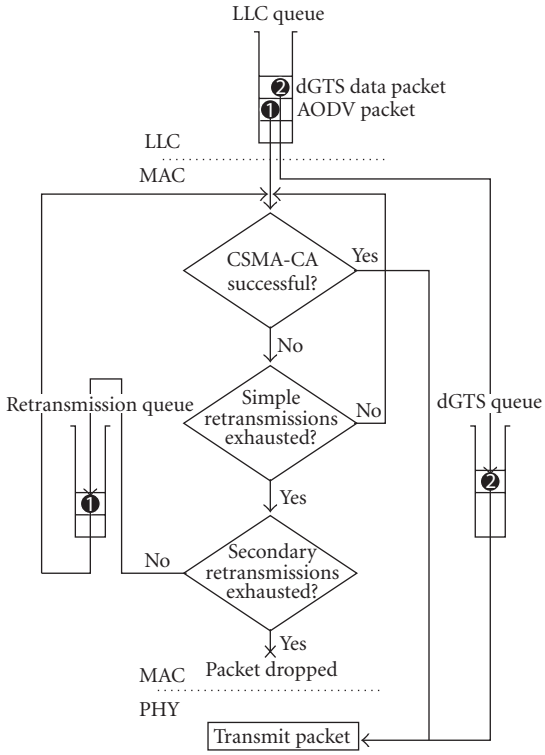

Figure 11: Packet queues.

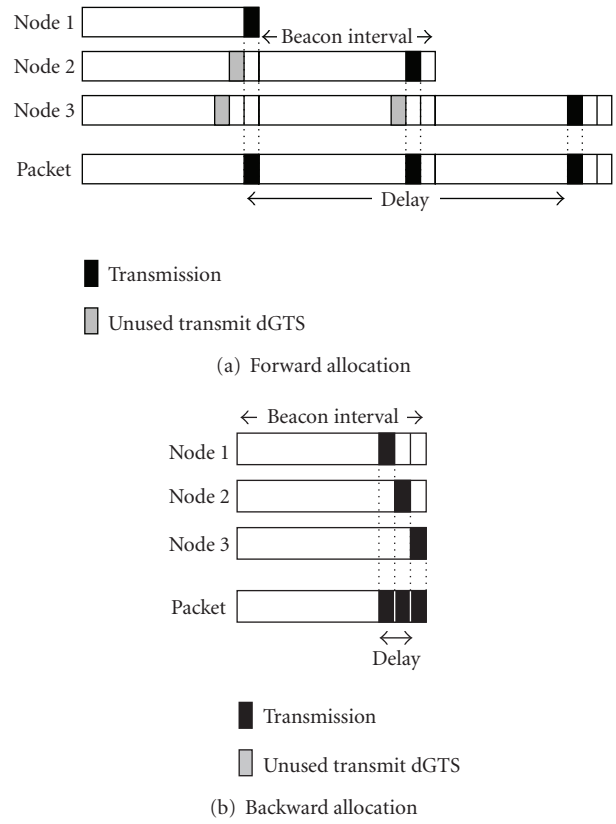

Figure 12: End-to-end delay. Node 1 is closest to the source and node 3 is closest to the destination, but is not the destination.

experiments, pending CSMA-CA transmissions and packets waiting for CSMA-CA is the most critical, so they are always handled first if there is any. If no pending CSMA-CA transmissions are found, any pending deallocations are started. If none are found, the retransmission queue is checked and if any packet that does not cause task overflow is found, it is dequeued and transmitted.

Note that a packet whose backoff timer has been started previously has a higher priority compared to packets waiting for CSMA-CA to finish handling other packets. This is essential to keep packet handling strictly sequential without any interleaving. Interleaved packet handling corrupts duplicate packet detection at the destination if both packets are destined to the same node. For example, consider the case where a packet is first transmitted and received correctly at the destination, but its ACK is not received. Then, another packet is transmitted to the same destination, and the first packet is retransmitted afterwards. In this case, the destination will not be able to detect that the retransmitted packet is a duplicate of the first since duplicates are identified using the sequence number of the last packet received from a given neighbor, which is no longer the same packet in this scenario. Proper duplicate packet detection is important because whenever a duplicate packet is received, it is acknowledged again to suppress further retransmissions. Therefore, erroneous duplicate packet detection may cause numerous unnecessary retransmissions, and can lead to transaction corruption in some scenarios.

When a packet is retrieved from the retransmission queue, it is transmitted normally as if it were the first time. This includes the simple retransmissions, provided that ACK frames are not received. However, if the packet remains unacknowledged after exhausting the simple retransmissions, no more retransmissions are attempted and the packet is dropped.

4.6. Integrated dGTS and routing

Being part of the MAC sublayer, the dGTS protocol is independent of the routing protocol. However, exchanging some information with the routing protocol allows both protocols to achieve better performance. The following subsections outline our proposed optimizations as well as their advantages and limitations. Basically, dGTSs can be allocated in a forward order or in a backward order. Using the forward order of allocation, dGTSs are reserved at the source first, followed by the next hop, and so on until the final hop. In the backward order of allocation, dGTSs are reserved at the last hop first, followed by the next-to-last hop, and so on until the first hop. If the allocation triggering mechanism results in dGTSs being allocated in the forward order, the source will start by allocating the latest available slots in its superframe, forcing subsequent hops to use earlier slots. This increases the delay between every two subsequent hops along the route, as shown in Figure 12(a). On the contrary, dGTSs allocated in the backward order may incur next-to-none delay as shown in Figure 12(b), subject to slot availability.

4.6.1. Forward (data-triggered) dGTS allocation

The dGTS protocol's internal allocation mechanism basically triggers the dGTS allocation procedure whenever a data packet, received from higher layers, is to be transmitted during the CFP, but there is no allocated dGTS suitable for that packet's transmission. This implies that all routing decisions are made independently of dGTS availability along candidate routes, and it is only when the packet arrives at the MAC sublayer that the dGTS is allocated. This usually works because dGTS transmissions are designed for regular traffic; if the first packet triggered an allocation task that did not succeed, there will be subsequent packets that belong to the same flow that will eventually establish the dGTS. The major advantage of this approach is that it is an OSI-model-compliant solution. However, this dGTS assignment policy will inadvertently result in the forward allocation of dGTSs (as shown in Figure 12(a)), likely contributing to enlarging the route's end-to-end delay.

4.6.2. Backward (route-triggered) dGTS allocation

In contrast to forward dGTS allocation, consider the case where a node attempts the dGTS allocation whenever it receives a route reply (RREP) packet, and only forwards that RREP to its downstream neighbor if a dGTS is successfully allocated. Obviously, the route's destination readily constructs and transmits, as per slotted CSMA-CA, the RREP packet as soon as it receives the route request (RREQ). Now every RREP received at the route's source node is guaranteed to have a dGTS path along its route. Such a guarantee is not present when dGTS data triggered allocation is used. In effect, this mechanism deals with dGTS allocation as part of the route discovery process, and a route is not discovered if dGTSs could not be allocated along its hops. Another major advantage is the fact that dGTSs are allocated at the destination first and backwards along the discovered route to the source [14]. dGTSs are usually allocated slots from the end of the superframe, so this reversed order of allocation along the route significantly minimizes end-to-end delay [14], as illustrated in Figure 12. These built-in guarantees come at a cost though. First, if a dGTS gets deallocated at a later time due to some conflict for example, a secondary mechanism is required to reallocate it, or else packets that use that dGTS will get stuck in the dGTS queue and consume the queue space while other packets with active routes may get dropped for lack of queue space. Second, when a RREQ packet reaches a node that is already on a dGTS route that traverses, or ends at, the RREQ's destination, that node will most likely be the first to reply to the RREQ. In particular, it will send its RREP packet sooner than any node with no dGTS routes to the RREQ's destination. Consequently, routes will tend to *aggregate* at such nodes, which will be overused for routing. Besides negatively affecting the network lifetime, this behavior may result in considerable packet dropping. This is because that node will not need to receive an RREP since it is already aware of the route to the destination, and hence will not allocate a separate dGTS for the new route. It will just send an RREP to its downstream neighbor and, in effect, use its existing dGTS for both the old and new routes; it will have a single transmit dGTS handling the traffic coming from two (or more) receive dGTSs. Obviously, a dGTS allocated initially for a single flow will not be able to accommodate multiple flows. Expanding such dGTSs to accommodate multiple flows requires more sophisticated MAC-routing cooperation (since the MAC sublayer is unaware of route destinations) that is better considered for avoiding route aggregation and promoting multiple routes instead. Multiple-route capability is a routing protocol feature [15]. Hence, the dGTS protocol should be used with a multiple-route-capable routing protocol that can evaluate multiple routes instead of one as a result of broadcasting a single RREQ packet, and then choose one (or more) based on some criteria [15]. One such criterion could very well be dGTS availability.

