
Vinod Namboodiri ∗
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA
vnambood@ecs.umass.edu

Abtin Keshavarzian
Robert Bosch Research and Tech. Center
4009 Miranda Ave., Palo Alto, CA
abtin.keshavarzian@us.bosch.com

Abstract

Collection of rare but delay-critical messages from a group of sensor nodes is a key process in many wireless sensor network applications. This is particularly important for security related applications like intrusion detection and fire alarm systems. An event sensed by multiple sensor nodes in the network can trigger many messages to be sent simultaneously. We present Alert, a MAC protocol for collecting event-triggered urgent messages from a group of sensor nodes with minimum latency and without requiring any cooperation or pre-scheduling among the senders or between senders and receiver during protocol execution. Alert is designed to handle multiple simultaneous messages efficiently and reliably minimizing the overall delay to collect all messages along with the delay to get the first message. Moreover, the ability to handle a large number of simultaneous messages does not come at the cost of excessive delays when only a few messages need to be handled. We analyze Alert and evaluate its feasibility and performance with an implementation on commodity hardware. We further compare Alert with existing approaches through simulations and show the performance improvement possible through Alert.

1 Introduction

With the transition of many automated tasks from a wired to a wireless domain, wireless sensor networks (WSNs) are increasingly being subjected to new application domains. Applications of critical nature have been the forte of wired networks due to their reliability. The ever increasing reliability of WSNs coupled with cost-effectiveness has led to their gradual adoption for such critical applications as well. The nature of such applications, however, require new MAC protocols for WSNs that meet the requirements and inspire sufficient confidence about their usage.

The requirements for applications of critical nature can be fundamentally different from the applications for which current MAC protocols are designed for. For example, energy is a valuable resource in sensor devices and most existing MAC protocols are optimized to conserve energy, trading off latency, throughput and other similar performance metrics in the process. These same protocols are typically not suitable when the application demands better performance at the expense of some additional energy. If latency is to be minimized with energy consumption only a secondary issue, protocols need to be re-designed from that application perspective.

In this paper we consider applications that require all wireless sensors to convey urgent messages to a centralized base station with minimum delay from the time they are generated. These messages are triggered by events detected by sensor nodes and their task is to inform the base station for possible action. Such messages are triggered very rarely, and the aim is to focus on minimizing latency when they are triggered, even if some additional energy is expended during those times. Intrusion detection and fire alarm applications are some examples which require such a solution. Even though the messages are typically correlated, the collection of all messages as opposed to just a subset of them provides valuable information which can be used for detection of false positives or post event analysis. For example, the European Standard EN 54-25 for fire alarm systems specifies the duration within which the first alarm should be reported and by when all alarms must be received at the base station [1]. The challenges in designing WSN MAC protocols for such applications are the handling of a number of simultaneous messages at the same time without knowledge of how many, and planning for possible interference. Additionally it is also important to ensure implementation feasibility taking into account the additional constraints imposed on WSNs like time synchronization and limited computation and storage capabilities.

*The work was performed when the author was a research intern at Bosch Research and Technology Center.
We demonstrate the feasibility of implementing Alert and validate our analytical results. We further compare Alert with two other existing MAC protocols to point out its advantages. Section 6 presents some related work and concluding remarks are made in Section 7.

2 Protocol Description

The protocol actions can be divided into those at a sender, a node that has a message to send, and a receiver, a node whose task is to collect these messages. The time is slotted\(^1\) into what we refer to as an Alert slot or simply a time slot. Each Alert slot can be used to exchange one data packet and its acknowledgment between a sender-receiver pair.

In each time slot, multiple frequency channels can be used by the senders or receiver. We denote the number of frequency channels in each slot by \(M\). These channels have different priorities. The receiver samples them one by one based on their priority level and tries to receive a packet from one of the senders.

