Opportunistic Aggregation over Duty Cycled Communications in Wireless Sensor Networks

Jonathan Benson, Tony O'Donovan, Utz Roedig[+], Cormac J. Sreenan.
Mobile and Internet Systems Laboratory, Dept. of Computer Science , University College Cork, Ireland.
[+]InfoLab21, Lancaster University, UK.

***Abstract*—To implement duty cycles with packet based transceivers, a sender transmits a trail of identical packets (which we call framelets) of which the receiver is able to catch one in its active listening phase. This communication concept is used in the standard low power listening (LPL) protocol shipped with TinyOS 2.x. This existing solution has many shortcomings which result in a very limited network performance. In this paper, we firstly present an alternative framelet based low power listening implementation called Framelet Communications (FrameComm) that eliminates these shortcomings. Secondly, we present a novel additional improvement to FrameComm - Interception and Aggregation of Framelet Communications (i-FrameComm) - that further improves network performance by opportunistically aggregating packets over the radio channel. A prototype implementation of the proposed FrameComm mechanism in TinyOS 2.02 is used for evaluation and comparison. The experiments show that the interception and aggregation method increases network throughput and lifetime as communication resources are used more efficiently.**

I. Introduction

A major constraint in the design of wireless sensor networks is the need of autonomous, untethered operation for extended periods of time. The system lifetime is ultimately defined by the energy-efficiency of the design which is especially affected by the way the communication system is operated. Generally, a sensor node transceiver can be set to one of four states: *transmitting*, *receiving*, *listening* or *sleeping*. Energy efficient operation of transceivers is achieved essentially by keeping them in sleeping mode as often as possible. The sleeping state generally consumes orders of magnitude less energy than the active states (transmitting/receiving/listening). However, as communication cannot take place between nodes while the transceivers are in sleeping state, sender and receiver actions must be synchronised for successful transmission of messages to take place.

Different strategies can be used to provide transmitter-receiver synchronisation such as the use of *wake-on radios*, *shared time basis* or *radio duty cycles*. The common element of all these techniques is the need to establish, in an efficient way, an intersection of data transmission and listening activities enabling effective communication between transceiver and receiver. Such an intersection can be achieved at different costs by each technique or a combination of them.

Due to its simplicity, many sensor networks implement a radio duty cycle approach. Thus, additional hardware and complex mechanisms needed to implement wake-up radios or time synchronisation can be avoided. In a simple radio duty cycle approach, a potential receiver alters the transceiver state periodically between listening and sleeping. The ratio between listening and sleeping periods dominates the communication related energy consumption pattern of the node. The sender must keep transmitting long enough such that the receiver is guaranteed to enter its listening period during the transmission.

Currently wireless sensor nodes such as the MICAz or TelosB platforms incorporate packet based transceivers (e.g. the Chipcon CC2420 or the Nordic nRF2401). These transceivers handle complete data packets independently from the microcontroller. As consequence, channel access can only be controlled on a per packet level. To implement radio duty cycles with packet based transceivers, a sender might transmit a trail of identical packets (which we call framelets [14]) of which the receiver is able to catch one in its active listening phase. This received packet might already contain the data that has to be transmitted or it might only be used to synchronise sender and receiver for the actual data transmission. To practically use the described framelet based radio duty cycle technique, it has to embedded efficiently within a MAC protocol.

In this paper we present our own framelet based communication framework (which we name FrameComm). The implementation of FrameComm is based on previous work in this area described in [14]. FrameComm is quite similar to Low Power Listening (LPL) which is a well known implementation of a duty cycled MAC protocol using framelet based communication. LPL is shipped with TinyOS 2.x for nodes using the CC2420 transceiver. However, our experimental evaluation of the LPL revealed that its usage in a realistic deployment scenario results in a very high packet loss rate. These high losses are caused by the applied clear channel assessment methods and back-off strategies. These problems are avoided in our implementation of FrameComm.

FrameComm represents an efficient and simple framelet based communication framework. However, many more optimisations and improvements are possible; some of these are sketched in [14] and [15]. One possible optimisation is explored within this paper: transmission interception and aggregation. A sensor node in the vicinity of two other communicating nodes might be able to overhear an ongoing framelet transmission. This is likely to happen during carrier sense prior to sending or at a scheduled listen period. The overhearing node can intercept an ongoing transmission if it has data to transmit to the same destination. The interception is done by interrupting the framelet transmission before the originally intended receiver is able to receive it. Subsequently, the overhearing node is able to take control of the channel and continue the intercepted transmission including its own data. Thus, the intercepting node is able to gain faster access to the channel, perform in-communication data aggregation and reduce contention. Within the paper we present a modification and evaluation of the basic FrameComm that includes the interception and aggregation mechanism. We refer to this modified FrameComm as i-FrameComm. As the evaluation shows, i-FrameComm improves throughput and network lifetime as communication resources are used more efficiently.

In summary, the contributions of this paper are:

- FrameComm: a framelet based communication framework that remediates shortcomings of the standard TinyOS2.x CC2420 LPL
- i-FrameComm: a framelet based communication framework that improves FrameComm performance by integrating interception and aggregation techniques
- A comparative evaluation of LPL, FrameComm and i-FrameComm

The rest of this paper is organised as follows. Section II discusses related work in the area of duty cycles and data aggregation. Section III describes the basic concept of framelet based communication. Section IV describes the FrameComm concept and its improvements compared to the standard TinyOS LPL. Section V explains the additional improvements that are included in i-FrameComm. Section VI details the implementation of FrameComm and i-FrameComm in TinyOS. Section VII presents the comparative evaluation of LPL, FrameComm and i-FrameComm. Finally conclusions and future work are discussed in Section VIII.

II. 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 typical trade offs are presented including latency and throughput versus energy efficiency.

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 [3] coordinates the sleep cycles for groups of neighbouring nodes by exchanging schedules and synchronisation messages. Like S-MAC, T-MAC [8] 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 near full sending queues. RMAC [2] 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 in addition 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 the relaying nodes to remain awake for data transfer while any other nodes return to sleep mode. Unlike these approaches our protocols do not use synchronisation and its associated overhead.

An alternative approach is to use out of band signalling such as that described in PAMAS [4] 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 [9]. Note that both of these approaches assume the presence of additional hardware which is a major drawback to their deployment.

It is also possible to implement an 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. One such protocol is B-MAC described in [7] which is implemented under TinyOS 1.x. WiseMAC [5] 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.

Methods of implementing asynchronous duty cycles in packetised radios are described in the X-MAC [6] and CSMA-MPS [1] protocols. These approaches reduce the amount of preamble that must be both sent and received by leaving gaps in the preamble so that the receiver can send an acknowledgement and data transfer can begin without further delay and further unnecessary preamble. The principal difference compared to [14] and our protocols is that these protocols do not send full data packets all the time but send a sequence of headers or strobes before sending a packet containing any payload. When one of these strobes is heard by the receiver an acknowledgement is sent and data transmission begins. Finally it must be stated that none of the above protocols attempts to perform aggregation of data over an active radio channel. To our knowledge this is an entirely novel approach.

