Computer Networks 55 (2011) 2558–2575

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Self-adaptive framelet-based communication for wireless sensor networks

Tony O'Donovan [a,*], Utz Roedig [b], Jonathan Benson [a], Cormac J. Sreenan [a]

[a] MISL, Department of Computer Science, University College Cork, Ireland
[b] School of Computing and Communications, Lancaster University, United Kingdom

ARTICLE INFO

Article history:
Received 10 March 2010
Received in revised form 19 April 2011
Accepted 28 April 2011
Available online 8 May 2011
Responsible Editor: S. Oktug

Keywords:
Medium access control
Wireless sensor network
Prioritisation
Aggregation
Adaptive duty cycle

ABSTRACT

Wireless sensor nodes employ a duty cycle to conserve energy. To implement a duty cycle, a sensor node constantly switches the communication transceiver between listen and sleep states. If a listen/sleep cycle of the receiver is known, a sender can transmit a trail of identical packets, called framelets, of which the receiver is guaranteed to receive one. Such framelet-based communication mechanisms are currently used in sensor networks. However, the framelet communication mechanisms that are currently used are static and unable to adapt to changing traffic requirements or traffic bursts. In this paper, we present three new framelet communication enhancements that can be used to overcome this limitation and allow us to construct a self-adaptive framelet-based communication protocol. Our framelet mechanisms are evaluated using testbed and simulation experiments. The results show that our self-adaptive communication protocol is able to accommodate varying traffic patterns with low energy cost.

1. Introduction

A sensor node spends most of its scarce battery power on operating the transceiver (radio) chip. The employed radios generally have the following four modes of operation: listen, receive, transmit, and sleep. Energy efficiency is maximised by spending as much time as possible in sleep mode, only drawing current on the order of μA, as opposed to mA in the other modes. However, since communication between two nodes cannot occur if the radio of either node is in sleep mode, a mechanism is required to ensure the transmit operation of the sender and listen operation of the receiver coincide. This coordination of send and listen activities is known as *transmitter–receiver rendezvous*. Different strategies can be used to achieve transmitter–receiver rendezvous, such as the use of wake-up radios, shared time base, application-layer knowledge, or duty cycles. This paper focuses on transmitter–receiver rendezvous techniques using duty cycles, taking advantage of recent advances in transceiver technology.

To implement a duty cycle, a sensor node continually switches its radio transceiver between listen and sleep states. If the listen/sleep cycle of the receiver is fixed and known then a transmission can be simply extended such that an overlap between the transmission and the listen phase is guaranteed to occur. Traditionally, this was achieved using a long preamble immediately followed by the packet payload. A receiver hearing the preamble would keep its radio in a listen state until the packet payload was received. This duty cycle concept was implemented in B-MAC [1] and relies on transceivers such as the TR1000 [2] that can be used to control individual bit transmissions. Modern radios such as the nRF2401 [3] and the commonly used CC2420 [4] provide a number of additional enhancements such as automatic packet header processing and CRC computation and are normally used to send complete packets rather than individual bits. With these transceivers, it is not possible to use a long preamble to achieve transmitter–receiver rendezvous, and a slightly modified approach is required. One solution is to transmit a series of identical packets, which we call *framelets*, in such a way that the receiver is guaranteed to catch at least one. Despite the transmission overhead, the scheme is as en-

* Corresponding author.
E-mail address: t.odonovan@cs.ucc.ie (T. O'Donovan).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.04.016

ergy- and bandwidth-efficient as the classical long preamble approach because the new transceivers provide much higher data rates, while consuming roughly the same amount of energy. This duty cycle concept was proposed by Barroso et al. [5], and Mahlknecht and Bock [6] and is also included in the TinyOS 2.0.2 distribution [7], where it is known as the low power listening (LPL) module.

The outlined framelet communication mechanism is successfully used in many WSN deployments. However, this basic mechanism is static and unable to adapt to fluctuating traffic requirements or traffic patterns. For example, the basic framelet communication mechanism can neither handle high priority packets, nor deal with traffic bursts. This paper shows how this inflexibility can be overcome and how it is possible to construct a *self-adaptive framelet-based medium access control protocol*. The paper describes three new communication enhancements that introduce self-adaptive behaviour to a framelet-based medium access control protocol. The first feature is named priority interrupt and is used to implement priority level dependent message forwarding in the network. The second is named *opportunistic aggregation* and allows us to deal with temporarily increased traffic loads. The third feature is called *adaptive duty cycle* and is another option for handling traffic bursts. The aforementioned three enhancements can be used individually or in combination to achieve the required MAC layer flexibility demanded by the target WSN application.

The self-adaptive framelet-based medium access control protocol presented in this paper is called *FrameComm*. Some initial research results on specific FrameComm aspects were previously presented in [8] (opportunistic aggregation) and [9] (priority interrupt). This paper describes the three FrameComm enhancements and their combination in detail and gives a comprehensive evaluation. The specific contributions of this paper are the following:

(1) New additional features such as adaptive duty cycles are presented.
(2) All FrameComm features and enhancements are employed in a single communication protocol.
(3) The FrameComm protocol is evaluated comprehensively using both simulation and test bed experiments.

The FrameComm protocol presented in this paper was designed to be used in real-world sensor network deployments. Its design was influenced by our experience with a number of such deployments [10], and so in making key design decisions, we opted for choices that were pragmatic and relatively simple. It has to be noted that FrameComm needs to be implementable on small embedded devices with limited capabilities. For example, the number of protocol states is limited and complex random number generators are not available. Additionally, the protocol design should not be too complex as it must be possible to debug the protocol in a real-world deployment.

Throughout this paper, the FrameComm protocol is described. In some cases, one or more of the optional enhancements will be enabled. While it is always the same protocol being discussed, different names will be used to denote which enhancements, if any, are enabled. See Table 1 for details.

Table 1
Framelet nomenclature.

	Priority interrupts	Opportunistic aggregation	Adaptive duty cycle
FrameComm	Off	Off	Off
FrameCommPI	On	Off	Off
FrameCommOA	Off	On	Off
FrameCommAD	Off	Off	On
FrameComm+	On	On	On

The next section describes the basic framelet communication mechanism. Section 3 details the priority interrupt, opportunistic aggregation and adaptive duty cycle enhancements, which constitute the first contribution of the paper. Sections 4 and 5 describe evaluations performed on a small laboratory test bed and on a simulator, respectively. Section 6 gives an overview of related work and Section 7 concludes the paper.

2. The framelet communications mechanism

The framelet communications scheme is a lightweight, energy-efficient MAC protocol. By employing a duty cycle, the radio transceiver spends much of the time in sleep mode, providing a significant saving in energy. Transmitter–receiver rendezvous is achieved by sending the data repeatedly in a manner that ensures at least one of the messages sent coincides with the listen period of the destination node.

2.1. Assumptions and definitions

It is assumed that the clocks of the transmitter and receiver operate at approximately the same rate. Note that this does not imply time or sleep cycle synchronisation, rather the clock drift between any two nodes is insignificant over a short period. It is also assumed that a fixed rate radio duty cycle is used, i.e. each node periodically activates its radio for a fixed time interval to monitor activity in the channel. The duty cycle period is represented as $P = \Delta + \Delta_0$, where Δ is the time the radio remains active and Δ_0 is the time the radio is in sleep mode. The duty cycle ratio, or duty cycle for short, is defined as:

$$\text{Duty Cycle} = \frac{\Delta}{P} = \frac{\Delta}{\Delta + \Delta_0}. \quad (1)$$

2.2. Rendezvous using framelets

In general, framelets can be described as small, fixed-sized frames that can be transmitted at relatively high speeds. Certain types of ultra low-power transceivers, such as Chipcon's CC2420, are able to transmit small frames at speeds of up to 250 kbps. Framelets are defined as having a fixed or limited size. For example, the maximum size of a CC2420 packet is defined at compile time (despite having a 128-byte buffer, the default for TinyOS is a 28-byte

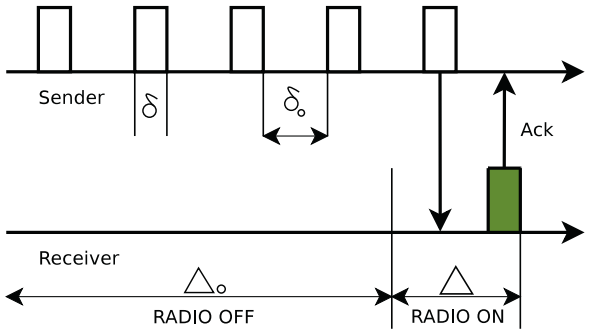


Fig. 1. Framelet rendezvous.

payload plus the message header) and cannot be exceeded during run time without fragmentation. Rendezvous requires the repeated transmission of several frames containing the entire payload as depicted in Fig. 1. If the receiver captures one of these, the payload is delivered. The trail of framelets is defined by three parameters:

- Number of transmissions: n.
- Time between framelets: δ_0.
- Framelet transmission time: δ.

In order to ensure rendezvous, a proper relationship between the parameters Δ, Δ_0, n, δ, and δ_0 must be obeyed. First, the listening phase of the duty cycle Δ must be such that:

$$\Delta \geqslant 2 \cdot \delta + \delta_0. \tag{2}$$

This ensures that at least one full framelet will be intercepted during a listen phase. Furthermore, to ensure overlap between transmission and listening activities, the number of retransmissions n needs to comply with the following inequality when $\Delta_0 > 0$:

$$n \geqslant \left\lceil \frac{\Delta_0 + 2 \cdot \delta + \delta_0}{\delta + \delta_0} \right\rceil. \tag{3}$$

This ensures that a framelet trail is sufficiently long to guarantee rendezvous with the listening phase of the receiver and ensures that at least one framelet can be correctly received.

In general, the values of δ and δ_0 should be as small as possible, as this influences (according to Eq. 2) the smallest possible active time, Δ, of the duty cycle. The duration of the time Δ determines message delay, throughput, and energy savings.

2.3. Acknowledgments

An acknowledgment mechanism can be added to the framelet communication scheme. The radio switches to a listen state in the transmission gaps, δ_0. Once the destination has received one of the framelets, it sends an acknowledgment (ACK) message in response. After reception of this acknowledgment, the sender ceases transmission of its framelet trail. Thus, using acknowledgments, most transmissions will not require the full n framelets. As a result, a transmission will occupy the channel for a shorter period of time, this provides further energy savings while reducing contention and increasing throughput. To accommodate acknowledgments, δ_0 must be long enough to cater for transceiver switching times as well as transmission and receipt of the acknowledgment message. An *acknowledgment request* flag in the message header is used to indicate whether or not the sender expects an acknowledgment. This optimisation is only intended for unicast messages; broadcast messages should send all n framelets.