Each sender selects a frequency channel randomly and independently of all other senders, but the channels are not selected with equal probability. Less chance is given to select a higher priority channel, i.e., the selection probability decreases as we move towards higher priority channels. An example is shown in Figure 1 with \(M = 3\) channels and channel selection distribution \((p_1, p_2, p_3) = (0.1, 0.3, 0.6)\). This non-uniform distribution is pre-specified and known to all senders. It is designed to reduce the chance of collision among the senders. We discuss its effect and how to find the optimum distribution in detail in subsequent sections.

Once a sender selects a frequency channel randomly based on the pre-specified channel selection probabilities, it switches to the selected frequency and sends a long preamble\(^2\) before sending its data packet. After the data packet, the node expects an acknowledgment packet (ack) from the receiver. If the ack packet is received correctly, the sender stops, otherwise, it tries to send its message again in the next time slot.

At the beginning of each time slot, the receiver samples the signal level on each of the \(M\) frequency channels starting with the highest priority channel. If high signal level (high RSSI\(^3\)) is sensed by the receiver, it stays on the same frequency (locking to that channel) and stops sampling any more channels. Then the receiver waits to receive a packet. If a packet is received correctly, it sends an acknowledg-

---

\(^1\)We assume all the nodes in the network are time synchronized with each other. We use periodic broadcast beacons from our base station in our implementation in Section 5.

\(^2\)Note that the diagrams in Figures 1 and 2 do not represent the actual timing scales within a time slot. To present the idea, the sampling and preamble durations are shown much longer than their actual value compared to the packet and ack exchange duration.

\(^3\)RSSI stands for Received Signal Strength Indicator.
messages from all senders in as few time slots as possible. Therefore, the protocol can collect and its energy consumption is not an issue. The rarity of alarms allows for greater effort to be put into re-
sending fire alarms only once or twice a year. The example, a typical system may encounter a situation that requires sending a message very infrequently. For a sender has a message to send very infrequently. For

Note that a transmitter is not aware if a receiver has successfully ‘locked’ onto its selected frequency and will thus always transmit the packet even if the receiver is waiting on some other channel. For the applications under consideration, a sender has a message to send very infrequently. For example, a typical system may encounter a situation that requires sending fire alarms only once or twice a year. The rarity of alarms allows for greater effort to be put into reducing latency even at the cost of some extra energy. The receiver, or centralized base station, is typically ac-powered and its energy consumption is not an issue.

While the number of channels, $M$, remains the same, the frequency of channels (and their priority) can change across time slots. This is illustrated in Figure 2. The frequency table shown in this example is from a simple expression where we have 16 channels numbered from 0 to 15 and $f_m(k)$ represents the $m$th frequency channel in $k$th Alert slot: $f_m(k) = [5k + 9(m - 1)] \mod 16$ for $m = 1, 2, 3$. We assume that the frequency table is pre-specified, and all the nodes in the network know this pattern. Varying frequency channels after each time slot increases the reliability and robustness against channel fading and interference.

Alert is designed to avoid collision among senders such that in most time slots (with high probability) one message is received correctly. Therefore, the protocol can collect messages from all senders in as few time slots as possible. If there is only one sender, there will be no collision and the frequency channel selected by the sender does not matter as the receiver will find and lock to the frequency picked by the sender. If two nodes are contending to send their messages, the node that selects the higher priority channel will be successful since the receiver will hear its preamble/tone first and stay in the channel awaiting its packet. If both select the same channel, a collision occurs. The channel selection probabilities are designed to reduce the chance of collision. As the number of senders increases, it becomes more probable that one of the senders selects a higher priority channel, and if only one node selects the higher priority channel, the message from that sender will be successfully received.

Based on the protocol description, it should be clear that the two key design parameters are the number of frequency channels to use in the protocol and the probability distribution over the channels that minimizes the overall time to read all messages. A larger number of channels should decrease contention among the nodes that have messages to send. But this increases the size of a time slot and results in fewer time slots within a given period of time, exposing a tradeoff. The channel probability distribution controls the contention among the nodes. When the number of nodes with messages is large, assigning small probabilities to the higher priority channels ensures lower contention for those channels. This increases the chance that only one node chooses that channel. On the other hand, when the number of simultaneous messages is small, i.e., the load is low, assigning small probabilities to the higher priority channels could lead to under-utilization of these channels and higher utilization on the lower priority channels resulting in collisions and an increase in the overall latency. In the following section we construct a theoretical framework to select these design parameters to optimize the performance of our protocol.