Aggregation in wireless sensor networks has been a subject of intensive research as there are a great many benefits in terms of energy saving by using this approach. A common approach implement 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 [11], [12], [13], [10]. 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. [16] discusses how and where best to spend time waiting if each message has a latency bound. [12]

implements a cascading system 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.

III. Framelet Based Data Transmission

A basic problem introduced by the use of radio duty cycles as an energy saving technique is the need to establish rendezvous between transmitter and receiver. Since communication can only take place when the receiver's radio is active, the transmission of frames needs to somehow overlap with this active period. Transmitter-receiver rendezvous is the overlapping of data transmission and listening activities enabling effective communication.

To implement the duty cycle approach, no time synchronisation between communicating nodes is necessary. However, this can only be achieved at the expense of extra overhead per frame communicated. In the following paragraphs, an implementation of rendezvous in duty-cycled systems is described for packet based radios.

A. Assumptions and Definitions

It is assumed that the clock of transmitter/receiver operates 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 sleeping mode. The duty cycle ratio , or duty cycle for short, is defined as:

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

B. 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 the Nordic nRF2401, are able to transmit small frames at speeds of typically 1Mb/s. A sensor network frame of 32 bytes can therefore be transmitted in 1/4 of a millisecond. Framelets are defined as having a fixed or limited size. For example the maximum size of a CC2420 packet is defined at compile time (the default is 40 bytes in total including the header) and cannot be exceeded during run time without fragmentation. Instead, rendezvous requires the repeated transmission of several frames containing the entire payload as depicted in Fig. 1. If the receiver captures one framelet, the payload is received. The trail of framelets is defined by three parameters:

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

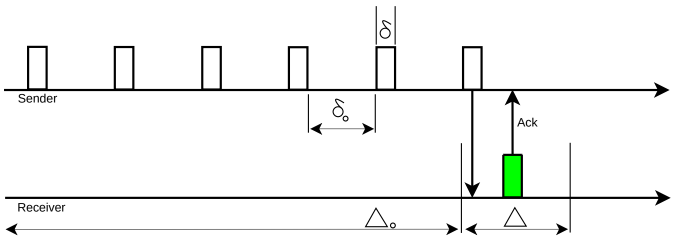

Fig. 1. Framelet-rendezvous

In order to ensure rendezvous, a proper relation between the parameters and Δ, Δ_0, n, δ, and δ_0 must be obeyed. First, the listening phase of duty cycle Δ must be such that $\Delta \geq 2 \cdot \delta + \delta_0$. 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 $\Delta_0 > 0$

$$n \geq \left\lceil \frac{\Delta_0 + 2 \cdot \delta + \delta_0}{\delta + \delta_0} \right\rceil \quad (2)$$

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 equation 1 the minimal possible active time Δ of the duty cycle. The duration of the time Δ determines message delay, throughput and energy savings as shown in the next section.

C. Acknowledgements

Framelets can be individually acknowledged by the receiver as shown in Fig. 1. Combined with the use of data replica framelets, this technique allows the transmitter to stop re-sending framelets shortly after rendezvous is established thus minimising energy expenditure and freeing the data channel. If acknowledgements are not used, or are used only after the successful delivery of the last frame, the transmitter is forced to resend no less than the number of frames specified in inequality 2.

D. Fragmentation

Although many applications of wireless sensor networks are expected to require very small frames, some deployments will involve larger payloads. If the physical layer of the sensor node only supports framelets, a fragmentation layer becomes necessary. The first fragment may be transmitted multiple times to guarantee rendezvous with the receiver. After rendezvous is established and the first fragment is acknowledged, each remainder fragment can be subsequently transmitted. It is assumed that the first framelet trail indicates of how many individual framelets the whole message consists. This allows the receiver to determine how long the radio must be kept active.

IV. FrameComm Design

The previously described method to implement duty cycles with framelets must be embedded in a MAC protocol. The following describes one possible instance of a framelet based MAC protocol that was used as a starting point to implement the interception technique described next. Another implementation is Low Power Listening which is part of the TinyOS 2.x suit of protocols which come with the standard distributions. Details of the implementation of LPL are described in Section VI.

By far the most common MAC protocol found on wireless sensor networks is CSMA. This is primarily due to its simplicity and low implementation requirements. For this reason we choose a CSMA style MAC protocol and modify it to suit the particular characteristics of framelet based transmission. To do this we must ensure correct carrier sense and choose a suitable backoff strategy.

A. Carrier Sense

There are a number of methods and techniques of performing carrier sense using wireless radios. First the channel may be briefly sampled just long enough to detect a signal above the noise threshold. This is commonly used on the CC2420 radios as they support clear channel assessment (CCA) which provides this feature. This is a very energy efficient approach as the channel is sampled for only the briefest of times. Note that this time is not long enough for a packet to be received.

A second approach is to activate the radio for a fixed period of time. If data is received then it can be concluded that the channel is busy. This is a simple way of assessing the channel but is more costly as the radio is kept active for slightly longer.

In order to perform carrier sense when framelet based transmissions are used it is necessary to have the radio listen for the duration Δ. This is the only way to guaranteed successful rendezvous between the listening radio and the transmitter. There are two possible methods to perform this listen which will be discussed next.

The CCA method can easily be adapted to this scenario. To do this a CCA must be performed every δ for the duration of Δ. If any CCA checks should fail then the channel will be deemed to be busy and a backoff will occur.

Alternatively the radio could be kept in a listen mode for the duration Δ and this is the approach that we use in this paper. Under most circumstances a packet trail will be intercepted and a whole packet will be received. If a packet arrives a backoff will occur. However there are circumstances where another node may begin transmitting near to the end of the listen phase. In this case an entire packet will not be received, no backoff will ensue, and a collision may occur. To avoid this a CCA check is made at the end of the listen phase to ensure a new framelet trail has not started. If the channel is not clear the listen phase is extended.

At first it may seem that using solely the CCA method is clearly better than simply listening for the duration Δ. While it is true that the CCA method is more efficient, activating the radio to receive data has a number of advantages. First the current sender on the channel may be attempting to send a packet to the listening node (who also wishes to send and is hence performing a carrier sense listen). This packet can be acknowledged and the channel will become clear thereafter whereupon transmission can begin without further delays. Second, since an unscheduled listen has been performed during carrier sense the next scheduled listen period can be rescheduled to a time Δ_0 afterwards thus minimising the time a radio is active. In addition if a node wishes to send but is already in a scheduled listening period the scheduled listening period itself can be used for carrier sense. Third, the data from overheard packets can be used to influence the backoff times. Forth, overheard data can be used for aggregation which is discussed in detail later.

B. Backoff

The second problem is how to determine the correct backoff strategy and backoff times. We adopt the non persistent backoff approach which is commonly used in wireless communications. That is, if the channel is busy the node will backoff for a period of time and the repeat the carrier sense. This process is repeated until the node gives up or the channel is acquired. The question of how long to backoff for is a little more complex.