2.4. Backoff

Another problem, aside from achieving rendezvous, is how to determine the correct backoff strategy and backoff times. We have developed a backoff strategy that takes into account the fundamentals of framelet-based transmission and leverages overheard data to influence backoff times. Unicast transmissions requesting acknowledgment receive a shorter backoff because it is likely that an acknowledgment will arrive and cut short the framelet trail possibly leaving the channel free thereafter. Messages that do not request an acknowledgment, like broadcast messages, are longer, since the framelet trail must run to completion before releasing the channel. The scheme is described in detail in [8].

2.5. Fragmentation

One of the requirements for this framelet communication scheme is a maximum limit on the packet size, δ. Various factors such as the operating system and the specification of the radio transceiver of the node may have an impact on the value chosen for δ. If the size of the data a node has to send exceeds δ, then it becomes necessary to split the data into several fragments that are smaller than the limit imposed by δ and send each piece in a separate message. This can be done by including a fragmentation layer, see [9] for a description of the fragmentation layer.

2.6. Implementation details

FrameComm was implemented in TinyOS 2.0.2 and programmed into Tmote Sky [11] nodes, which were used in a small-scale deployment. Like *low power listening* (LPL), the de facto standard power saving communication protocol in the TinyOS 2.0.2 release and the wireless sensor network community, FrameComm uses framelet trails to ensure rendezvous. The listening cycle length of both FrameComm and LPL is dictated by the duty cycle and the times δ and δ_0.

The key differences between FrameComm and LPL are the following:

(1) To check for incoming messages LPL performs repeated clear channel assessments (CCA), whereas FrameComm performs a full listen of length, Δ, which should guarantee interception of any packets in transmission.

(2) Before beginning transmission of a message FrameComm uses a full listen to detect channel activity; LPL merely uses a brief CCA before beginning its transmissions. A CCA is not sufficient and can lead to problems when there is contention for the channel.
(3) FrameComm employs a full listen to check for ongoing transmissions and exploits the overheard information to influence its backoff strategy. LPL's backoff is unrelated to the duty cycle and results in numerous backoffs, which are insufficient to remove unnecessary contention for the channel.

3. Framelet enhancements

The framelet scheme is a reliable, lightweight contention-based protocol that has been shown to be energy efficient [5]. This section describes optional enhancements to the scheme such as aggregation, prioritised message handling, and self-adaptivity to different traffic requirements and patterns. These enhancements can be employed individually or in combination according to the requirements of the application.

3.1. Priority interrupts

An ongoing framelet transmission can be interrupted by any node in communication range by sending a short message similar to the acknowledgment described in Section 2.3. This feature may be used by nodes that have to send high priority data and need to access the channel immediately.

3.1.1. Interrupt concept

If a node has data to send, it will attempt to access the channel. To do this, it must first perform a carrier sense of duration Δ to ensure no other node is transmitting. If, during this listen phase, it receives a single framelet sent by a current transmitter, it examines the header of this framelet. If this header indicates a priority equal to or higher than the message in the transmit buffer, then the usual backoff occurs. On the other hand, should the priority of the received framelet be lower than the message awaiting transmission, then a priority interrupt will be performed. The node sends an interrupt packet to the current transmitter. The current transmitter receives this interrupt packet as it would be waiting for an acknowledgment from the destined receiver of the framelet transmission. If the current transmitter correctly receives this interrupt it will send an interrupt acknowledgment, cease transmissions, and then enter a backoff phase. After this backoff, the interrupted node will attempt to send its message again. Upon receipt of the interrupt acknowledgment, the interrupter will immediately begin transmitting its own framelet trail. If an interrupt acknowledgment is not received, the interrupter will enter a backoff phase.

The mechanism can be seen in Fig. 2. Node *B* has a high-priority message to send and interrupts Node *A*, who is transmitting lower-priority messages.

The basic scheme described here assumes that high priority messages will always take precedence over low priority messages. As with other prioritisation schemes, however, this can lead to low priority messages being delayed or in some cases dropped. Modifications could be made to the scheme to avoid flow starvation and excessive numbers of low priority packets being dropped. For example, a node could increase the priority of its message after it has been interrupted to prevent subsequent interruptions or it could simply choose not to cede access to the channel by not sending an interrupt acknowledgment.

3.1.2. Collision avoidance

Usually, the number of neighbouring nodes accessing a shared radio channel is limited in order to reduce collisions. Sampling the channel before transmission is an effort to eliminate collisions among one-hop neighbours. Note that when interrupts of framelets are used there is an increased chance of collisions among neighbouring nodes seeking to interrupt the current framelet trail. Should two interrupts be sent simultaneously from different nodes (Node *B* and Node *C*, for example), a collision may occur. There is a possibility that Node *A* (the current sender), Node *B*, and Node *C* will all try to send their framelet trail simultaneously. This situation can be avoided by the use of a handshaking mechanism similar to an RTS-CTS method.

The method works as follows. After sending an interrupt message to Node *A*, both nodes will wait and listen for Node *A* to acknowledge the interrupt or to send its next packet. If either node should be successful in its attempt to interrupt Node *A*, then Node *A* will send an acknowledgment to the successful interrupter. Both Node *B* and Node *C* will hear the acknowledgment and know if their bid to interrupt Node *A* was successful or not. The winning node (Node *B*) will assume control of the channel, while the losing node (Node *C*) will back off and retry at a later time to interrupt the current sender. If neither packet is successful due to both attempted interrupts colliding, then Node *A* will continue its framelet trail as normal, and both Node *B* and Node *C* will use a short random backoff before trying to interrupt again.

3.1.3. Arbitration

This priority interrupt mechanism can be used with several levels of priority, allowing multiple interruptions in a single duty cycle period. Any node that has a message to send can interrupt an ongoing lower priority framelet trail from a neighbouring node. A node, Node *B*, that successfully interrupts another, Node *A*, can subsequently be interrupted by a third node, Node *C*, provided that Node

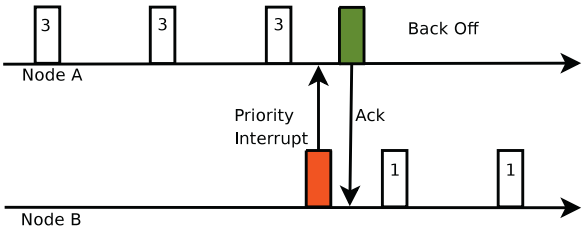


Fig. 2. High-priority interrupts.

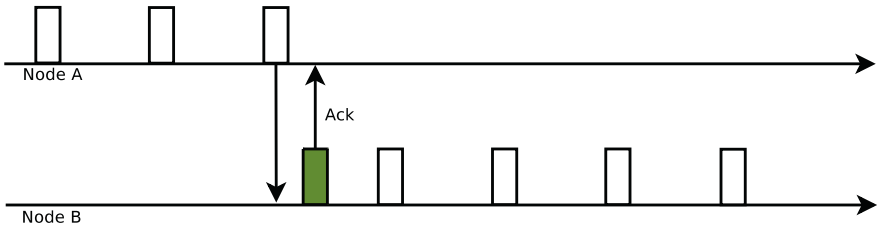


Fig. 3. Gaining the channel after carrier sense.

C's message priority exceeds that of Node *B*. In effect, this provides priority arbitration, ensuring the highest priority message gets access to the channel. A more detailed explanation can be found in [9].

3.1.4. Expected benefits

The process of interrupting another node takes place very quickly and the interrupting node can seamlessly take control of a busy channel to transmit its own high-priority message. Multiple interrupts may take place among a group of nodes (e.g., A interrupts *B*, which is then interrupted by *C*, etc.). Thus, it is ensured that at a given time the highest priority message is very likely to gain access to the channel.

3.1.5. Implementation details

The basic FrameComm implementation was modified to include Priority Interrupts as described in Section 3.1. The TinyOS message header contains a message type field that is used to determine the priority of a message, and a different message type is assigned to each priority. As in the basic FrameComm scheme, a node must perform a listen operation in order to check the channel before it can send a message. In the event of a message being received during this 'pre-send' listen, its priority is obtained from the message type and compared to that of the message waiting to be sent. If the priority is equal to or higher than the message in the transmit buffer, then the usual backoff occurs. On the other hand, should the priority of the received framelet be lower than the message awaiting transmission, a priority interrupt packet is generated and sent to the current transmitter.

Upon receipt of a priority interrupt packet, a sender ceases transmission of its framelet trail and sends an interrupt acknowledgment; this is little more than a message header addressed to the source of the priority interrupt. The interrupted sender then enters a backoff phase similar to a congestion backoff, after which it attempts retransmission of the interrupted message.

Once the higher priority sender receives the acknowledgment of its interrupt, it knows the channel is now available and begins its transmission.

3.2. Opportunistic aggregation

In addition to providing prioritised message handling, the interrupt concept described in Section 3.1.1 can be used to introduce an aggregation scheme to the framelet protocol.

A node (called Node *B*, for example) with a message to send (to Node *C*) must first sample the channel for a fixed duration to ensure that the channel is clear. During this phase, its radio transceiver is on, and messages on the channel can be heard. If the channel is busy, Node *B* will keep its radio active long enough to receive a single framelet of the ongoing transmission.

If Node *B* receives a packet (from Node *A* for example) during this listen it checks the address of that packet. Should the packet be addressed to Node *B*, an acknowledgment will be sent to Node *A* in the normal way and the channel will then be clear for Node *B* to transmit its packet, as seen in Fig. 3. If the received packet is also addressed to Node *C*, the destination of the message it is currently trying to send, then both messages may possibly be aggregated. Node *B* checks to see that aggregation[1] is possible and builds an aggregate packet if it is. An interrupt is then sent to the current sender (Node *A*), offering to include its data in an aggregated packet. Upon receipt of this interrupt, Node *A* can cease its transmissions, and Node *B* will take over the channel and begin to transmit a framelet trail of its own that is addressed to Node *C* and contains the aggregated data of both Nodes *A* and *B*, as in Fig. 4. If aggregation is not possible or the address of the received packet is neither Node *B* nor Node *C*, then the overheard message will be discarded and Node *B* will activate its backoff mechanism.

3.2.1. Expected benefits

In the scenario outlined above, there should be a significant advantage in terms of overall next hop message delivery latency (i.e., if there are many messages to be delivered from different nodes to a common destination, the time taken to deliver all messages should be shorter). Using a fixed duty cycle with no interrupts and aggregation, the time taken to deliver n messages from n peer nodes to a fixed duty cycled receiver will be approximately $(n-1)P$. Recall that $P = \Delta + \Delta_0$. The first message will be delivered somewhere between *0* and *P*, and all remaining messages will be delivered sequentially during the following listening periods and will therefore take an additional $(n-1)P$. Assuming that on average $0.5P$ will expire before rendezvous, the next hop latency may be given by the following approximation:

$$0.5P + (n-1)P.$$

[1] For illustrative purposes, a packet stuffing approach is taken. However, other forms of aggregation are possible and described in [8].