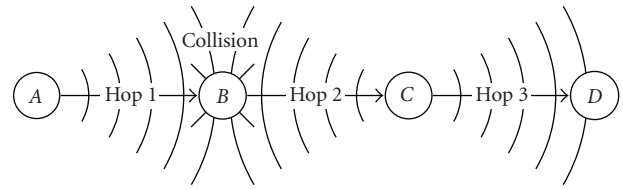

Figure 13: Slot reuse distance is equal to three.

4.6.3. Choosing dGTS superframe slots

In a MAC-only decision regarding which slots a dGTS should consist of, a reasonable policy is echoed in the standard: start allocating dGTSs from the end of the superframe. Although on the surface this may seem efficient, it suffers the short-sight problems of greedy algorithms, with nodes being likely to better utilize their superframe slots, but more likely to significantly worsen end-to-end delays. However, this is highly dependant on the adopted allocation triggering mechanism as well. In essence, dGTS allocation is a critical section whose resource is superframe slots. Thus, dGTSs along a route have to be allocated in sequence to avoid slot collisions. Note that the slot reuse distance is three. The worst-case delay scenario illustrated in Figure 12(a) applies to three subsequent hops only. The fourth hop, whose source is three hops away from the first source, have to pick slots that do not intersect with the second and third hops only, which could be the slots of the first hop. As illustrated in Figure 13, the third hop cannot use the first hop's slots because the source of the third hop, node *C*, is a neighbor of the destination of the first hop, node *B*, and its transmission will collide with the first hop's transmission by node *A*.

Nevertheless, in a dense network with relatively long routes, slot availability is very likely to be the dominant factor in deciding which slots are allocated to a dGTS. This may cause both allocation policies to converge to the same average delay. Sensor networks, the main application for IEEE 802.15.4 and, in turn, the dGTS protocol, do, by convention, meet these characteristics and, hence, the issue can safely be considered of moderate significance.

5. PERFORMANCE EVALUATION

In this section, we present various performance experiments characterizing the dGTS protocol. We use our framework's synchronization mechanism to provide a single superframe structure and identical slot timing throughout the network, eliminating the need for special-role nodes. Special-role nodes are avoided in the analysis because for a single given network topology, there could be numerous role configurations, each resulting in potentially different performance. More importantly, special-role nodes cause multiple beacons and multiple superframe schedules to be used simultaneously at different nodes in the network, which means that time is slotted in a local sense only, and transmissions to different beaconing coordinators behave, relative to each other, as if unslotted access is used. Therefore, to evaluate slotted random access, we use our framework's global synchronization mechanism to implement a flat topology that will behave consistently without further configuration. This mode of operation is referred to hereafter as *slotted mode*.

5.1. The simulator

Due to the availability of its source code, the *network simulator (ns-2)* [16] implementation of IEEE 802.15.4 (referred to as the *wpan* module) [4] is widely used by the research community. The dGTS protocol is implemented on top of the wpan module.

5.1.1. Limitations and proposed solutions

A few inconsistencies between the wpan module and the standard have been reported in [17]. We revisit a previously reported problem, discover a new problem, and provide efficient solutions for both. In particular, note the following two problems.

(1) Sometimes, slotted CSMA-CA remains idle for an additional backoff period following the second CCA (see Figure 14), while it is supposed to start transmission immediately after the second CCA is completed.

(2) For large BO and SO values (e.g., $BO = SO = 14$), networks perform extremely poorly because very few transmissions are carried out.

As for the first observation, the transceiver turnaround should finish no later than the backoff period boundary of the previous CCA. The standard dictates that the transceiver turnaround time is at most 12 symbols. The simulator uses that maximum value, which means that the turnaround time ends at exactly the backoff period boundary. This should work just fine. However, in the simulator, time is modeled as a double-precision floating-point number (C++ double) [18], which is subject to floating-point approximation errors. Hence, the absolute time at which the transceiver turnaround actually finishes in the simulator may sometimes be approximated to a slightly greater value than the upcoming backoff boundary, in which case the closest next backoff period boundary is another backoff period away. Note that in slotted CSMA-CA, transmissions are required to start at a backoff period boundary. The approximation error is of the order of magnitude of 0.1 femtosecond (10^{-16} second). Since the approximation may result in a slightly smaller value as well, it is not always the case that there is an idle backoff period prior to the transmission.

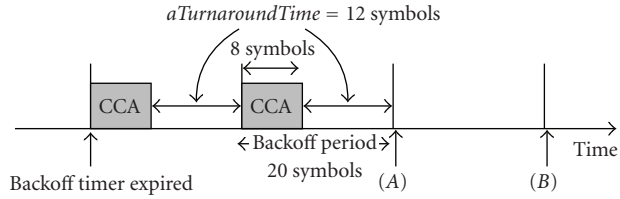

Figure 14: CCA in slotted CSMA-CA—idle channel. Transmission should start at (A). In ns-2, it sometimes erroneously starts at (B).

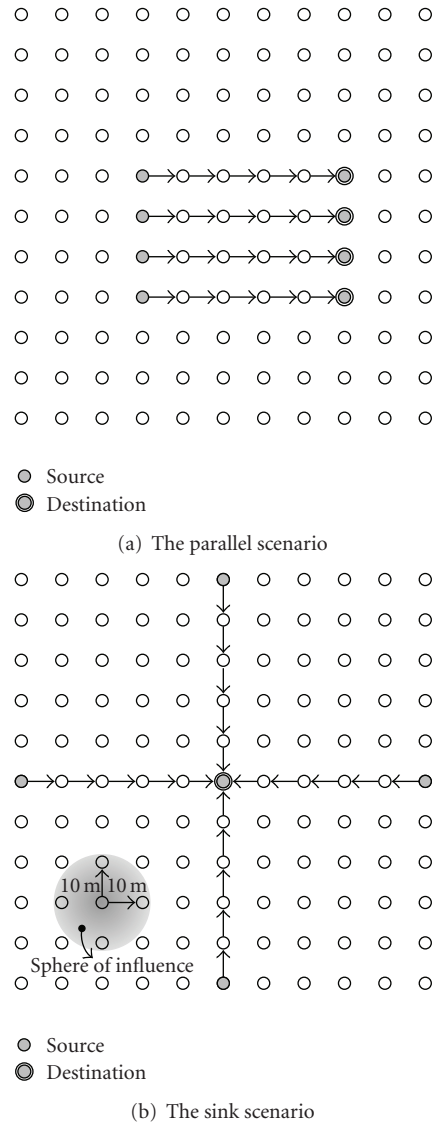

Figure 15: Simulation scenarios.