Ideally when a node finds the channel to be busy its radio will be deactivated and it will awake as close to the end of the previous transmission as possible and attempt to gain the channel. (Naturally, some randomisation among competing nodes is necessary here to avoid collisions.) This behaviour minimises the time that the channel is unnecessarily idle and minimises latency. With this in mind we have developed a backoff strategy that takes into account the fundamentals of framelet based transmission and leverages overheard data to influence backoff times. The longest backoff times are when broadcast transmissions and unicast transmissions not requesting an acknowledgement are detected. These transmissions must span a full period P and must run to completion before releasing the channel thus requiring relatively long backoffs when encountered. Unicast transmissions requesting acknowledgement received a shorter backoff since it is likely that an acknowledgement will arrive and cut short the framelet trail possibly leaving the channel free thereafter. In both cases when any backoff occurs a record of the sender ID and the sequence number of the packet is kept. After the first backoff occurs if the same packet as was previously detected is still being transmitted then it is likely that this transmission will end very soon and only a very short backoff is needed. The backoff times become progressively smaller if the same packet is encountered repeatedly. The following is the formula used to calculate the backoff times,

$$T_{backoff} = \frac{P}{2^b} + \frac{Rand[0,\, P]}{2^b} \tag{3}$$

where b, $0 < b < 5$, is the number of times a node has backed off due to a particular transmission. The value of b is initialised to 2 when a unicast packet requesting acknowledgement is first detected and is initialised to 1 otherwise. For

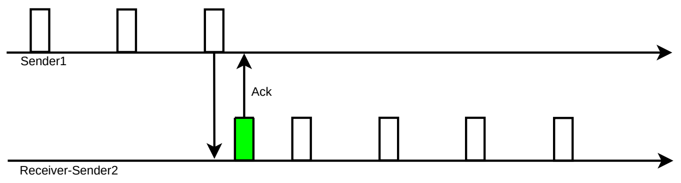

Fig. 2. Gaining the channel after carrier sense

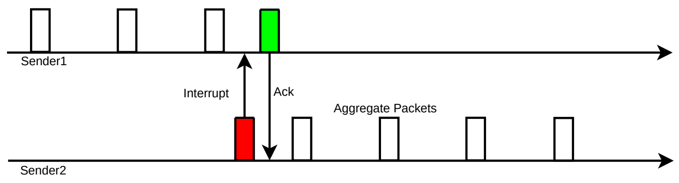

Fig. 3. Interrupts

each subsequent detection of that particular transmission b is incremented by 1 up to a maximum of 4. When a different packet is detected b reverts to its original value and the sender id and sequence number are overwritten with the data from the newer packet. Note that the random element along with the fact that an "educated guess" is made regarding when a transmission will be finished ensures that competing nodes to not all try to gain access to the channel simultaneously after another transmission has completed. Also if b exceeds its bounds it is also reset to its initial value (i.e. either 1 or 2). This ensures that if a node within the network is malfunctioning and causing other nodes to backoff that the backoffs do not become excessively small increasing contention to unacceptable levels.

V. i-FrameComm Design

A node (called Sender2 for example) with a message to send 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 a node will keep its radio active long enough to receive a single framelet. Should Sender2 receive a packet addressed to it (from Sender1 for example) an acknowledgement will be sent in the normal way and the channel will then be clear to transmit the original packet as seen in Fig. 2. If the received packet is not addressed to the node it must check if the addressee is the same addressee of the message it is currently trying to send (i.e. a message is being sent by a peer node to the same parent). If so then both messages can possibly be aggregated. Sender2 checks to see that aggregation is possible (i.e. there is adequate space in the payload) and builds an aggregate packet. An interrupt is then sent to the original sender (Sender1). Upon receipt of this interrupt Sender1 will cease its transmissions and Sender2 will take over the channel and begin to transmit a framelet trail of its own which continues from the previous framelet trail as in Fig. 3. Should the received packet not be addressed to either Sender2 or its parent then the message will be discarded and Sender2 will activate is backoff mechanism.

A. *Collision avoidance*

Under normal circumstances the number of neighbouring nodes accessing a shared radio channel is limited in order to reduce collisions. In particular sampling the channel before transmission is a means to attempt to eliminated 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 occur simultaneously from different nodes (Sender2 and Sender3 for example) a collision may occur. There is the potential that the current sender (Sender1) and Sender2 and Sender3 will all try to send their framelet trail simultaneously. This situation can be avoided by the use of a handshaking mechanism similar to the CTS-RTS method as in Fig 3. The method works as follows. After sending an interrupt message to Sender1 both nodes will wait and listen for Sender1 to acknowledge the interrupt or to send its next packet. If either node should be successful in its attempt to interrupt Sender1 then Sender1 will send an acknowledgement to the successful interrupter. Both Sender2 and Sender3 will hear the acknowledgement and know if their bid to interrupt Sender1 was successful or not. The winning node (Sender2) will assume control of the channel while the losing node (Sender3) will back off and retry at a later time to interrupt the current sender. If neither packet were successful due to both attempted interrupts colliding then Sender1 will continue its framelet trail as normal and both Sender2 and Sender3 will use a short random backoff before trying to interrupt again.

B. *Expected benefits*

Conceptually it is relatively easy to see the situations in which this form of aggregation are advantageous. First there must be a reasonable amount of traffic so that a framelet trail may be interrupted by its peers. Second as duty cycles become more aggressive the length and time span of the framelet trails become longer thus increasing the likelihood that an interrupt may occur during this period. Third the chances of an interrupt occurring will also depend on the number of peers a transmitting node has. For example, a bushy tree should produce more favourable results than a sparse tree. Therefore if a very short duty cycle is assumed in a bushy tree and traffic occurs in sporadic bursts we should expect our protocol to be particularly effective.

Note that 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 the time taken to deliver all messages should be shorter). Using a fixed duty cycle with no interrupts and aggregation the best case 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 and additional $(n-1)P$. Assuming that on average $0.5P$ will expire before rendezvous the the next hop latency can be given by the following approximation:

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

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 that the $7.5P$ it would take under normal conditions. Note that this approach is also more energy efficient for the senders since less time need collectively be spent with their radios active. 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 8 messages are sent in time $2.5P$ compared to $7.5P$ under normal circumstances. The use of interrupts and interception allows a variable throughput despite having a fixed length listen period. Previous research allowed for variable throughput by increasing the listening time of the receiver to cope with additional data transfer. 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 that the radio must be kept on for some minimum additional time regardless of whether there is additional data or not. If bursts of high traffic are rare then this can 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.