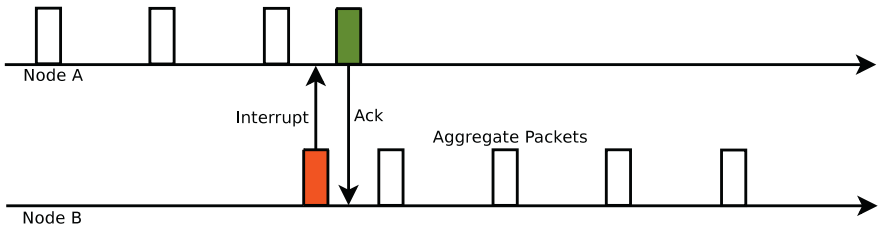


Fig. 4. Opportunistic aggregation.

In comparison, if we assume that m messages, can be aggregated on average during a period Δ, then the total next hop latency reduces to the following approximation:

$$0.5P + \left\lceil \frac{n - 0.5m}{m} P \right\rceil.$$

For example, if we have $n = 8$ messages from different peers and $m = 4$ messages can be aggregated during a time period, the latency is $2.5P$.

This is significantly better than the $7.5P$ it would take under normal conditions. Note that this approach is also more energy efficient for the senders because the time they need to spend with their radios active is collectively less. An alternative way of looking at this is to consider that the individual packet latency reduces to the latency that would be experienced on a contention-free channel (assuming that there is always room to aggregate packets and only peer nodes are contending to send).

The data throughput of the network is also increased significantly. In the example above, eight messages are sent in $2.5P$, as opposed to $7.5P$ under normal circumstances. The use of interrupts and interception allows a variable throughput, despite having a fixed-length listen period. An alternative method for variable throughput that increases the listening time of the receiver to cope with additional data transfer is shown in Section 3.3. In order to modify the listening period, a node must first wake up from its sleep cycle. This means that there is a latency directly related the length of the duty cycle before any extra traffic can be handled. In addition, the increasing of the listening period requires that when a packet is received, the radio must be kept on for some minimum additional time regardless of whether or not there is more data. If bursts of high traffic are rare, then this may present a significant overhead. Our protocol does not need such modifications and is highly reactive to bursty traffic. The advantage of this approach is that a network can be configured to operate in an extremely low-power duty cycle mode, yet deal seamlessly with increases in traffic due to local sensor events.

3.2.2. Implementation details

Similar to FrameCommPI, described in Section 3.1.5, a listen is required to check the channel status prior to any transmission by a node, say, Node *A*. If the channel is found to be clear, then the message is sent; however, if a packet is received during this 'pre-send' listen, a check is performed to see if aggregation is possible. First, the message header of the overheard packet, from Node *B*, is examined to see if both messages have the same destination. If so, the combined payload length is calculated to decide if both messages will fit in a single TinyOS message. TinyOS 2.0.2 allows 28 bytes of data by default. In the event of both senders having a common destination and there being sufficient space in Node *A*'s message to include the overheard message from Node *B*, an interrupt message is generated by Node *A* and sent to Node *B*. When Node *B* receives the interrupt, it first sends an interrupt acknowledgment to Node *A*, it then triggers a sendDone event, indicating its transmission is complete, before finally entering sleep mode.

A packet stuffing approach is used, whereby spare space in the packet payload was filled with data from interrupted packets. Note that additional and alternative forms of aggregation could be implemented and are described in [8]. Information from the CC2420 header regarding the size of the payload is used to quickly ascertain if aggregation is feasible.

In the implementation of FrameCommOA, it is vital that the different types of packets be quickly and easily distinguishable from each other. This is done by examining the CC2420 packet header which, contains a 1-byte type field. Thus, interrupts are quickly distinguished from data packets, acknowledgments, or control messages.

3.3. Adaptive duty cycle

Regardless of the speed capabilities of the radio transceiver, only a single message can be handled by a forwarding node per duty cycle period P. This results in increased data delivery times and reduced throughput in the network. In some situations, such as bursts of heavy traffic, it may be desirable to increase the throughput or reduce the latency for a short period at the cost of some additional listen time at the forwarding node. This can be achieved by temporarily adapting the duty cycle so that more time is spent active.

3.3.1. Traffic-aware dynamic duty cycle

The major source of energy waste in wireless communications protocols is idle listening. This is where a node spends extended periods listening to an empty channel waiting to receive data. Duty-cycled protocols dramatically reduce the amount of idle listening by putting the transceiver in sleep mode most of the time. This works well when the traffic load is low; however, if the network experiences bursts of heavy traffic, the energy saved on the forwarding nodes is offset by the extra time the leaf nodes

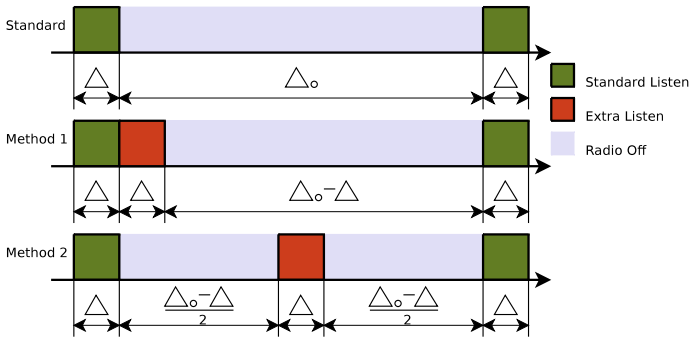


Fig. 5. Adaptive duty cycle.

must spend transmitting. During such periods of bursty traffic, a temporary increase in the amount of listening time at a forwarding node will reduce the power used by the leaf nodes. In addition to energy savings on the leaf nodes, this increase in listening time will result in significantly lower delivery times.

There are two possible approaches to modifying the duty cycle, as shown in Fig. 5. Method 1 extends the node's standard listen time, while Method 2 adds an extra listen halfway through the node's sleep time. Consider the case where Node *A* has a message to send but finds the channel already occupied by Node *B*. Node *A* will back off for a short time to allow Node *B* to complete sending. Node *B*'s transmission will complete when Node *C*, the receiving node, wakes up for a scheduled listen. Having noticed an increase in incoming messages Node *C* modifies its duty cycle to allow more messages through. With Method 1 Node *A* only has the length of the extra listen time, Δ, to return from back off, gain access to the channel and begin transmission if it is to avail of Node *C*'s extra listen time. However, with Method 2 Node *A* has $\frac{\Delta_0-\Delta}{2}+\Delta$ to return from back off, gain access to the channel and begin transmission to benefit from Node *C*'s extra listen time. That is, Method 2 allows more time than Method 1 for a sending node to gain access to the channel and begin sending if it is to be heard during the extra listen time at the receiver. More than 1 extra listen time can be added, for n listens Method 1 gives $(n-1)\Delta$ and Method 2 gives $\frac{\Delta_0-n\Delta}{n+1}+\Delta$ for a sender to avail of the extra listen time.

3.3.2. Expected benefits

During bursts of heavy traffic, a small increase in the listen time of a forwarding node can significantly reduce the delivery delay of a message and the energy required to send it. Consider the situation where Node *A* wishes to send to Node *B*, and Node *B* has a duty cycle period of *P*, with a listen time Δ as in Eq. 1. On average, sending a message from *A* to *B* will take *P*/2. By introducing an extra listen of duration Δ to the sleep interval of Node *B* it incurs a small increase in energy spent. However, this increase is offset by a reduction Node *A*'s average sending time to *P*/4, reducing both the energy spent by Node *A* and the message latency. In a multihop network, these savings will be experienced at each hop.

3.3.3. Implementation details

While the default duty cycle is suitable for standard low data rate network conditions, it is sometimes desirable to temporarily modify the duty cycle to allow more efficient handling of bursts of heavy traffic, as described in Section 3.3. Implementation of this feature has two requirements: first, bursts of heavy traffic must be detectable, and second, the duty cycle itself must be dynamically modified.

Detection of heavy traffic bursts is achieved by recording whether or not consecutive scheduled listen operations result in a message being received. If this is the case, the sleep interval is reduced, giving sending nodes an extra opportunity to have their message delivered. This reduces the amount of energy required to transmit a message on the sender side and allows any backlog of messages that may have built up to be cleared more quickly. Once the traffic rate returns to normal, the forwarding node will be carrying out listens where nothing is heard; when this is observed, the duty cycle reverts back to the default.

Different rates of duty cycle modification are possible; these are evaluated in Section 4.6.

3.4. Summary

The basic framelet scheme can be enhanced with the addition of a priority interrupt, opportunistic aggregation, or adaptive duty cycle feature. These optional enhancements can be used individually or in combination to provide prioritised message handling and traffic aware congestion control. A practical example of how these options can be employed follows.

A physical intrusion detection system is being developed to secure an office building [12]. A number of wireless intrusion detection sensors are distributed in the building and report their observations to a central sink for data analysis. As well as generating alerts the sink is responsible for network control and security key updates.

To ensure timely detection of intruders, it is necessary to transport messages with positive detection events with the highest priority. Heartbeat and network maintenance messages are not time critical. Thus, an efficient network-wide scheduling mechanism is needed.

Part of the solution is a priority scheduler on each node, which is able to correctly schedule messages locally. However, a scheduler alone is not sufficient, as it is not possible

to claim the channel immediately if another node is already in the process of transmitting a low priority message. For example, it is frequently observed that a control message or key update is sent from the sink to all nodes (broadcast, downstream), blocking the channel needed for an event detection to be sent to the sink (unicast, upstream). Including priority interrupts in FrameComm allows us, in such cases, to clear the path for an important message travelling upstream.

When a node detects an intrusion, there is a high probability that neighbouring nodes will also observe the event, generating a sudden burst of traffic. In such situations either the opportunistic aggregation or adaptive duty cycle enhancements (or both) can be used to reduce the latency caused by the heavy traffic load, ensuring messages are delivered in a timely manner.

4. Test bed evaluation

In this section, the behaviour of a small number of nodes is observed in detail. Evaluating the protocols without any obscuring factors (such as network dynamics, background traffic, or queueing effects) allows us to gain a clear insight into their effectiveness. While this approach does not accurately represent the potential "in-the-field" performance of the protocols, it does offer a transparent and fair evaluation.

4.1. Experiment setup

The experiments were carried out using Moteiv Tmote Sky and Tmote Invent sensor nodes running TinyOS 2.0.2. A small tree topology consisting of some leaf nodes, a forwarding node, and a base station (sink) as shown in Fig. 6 was used. The leaf nodes and the forwarding node were duty cycled, while the base station was always on.

4.2. Evaluation metrics

A variety of experiments were carried out to assess the performance of FrameComm and its enhancements. A number of different measurements were required for comprehensive results.