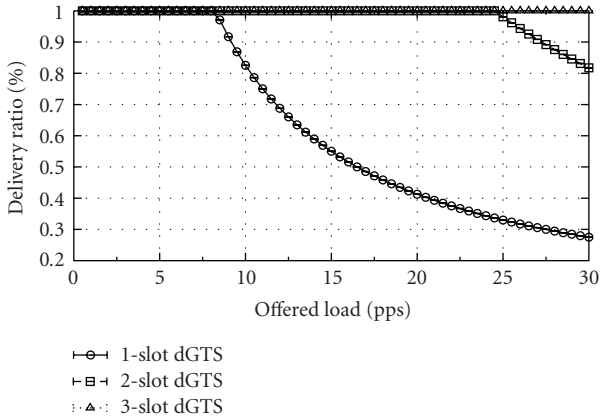

Figure 16: Delivery ratio, parallel scenario, manual routing.

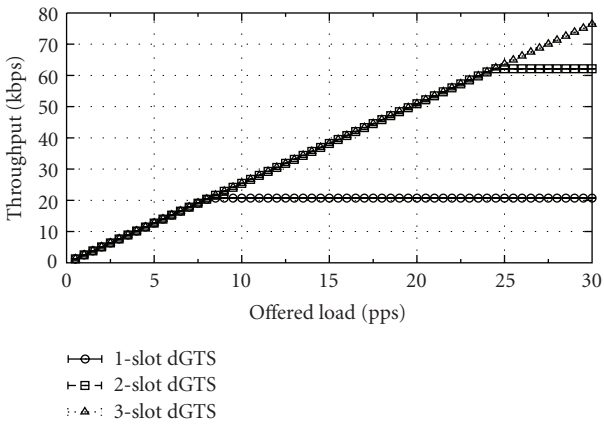

Figure 17: Throughput, parallel scenario, manual routing.

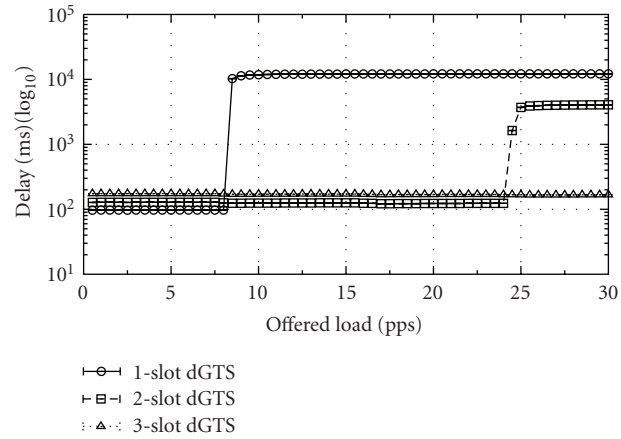

Figure 18: End-to-end delay, parallel scenario, manual routing.

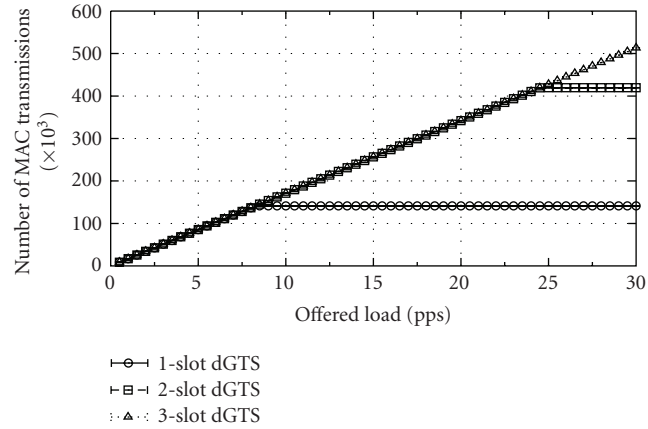

Figure 19: MAC transmissions, parallel scenario, manual routing.

There are two simple solutions for this problem: (1) in the simulation, use a value smaller than the maximum *aTurnaroundTime* for the actual transceiver turnaround, say 10 symbols; or (2) compare the difference between the current time and the time corresponding to the next backoff period boundary to a logical zero instead of an absolute zero, where a logical zero is a very small number that is close to zero but not exactly zero, such as 1 picosecond.

The second observation is due to underestimated integer values. In the CSMA-CA implementation, there are a couple of 16-bit integer variables that hold the number of remaining backoff periods in the current CAP so that the algorithm can decide whether a transaction would fit in the remaining time of the CAP or should be deferred to the next superframe. For large *BO* and *SO* values, the number of remaining backoff periods can get too large to fit into a 16-bit integer variable, causing an overflow and, in turn, an unpredictable behavior. For example, at the midpoint of a superframe, where $BO = SO = 14$, the number of remaining backoff periods until the end of the superframe is

$$\frac{aBaseSuperframeDuration \times 2^{14}}{2 \times aUnitBackoffPeriod} = 393216 > 65536 = 2^{16}. \quad (3)$$

Simply changing these variables from 16-bit to 32-bit integers fixes the problem and significantly boosts the performance of large *BO*/*SO* configurations.

While implementing the dGTS protocol for the ns-2 simulator, a few modifications had to be made to the wpan module. For example, the dGTS protocol broadcasts its messages but requires ACK frames based on a secondary destination address field. Also, since it operates in a flat topology, the addresses are always 64-bit long and short addressing is not used. However, frames with 64-bit destination addresses received by a node whose address does not match the destination address are dropped in the original wpan module implementation, even if they are broadcast frames. An exception had to be implemented to allow dGTS commands only to be received when in the synchronized peer-to-peer mode. A modification related to logging the simulation events to ns-2 trace files was also required. The wpan module implementation shares the same MAC headers as the IEEE 802.11 ns-2 implementation for trace support purposes. This causes some of the additional commands introduced by the dGTS protocol to be logged as 802.11-specific packets that do not exist in 802.15.4 such as RTS. To correct this behavior, we created a wpan-specific packet type space and added

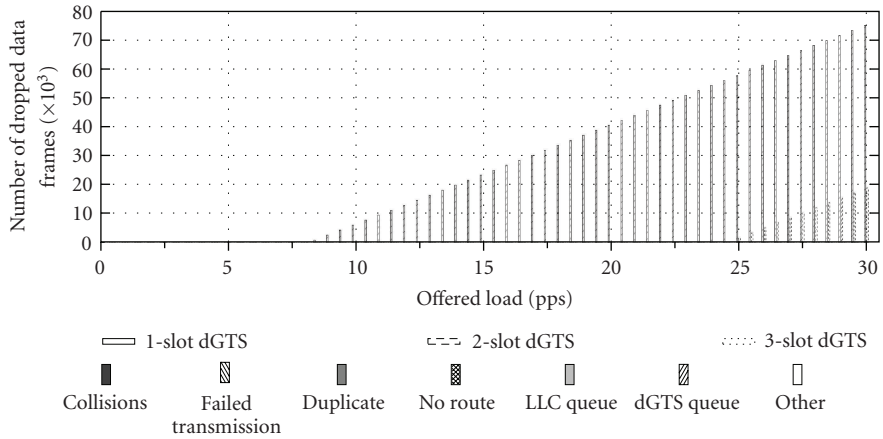

Figure 20: Dropped data frames, parallel scenario, manual routing.

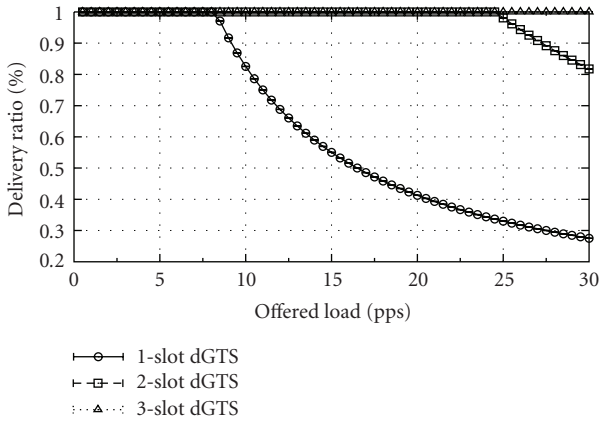

Figure 21: Delivery ratio, sink scenario, manual routing.