C. Alternative aggregation methods

Rather than aggregate the data payload of a packet it is possible to aggregate multiple data packets into a framelet trail in a number of ways. One such way is to simply append the extra packets onto the end of the framelet trail. The receiver can deduce that there are additional packets to be receiver by examining a flag in the header field. Whenever a node has additional packets to send it would set the ADDITIONAL_PACKETS flag to 1 in each header. For example a sender with packets 1, 2 & 3 to send would send a trail of framelets consisting of packet 1 with the ADDITIONAL_PACKETS flag set to 1. On waking and hearing the first packet a receiver would acknowledge packet 1 and extend

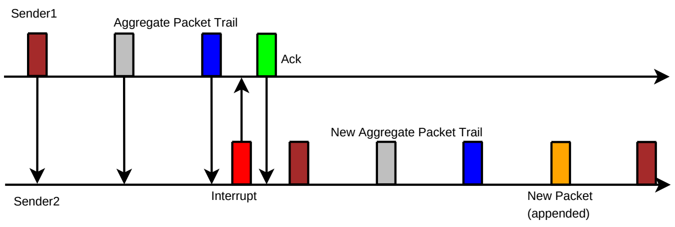

Fig. 4. Alternative Aggregation Method

its listening period to receive packet 2 which would also have its ADDITIONAL_PACKETS flag to 1. Again the receiver would acknowledge packet 2 and extend its listening period. Finally packet 3 would be received and the listening period would end since the ADDITIONAL_PACKETS flag is set to 0.

It is apparent that when data is combined in this manner that energy savings similar to aggregation occur with very little overhead. This is particularly the case with very aggressive duty cycles where the length of the framelet trails are very long. Note that data types not suitable for aggregation can be handled in this manner, a significant advantage over traditional aggregation approaches.

It would be especially beneficial if we could use our interrupt approach to combine these messages on the fly. However this is difficult to do when the sending node has multiple packets to send. If a single packet is being sent an interrupt can be used in the normal manner. Otherwise all messages must be transfered to the interrupter. One method to do this is to perform a normal interrupt and the send the additional messages to the interrupting node sequentially. There are several problems with this approach. First the transfer period may overlap with the original destinations listen period and the original destination node may resume a sleep state before the transfer has been completed. Second the interrupter may have a single message while the original sender may have several. Transferring several messages to be aggregated with one is not energy efficient. An alternative approach in this particular may be to use some kind of handshaking so that the node with less messages passes them to the node with more messages. This has the disadvantage of increasing the complexity of the protocol and does not solve the first problem.

We propose the following alternative as a solution to the problems outlined above. Rather than send each packet trail sequentially the framelet trail will alternate between each packet that has to be sent. For example if packets 1, 2, and 3 are to be sent as in the previous example the framelet trail will be 1,2,3,1,2,3.........etc. Header fields in each will indicate that each is a member of a trail and its order in the trail (i.e. 1 of 3, 2 of 3, etc.). A node wishing to interrupt will realise that it must listen to all packets in the sequence before an interrupt can be sent. When all packets have been received an interrupting node can send and interrupt and the usual mechanism is followed. This procedure is illustrated in Fig. 4. In order to avoid conflicts the responsiveness of the intended receiver is greater that any potential interrupting node. Upon hearing a packet the intended receiver sends an acknowledgement for that packet immediately whereas an interrupting node must wait for the full sequence to complete. If an ack is received by the sender before an interrupt arrives the sender will ignore any further attempted interrupts and complete delivery to the intended receiver. Any interrupting nodes will backoff and retry. At this stage all transmissions will be finished or acknowledgements from the intended receiver will be heard. In either case the additional packets will be dropped and the attempted interrupter will resume normal operation. This method is not implemented in this paper but will form part of our future work.

VI. Implementation

FrameComm and i-FrameComm were both implemented in TinyOS 2.02. and both of these were compared in Section VII where telosb nodes were used in a small deployment, with Low Power Listening which is the default duty cycle protocol in the Tiny0S 2.02 release. FrameComm and i-FrameComm have the same components but the difference lies in their usage. In Fig. 5 the component graph for LPL is shown. Note that this is very similar to the implementation of FrameComm in Fig. 6. The primary difference is that the DefaultLPLC component and it's associated components are replaced by a CC2420DutyCycleC component. Thus the inclusion of FrameComm does not upset the existing TinyOS 2.02 architecture to any significant degree.

Since FrameComm and i-FrameComm are very similar (the former is a subset of the functions of the latter) they are both contained within the same component. It must be decided at compile time which protocol to use. At present this is done by altering some of the code. In future versions of the code we will adopt the TinyOS 2.x methodology of having variable components. This can be seen in the use of dummy components that are wired into place according to preprocessor definitions at compile time. For example if LOW_POWER_LISTENING is not defined an alternative component called DummyLPLC is wired into the configuration rather than the DefaultLPLC component. Both components provide the same i-faces to higher level components and are aliased as LplC.

The implementation of FrameComm and LPL are very similar in a number of ways but there are some important differences. In terms of similarities they both use framelet trails to ensure rendezvous and the listening cycle length is dictated by the duty cycle and the times δ and δ_0. LPL however does not listen in the same manner as the implementation of FrameComm. LPL repeatedly polls the CCA to detect if the channel is busy for a fixed time of 11ms. If during this time a packet is detected the listening phase is extended to ensure that the packet is received correctly. FrameComm on the other hand performs a normal listen for 12ms which should guarantee interception of any packets transmitting and performs a brief CCA at the end of the listen period in case another framelet has begun transmitting. If so the listen period is extended. As a result of the longer listen period Δ FrameComm had a longer sleep period Δ_0 than LPL. A 2% duty cycle using FrameComm has a period P=600ms whereas LPL has a period $P = 550ms$. Therefore LPL should theoretically be able to handle more traffic than FrameComm at the same duty cycle.

Another major difference is the way in which the carrier sense is performed. Whereas FrameComm uses a full listen, as described above, LPL merely uses a brief CCA before beginning its transmissions. The use of this brief CCA is insufficient to ensure that the channel is in fact clear and a number of problem arose when LPL experienced contention for the channel. These problems are described in sectionVII.

Yet another difference between the two implementations was the backoff strategy. FrameComm make use of a full listen and exploited the overheard information to influence it backoff strategy. LPL does not do this since it does not receive packets during its carrier sense phase. In addition the backoff strategy of LPL is unrelated to the duty cycle and resulted in numerous backoffs which were insufficient to remove unnecessary contention for the channel.

With regard to the implementation of i-FrameComm we adopted a packet stuffing approach 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 as long as any operations are sufficiently fast. Information from the CC2420 header regarding the size of the payload was used to quickly ascertain if aggregation was feasible.

In the implementation of i-FrameComm it was vital that the different types of packet could be quickly and easily distinguished from each other. This was done by examining the CC2420 packet header which contained an 1 byte type field. Thus interrupts could be quickly distinguished from data packets or control messages.

VII. Experimental Evaluation

In this section the behaviour of just a few nodes is observed in detail. The reason for this is to gain a clear insight into the effectiveness of the protocols evaluated without any obscuring factors such as network dynamics, background traffic or queueing effects. While this approach does not accurately represent the potential "in the field" performance of the protocols, it does offer an extremely transparent and fair evaluation. In all of the experiments a simple and small tree-like structure was used. This consisted of a base-station and an intermediate forwarding node and two to four leaf nodes. All nodes are unsynchronised. 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 on aggregation events. Each experiment run was repeated three times for each data point.