4.2.1. Losses

While energy efficiency, latency, and throughput are significant metrics for communication protocols, particularly in wireless sensor networks, reliability is critical for any protocol that is to be deployed in a real world application. Packet losses were used as an indication of reliability. Any experiment where reliability was a factor of interest involved sending a specific number of packets from each leaf node and logging those received by the sink. Given the number of packets sent and received, the calculation of losses is trivial.

4.2.2. Latency

Another measurement of interest is the node-to-node delivery latency, t, of messages. However, without a common time source or complex synchronisation mechanism it is difficult to measure this latency. With this in mind, we decided to measure the latency locally on each node by comparing the time the node decides to schedule a message for transmission, t_1, and the time the node receives an acknowledgment, t_2, ($t = t_2 - t_1$). More specifically, within the TinyOS implementation, we measure the period between the application calling a *send* and receiving a *sendDone* signal. While there is some overhead associated with a message being sent down the TinyOS communication stack and the *sendDone* being delivered, this time does not vary between experiments or across nodes. The message size is constant throughout and the nodes are not carrying out any other operations or activities that could affect these times. As a result, the only variable is the time taken to access the channel and deliver the message.

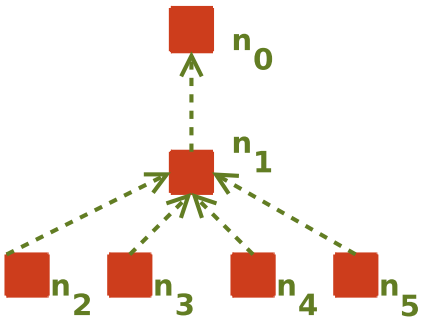


Fig. 6. Test bed topology.

4.2.3. Radio-on time (energy efficiency)

Measuring energy-efficiency or power consumption presents difficulties, especially when only the power consumed by the radio transceiver is of interest. With this in mind, we decided to measure the amount of time the radio transceiver was operational, i.e., not sleeping. Since the energy used by the radio during listen, receive, and send activities is similar, the radio-on time measurement is representative of the energy consumed. The unit for this metric is milliseconds (ms).

4.3. Comparison to TinyOS's LPL

Initially, the basic FrameComm scheme was compared to the default TinyOS 2.0.2 low power listening (LPL) component for benchmarking purposes.

4.3.1. Experiment 1

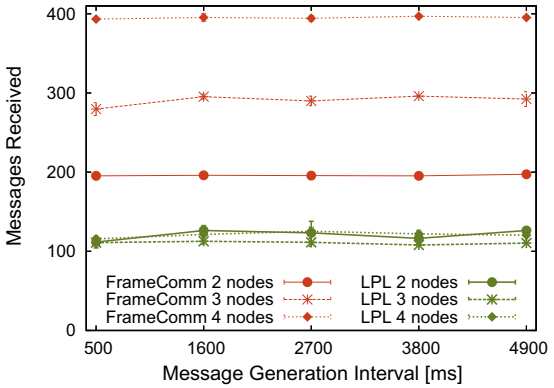
Data was generated at each leaf node periodically at varying rates, namely every 500, 1600, 2700, 3800 and 4900 ms. These message generation intervals were chosen to test the performance of the schemes at different levels of contention, ranging from low (when data is generated every 4900 ms) to in excess of the channel capacity. Each node generated 100 five-byte sensor messages per run. The five-byte sensor message included a 16-bit node ID, an 8-bit sequence number, and a 16-bit data reading.

One striking observation that can be made from the graph in Fig. 7(a) is that the data losses of FrameComm are much less than those of LPL. LPL appears to be only

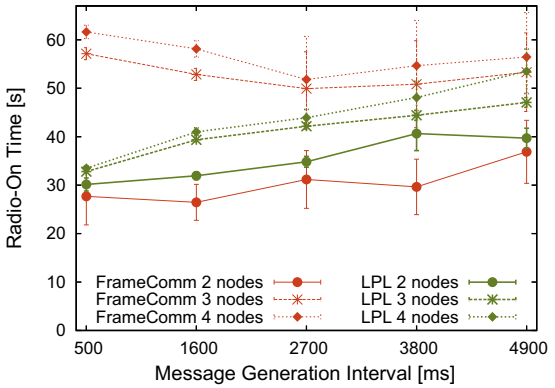
able to deliver approximately 120 sensor messages, irrespective of topology or data generation rate. It must be noted that LPL appears to have significant problems with interleaving and its backoff strategy. The problems with the interleaving of messages, evident from data traces generated during the experiments, seem to stem from the fact that LPL does not perform an adequate listen before sending. LPL merely performs a very brief clear channel assessment (CCA) before sending each packet. If this CCA falls in the gap between successive transmissions from a neighbouring node, as it often does, the busy channel appears free, and interleaving occurs. This interleaving often results in the loss of packets from one or both of the nodes.

Also, it is apparent that FrameComm is handling much more packets than would seem theoretically possible. This is due to the fact that prior to sending, a listen period is used by the intermediate forwarding node, n_1. During this period, another packet may be received. The first packet is quickly sent (since the base station is always on), and the second packet is passed back down to the lower layers for sending. Again, a listening period is used prior to sending, and this process continues until nothing is received during the listen period. Therefore, at n_1, there are more listening periods than the duty cycle would suggest. Note that this phenomenon does not happen all the time, but the likelihood of such occurrences increases with the traffic. Therefore, our implementation of FrameComm seems to be somewhat traffic aware in some conditions, albeit inadvertently, and modifies the amount of listening periods used accordingly.

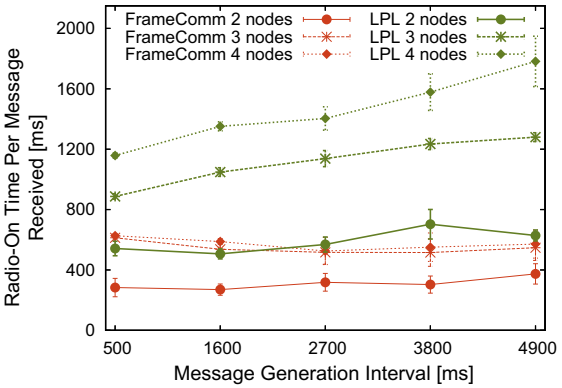
In terms of radio-on time, and thus energy consumption, it appears from the graph in Fig. 7(b) that LPL performs better than FrameComm as the number of leaf nodes increases. Note, however, that LPL's radio-on time is relatively constant. This is primarily because the data losses when using LPL were so high. Indeed, LPL tended to deliver approximately 120 messages, regardless of the number of leaf nodes or the data generation rate. The rest of the data is lost due to a mixture of interleaving and dropped packets. Since so many packets were not sent correctly relative to the other protocols, this is a somewhat unfair comparison. A fairer comparison is shown in Fig. 7(c), where the radio-on time per sensor message received is shown. Here, we can see that FrameComm significantly outperforms LPL.



(a) Data delivered by FrameComm and LPL for 2, 3 and 4 leaf nodes



(b) Radio-On Time for FrameComm and LPL for 2, 3 and 4 leaf nodes



(c) Radio-On Time per Message for FrameComm and LPL for 2, 3 and 4 leaf nodes

Fig. 7. FrameComm vs. LPL.

4.4. FrameCommPI

This experiment examines the behaviour of FrameCommPI in terms of average latency t with various combinations of normal- and high-priority senders. This gives an indication of how FrameCommPI performs in a larger scale multihop network with a mixture of priorities, particularly in upstream forwarding nodes, where data converges approaching the sink. Each run of this experiment had four leaf nodes generating messages. Messages are generated at random intervals, which have an average of 4×500 ms = 2000 ms = 2 s. To begin with, all four leaf nodes sent messages of normal priority. In each subsequent variant of the experiment, the priority of one sender was increased, giving three normal and one high priority, then two normal, two high, and finally one normal and three high. The results of this experiment are shown in Fig. 8. A cumulative distribution for the latency of both normal and high priority traffic is presented for each variant of the experiment.

With a single high-priority sender, an average latency of very close to $P/2$ = 300 ms = 0.3 s is achieved, despite the extra network traffic. This value increases with each additional high-priority sender; however, even with three high-priority senders, the average latency for high-priority messages is still half that of when the priorities of all messages are equal. A more dramatic jump for the latency of low-priority messages can also be seen. This is expected as a larger proportion of high-priority messages leads to more interruptions and more backoffs for messages with a low priority.

4.5. FrameCommOA

The same small tree structure was used again when testing the opportunistic aggregation that is part of the FrameCommOA variation.

In all experiments, 100 messages are generated by each leaf node. A debug packet was sent at the end of each run by every leaf node, which logged various attributes. These attributes included the total number of packets sent, the total radio-on time, the total backoff time, the number of backoffs, and, where applicable, data about aggregation events. Each experiment run was repeated three times for each data point.

We consider three individual experiments. The first experiment examines how many messages are successfully delivered to the base station at varying message generation intervals. We do not use a buffer to hold excess messages. If a message is generated while a node is already sending, the message is simply dropped. The power consumption is also examined here, and we use the time the radio is active as our metric.

The second experiment considers the throughput of the three protocols for a fixed message generation rate. Here,