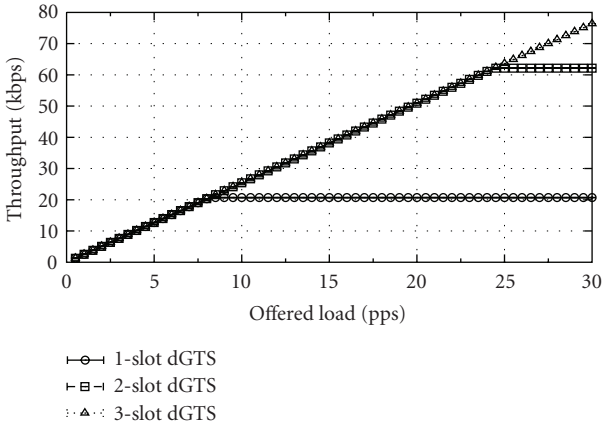

Figure 22: Throughput, sink scenario, manual routing.

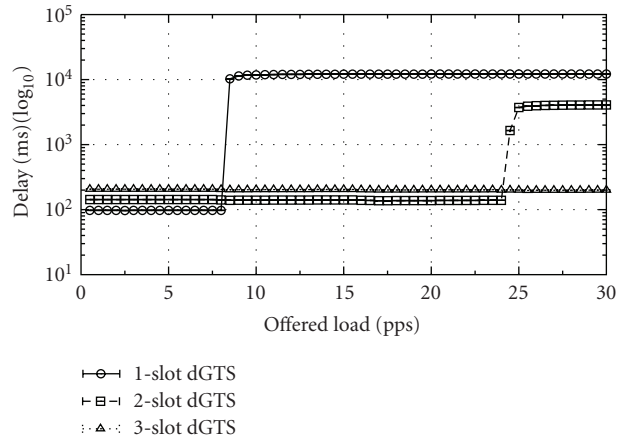

Figure 23: End-to-end delay, sink scenario, manual routing.

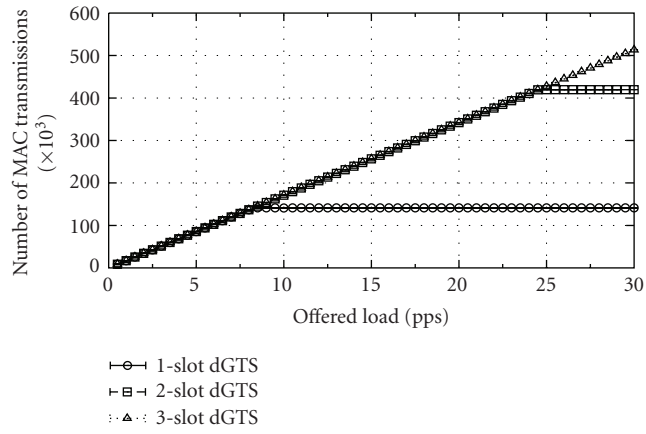

Figure 24: MAC transmissions, sink scenario, manual routing.

the required code to distinguish wpan from IEEE 802.11 packet types.

While observing the network behavior in the simulations, the number of packets being dropped by ARP was rather not negligible. In ns-2, the ARP protocol is implemented in BSD style, where only a single packet may be buffered for each destination hardware address until the address is resolved. If an additional packet for the same destination is received by the ARP module, the buffered packet is dropped and replaced with the recent

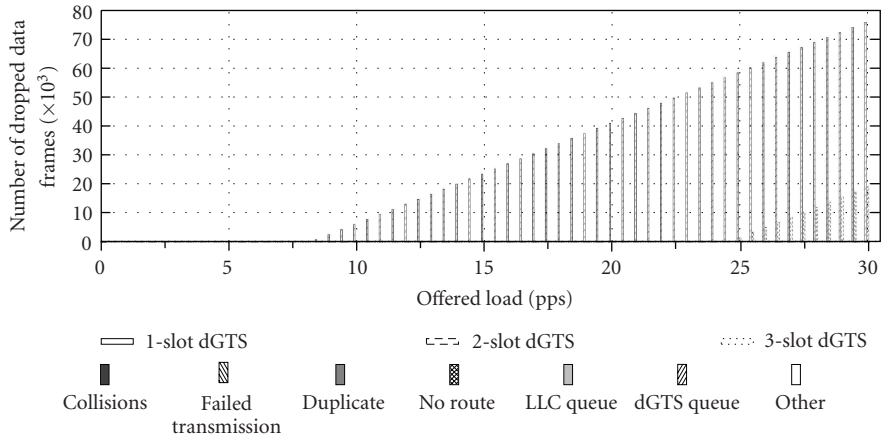

Figure 25: Dropped data frames, sink scenario, manual routing.

one [18]. Therefore, following the wpan module authors' recommendation [4], we disabled the ARP protocol for performance evaluation purposes.

5.1.2. Protocol implementation

Although the dGTS protocol is built on top of the IEEE 802.15.4 standard, it merely uses a few of its infrastructural features, namely, the superframe structure and the CSMA-CA algorithm. Unlike the standard's nonbeacon-enabled mode, in our proposed synchronized peer-to-peer mode (which is designated for dGTS operation), access is slotted and superframes are used. Synchronized peer-to-peer mode is also unlike the standard's beacon-enabled mode; there is no association and nodes operate independently in a flat topology without assuming special roles. We implement the synchronized peer-to-peer mode with nodes maintaining a beacon-less superframe structure while being part of a peer-to-peer topology. A node transits to this mode by setting the superframe parameters *BO* and *SO* manually at the simulation scenario creation time. The node is not started as a coordinator or a non-coordinator device, as that would initiate association/PAN creation, which are no longer required. Like peer-to-peer nonbeacon-enabled networks, none of the standard's MAC commands are used. Instead, a few dGTS-specific commands are used only to setup dGTSs, if required. If no dGTSs are used, the node uses the slotted access and controlled duty cycle features of superframes, in which case it is considered to be operating in the slotted mode described above. We simulate the dGTS protocol component of our framework only. As for the synchronization component, perfect synchronization is assumed hereafter. The synchronization protocol overhead can be modeled at each node as a constant bit-rate (CBR) traffic flow, but is neglected in the performance evaluation experiments.

The dGTS implementation required designing some components that are not part of the protocol, yet are essential for proper protocol operation. The dGTS table (Section 4.4.3), the dGTS queue (Section 4.4.4), and the retransmission queue (Section 4.4.5) are examples of such components. The dGTS and retransmission queues are simple queues with a custom dequeue operation. In the dGTS queue, the packet to be dequeued is first searched for based on the destination address. The retransmission queue distinguishes queued packet types and may be requested to dequeue a packet of a particular type to avoid transaction overflow (i.e., conducting two dGTS allocations simultaneously). The dGTS table, however, is a more interesting structure, especially the neighbor table.

The own dGTS table is implemented as a number of arrays with *aNumSuperframeSlots* elements, where each element corresponds to a superframe slot. The arrays are (1) *transmit dGTS* flag: set if the corresponding slot is part of an own transmit dGTS; (2) *receive dGTS* flag: set if the corresponding slot is part of an own receive dGTS; (3) *dGTS length*: if the corresponding slot is the starting slot of an own dGTS, regardless of its direction, then this entry is set to that dGTS's length in slots, otherwise, it is set to zero; (4) *partner address*: if the corresponding slot is the starting slot of an own dGTS, regardless of its direction, then this entry holds the address of the partner node for the dGTS; (5) *idle superframes*: if the corresponding slot is the starting slot of an own dGTS, regardless of its direction, then this entry holds the number of superframes for which that dGTS has been idle, to enable the detection and expiry of abandoned dGTSs; and (6) *deallocate* flag: set if the corresponding slot is the starting slot of a dGTS awaiting deallocation (i.e., not yet deallocated) to avoid using it.