We consider 3 individual experiments. The first experiment examines how many messages are successfully delivered to the base station at varying message generation intervals. We

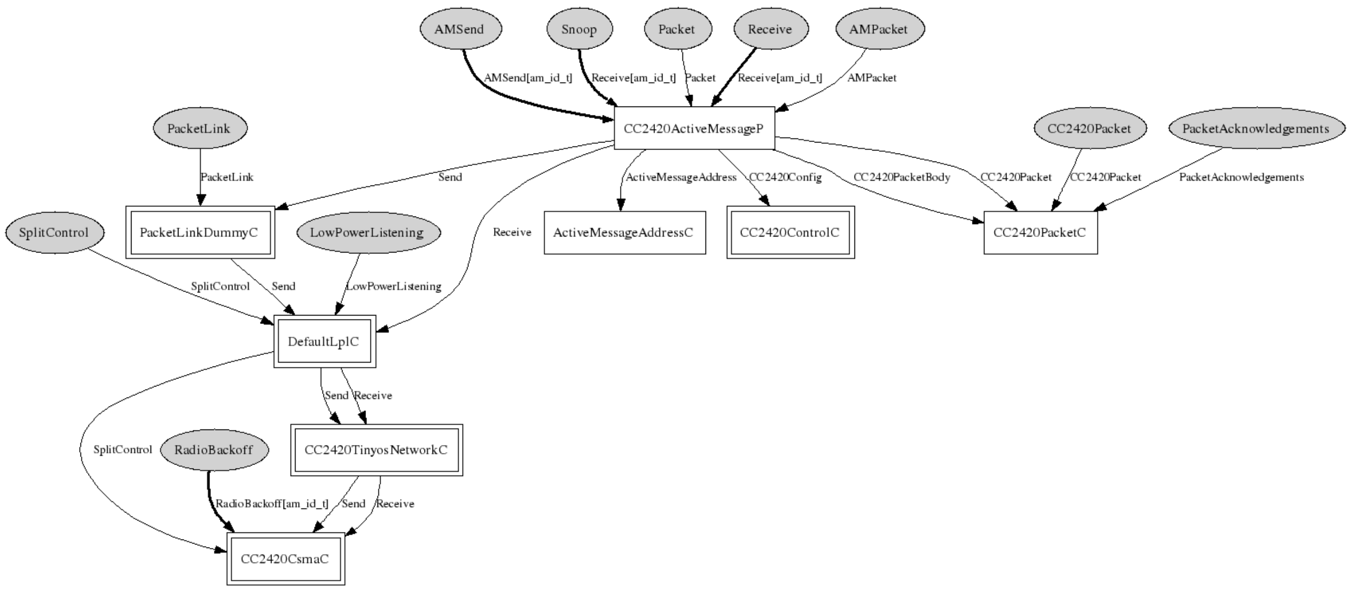

Fig. 5. TinyOS implementation of Low Power Listening (LPL)

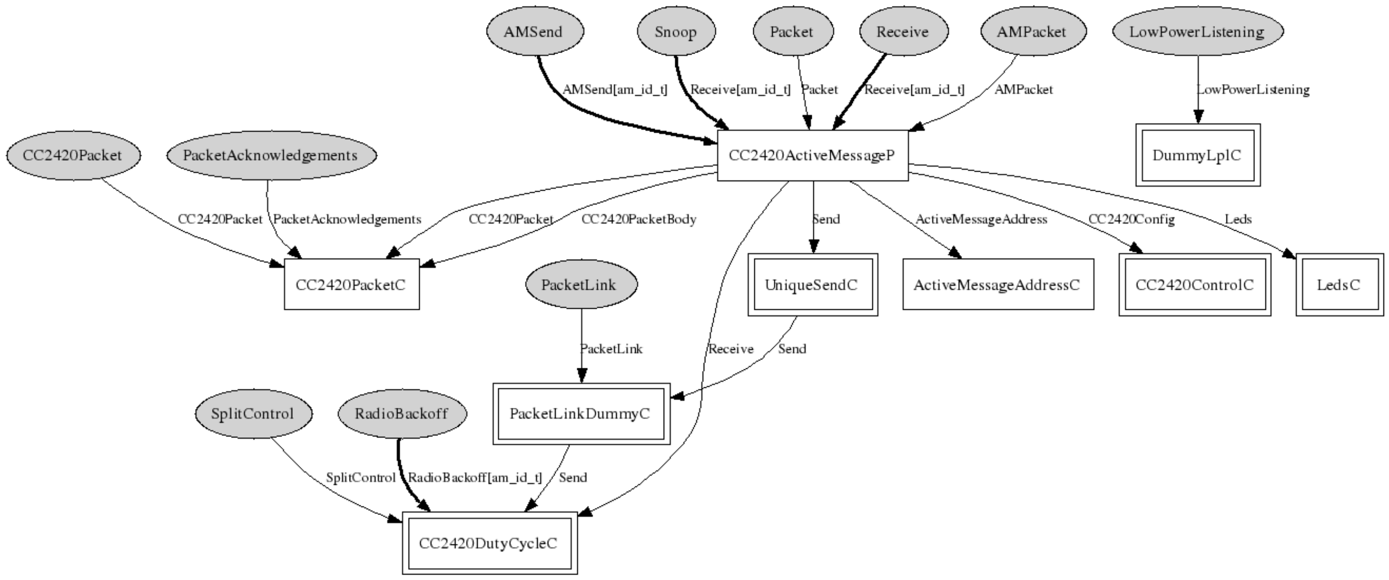

Fig. 6. TinyOS implementation of FrameComm

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 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 that additional time is used to complete the sending of the 100 messages. This provides an accurate and fair way to gauge 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 i-FrameComm.

A. Experiment 1

Data was generated at each leaf node periodically at varying rates. Every 500, 1600, 2700, 3800 and 4900 milliseconds respectively. Each node generated 100, 5 byte, sensor messages per run. The 5 byte sensor message included a 16 bit node ID an 8 bit sequence number and a 16 bit data reading. A maximum of 5 packets could be aggregated into a 28 byte payload.