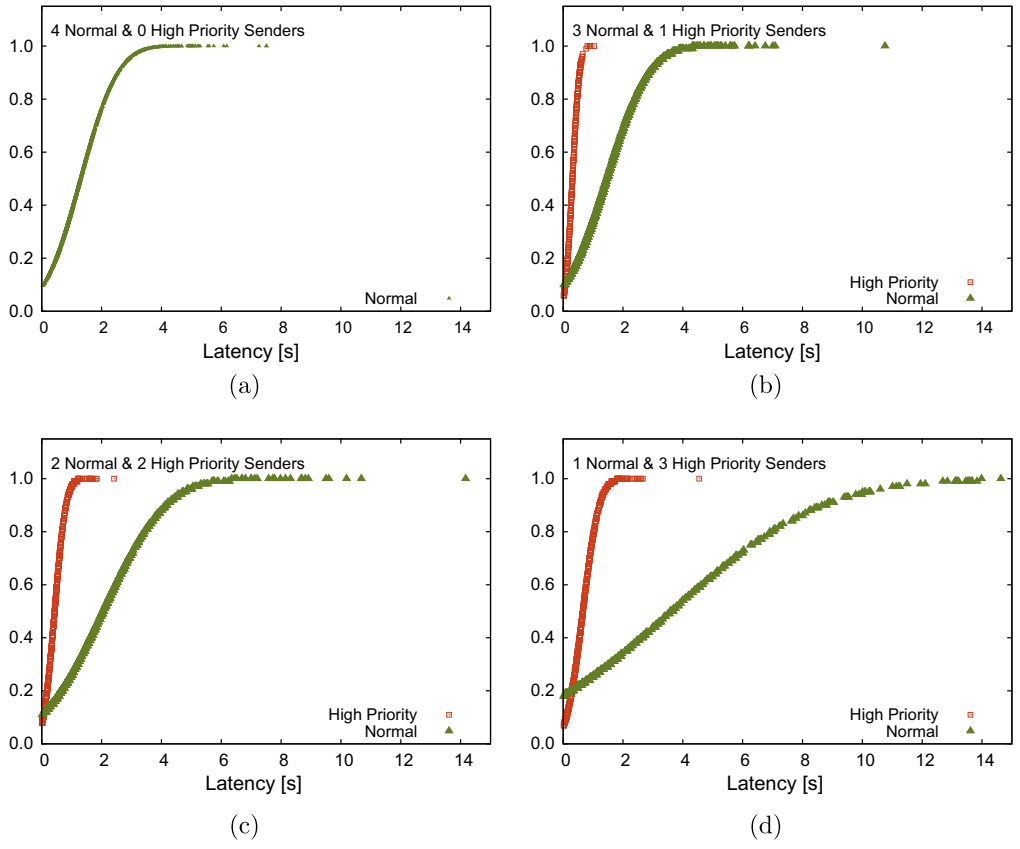
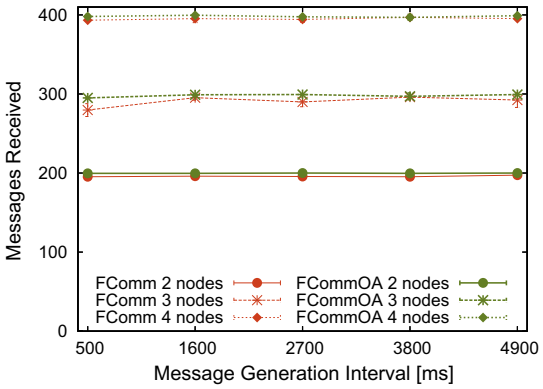
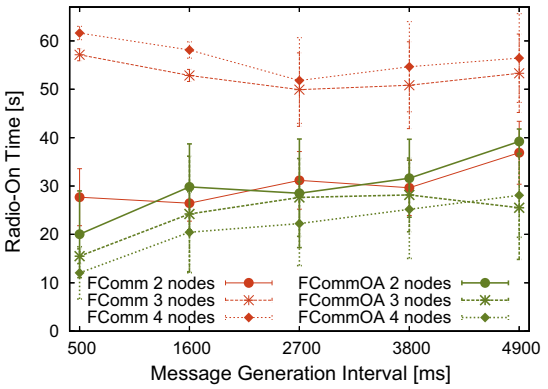


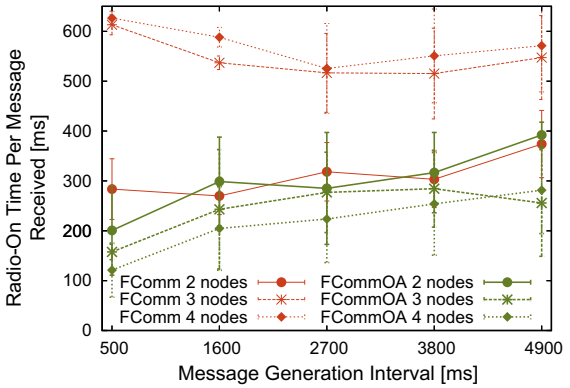
Fig. 8. Cumulative distribution of latency.



(a) Data delivered by FrameComm and FrameCommOA for 2, 3, and 4 leaf nodes



(b) Radio-on time for FrameComm and FrameCommOA for 2, 3, and 4 leaf nodes



(c) Radio-on time per message received for FrameComm and FrameCommOA for 2, 3, and 4 leaf nodes

Fig. 9. FrameComm vs. FrameCommOA.

we use a virtual infinite buffer unlike the previous experiment where there is no buffer. This means that no packets are lost at the node due to queue overflows, but additional time is used to complete the sending of the 100 messages. This provides an accurate and fair way to determine the characteristic throughput of each protocol.

In the final experiment, we adopt a fixed message generation rate and progressively reduce the duty cycle. As in the first experiment, there are no buffers. Data losses are examined with respect to duty cycle for FrameComm and FrameCommOA.

4.5.1. Experiment 1

The setup of this experiment is the same as that of Section 4.3.1; again, each node generated 100 five-byte sensor messages per run. The five-byte sensor message is made up of a 16-bit node ID, an 8-bit sequence number, and a 16-bit data reading. The experiment was run for the same range of message generation rates, i.e., a message generated every 500, 1600, 2700, 3800, and 4900 ms, with 2, 3, and 4 leaf nodes producing data.

As can be seen from Fig. 9(a), the data losses for FrameComm and FrameCommOA are very low. Again, as in Section 4.3.1, the extra listens performed by the forwarding node allows FrameComm to handle more messages than should be theoretically possible. This is one reason FrameComm and FrameCommOA perform quite similarly with respect to packet losses. Another reason is that any losses of an aggregate mean that the equivalent of several sensor messages are lost. This fact tends to skew the results against FrameCommOA somewhat. Examination of the raw data verifies that FrameCommOA has far fewer incidences of loss, but the magnitude of individual losses tend to be greater.

In terms of radio-on time, and thus energy consumption, FrameCommOA appears to be the clear winner and significantly outperforms FrameComm at higher data rates and denser topologies. It is clear that the multiplexing ability of the FrameCommOA approach drastically shortens the transmission times, compared to simply transmitting the data sequentially. Some of the results may be unduly misleading, however. For example, where two neighbouring

nodes are generating data at a high rate, the message will be passed back and forth as each node interrupts the other multiple times, adding a sensor message each time. This scenario would seem to be unlikely in a real deployment.

It can also be seen from Fig. 9(b) that the plots for FrameCommOA have very large error bars. This is due to the fact that the workload on the nodes is uneven. Some nodes will typically aggregate more than others. Thus, some nodes may do a lot less work than their counterparts, leading to the large error bars seen on the graphs.

Note that in Fig. 9(b) and (c), there is no significant benefit to using FrameCommOA over FrameComm at the slower data generation rates with only 2 leaf nodes. However, a significant difference can be seen when the generation rate is at its fastest, and therefore, there are more opportunities to aggregate. Likewise, it can be easily seen that FrameCommOA easily outperforms Framecomm when there are more senders.

4.5.2. Experiment 2

In this experiment, we examine the time taken by the protocols to deliver a fixed amount of messages. As before, the experiment is performed for 2, 3 and 4 leaf nodes and 100 messages are generated and sent. A message is generated at each node every 300 ms. Should the radio be busy, the message attempts to resend again in another 300 ms. The message generation process will continue until 100 messages are sent by each leaf node. The time taken to complete the operation is a good indication of the potential throughput of each of the protocols. Fig. 10 shows the time taken for all nodes to send all 100 messages. It is also clear that FrameCommOA offers significant benefits over FrameComm, sending the required number of messages in a fraction of the time.

4.5.3. Experiment 3

As previously mentioned, a possible benefit of FrameCommOA is that it can support much lower duty cycles, while delivering similar throughput compared to another protocol using a less aggressive duty cycle. Imagine the maximum message generation rate per node is known and bounded. If the topology is known, then it is possible to implement a minimal duty cycle sufficient to handle the maximum amount of possible traffic. In this experiment, we continually introduce a progressively lower duty cycle to a fixed topology using three leaf nodes. Naturally, if a particular duty cycle cannot sustain the volume of traffic generated, then the packets will be dropped. Thus, our success criterion is that a fixed percentage of packets must be successfully received. In this experiment, we adopt a similar topology to the one used in the previous experiments. Every 300 ms, three leaf nodes generate a 4-byte message, thereby allowing a possible 7 messages per payload while aggregating, which is then forwarded to an intermediate node. As before, this node forwards the received messages to a base station. Data is not buffered, as described in Section 4.5.

From Fig. 11, it is obvious that FrameCommOA is able to support a much lower duty cycle, while maintaining a low data loss rate.

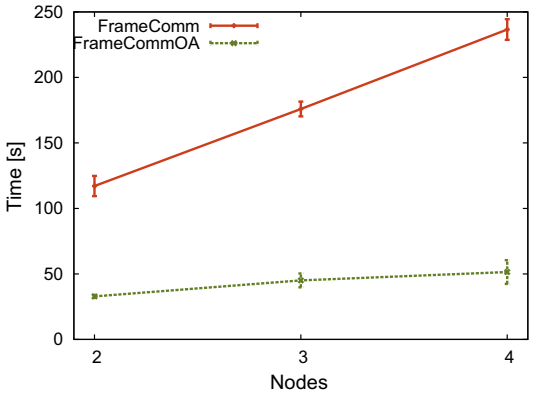


Fig. 10. Time taken to deliver 100 messages per node for 2, 3, and 4 leaf nodes.

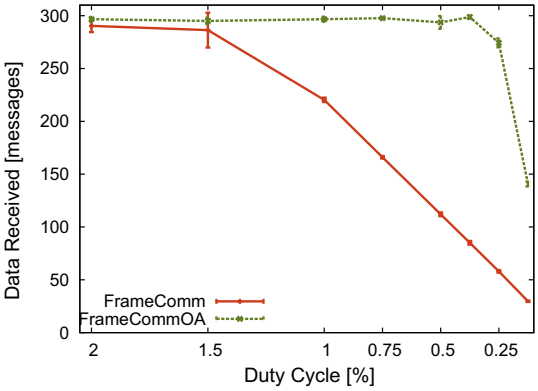


Fig. 11. The effects of reducing the duty cycle on data reception.

4.6. FrameCommAD

Several approaches are possible when choosing the ideal rate at which the duty cycle should be modified when a burst of traffic is detected. With this in mind, the adaptive duty cycle enhancement was evaluated in terms of duty cycle modification rate. This experiment used the same small tree topology shown in Fig. 6, this time with three leaf nodes generating data. To create conditions where the network experienced bursts of heavy traffic, a random message generation rate was used. The leaf nodes generated a message each 2 s on average (the range was ±1 s). The forwarding node, n_1, began with a default duty cycle of 2%; in the event of heavy traffic, this was increased to 10%. The rate at which the duty cycle changed between the upper and lower bounds was varied.

A fast increase and fast/slow decrease were tested. With a fast increase, the duty cycle was changed to the upper and lower bounds immediately; a slow decrease, on the other hand, involves a gradual change, with the duty cycle reducing by half if no messages are heard in the next listen period. In addition to fast increase, fast decrease (FI, FD) and fast increase, slow decrease (FI, SD), the experiment was run with no increase, no decrease (NI, ND). The NI,

Table 2
Adaptive duty cycle experiments.

	Latency (ms)	Radio-on time forwarding node (ms)	Radio-on time leaf nodes (ms)
FI, FD	547.75	9603.00	41010.56
FI, SD	478.40	10124.00	36691.11
NI, ND	645.84	7731.67	47299.56

ND option has a static 2% duty cycle, i.e., the basic FrameComm with no enhancements enabled.