The neighbor dGTS table holds less data about a dGTS, but is tasked with the additional requirement of keeping track of multiple identical dGTSs and nonidentical dGTSs starting at the same superframe slot. To meet this requirement, this table is implemented as two triangular matrices: one for transmit dGTSs and one for receive dGTSs. Each matrix consists of *aNumSuperframeSlots* -1 rows and an equal number of column. The row index in each matrix represents the starting slot index, while the column index represents the dGTS length. The starting slot index is greater than 0 since the first slot (whose index is equal to 0) is reserved for the CAP. The value stored at row i and column j is the current number of dGTSs starting at slot i whose

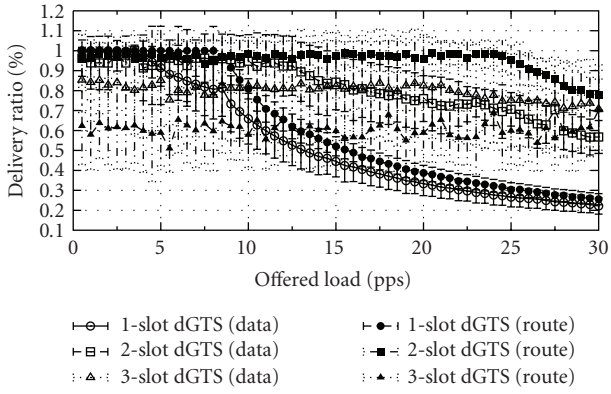

Figure 26: Delivery ratio, parallel scenario, AODV.

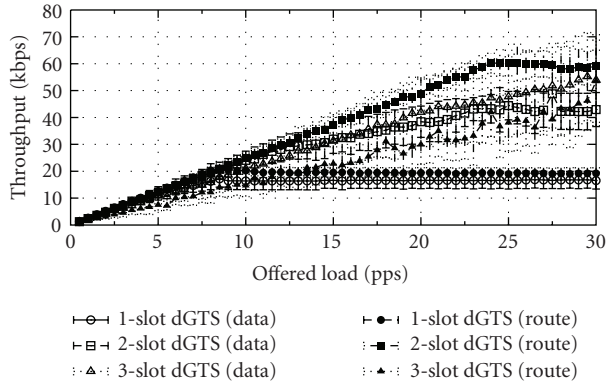

Figure 27: Throughput, parallel scenario, AODV.

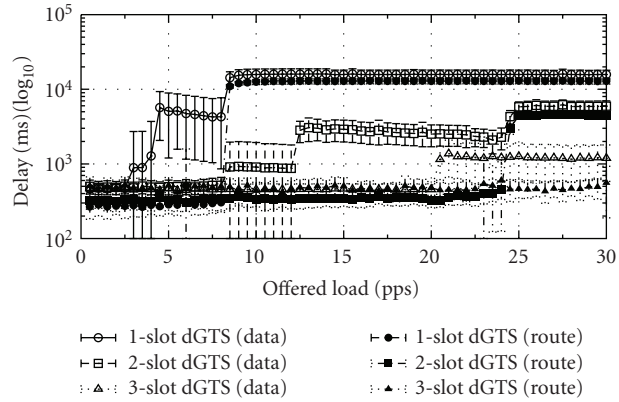

Figure 28: End-to-end delay, parallel scenario, AODV.

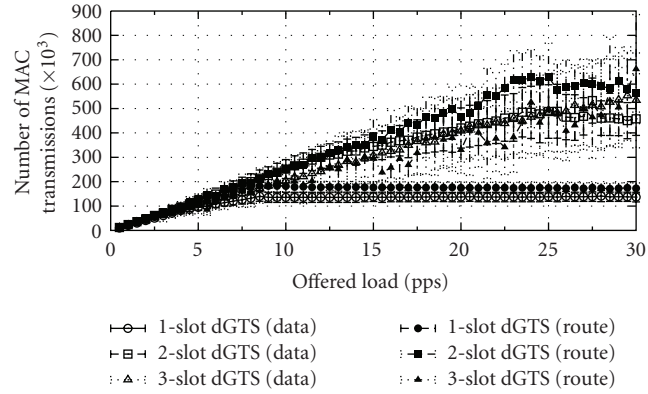

Figure 29: MAC transmissions, parallel scenario, AODV.

length is j slots. Thus, each matrix entry corresponds to the number of multiple identical dGTSs allocated to neighbors, while different entries belonging to the same row correspond to dGTSs starting at the same slot but having different lengths. A dGTS starting at slot 1 can be anywhere from 1 to *aNumSuperframeSlots* -1 slots long, while one starting at slot *aNumSuperframeSlots* -1 can be 1 slot long only. Therefore, each row i in the matrix is a vector of *aNumSuperframeSlots* -1 elements representing all possible lengths of dGTSs starting at slot i. Hence, the two matrices are triangular.

5.1.3. Performance evaluation scenarios, parameters, and metrics

We consider two basic traffic scenarios in the performance evaluation study. As shown in Figure 15, the network consists of 121 sensor nodes distributed in an 11×11 grid spanning a $100 \times 100\,\text{m}^2$ area with 10 meters separating adjacent nodes. The nominal wireless transmission range is set to 12 meters, enabling each node to have at most four neighbors.

The first traffic scenario, referred to as the *sink scenario*, simulates the behavior of wireless sensor networks around the network sink, where all sensory data is eventually collected. As depicted in Figure 15(b), it involves four 5-hop routes ending at the same destination. On the other hand, the second traffic scenario, referred to as the *parallel scenario*, simulates the behavior of wireless sensor networks far from the network sink, where virtually all network traffic flows towards the same direction, in an attempt to get closer to the sink. As illustrated in Figure 15(a), four 5-hop routes are used in the parallel scenario as well.

Radio propagation is modeled using the two-ray propagation model [19, 20]. Nodes use the IEEE 802.15.4 PHY along with our modified MAC and the link layer queue size is set to 50 packets. CBR traffic is generated by the application layer at the source nodes and each packets is equal to 80 bytes. Given its recurrent nature, CBR traffic adequately simulates sensory data. A superframe structure for which $BO = SO = 3$ is used with the dGTS protocol. Also, the dGTS queue size is set to 100 packets and the retransmission queue size is set to 5 packets. Data packets (and all other higher-layer packets) are assumed to require acknowledgment.

In our simulations, we start the traffic generation at the source nodes sequentially (about 3 seconds apart). This is done to minimize the chances of routing packet collisions during route discovery when the first data packets are generated at the traffic sources. Unsuccessful route discovery drastically degrades the performance of the network regardless of the MAC protocol being used.

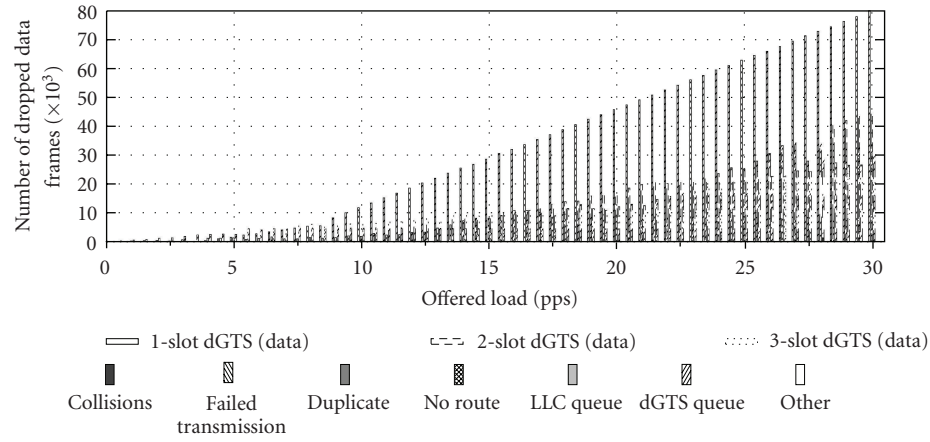

Figure 30: Dropped data frames, parallel scenario, AODV, data-triggered allocation.