3 Analysis and Design

For selection of the right design parameters to use in the Alert protocol we need to consider the probability of success that a message is sent successfully in a single slot and extend that to quantify the number of slots that will be required to read all outstanding messages. We begin our analysis by obtaining an expression for the success probability of a single slot.

Let $\mathbf{p} = (p_1, p_2, \ldots, p_M)$ represent a row vector of channel probabilities corresponding to each channel $1, 2, \ldots, M$. In this section, we assume that the same channel probabilities $\mathbf{p}$ will be used by each node in all slots; i.e., $\mathbf{p}$ is not updated during the protocol execution. This allows for a simpler implementation. Also, the results of this section can be built upon by more sophisticated approaches (we show one such approach in the next section).

We assume $n$ nodes are contending to send their message
interference on any of the channels. They select \( M \) channels based on the probability distribution \( p = (p_1, p_2, \ldots, p_M) \). Let random variables \( X_m \) (for \( m = 1, \ldots, M \)) denote the number of nodes deciding to transmit on channel \( m \). The variables \( (X_1, \ldots, X_M) \) will have the joint multinomial distribution

\[
P(X_1 = x_1, \ldots, X_M = x_M) = \begin{cases} \frac{n!}{x_1! \cdots x_M!} p_1^{x_1} \cdots p_M^{x_M}, & \text{if } \sum_{m=1}^{M} x_m = n, \\ 0 & \text{otherwise}, \end{cases}
\]

for non-negative integers \( x_1, \ldots, x_M \).

In order to model interference, we use an indicator random variable \( Y_m \) to specify if channel \( m \) has been interfered with or not, if and when the receiver samples the channel. We assume \( Y_m \)s are independent and have the Bernoulli distribution

\[
P(Y_m = y) = \begin{cases} Q & \text{if } y = 0 \text{ (no interference)}, \\ 1 - Q & \text{if } y = 1. \end{cases}
\]

Parameter \( Q \) represents the probability that a channel sees no interference. The value of \( Q = 1 \) corresponds to the ideal case of no interference on any channel. The assumption of independent interference on different channels is strengthened by the periodic switching of channels by all nodes in Alert protocol\(^5\).

Assume that channel \( c \) is the first channel, in order of priority, that is selected by at least one of the nodes and node \( u \) is transmitting on this channel. If any other node had transmitted or interference had caused a high signal on another channel before \( c \) the receiver would never have been waiting on \( c \) to receive \( u \)'s packet. If another node \( v \) also selects channel \( c \), the message from these two nodes will collide at the receiver. Ignoring the possibility of capture effect, the collision results in a slot failure with no message succeeding in the whole slot since the receiver will remain on channel \( c \) for the whole slot. Thus, \( u \) succeeds on channel \( c \) if it is the only node transmitting on that channel (with no interference as well). So we have the following rule: *If there is only one node transmitting in the first non-idle channel (in order of priority), an Alert time slot will be successful.*

So a time slot is successful if for any value of \( m \) (between 1 to \( M \)), only one node selects the \( m \)th channel (event \( E_1 = \{X_m = 1\} \)) and no node selects any of the first higher priority (\( m - 1 \)) channels (event \( E_2 = \{X_1 = \cdots = X_{m-1} = 0\} \)), and there is no interference on any of the \( m \) channels, i.e., \( E_{\text{int}} = \{Y_1 = \cdots = Y_m = 0\} \). So the probability of success of a slot is

\[
P_n(p) = \sum_{m=1}^{M} P(E_1 \cap E_2) P(E_{\text{int}}),
\]

where we assume that interference is independent of the activities of the node, and \( P(E_{\text{int}}) = P(Y_1 = \cdots = Y_m = 0) = Q^m \). Combining this with (1) we find the the probability of success of a slot for a given \( Q, M \), for known number of senders \( n \) and channel probability distribution \( p = (p_1, \ldots, p_M) \) as

\[
P_n(p) = n \sum_{m=1}^{M} p_m Q^m \left(1 - \sum_{k=1}^{m} p_k\right)^{n-1}.
\]