One striking thing that can be seen from the graphs (Fig

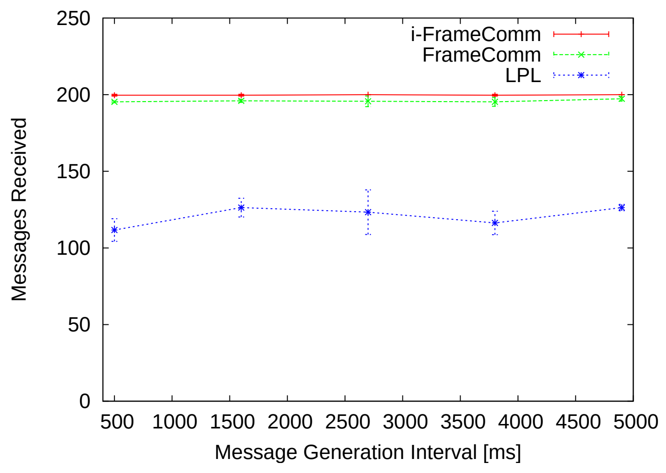

Fig. 7. Data delivered with 2 leaf nodes

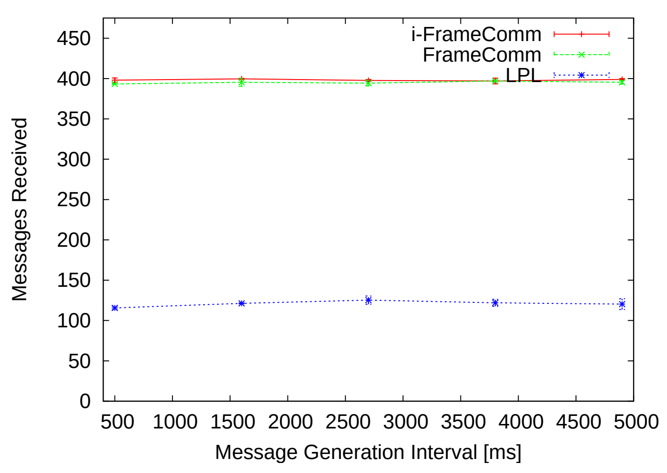

Fig. 9. Data delivered with 4 leaf nodes

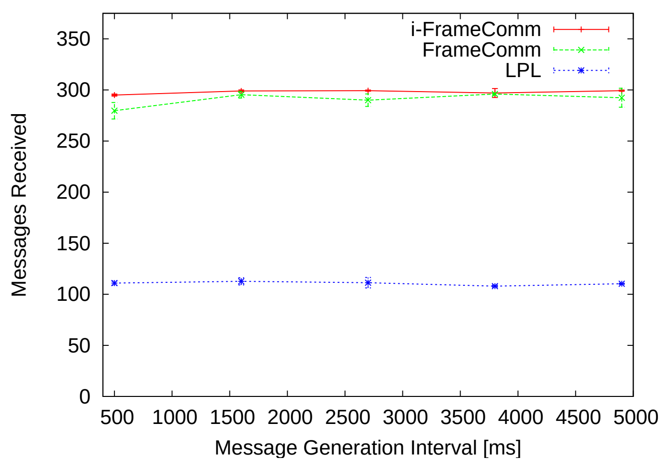

Fig. 8. Data delivered with 3 leaf nodes

7, 8 and 9) is that the data losses for FrameComm and i-FrameComm is much less than that of LPL. LPL appears to be only able to deliver an 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 interleaving of messages, evident from data traces generated during 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, then interleaving will occur. This interleaving often results in one or both of the packets being lost.

Also, is 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. 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 there are more listening periods at the intermediate forwarding node than the duty cycle would suggest. Note that this phenomenon does not happen all the time but the likelihood of such occurrences increases with respect to the traffic. Therefore our implementation of FrameComm seem to be somewhat traffic aware, albeit inadvertently, and modifies the amount of listening periods used accordingly. This is one reason why FrameComm and i-FrameComm correspond quite closely 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 i-FrameComm somewhat. Examination of the raw data verifies that i-FrameComm 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, i-FrameComm appears to be the clear winner and significantly out performs both FrameComm and LPL at higher data rates and bushier topologies. It is clear that the multiplexing ability of the i-FrameComm 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 node 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. 10,**??** and 12 that the plots for aggregation 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 their peers while others will aggregate less. Thus some nodes may do a lot less work than their counterparts leading to the large error bars seen on the graphs.

The radio on time from the LPL experiments in Fig. 10, 11 and 12 is unexpected. It appears from the graphs that the