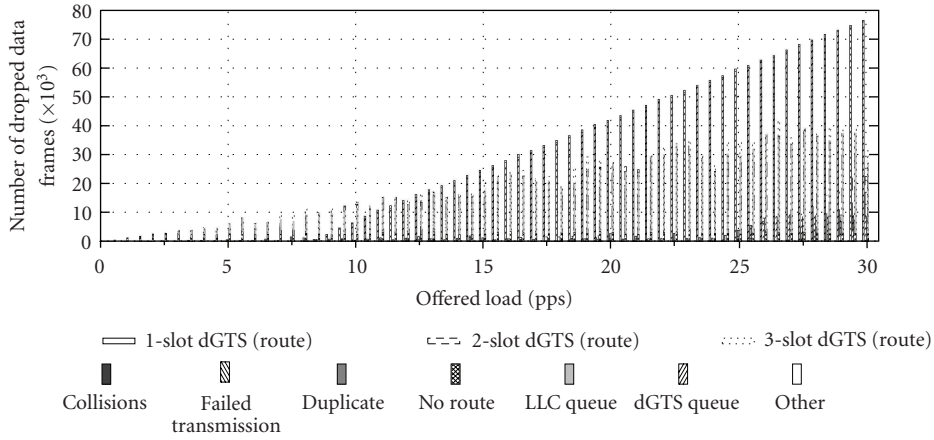

Figure 31: Dropped data frames, parallel scenario, AODV, route-triggered allocation.

The performance of the network is presented in the form of statistical plots. Performance metrics of interest include (1) packet delivery ratio: the percentage of application layer data packets received at the routes' destinations relative to those generated at the routes' sources; (2) aggregate network throughput: the number of kilobits successfully transferred over all routes per second; (3) end-to-end delay: averaged over all routes and all frames successfully received by the routes' destinations, the time it takes a packet in milliseconds to reach the route's destination from the route's source; (4) number of data MAC-level transmissions: summed over all routes, the number of transmissions and retransmissions performed by the MAC sublayer for application layer data packets only; and (5) number of data packets dropped: summed over all routes, the number of application layer data packets dropped by the MAC sublayer. The packet delivery ratio measures the reliability of the network, while the aggregate throughput and the number of dropped data packets measure how efficiently the bandwidth is used see Figure 26. Since transmission accounts for most of the power consumption in IEEE 802.15.4 nodes [21], we use the total number of MAC transmissions (including retransmissions) to measure the energy efficiency of the network. We vary the offered load (i.e., the number of 80-byte packets generated per second by the application layer at the route's source) and plot each of the aforementioned performance metrics accordingly. Each point in the plot is the average of a 20-run sample, which is a fair estimation of the population mean: the true expected value corresponding to the metric. Also, we represent the sample standard deviation by error bars where appropriate, to reflect the stability of the performance metric across runs. The sample mean and standard deviation are computed, as estimators for the corresponding population statistics, according to the formulas

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i, \qquad S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2, \tag{4}$$

where $\overline{X}$ is the sample mean, S is the sample standard deviation (the positive square root of the sample variance S^2), n is the sample size (20 in our setup), and X_i is the ith measurement in the sample.

5.2. Performance with manual routing

To evaluate the dGTS protocol independently of routing, we setup manual routing using the *no ad hoc routing agent*

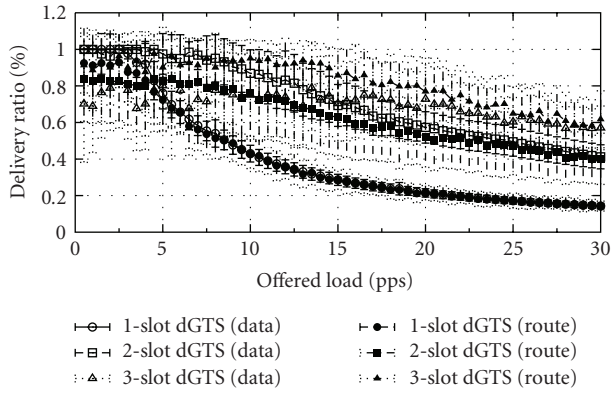

Figure 32: Delivery ratio, sink scenario, AODV.

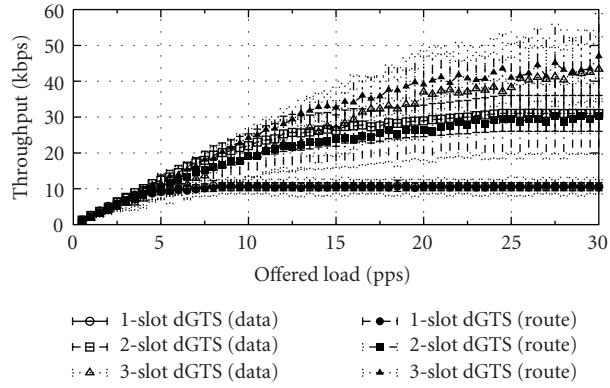

Figure 33: Throughput, sink scenario, AODV.

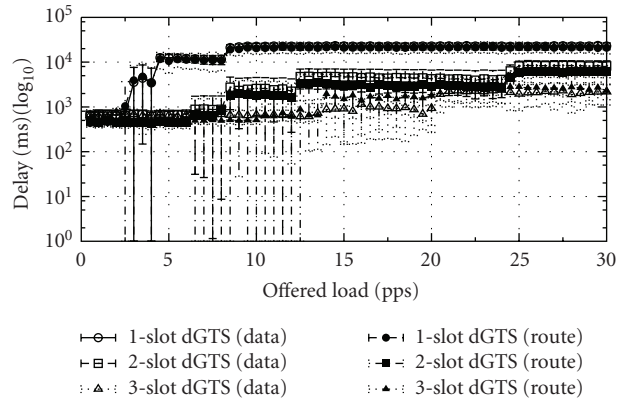

Figure 34: End-to-end delay, sink scenario, AODV.

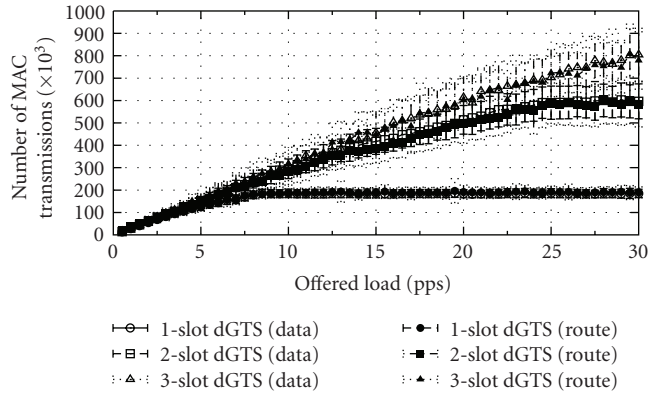

Figure 35: MAC transmissions, sink scenario, AODV.

(NOAH) ns-2 extension, which enables creating manual routes at simulation scenario creation time [22]. Static routes are setup along the shortest path, as shown in Figure 15.

5.2.1. Parallel scenario

Figure 16 shows how the delivery ratio reacts to varying the offered load when manual routing is used along the shortest paths in the parallel scenario.

There is no dropping due to collisions, as expected of a contention-free access scheme. When a single superframe slot is used for the dGTSs, saturation is reached relatively quickly (at around 9 pps) due to the short time available for transmission. However, as shown in Figure 19, there is no increase in the number of transmissions as the load exceeds saturation, since, besides limiting the throughput, saturation limits the transmission attempts in dGTS.