Any changes in latency and radio-on time for the leaf nodes relative to the radio-on time of n_1 was of interest. As can be seen from Table 2 the basic FrameComm option (NI, ND) has the worst latency and radio-on time for leaf nodes, but the best radio-on time for n_1. For a small amount of extra listen time (and energy) at n_1, there is a significant saving in both latency and radio-on time at the leaf nodes.

The best option tested was fast increase, slow decrease (FI, SD) which was expected. This option increased the duty cycle to 10% when a burst was detected, but gradually decreased afterwards. As a result, it was also most costly at n_1.

Losses for the experiment were under 1%. The test was rerun after reducing the range for the message generation interval from ±1 s to ±0.25 s; the findings for these runs were consistent with the first. All options are available in the implementation but, fast increase, slow decrease (FI, SD) is the default as it performs best.

5. Simulation evaluation

The lab experiments described in the previous section have some limitations. First, all nodes in the lab experiment are placed in the same collision domain. Thus, the lab experiments cannot show how the FrameComm protocol performs in dispersed deployments where not all nodes are in communication range of each other. Effects such as the hidden terminal problem cannot be investigated in our limited size deployment. Second, the lab experiments consist of a small number of nodes, and effects only visible in deployments with a large number of nodes cannot be investigated. For example, our lab deployment is limited to two hops and does not allow the investigation of larger multihop networks.

To overcome the aforementioned evaluation limitations, a simulation environment is used to evaluate FrameComm. In addition, the simulation environment allows us to compare the effect of the three different FrameComm enhancements (priority interrupt, opportunistic aggregation, and adaptive duty cycle) within a standardised sensor network deployment scenario.

To carry out the simulation, we implemented our own lightweight simulation tool with a FrameComm stack implementation. We duplicated the lab experiments shown in Section 4 to ensure that the simulation implementation of FrameComm matches the TinyOS implementation of Framecomm used in the lab experiments.

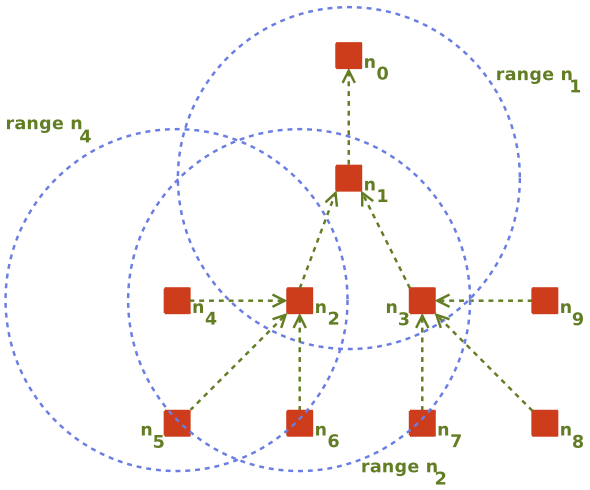


Fig. 12. Simulation topology.

5.1. Simulation evaluation setup

Fig. 12 shows the experimental setup. In the deployment, $n = 10$ nodes are used. Node n_0 is used as the sink. The radio of the sink node is always on. All other nodes use a 2% duty cycle as was used in the lab experiments. All nodes are organised in a tree topology rooted at the sink node n_0. Data is forwarded along the tree towards the sink. neighbouring nodes are in communication range of each other. For example, n_4 can transmit messages to n_2, n_5, n_6 (shown as range *range* n_4 in Fig. 12).

In all experiments, the leaf nodes n_4, n_5, n_6, n_7, n_8, and n_9 are used to generate traffic. Node n_6 is selected to emit packets of high priority which should receive transport priority. The node n_1, n_2, and n_3 are solely used as forwarders. In the experiments, a traffic burst is simulated in which all leaf nodes start to periodically generate packets with an inter-packet interval randomly selected from a uniform distribution between 1 and 3 s, resulting in an average packet generation rate of one packet every 2 s. During this traffic, burst the network is operating at its capacity limit. The aim of the experiments is to evaluate how the three FrameComm communication enhancements impact the selected traffic scenario.

All nodes have a queue size of *3* to buffer messages. Messages are queued according to their priority to ensure that high-priority messages are processed first by FrameComm. The simulation environment models a lossless channel. Framelets can be lost due to collisions; messages can be lost due to buffer overflows.

Within the experiments, the following parameters are investigated:

- Average message delay D_n: the average time for messages travelling from each leaf node n_n to the sink is recorded. Unlike the testbed the simulator allowed end-to-end delay to be measure; this differs from the latency metric described in Section 4.2.2.

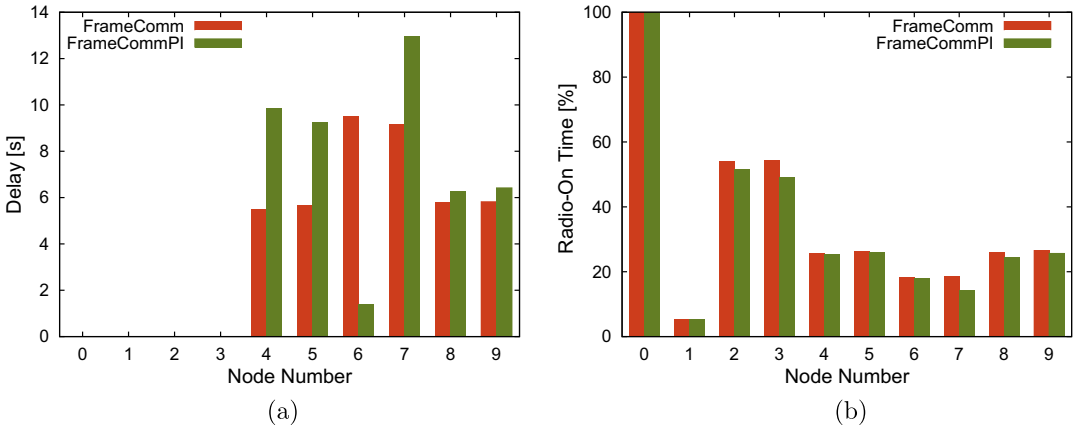


Fig. 13. FrameComm vs. FrameCommPI (high priority interrupts).

- Radio-on time R_n: The percentage of radio-on time for each node n_n is recorded. This is the same metric as in Section 4.2.3, but uses a percentage instead of being measured in milliseconds.

5.2. Simulation experiments

The results for the simulation experiments follow. Each of the enhancements are first plotted separately, then all enhancements enabled together. All plots use FrameComm as a benchmark for comparison purposes.

5.2.1. FrameComm

In the first experiment, the standard FrameComm protocol without any of the three new enhancements is used. The simulation results are shown in Fig. 13(a) and (b). Packets are queued at forwarding nodes, which accumulates to a very high delay.[2] Nodes n_6 and n_7 have a highly increased message delay compared to the other leaf nodes, as they overhear 5 neighbouring nodes. All other leaf nodes overhear messages of only 3 neighbouring nodes. Nodes n_6 and n_7 have a lower data throughput than other leaf nodes (0.34 packets per second as opposed to 0.41 packets per second), as they encounter a busy channel more frequently. An overall message loss rate of $L = 10.2\%$ is recorded. Radio-on times are significantly higher than the 2% that are consumed in an idle network. Forwarding node n_2 and n_3 consume more energy than leaf nodes; n_1 has a low energy consumption as n_0 is always on. As a consequence, framelet trails emitted by node n_1 consist of only one framelet (if no collision occurs), which explains the low energy consumption of node n_1.

5.2.2. FrameCommPI

In this experiment, the high priority interrupt feature is enabled. The simulation results are shown in Fig. 13(a) and (b). Obviously, the network adapts to the fact that messages from node n_6 have to be forwarded with high priority. Other leaf nodes have to pay the price in terms of slightly increased message delays; especially, n_7 has to pay a high price as its packets are interrupted on transit from n_7 to n_3 and from n_3 to n_1. Nodes n_7, n_8, and n_9 experience high packet loss rates. This is due to the fact that forwarding node n_3 is interrupted frequently by high priority messages emitted by n_6 and forwarded by n_2. Thus, n_3 has to frequently drop messages due to queue overflows. An overall message loss rate of $L = 19.9\%$ is recorded. However, node n_6 experiences a lower loss rate of $L = 8.1\%$.

5.2.3. FrameCommOA

In this experiment, the opportunistic aggregation feature is enabled. A maximum of 3 messages can be aggregated into a single packet. The simulation results are shown in Fig. 14(a) and (b). The aggregation feature has several beneficial effects. The data transport delay is significantly reduced, and the energy consumption of each node is reduced as well. Both effects can be attributed to the fact that aggregation reduces the number of packets that need to be forwarded. An overall message loss rate of $L = 13.4\%$ is recorded.

5.2.4. FrameCommAD

In this experiment, the adaptive duty cycle feature is enabled. A minimum duty cycle of 2% and a maximum duty cycle of 16% is allowed. The simulation results are shown in Fig. 15(a) and (b). Compared to the basic FrameComm, the message delay is largely improved. Also, the energy consumption of the nodes is improved, despite the fact that nodes will employ more listen periods (temporary duty cycle of 16%). The radio-on times are reduced as a node has to transmit less framelets, before sender-receiver synchronisation is achieved. Only node n_1 uses more energy with adaptive duty cycles. This is due to the fact that n_1 transmits to n_0, which is always on. Thus, the framelet trails of n_1 are not shortened, but n_1 has to invest more energy for the additional listen periods. Basically, forwarding nodes n_2 and n_3 benefit in terms of energy from the adaptive duty cycle, as saved transmission energy is larger than invested energy for additional listen periods. Forwarding node n_1 has to invest more energy in additional listen periods, but does not save energy for transmissions. Compared

[2] In an empty network, messages can travel, on average, within $D = P = 0.6$ s to the sink.

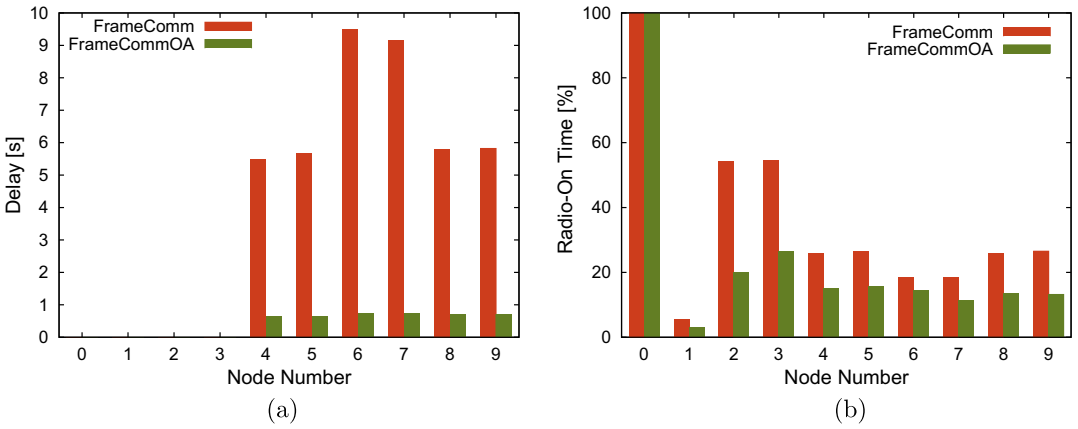


Fig. 14. FrameComm vs. FrameCommOA (opportunistic aggregation).