We use random variable \( T_n \) to denote the number of time slots required to collect \( n \) messages. Next, we find the distribution of \( T_n \) for a given probability distribution \( p \).

If we start with \( n \) nodes, in first time slot, with probability \( P_n \), one message is successfully received and \((n-1)\) nodes remain to go in the next time slots, or with probability \((1 - P_n)\) no node is successful and we still have \( n \) messages left. Hence, we can write the following recursive expression

\[
\xi(n, k) = P_n \xi(n - 1, k - 1) + (1 - P_n) \xi(n, k - 1),
\]

where \( \xi(n, k) \triangleq P(T_n = k) \) is defined as the probability that it takes \( k \) time slots to collect all messages from \( n \) nodes. We can solve (4) numerically using the following initial condition \( \xi(0, i) = \xi(i, 0) = 0 \) for all \( i = 1, 2, \ldots \), and \( \xi(0, 0) = 1 \).

We define the moment generating function (MGF) of \( T_n \) as \( \Phi_n(z) = E(z^{T_n}) = \sum_{k=0}^{\infty} P(T_n = k) z^k \), then from (4) we get

\[
\Phi_n(z) = \prod_{k=1}^{n} \frac{P_k z}{1 - (1 - P_k) z}.
\]

This shows that the random variable \( T_n \) is the sum of \( n \) independent Geometrically distributed random variables with parameter \( P_k \), i.e.,

\[
T_n \sim \sum_{k=1}^{n} \text{Geom}(P_k)
\]

where the Geometric distribution \((X \sim \text{Geom}(\alpha))\) shows the number of Bernoulli trials needed to get the first success with \( P(X = k) = \alpha (1 - \alpha)^{k-1} \) for \( k = 1, 2, \ldots \), \( E(X) = \frac{1}{\alpha} \), and \( \Phi_X(z) = E(z^X) = \frac{\alpha z}{1 - (1 - \alpha) z} \). So from (6) we have

\[
E(T_n) = \sum_{k=1}^{n} \frac{1}{P_k}, \quad \text{Var}(T_n) = \sum_{k=1}^{n} \left( \frac{1 - P_k}{P_k^2} \right)
\]

There is a simple intuitive explanation behind the distribution in (6): When we start with \( n \) nodes, the chance of success in each slot is \( P_n \), so it takes Geom(\( P_n \)) trials/slots for the first message to go through. With the first message received, there remain \((n-1)\) senders/messages, so the second message requires an extra Geom(\( P_{n-1} \)) slots. So in general the \( k \)th message requires Geom(\( P_{n-(k-1)} \)) slots.
3.1 Optimum Probability Distribution

The optimum distribution p depends on the available information about number of nodes and interference level, and the performance metric that is optimized. In this subsection, we present different methods to find the distribution p and rationale behind each case.

Case 1) Maximizing $P_n$ for a given $n = N$:

If we know (or estimate) the number of senders to be $N$, we can select $p$ such that the probability of success of one time slot is maximized for the given value of $n = N$. By maximizing $P_n$ we guarantee that the average delay of receiving the first message is minimized when $N$ nodes are contending to send. The number of time slots needed to successfully receive the first message is a random variable with geometric distribution and mean $1/P_n$. The optimum distribution $p$ in this case can be found using the following recursive expression:

$$p_m = \left( \frac{Q - \gamma_m}{NQ - \gamma_m} \right) \left( 1 - \sum_{k=1}^{M-m-1} p_k \right) \tag{8}$$

where $\gamma_1 = 0$ and for $m = 2, \ldots, M-1$, we have

$$\gamma_m = Q^{N+1} \left( \frac{N - 1}{NQ - \gamma_{m-1}} \right)^{N-1} \tag{9}$$

This case is very similar to the results in Sift [2] and our result is a generalization of their solution with addition of the effect of interference through parameter $Q$. For proof of the above results please refer to [3].