As the number of slots in the dGTS increases, the throughput increases accordingly until saturation is reached. Saturation forces more packets to be stored in the dGTS queue, ultimately resulting in packet dropping (see Figure 20). While a single-slot dGTS reaches saturation at around 9-pps offered load and 20 kbps throughput, a 2-slot dGTS reaches saturation at 25-pps offered load and above 60 kbps throughput: more than three times the single-slot dGTS throughput (see Figure 17). A 3-slot dGTS does not reach saturation for the values of the offered load used in the simulations.

As the number of slots constituting the dGTS increases, the average delay increases provided that the load is below saturation. This is because with a longer dGTS, a packet may wait longer at a node before it is transmitted although it is transmitted in the same superframe in both cases. For example, for two dGTSs starting at the same slot where one is 1 slot long and the other is 2 slots long, a packet received at the beginning of the dGTS may wait for at most 1 slot in the 1-slot dGTS case while it may wait for 2 slots in the 2-slot dGTS case, given that it is to be forwarded within the same superframe. Note that the average delay of a dGTS of a given length can be manipulated by changing *BO* and *SO*, where smaller *BO*/*SO* values yield shorter average end-to-end delay, but less throughput (see Figure 18).

5.2.2. Sink scenario

In contrast to the parallel scenario, the sink scenario involves a bottleneck at the sink node, where all routes end.

Figure 21 shows how the delivery ratio reacts to changes in the offered load when manual routing is used along the shortest paths in the sink scenario. The performance of dGTS in the sink scenario is consistent with its performance in the parallel scenario, demonstrating its inherent congestion

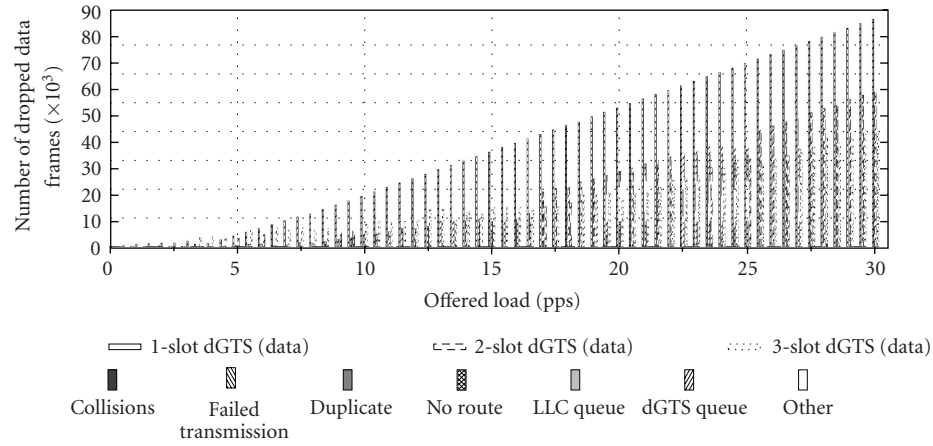

Figure 36: Dropped data frames, sink scenario, AODV, data-triggered allocation.

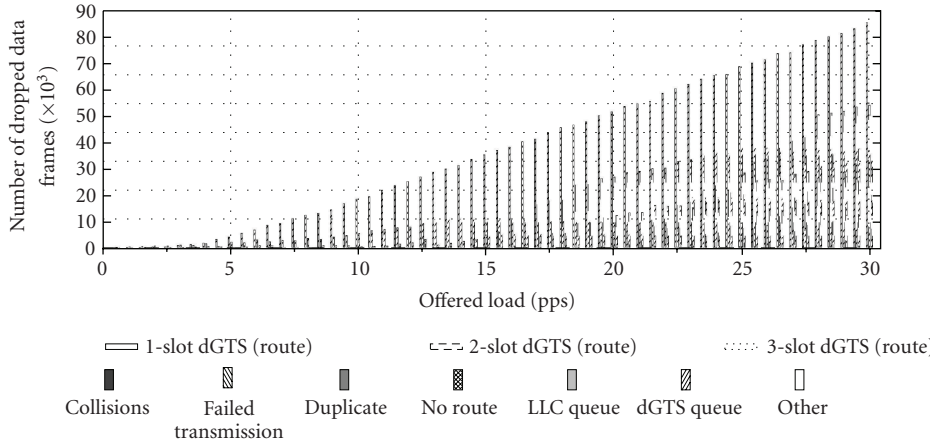

Figure 37: Dropped data frames, sink scenario, AODV, route-triggered allocation.

tolerance features (see Figure 22). Larger dGTS lengths accommodate higher traffic loads with little delay penalty, as demonstrated in Figure 23. As expected, the delay increases drastically when the traffic exceeds the saturation load, depending on the size of the dGTS queue. The larger the dGTS queue, the higher the average end-to-end delay for beyond-saturation offered loads. Furthermore, for beyond-saturation offered loads, the shorter the dGTS, the smaller the average end-to-end delay, since queued packets have shorter periods of time (the dGTSs) during which they may be dequeued and transmitted (see Figures 24 and 25).

5.3. Performance with dynamic routing

In this section, we evaluate the performance of dGTS in conjunction with dynamic routing using the *ad hoc on-demand distance vector* (AODV) protocol [23, 24]. AODV enables nodes to operate as routers. Being a reactive protocol, routes are obtained as needed, or on demand. AODV offers quick adaptation to dynamic link conditions, low processing and memory overheads, and low network overhead.

As explained in Section 4.6, dGTSs are allocated along the discovered routes using forward (data-triggered) allocation or backward (route-triggered) allocation.

Ideally, when using dGTS with a dynamic routing protocol such as AODV, at each node belonging to a set of (one or more) routes as a source or an intermediate node, a number of dGTSs equivalent to the number of routes are assigned. However, as will be shown in Figures 30 and 31, almost all dropped packets are attributed to the dGTS queue being full, which has two causes. First, if a single route traverses an intermediate node (i.e., no dGTS overloading) and saturation is reached, the dGTS queue will become full. Secondly, if multiple routes traverse an intermediate node and dGTS overloading takes place, saturation will result and the dGTS queue will become full. dGTS overloading is the result of using the same dGTS for multiple routes instead of allocating additional ones. This happens whenever there are common nodes in two or more routes. This behavior is difficult to overcome because dGTSs are managed in the MAC sublayer, which is unable to distinguish between flows pertaining to different routes.

5.3.1. Parallel scenario

Figure 27 shows that data-triggered dGTS reaches saturation before route-triggered dGTS. Since data-triggered dGTS employs forward dGTS allocation, nodes process RREP packets faster. The backward allocation policy used in conjunction with route-triggered dGTS means that it takes nodes longer to complete the processing of incoming RREP packets due to awaiting the allocation of suitable dGTSs. Consequently, data-triggered dGTS is more prone to dGTS overloading compared to route-triggered dGTS. Hence, the saturation load and throughput of route-triggered dGTS are almost identical to the saturation load and throughput of dGTS with manual routing.

Expectedly, compared to data-triggered dGTS, route-triggered dGTS achieves shorter end-to-end delays, which is consistent with the analysis presented in Section 4.6 (see Figures 28 and 29).

5.3.2. Sink scenario

This is the most challenging among all the scenarios considered in this study. It involves a bottleneck where four nodes, all hidden from each other, are transmitting to the same destination. Also, dynamic routing further complicates the situation.