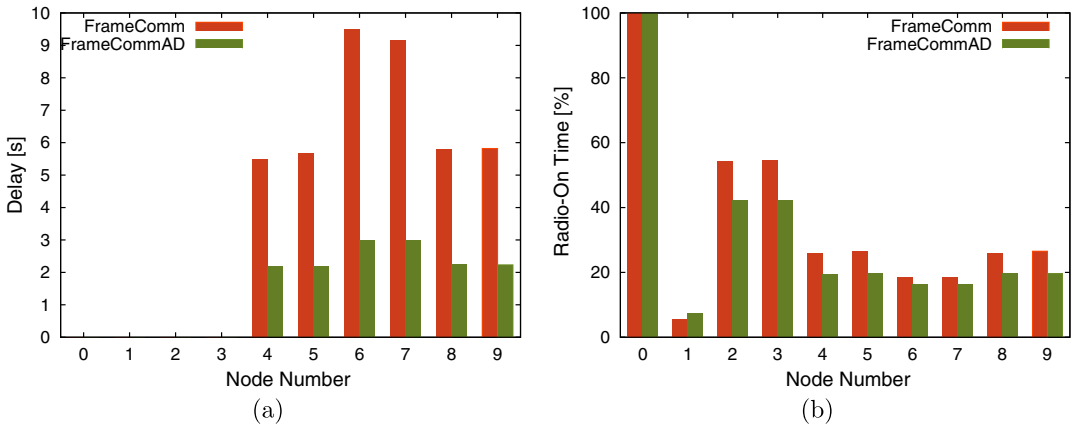


Fig. 15. FrameComm vs. FrameCommAD (adaptive duty cycles).

to the basic FrameComm protocol, an improved overall message loss rate of $L = 1.25\%$ is recorded.

5.2.5. FrameComm+

In this final experiment, all three FrameComm enhancements are employed at once. The simulation results are shown in Fig. 16(a) and (b). It can be seen that the positive effects of adaptive duty cycles and opportunistic aggregation on message delay and transceiver on time are additive. In addition, node n_6 receives the required preferred service. An overall message loss rate of $L = 7.7\%$ is recorded, and node n_6 does not experience any loss.

6. Related work

A great deal of research has been devoted to energy-efficient methods of communication in wireless sensor networks. One particular aspect of this body of research deals specifically with duty-cycled communications and medium access control (MAC) protocols which exhibit duty-cycled or power-saving behaviour. Where duty cycles are used, trade-offs are made typically between energy efficiency and latency or throughput. There are three well-known approaches for duty-cycled communications that we now review: internode synchronisation, out-of-band signalling, and asynchronous sampling. Then, we will discuss published work specifically related to prioritised transmission, aggregation, and adaptive communication.

One approach commonly used to improve the efficiency of sleeping nodes is to synchronise or coordinate the wake up phases of sensor nodes. For example, S-MAC [13] coordinates the sleep cycles for groups of neighbouring nodes by exchanging schedules and synchronisation messages. Like S-MAC, T-MAC [14] operates using synchronisation messages but extends the active listening period when there is additional traffic for a particular node and allows for limited prioritisation of the channel for nodes with full or nearly full sending queues. RMAC [15] uses Pioneer (PION) frames to set up relaying or forwarding nodes to wake up at a particular time so that data can be quickly moved through the network. The PION messages are sent across multiple hops prior to the initiation of data transfer and inform forwarding nodes when they should be awake to handle the arriving data. This has the effect of decreasing the overall latency experienced and helps to remove contention in areas with high traffic loads. Contention is also avoided by transferring data during the SLEEP period. The AWAKE period is simply used to send PIONs to set up

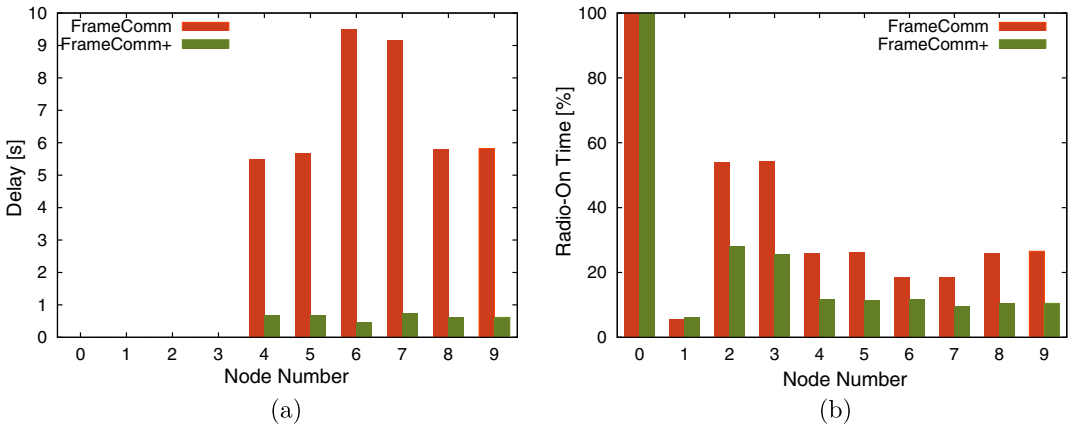


Fig. 16. FrameComm vs. FrameComm+ (high priority interrupts, opportunistic aggregation, and adaptive duty cycles all enabled).

the relaying nodes to remain awake for data transfer while any other nodes return to sleep mode. Synchronised approaches require a sensor network to implement explicit coordination between all nodes, and thus involve additional software and/or hardware, in addition to ongoing message overhead for maintaining close synchronisation. Synchronised approaches are not suitable for distributed wireless sensor networks.

An alternative approach is to use out-of-band signalling such as that described in PAMAS [16] which uses probe packets to determine when and for how long it should deactivate its radio transceiver. Other approaches assume the presence of an ultra-low power wake-on-radio that can be used to activate the main radio when signalled [17]. Note that both of these approaches assume the presence of additional hardware on each node in the sensor network, thus adding to the total node cost and complexity.

It is also possible to implement asynchronised duty cycle behaviour. One of the advantages of this approach is much simpler than synchronous approaches and does not require any additional hardware to operate. The most common scheme is to precede packet transmission with a relatively long preamble that can be used by the receiver to extend its listening duration. An example of an asynchronous protocol is B-MAC, described in [1], which is implemented on TinyOS 1.x. WiseMAC [18] also operates in an asynchronous manner but learns when potential receivers will wake up in order to reduce the amount of preamble transmitted and received before payload data transmission occurs. SCP-MAC [19] also reduces the preamble size; it uses synchronisation to achieve this. The synchronisation can be particularly effective in multihop systems reducing latency considerably. This comes at the cost, however, of keeping schedules of neighbours.

The use of packetised radios requires a fresh approach of implementing asynchronous duty cycles. Some schemes use the same concept of framelet trails as our approach does. The current default energy saving protocol in TinyOS is based on the Low Power Listening component of [1], but uses message retransmission instead of a long preamble in order to accommodate packet-based radios. X-MAC [20] also uses framelets to establish rendezvous between sender and receiver but only retransmits a message header, the payload is sent only after one of the replicas has been acknowledged and the sender knows that the destination is listening. Other related duty-cycled schemes include Koala [21] and CSMA-MPS [6].

Turning specifically to support for prioritised transmission, there are a number of papers that are relevant. The requirements for low-latency, hard real-time applications and the potential for 802.15.4-based radios to satisfy them is examined in [22]. The problem of scheduling access to the wireless medium is the focus of most of the related work, ensuring messages with high priority can gain access to the channel first as in [23]. PR-MAC [24] is among many protocols that use arbitration inter-frame space (AIFS) from IEEE 802.11e [25] to prioritise channel access. Adaptive distributed fair scheduling (ADFS), described in [26], incorporates a weighted queue to schedule messages based on priority; however, the protocol requires synchronisation of nodes to operate. Few works address the problem of interruptions on the shared medium. In particular, to the best of our knowledge, no work addresses the problem of interrupting an ongoing transmission in the context of duty cycled wireless sensor networks. RTQS [27] presents an approach to conflict-free transmission scheduling and provides prioritised communications, however, it requires static priorities. CoBRA [28] is a framework that supports multiple classes of traffic in sensor networks by enforcing rate control using distributed cluster-based mechanisms. Prioritised transmission is provided in TO-MAC [29], a TDMA-based MAC protocol that uses non-destructive bitwise arbitration for message preambles. In a TDMA slot, all nodes with data to send simultaneously start to transmit their unique bit sequences as preambles. The transmission of a 0 is dominant, and as a result, the node with the lowest number transmitted as preamble will win and continue to transmit the data block. Channel access can be prioritised according to the bit numbers used as preamble. An ongoing transmission is not interrupted, instead, all transmitters start simultaneously, and the transmitter with the highest priority will continue to the end while others back off.

Aggregation has been a subject of intensive research, as it offers a significant saving in energy. A common approach

to implementing aggregation is to abstract aggregation from the underlying network operation by implementing a SQL-like query layer, which a programmer or end user can use to pose queries to the sensor network [30–33]. In essence, the sensor network is treated like a distributed database running distributed queries. Other forms of aggregation exist and many are application specific. Nevertheless, a common trait in all aggregation implementations is the need to store data at a point (typically, a parent node) and await the arrival of more data (typically, from the remaining child nodes) to perform aggregation. In [34], how and where best to spend time waiting if each message has a latency bound is discussed. In [32], a cascading system is implemented, whereby each node must wait before transmission so that all its children and sub-children can report. Unlike these approaches, neighbouring nodes can perform aggregation on the fly using our framelet interception method, thus reducing the overall latency necessary to perform aggregation.