Case 2) Minimizing $E(T_n)$ for a given $n = N$:

Here, we find the probability distribution $p$ such that the expected number of time slots needed to collect from $n = N$ senders is minimized. In this way, we guarantee that all messages are collected in as few time slots as possible. The optimization problem can be solved by using numerical methods and gradient descent techniques [11].

Case 3) Maximizing $\min P_n$ for a given range for $n \leq N$:

If we do not know the number of nodes in the system, we can select the distribution $p$ such that the probability of success of a time slot is high across a given range. Essentially we try to find the solution which maximizes the minimum value of $P_n$ in the given range, i.e., maximize $\min_{n=1}^{N} P_n$. We used sub-gradient method [11] to solve this optimization problem numerically.

To compare the three different methods we present a numerical example. Figure 3 shows the average number of time slots required to receive the first message or all the messages in the network for the three cases. The results are calculated for $N = 50$ messages, with $M = 3$ channels, and $Q = 0.95$ (interference probability of 5%). Note that there are two variables representing number of nodes or senders: $N$ denoting the estimated number of senders based on which the distribution $p$ is calculated and $n$ which is the actual number of senders (the horizontal axis). For correct number of senders $n \approx N$, we see that case 2 (minimizing $E(T_n)$) achieves the smallest number of time slots (on average) to collect all messages; case 3 follows closely, then we have case 1. This is expected as the optimization parameter in case 2 is the delay to collect all nodes. Case 1 (maximizing $P_n$) gives the best delay for collecting the first message at $n \approx N$ and for $n > N$, but for smaller values $n < N$ the delay of the first message and the overall delay to collect all messages are worse than the other two cases. Case 3 (maximizing $\min_{n=1}^{N} P_n$) gives an overall good performance between the two other cases and tries to keep delay of collecting all messages and the delay of the first message low for the whole range of $1 \leq n \leq N$.

Asymptotic limit of $P_n$ for $n \to \infty$:

It is interesting to see if $n \to \infty$, how the probability of success in a time slot scales. If we fix the probability distribution $p$, it is easy to see that $\lim P_n = 0$ as $n \to \infty$. However, if we let the distribution $p$ to change with $n$, we can get non-zero limits for the probability of success. These limits represent the asymptotic bounds on the performance of Alert. The distribution $p_n = (p_1^{(n)}, \ldots, p_M^{(n)})$ which gives a non-zero limit $\lim_{n \to \infty} P_n^{(p_n)}$ should have the following form: $p_m^{(n)} = \alpha_m/n$ for $m = 1, \ldots, M-1$, and $p_M = 1 - \sum_{k=1}^{M-1} p_k$, which gives

$$P_\infty \triangleq \lim_{n \to \infty} P_n^{(p_n)} = \sum_{m=1}^{M-1} \alpha_m Q^m \exp(-\sum_{k=1}^{m} \alpha_k). \tag{10}$$

Figure 3. Expected number of time slots to collect all message or first message.
We claim that the values of $\alpha_m$ which maximize $P_\infty$ can be found from the recurrence $\alpha_m = 1 - Q e^{-\alpha_{m+1}}$ for $m = 1, \ldots, M - 2$, with $\alpha_{M-1} = 1$. These values give the following asymptotic (upper) bound on the probability of success of a time slot

$$P_\infty = Q e^{-\alpha_1} = \frac{Q}{e} \frac{\alpha_1}{(M-2)}$$ \quad (11)

For proof of above result, please refer to our report in [3].

We get close to this asymptotic bound for relatively small values of $n$, e.g., with $Q = 0.9$ and $M \geq 3$ the maximum possible $P_n$ is close to $P_\infty$ for $n \geq 20$.