The results are shown in Figures 32, 33, 34, 35, 36, and 37. Unlike route-triggered dGTS, data-triggered dGTS achieves 100% delivery under light traffic load. This is due to the fact that route-triggered dGTS is subject to deallocating dGTSs after they have been successfully allocated at the time of route discovery, and it cannot get them reallocated again afterwards. This happens whenever a node receives a conflict command declaring a conflict with a dGTS of its own, forcing the deallocation of the dGTS causing the conflict. Since dGTSs are allocated at the time routes being discovered, and since a route has already been computed, the node cannot allocate a new dGTS. This problem has been also discussed in Section 4.6. Data-triggered dGTS is not subject to this problem because dGTSs are allocated whenever a data packet requires one (that does not exist). The effect of this problem is more visible in this scenario than in the parallel scenario due to the bottleneck at the sink node, where more allocations are likely to cause conflicts. Besides this distinction, data-triggered dGTS and route-triggered dGTS perform almost identically.

The dGTS sharing and overloading problem is visible in this scenario as well. The fact that all four routes have a common destination causes RREQs to be more likely to come across nodes belonging to other routes that are aware of the required destination.

6. CONCLUSION

Peer-to-peer topologies provide the flexibility required for large wireless sensor network deployments. Nevertheless, most of the IEEE 802.15.4 low-power and contention-free features are exclusive to star networks.

In this paper, we show that beacons are not suitable for multihop networks due to the drastic increase in the collision probability in sensor networks with large beaconing coordinator densities. Therefore, we propose a framework to implement global synchronization with high accuracy in a distributed manner. This is achieved by devising a new IEEE 802.15.4 operational mode, namely, the synchronized peer-to-peer mode. In this new mode, all the features of the standard's beacon-enabled mode, such as the superframe structure and slotted access, are preserved without being restricted to the use of beacons and its limitations.

Distributed global synchronization provides the infrastructure to implement and deploy distributed GTS management in peer-to-peer IEEE 802.15.4 networks, which, in turn, provides self-managed contention-free access. Contention-free access in multihop sensor networks can be very useful, especially in applications involving transmitting regular sensory data. Our distributed GTS management scheme enables any two neighboring nodes in the sensor network to allocate an agreed-upon number of interference-free superframe slots (in the form of a dGTS).

Our simulation experiments reveal that dGTS achieves 100% delivery ratios when routes are disjoint. Packet loss in dGTS is mostly attributed to dGTS overloading, which is usually caused by nondisjoint routes. While route-triggered dGTS (backward) allocation minimizes the interhop delay, further improvements are required to recover from broken routes caused by deallocating conflicting dGTSs.

Packet dropping in dGTS does not take place for below-saturation offered loads. For beyond-saturation offered loads, additional transmissions do not take place either; excess traffic is efficiently discarded.

Our future work entails investigating the dGTS integration with various optimizations [14] and routing technologies. Putting the elevated integration and design complexity aside, we anticipate that this will have a favorable impact on the end-to-end network performance.

NOMENCLATURE

MAC:	Medium Access Control
WPAN:	Wireless Personal Area Network
RF:	Radio Frequency
POS:	Personal Operating Space
CSMA-CA:	Carrier Sense Multiple Access
SD:	Superframe Duration
BI:	Beacon Interval
SO:	Superframe Order
BO:	Beacon Order
CAP:	Contention Access Period
CFP:	Contention-Free Period
GTS:	Guaranteed Time Slot
CCA:	Clear Channel Assessment
dGTS:	Distributed GTS management protocol
LLC:	Link Layer Control
ARP:	Address Resolution Protocol
CBR:	Constant Bit Rate
kbps:	Kilo Bit Per Second
AODV:	Ad-hoc On-demand Distance Vector.

ACKNOWLEDGMENTS

This research is led by Dr. Ahmed Safwat, the Principal Investigator. Dr. Ahmed Safwat's research is supported by the Natural Sciences and Engineering Research Council of Canada under Grant no. RGPIN/299023-2004, by the Canada Foundation for Innovation under Project no. 10480, by the Ontario Ministry of Research and Innovation under Project no. 10480, and by Queen's University under Grant no. 383000.

REFERENCES

[1] A. Safwat, H. Hassanein, and H. Mouftah, "Q-GSL: a framework for energy-conserving wireless multi-hop ad hoc networks," in *Proceedings of the IEEE International Conference on Communications (ICC '03)*, vol. 2, pp. 1101–1107, Anchorage, Alaska, USA, May 2003.

[2] IEEE Std 802.15.4-2003, "IEEE standard for information technology—part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs)," October 2003.

[3] J. Zheng and M. J. Lee, "A comprehensive performance study of IEEE 802.15.4," in *Sensor Network Operations*, chapter 4, pp. 218–237, IEEE Press, Wiley Interscience, Piscataway, NJ, USA, 2006.

[4] J. Zheng and M. J. Lee, "Low rate wireless personal area networks (LR-WPANs)—NS2 simulation platform," SAIT-CUNY Joint Lab, http://ees2cy.engr.ccny.cuny.edu/zheng/pub/.

[5] D. Mirza, M. Owrang, and C. Schurgers, "Energy-efficient wakeup scheduling for maximizing lifetime of IEEE 802.15.4 networks," in *Proceedings of the 1st International Conference on Wireless Internet (WICON '05)*, pp. 130–137, Budapest, Hungary, July 2005.

[6] ZigBee Alliance, "ZigBee specification," ZigBee Document 053474r06, Version 1.0, June 2005.

[7] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operaring Systems Review*, vol. 36, SI, pp. 147–163, 2002, in Proceedings of the 5th Symposium on Operating Systems Design and Implementation.

[8] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 138–149, Los Angeles, Calif, USA, November 2003.

[9] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 39–49, Baltimore, Md, USA, November 2004.

[10] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, vol. 36, no. 8, pp. 933–940, 1987.

[11] H. Dai and R. Han, "Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 125–139, 2004.

[12] D. Cox, E. Jovanov, and A. Milenkovic, "Time synchronization for ZigBee networks," in *Proceedings of the 37th Southeastern Symposium on System Theory (SSST '05)*, pp. 135–138, Tuskegee, Ala, USA, March 2005.

[13] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 214–226, 2006.

[14] A. M. Safwat, "On CAC, DCA, and scheduling in TDD multi-hop 4G wireless networks," in *Proceedings of the 23rd IEEE International Performance, Computing and Communications Conference (IPCCC '04)*, pp. 541–546, Phoenix, Ariz, USA, April 2004.

[15] A. Safwat, H. Hassanein, and H. Mouftah, "ECPS and E2LA: new paradigms for energy efficiency in wireless ad hoc and sensor networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '03)*, vol. 6, pp. 3547–3552, San Francisco, Calif, USA, December 2003.

[16] "The Network Simulator—ns-2," http://www.isi.edu/nsnam/ns/.

[17] C. K. Singh, A. Kumar, and P. M. Ameer, "Performance evaluation of an IEEE 802.15.4 sensor network with a star topology," *Wireless Networks*, vol. 14, no. 4, pp. 543–568, 2008.

[18] "The ns Manual (formerly ns Notes and Documentation)," ns-2.29 ed., January 2005.

[19] T. S. Rappaport, *Wireless Communications: Principles and Practice*, Prentice Hall, Upper Saddle River, NJ, USA, 2001.

[20] W. C. Lee, *Mobile Communications Engineering*, McGraw-Hill, Boston, Mass, USA, 1982.

[21] B. Bougard, F. Catthoor, D. C. Daly, A. Chandrakasan, and W. Dehaene, "Energy efficiency of the IEEE 802.15.4 standard in dense wireless microsensor networks: modeling and improvement perspectives," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 196–201, Munich, Germany, March 2005.

[22] J. Widmer, "NO Ad-Hoc Routing Agent (NOAH)," http://icapeople.epfl.ch/widmer/uwb/ns-2/noah/.

[23] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pp. 90–100, New Orleans, La, USA, February 1999.

[24] C. Perkins, E. Belding-Royer, and S. Das, "Ad-hoc on-demand distance vector (AODV) routing," RFC 3561, July 2003, http://www.ietf.org/rfc/rfc3561.txt.