Adaptive communication has been proposed as a suitable design methodology for sensor network protocols in many contexts. For example, in ALPL [35] all nodes are deployed with the same default duty cycle. A discovery phase is carried out and routing trees are calculated, after which a node modifies its duty cycle to reflect its role and the number of descendants it has. This information is advertised and nodes record in a table the following details of local neighbours: (1) duty cycle, (2) number of descendants, and (3) role. When a node has a message to send, it uses a cost function based on the neighbour table records to decide on the node that should be used to forward it. Energy efficiency and load balancing are the main motivations behind the scheme. Neugebauer et al. modify IEEE 802.15.4 to include an adaptive duty cycle in [36]. Relay nodes monitor the amount of traffic forwarded and calculate an appropriate duty cycle to handle it. Unlike AFC, where each node calculates its own duty cycle, this scheme requires a coordinator and has communication overheads. AEM is a MAC protocol for tiered sensor networks that use the Tenet architecture [37]. It aims to provide energy-efficient, robust and transparent communication with low delivery delays by employing static analysis to anticipate the application and modifying the duty cycle accordingly. X-MAC [20] attempts to improve energy-efficiency and latency by calculating the duty cycle of the receiver based on the probability of receiving a packet.

7. Conclusion and future work

We have shown that the three FrameComm enhancements presented in the paper can be combined to create an effective traffic-aware, self-adaptive communication protocol. The protocol is able to increase network capacity temporarily at a low energy cost if traffic bursts are detected. In addition, the protocol is able to automatically allocate transport capacity for high priority data messages. Finally, the protocol reduces the overall energy consumption in the network.

Communication protocols which outperform FrameComm in one aspect or another are available for sensor networks. Some protocols might achieve better energy savings or a lower data transport delay. However, many of these protocols are limited to specific application scenarios, network topologies, or traffic scenarios. The FrameComm protocol balances parameters of interest such as message delay and energy consumption well, especially considering the fact that the protocol is not restricted to a specific application scenario and that it can adapt to a variety of traffic scenarios.

In wireless environments, changing channel conditions are observed. Links might encounter increased loss rates at times or become temporarily unavailable. FrameComm needs to be able to accommodate these effects as well. We are currently addressing this aspect by using the presented adaptive duty cycle mechanism. The adaptive duty cycle can be used to deal with traffic bursts, but is potentially useful as well to deal with fluctuating channel quality. If increased loss rates are detected, additional listening periods can be introduced to increase the chance of receiving one framelet of a trail correctly. Duty cycle adaptation to deal with traffic bursts and duty cycle adaptation to deal with temporary decreased channel quality need to be balanced correctly to obtain a useful self-adaptive protocol.

Our FrameComm implementation for TinyOS and the simulation environment are available on request.

Role of the funding source

The sponsors of this work had no role in its design; in the collection, analysis or interpretation of data; in the writing of the report; or in the decision to submit the paper for publication.

Acknowledgments

This work was partially supported by Microsoft Research through its European Ph.D. Scholarship Programme and the EMBARK Initiative of the Irish Research Council for Science, Engineering and Technology.

References

[1] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: International Conference on Embedded Networked Sensor Systems (SenSys), 2004.
[2] TR1000 Hybrid Transceiver. <http://www.rfm.com/products/data/tr1000.pdf> (visited 15.02.2011).
[3] nRF2401 Radio Transceiver.<http://www.nordicsemi.com/index.cfm?obj=product&act=display&pro=64> (visited 15.02.2011).
[4] Chipcon CC2420 Datasheet. <http://www.ti.com/lit/gpn/cc2420> (visited 02.15.2011).
[5] A. Barroso, U. Roedig, C. Sreenan, Use of framelets for efficient transmitter-receiver rendezvous in wireless sensor networks, in: IEEE Conference on Local Computer Networks, 2005.
[6] S. Mahlknecht, M. Bock, CSMA–MPS: a minimum preamble sampling MAC protocol for low power wireless sensor networks, in: IEEE International Workshop on Factory Communication Systems, 2004.
[7] TinyOS 2.0.2. <http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html> (visited 15.02.2011).
[8] J. Benson, T. O'Donovan, U. Roedig, C. Sreenan, Opportunistic aggregation over duty cycled communications in wireless sensor networks, in: ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.
[9] T. O'Donovan, J. Benson, U. Roedig, C. Sreenan, Priority interrupts of duty cycled communications in wireless sensor networks, in:

International Workshop on Practical Issues in Building Sensor Network Applications (SENSEAPP), 2008.
[10] J. Benson, T. O'Donovan, P. O'Sullivan, U. Roedig, C. Sreenan, J. Barton, A. Murphy, B. O'Flynn, Car-park management using wireless sensor networks, in: International Workshop on Practical Issues in Building Sensor Network Applications (SENSEAPP), 2006.
[11] Moteiv Tmote Sky and Tmote Invent Motes. <http://www.sentilla.com/moteiv-transition.html> (visited 15.02.2011).
[12] T. Chung, U. Roedig, DHB-KEY: an efficient key distribution scheme for wireless sensor networks, in: IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2008.
[13] W. Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002.
[14] T. van Dam, K. Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks, in: International Conference on Embedded Networked Sensor Systems (SenSys), 2003.
[15] S. Du, A. Saha, D. Johnson, RMAC: a routing-enhanced duty-cycle MAC protocol for wireless sensor networks, in: IEEE International Conference on Computer Communications (INFOCOM), 2007.
[16] S. Singh, C.S. Raghavendra, PAMAS – power aware multi-access protocol with signalling for ad hoc networks, SIGCOMM Computer Communication Review 28 (1998) 5–26.
[17] E. Shih, P. Bahl, M.J. Sinclair, Wake on wireless: an event driven energy saving strategy for battery operated devices, in: International Conference on Mobile Computing and Networking (MOBICOM), 2002.
[18] A. El-Hoiydi, J.-D. Decotignie, C. Enz, E. Le Roux, Poster abstract: WiseMAC, an ultra low power MAC protocol for the WiseNET wireless sensor network, in: International Conference on Embedded Networked Sensor Systems (SenSys), 2003.
[19] W. Ye, F. Silva, J. Heidemann, Ultra-low duty cycle MAC with scheduled channel polling, in: International Conference on Embedded Networked Sensor Systems (SenSys), 2006.
[20] M. Buettner, G.V. Yee, E. Anderson, R. Han, X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks, in: International Conference on Embedded Networked Sensor Systems (SenSys), 2006.
[21] R. Musaloiu-E, C.-J.M. Liang, A. Terzis, Koala: ultra-low power data retrieval in wireless sensor networks, in: ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.
[22] K.K. Chintalapudi, L. Venkatraman, On the design of MAC protocols for low-latency hard real-time discrete control applications over 802.15.4 hardware, in: ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2008.
[23] P. Ansel, Q. Ni, T. Turletti, An efficient scheduling scheme for IEEE 802.11e, in: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2004.
[24] A. Firoze, L. Ju, L. Kwong, PR-MAC a priority reservation MAC protocol for wireless sensor networks, in: International Conference on Electrical Engineering (ICEE), 2007.
[25] Y. Xiao, Enhanced DCF of IEEE 802.11e to support QoS, in: Wireless Communications and Networking (WCNC), 2003.
[26] J. Fonda, M. Zawodniok, S. Jagannathan, S. Watkins, Adaptive distributed fair scheduling and its implementation in wireless sensor networks, in: IEEE International Conference on Systems, Man and Cybernetics (SMC), 2006.
[27] O. Chipara, C. Lu, G.-C. Roman, Real-time query scheduling for wireless sensor networks, in: IEEE International Conference on Real-Time Systems Symposium (RTSS), 2007.
[28] K. Karenos, V. Kalogeraki, S. Krishnamurthy, A rate control framework for supporting multiple classes of traffic in sensor networks, in: IEEE International Real-Time Systems Symposium (RTSS), 2005.
[29] A. Krohn, M. Beigl, C. Decker, T. Zimmer, TOMAC – real-time message ordering in wireless sensor networks using the MAC layer, in: International Workshop on Networked Sensing Systems (INSS), 2005.
[30] J. Gehrke, S. Madden, Query processing in sensor networks, IEEE Pervasive Computing 3 (2004) 46–55.
[31] P. Bonnet, J. Gehrke, T. Mayr, P. Seshadri, Query Processing in a Device Database System, Technical Report, Cornell University, 1999.
[32] S. Madden, M.J. Franklin, J. Hellerstein, W. Hong, TAG: a tiny aggregation service for ad-hoc sensor networks, in: Symposium on Operating Systems Design and Implementation, 2002.
[33] J. Beaver, M.A. Sharaf, A. Labrinidis, P.K. Chrysanthis, Power-aware in-network query processing for sensor data, in: Second Hellenic Data Management Symposium, 2003.
[34] U. Roedig, A. Barroso, C. Sreenan, Determination of aggregation points in wireless sensor networks, in: Euromicro Conference, 2004.
[35] R. Jurdak, P. Baldi, C. Lopes, Adaptive low power listening for wireless sensor networks, IEEE Transactions on Mobile Computing 6 (2007) 988–1004.
[36] M. Neugebauer, J. Ploennigs, K. Kabitzsch, Duty cycle adaptation with respect to traffic, in: Emerging Technologies and Factory Automation (ETFA), 2005.
[37] O. Gnawali, J. Na, R. Govindan, Application-informed radio duty-cycling in a re-taskable multi-user sensing system, in: ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN), 2009.

Tony O'Donovan is a researcher in the Mobile & Internet Systems Laboratory (MISL) research group in the Computer Science Department, University College Cork (UCC). He received an undergraduate degree in Computer Science in 1998 from UCC. After a period working in software development, he returned to UCC and received an M.Sc. degree in Software Development for Computer Networks in 2005. Upon completion of the M.Sc. degree he joined the MISL group to work on a Ph.D. programme in the area of Wireless Sensor Networks. Currently he is engaged in completing his Ph.D. while also working on the EU FP7 GINSENG Project.

Utz Roedig received a Ph.D. degree in Computer Science from Darmstadt University of Technology, Germany in 2002. Between 2002 and 2006, he was a postdoctoral researcher in the Department of Computer Science at University College Cork, Ireland. He currently works as a Lecturer at the School of Computing and Communications, Lancaster University, UK, which he joined in October 2006. His research work targets Embedded Systems and in particular Wireless Sensor Networks.

Jonathan Benson studied Computer Science as part of a Science Degree at Maynooth College and after a period working as a programmer decided to specialise in this area. He returned to education and completed a Higher Diploma in Software Engineering in University College Cork before becoming engaged in research, primarily in Wireless Sensor Networks. His research interests are in networks, particularly wireless sensor networks and the impact of real world contraints on performance, stability and scalability and practical methods of overcoming these problems. He is currently working in Seagate Technology as an Network Engineer.

Cormac J. Sreenan received the Ph.D. degree in Computer Science from Cambridge University. He is a professor of Computer Science at University College Cork (UCC) in Ireland. Prior to joining UCC in 1999, he was on the research staff at AT&T Labs–Research, Florham Park, New Jersey, and at Bell Labs, Murray Hill, New Jersey. At UCC, he directs the Mobile & Internet Systems Laboratory (MISL), which is a group of more than 10 research staff and students with research activities in Multimedia and Wireless Networking Systems.