### 3.2 Optimum Number of Channels

When multiple nodes contend to send messages in a slot, a larger value of $M$ (larger number of channels) decreases the contention among them by increasing the likelihood that they select different channels. Thus, one would think that we should use as many channels per slot as we can. However, there are practical considerations that present a tradeoff. For each channel used, the receiver has to sample it, and switch to the next channel. Thus, for each channel used, there is a sensing plus switching delay added to the size of a time slot. On one hand, as we increase $M$, the length of each time slot is increased, on the other hand, with larger $M$ we can get better success probabilities $P_n$ and therefore we can collect all messages in fewer time slots. Thus, selecting $M$ poses a tradeoff. We select $M$ to optimize this tradeoff and minimize the overall delay.

To find the optimum $M$, we need to have some timing constants from the radio. In general, we can write the length of a time slot as $\tau_{\text{tot}}(M) = M \tau_1 + 2 \tau_2$, where $\tau_1$ specifies the time duration required by the receiver to sample the presence of tone/preamble on one channel and switch to the next channel, and $\tau_2$ is the time for completion of the packet, ack message and other constant times in one time slot (see Figure 8 for more details). Since the number of channels is a positive integer value and bounded by a small number (total number of channels), we elected to pick the optimum $M_{\text{opt}}$ (and corresponding $p_{\text{opt}}$) as the one which minimizes the expected value of overall delay to collect from $N$ nodes, i.e., $E(D_N) \triangleq E(T_N) \times \tau_{\text{slot}}$.

Figure 4 shows the normalized delay ($E(D_N)/N$) as a function of $M$ for $Q = 0.95$. For timing constants, we used $\tau_1 = 0.4 \text{ ms}$ and $\tau_2 = 6.0 \text{ ms}$ which are representative values based on our measurements with the implementation on Bosch CC2420 node boards (see Section 5). The optimum $M$ which minimizes the delay is shown on each graph by a filled marker. The optimum value of $M$ depends on value of $N$ and $Q$. However, it is not very sensitive to $N$ as we see that close to $M_{\text{opt}}$, the curves are flat. So we can increase or decrease $M$ by one or two and expect to get almost the same performance.

### 4 Adaptive Algorithm for Alert

The previous section presented different theoretical approaches to minimize the number of time slots to collect all messages. These approaches focused on using a single set of channel probabilities $p$ throughout the execution of Alert. Though this results in a simpler implementation, we can do better by adapting these probabilities as the protocol executes. We present one such approach in this section where the probabilities to use are updated every time unit $t$. We thus seek a set of channel probabilities used throughout the protocol execution represented as a list of vectors $P = (p^{(1)}; \ldots; p^{(F)})$, or equivalently in matrix form

$$P = \begin{pmatrix} p^{(1)}_1 & \cdots & p^{(1)}_M \\ \vdots & \ddots & \vdots \\ p^{(F)}_1 & \cdots & p^{(F)}_M \end{pmatrix}$$

where $F$ is the number of time units required for reading all messages, with a new set of channel probabilities calculated for each unit. A time unit can be either a single time slot or a whole frame of time slots\(^7\) in the context of Alert. In this section we specifically look at the case where no knowledge of number of messages is assumed and present an adaptive

\(^7\)The actual implementation of Alert is based on the concept of frames. A specific amount of time per frame is allocated for reading messages, with the rest used for other operations like time synchronization, maintenance among others. This time can be divided up into any number of slots depending on the size of a slot.
algorithm (Algorithm 1). We believe this to be the most important case to handle for the protocol. For simplicity, we assume that all initial messages arrive simultaneously. This can be justified when messages are rare, but correlated, and usually occur due to some event observed by a subset of the nodes resulting in the generation of simultaneous messages. Generation of alarms by surveillance systems is such an example.