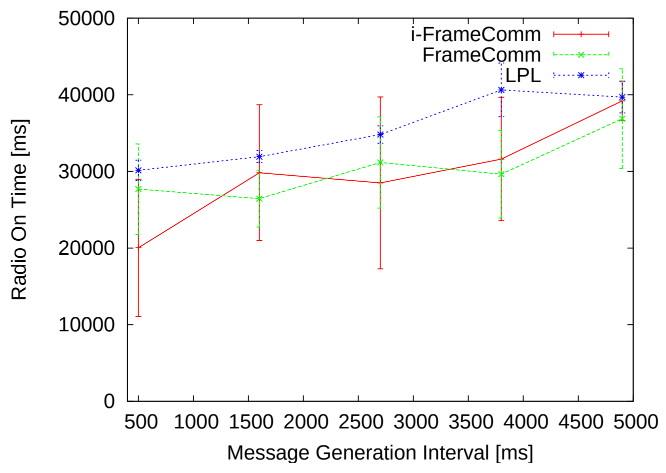

Fig. 10. Radio On Time with 2 leaf nodes

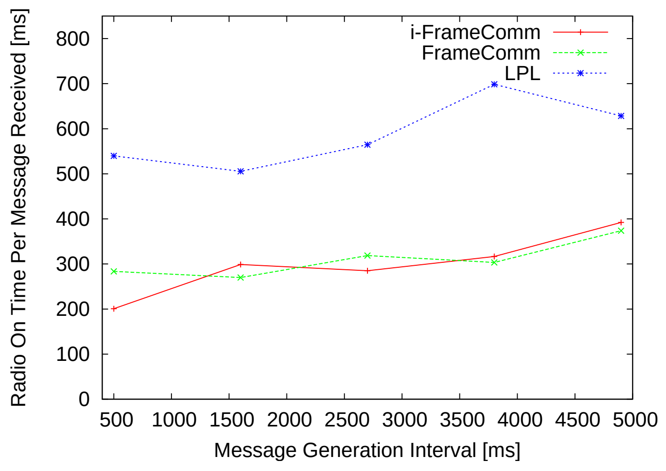

Fig. 13. Radio On Time per message with 2 leaf nodes

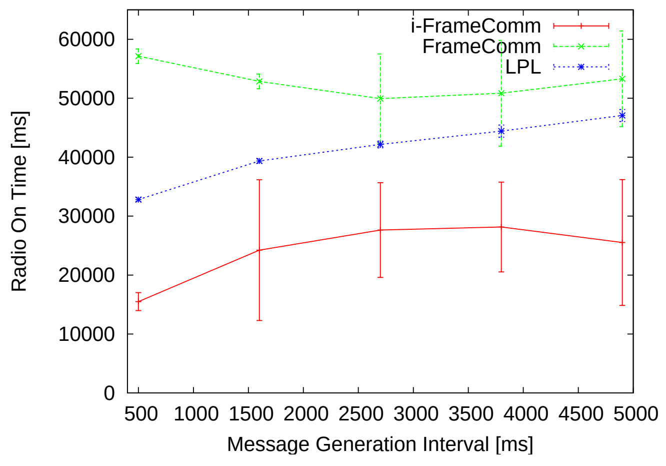

Fig. 11. Radio On Time with 3 leaf nodes

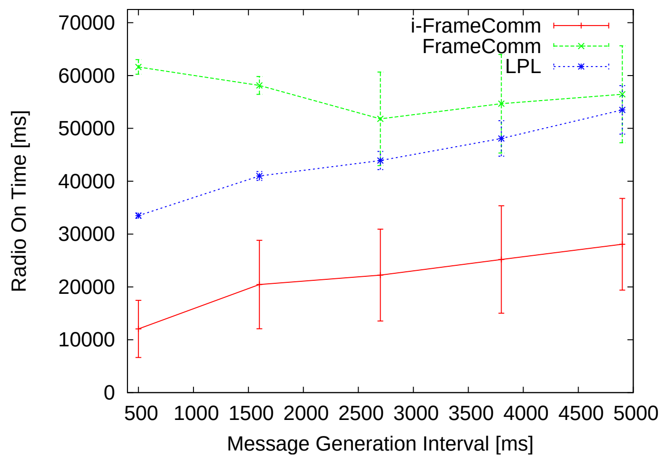

Fig. 12. Radio On Time with 4 leaf nodes

performance of LPL improves as the number of leaf nodes increase. Note however that LPLs radio on time is nearly constant between the 3 graphs. This is primarily because the data losses 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 and results from a mixture of interleaving and dropped packets. Due to the fact that so many packets were not sent correctly in relation to the other protocols it is a somewhat unfair comparison. A more fair comparison is shown in Fig. 13, 14 and 15 where the radio on time per sensor message sent is shown. Here we can see that both FrameComm and i-FrameComm significantly outperform LPL. Note that in Fig. 13 there is no significant benefit to using i-FrameComm over FrameComm at the slower data generation rates. However a significant distance can be seen when the generation rate is at it's fastest and therefore there are more opportunities to aggregate. Likewise it can be easily seen from Fig 14 and Fig. 15 that i-FrameComm easily outperforms the other approaches. This confirms our suspicions that the effectiveness of i-FrameComm method increase with respect to both the data generation rate and the number of active peer nodes (or bushiness of the topology). This is also reinforced upon examination of Fig. 16 which plots the number of aggregation events against the data generation interval for the various topologies.

B. Experiment 2

In this experiment we examine the time taken by each of 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 300ms. Should the radio be busy the message attempts to resend again in another 300ms. 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. 17 show the time taken for each

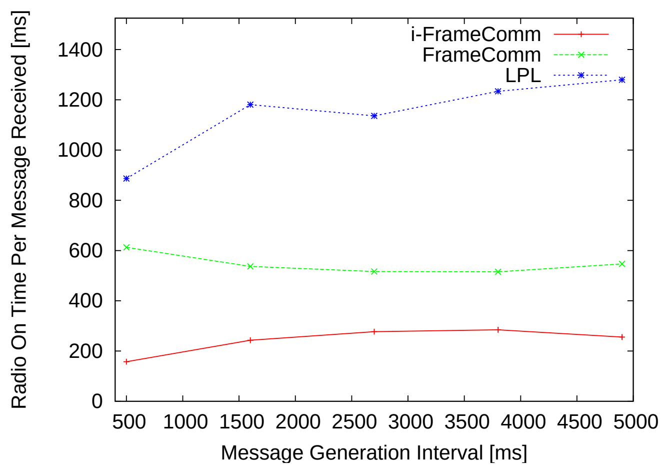

Fig. 14. Radio On Time per message with 3 leaf nodes

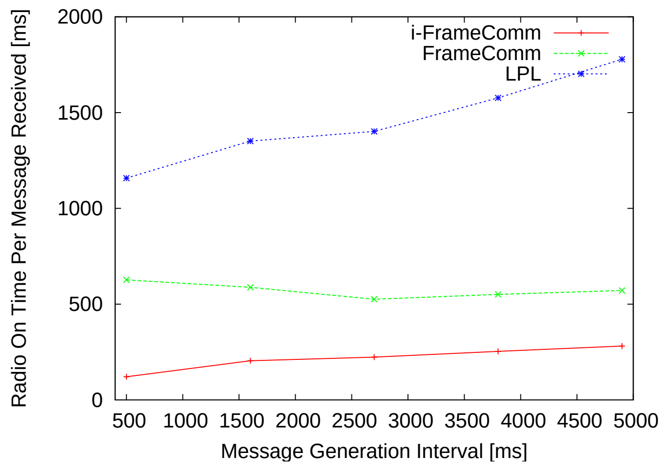

Fig. 15. Radio On Time per message with 4 leaf nodes

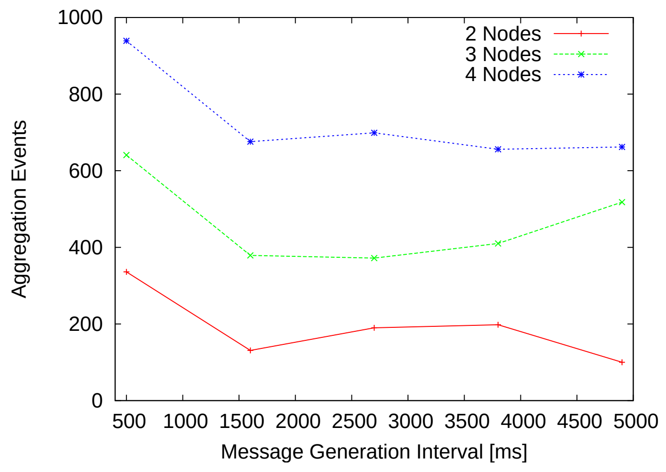

Fig. 16. Aggregation Events

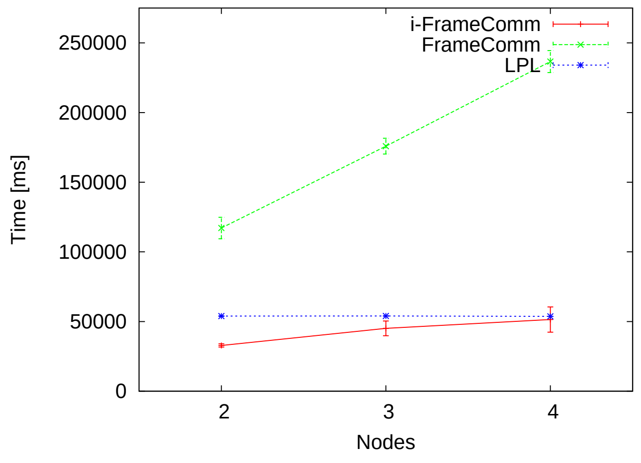

Fig. 17. Time taken to deliver 100 messages per node

node to send all 100 messages. Note that LPL has a fixed time on the graph. This is due to the fact that LPL fails to do a correct carrier sense as discussed previously and fails to do sufficient backoff. Therefore messages are pushed onto the network irrespective of whether it is busy or not causing collisions and packet losses. LPLs data losses during this experiment were very high and similar to those experienced in the previous experiment. This confirms that the majority of the losses experienced by LPL in the previous experiment were due to network failures/errors rather than dropped packets. It is also clear that i-FrameComm offers significant benefits compared to FrameComm.

C. *Experiment 3*

As previously mentioned a possible benefit of i-FrameComm 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 criteria is that a fixed percentage of packets are successfully received. In this experiment we adopt a similar topology to the one used in the previous experiments. Three leaf nodes generate a 4 byte message, thereby allowing a possible 7 messages per payload while aggregating, every 300 ms which is then forwarded to an intermediate node. As before this node forwards the received messages to a base station. All data is unbuffered as described in section VII-A.

Consider Fig. 18. Here FrameComm is compared to i-FrameComm. LPL was not considered due to its poor performance in previous tests. From the graphs it is obvious that i-FrameComm is able to support a much lower duty cycle while maintaining a low data loss rate. One particular scenario

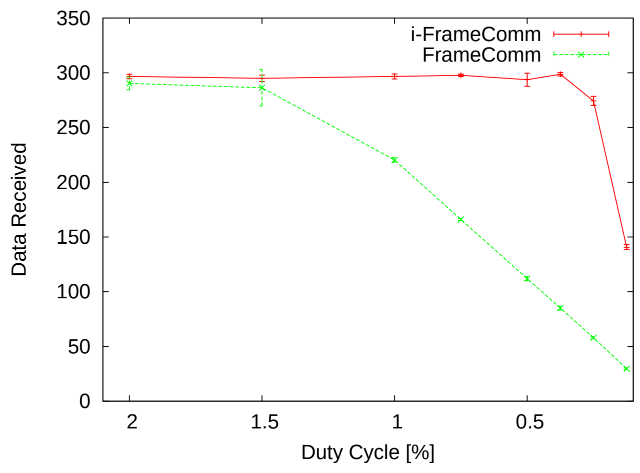

Fig. 18. The effects of reducing the duty cycle on data loss

where i-FrameComm would be especially beneficial is where an event based system is used. Should an event occur the network must be able to handle the traffic generated and yet conserve as much energy as possible when no events occur. Unlike other duty cycled approaches i-FrameComm offers a way to do this in a flexible and simple manner.

VIII. Conclusions and Future Work

A method of aggregating data on the fly over duty cycled communications, i-FrameComm, has been described and evaluated. To our knowledge this is an entirely novel approach to data aggregation in sensor networks. It is clear that this approach to data aggregation has significant benefits in sensor networks where energy efficiency and adaptability to variable traffic are a concern. In particular this work helps to address some of the fundamental tradeoffs inherent in implementing duty cycled communications. We have shown that our approach can deliver improved throughput and latency. Energy efficiency is also greatly improved compared to conventional duty cycled approaches. Our approach also show a great deal of flexibility in its ability to handed varying traffic loads and is particularly useful in event driven systems.

We have also evaluated FrameComm and LPL and shown that there are significant problems with the current implementation of LPL in TinyOS 2.x. It must be noted that FrameComm and LPL are based on a similar design concept and only differ in a few important design choices, primarily the length of the listen period and the implementation of the CSMA feature. We have also shown how a full and correct listen to implement CSMA leads to a more correct operation of the sensor network and less data loss. We believe that LPL may have much better performance if these issues are addressed. We have suggested a number of way in which data gleaned from a full CSMA listen can be exploited and proposed some optimisations to the backoff strategies used with duty cycled transmissions.

Our future work will exploit additional forms of aggregation such as that described in Section III. Apart from exploring the possibilities of varying types of aggregation there are a number of possibilities that arise from the use of interrupts. Priority channel capture using interrupts is a particular area which we will examine in our future work.

IX. Acknowledgements

Mr. O'Donovan is supported by Microsoft Research through its European PhD Scholarship Programme and the EMBARK Initiative of the Irish Research Council for Science, Engineering and Technology.

References

[1] S. Mahlknecht and M. Boeck. "CSMA-MPS: A Minimum Preamble Sampling MAC Protocol for Low Power Wireless Sensor Networks". In Proceedings of the 5th IEEE International Workshop on Factory Communication Systems (WFCS2004), Vienna, Austria, September 2004.

[2] S. Du, A. K. Saha and D. B. Johnson. "RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks". In Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007). May 2007 Page(s):1478 - 1486.

[3] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in IEEE Infocom 2002, Jun 23-27 2002, vol. 3, pp. 1567.

[4] S. Singh and C. S. Raghavendra, PAMAS: Power aware multi-acces protocol with signalling for ad hoc networks, ACM Computer Communications Review., vol. 28, no. 3, pp. 5-26, July 1998.

[5] A. El-Haiydi, WiseMAC, An Ultra Low Power MAC Protocol for the WiseNET Wireless Sensor Network, ACMSenSys, Los Angeles, 5-7 Nov 2003.

[6] Michael Buettner, Gary V. Yee, Eric Anderson, Richard Han: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. SenSys 2006: 307-320

[7] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In The Second ACM Conference on Embedded Networked Sensor Systems (SenSys), pages 95-107, November 2004.

[8] W. Ye, J. Heidemann, and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks. In IEEE Infocom 2002, Jun 23-27 2002, vol. 3, pp. 1567.

[9] Eugene Shih, Paramvir Bahl, Michael J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In Proceedings of the 8th annual international conference on Mobile computing and networking, MobiCom 2002.

[10] J. Gehrke, Y. Yao. Query Processing for Sensor Networks. IEEE Pervasive Computing 2004, vol 3, number 1, pages 46-55.

[11] P. Bonnet, J. Gehrke, T. Mayr, P. Seshadri. Query Processing in a Device Database System. Tech. Report, number 99-1775, Cornell University, Ithaca, NY, USA, 1999.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. Proc. of the 5th Annual Symposium on Operating Systems Design and Implementation, 2002.

[13] J. Beaver, M. A. Sharaf, A. Labrinidis, Panos K. Chrysanthis. Power-Aware In-Network Query Processing for Sensor Data. Proc. of the 2nd Hellenic Data Management Symposium, 2003.

[14] Andre Barroso, Utz Roedig, and Cormac J. Sreenan. Use of Framelets for Efficient Transmitter-Receiver Rendezvous in Wireless Sensor Networks. In Fifth International IEEE Workshop on Wireless Local Networks (WLN2005), Sydney, Australia, November 2005.

[15] Utz Roedig, Andre Barroso, and Cormac J. Sreenan. f-MAC: A Deterministic Media Access Control Protocol Without Time Synchronizati on. In Proceedings of the third IEEE European Workshop on Wireless Sensor Networks (EWSN2006), Zurich, Switzerland, February 2006.

[16] Utz Roedig, Andre Barroso, and Cormac J. Sreenan. Determination of Aggregation Points in Wireless Sensor Networks. In Proceedings of the 30th Euromicro Conference (EUROMICRO2004), Rennes, France, pp. 503:510, IEEE Computer Society Press, August 2004.