**Calculation of Channel Probabilities**

The distribution $p^{(t)}$ controls the contention of nodes with messages to send in a time unit $t$. Given $M$, the $p^{(i)}$ used for each time unit must control contention such that the probability of success in that time unit is maximized. As the protocol progresses, maintaining our assumption of simultaneous arrivals of messages, the number of remaining messages, $n$, decreases as they are successfully received by the base station. Thus, for subsequent slots, after the first slot, a new value of $p^{(i)}$ must be calculated before each slot $t$ taking into account the current value of $n$. For large initial $n$ the time to read all messages can be quite large, and the requirement of recalculation of $p^{(t)}$ before each slot $t$ can prove infeasible. Moreover, we would prefer that the values of $p^{(i)}$ used in each slot be pre-calculated and stored in memory. Thus, to the computational and memory limitations on wireless sensor devices, we adopt a per-frame recalculation strategy.

Let $R_n$ be the number of messages that go through in frame $t$ with $n$ messages contending at the beginning of the frame. We seek for frame $t$ the channel probabilities $p^{(i)}$ that maximize the number of messages read, i.e., the channel probabilities to use in a frame satisfies

$$\max_{p^{(i)}} E(R_n)$$

This can be solved through numerical methods using a recursive algorithm or through Markov chains in conjunction with equation (8). The distribution $p^{(t)}$ is updated at the beginning of each frame $t$. The number of remaining messages is updated at the end of each frame by subtracting the expected number of messages read in the frame for the $p^{(i)}$ used in the frame. Thus, if $M$ is the number of channels used in a slot and $F$ is the number of frames that are expected to be required by the protocol to read all messages, the distribution $P$ is in the form of a $F \times M$ matrix. Each row of $P$ is the distribution to use in the frame corresponding to that row. The steps to calculate $P$ are given as Procedure `getP`.

This procedure calculates the number of frames required to read all messages up to some small threshold, $N_{thr}$, and then repeats the last $p^{(i)}$ in the final frame $F$. The constant $N_{thr}$ is used to ensure that an under-estimate of number of messages remaining (compared to the actual value) does not result in using channel probabilities that will make it almost impossible for further messages to go through.

**Adaptive Algorithm Details**

When the actual number of messages to be sent is unknown, we desire that the protocol itself try to estimate it and calculate the corresponding design parameters for this estimate. We employ an adaptive approach where we vary a node’s estimate of number of senders, $N$, until it succeeds in sending its message. The algorithm starts out by selecting the number of channels to use per slot, $M_{opt}$, based on some guess on the initial number of nodes (or messages with our assumption of one message per node), $N_{guess}$ which need not be accurate due to insensitivity of $M$ to number of nodes. Further details of calculation of $M_{opt}$ are explained in Section 4. The algorithm sets its estimate of number of messages, $N$, to a small value $N_{init}$ as the starting point of its upward adaptation to larger values. Upward adaptation is chosen because changing to a different estimate only requires waiting a small time before realizing that the estimate may be incorrect. In the next step it finds the optimal channel probabilities, $P$, and the expected maximum number of frames $F$ within which the message will be read for this estimate. If, after $F$ frames of protocol execution, the node’s message is still not successful, it increases the estimate $N$ by a constant additive factor $C_{incf} (C_{incf} > 0)$ and re-calculates $F$ and $P$ to use in the subsequent frames.

The algorithm continues until the message is finally read, after which the node resets back to the initial state and on the next event will re-execute the algorithm. The algorithm is designed such that all the calculations of $M_{opt}$ and $P$ can be pre-stored and used from memory. The same initial $N$  

---

**Procedure `getP`**

1: for each frame $t$ do
2: Calculate (and store) $p^{(i)}$ for the given $M$ with current value of $n$ using equation (12)
3: Reduce $n$ by expected no. of messages read so far, i.e.,
   \[ n = n - E(R_n) \]
4: if $n \leq N_{thr}$ then
5: Store number of frames $F = t + 1$
6: Repeat last $p^{(i)}$, i.e., $p^{(F)} = p^{(i)}$
7: Break from for loop
8: end if
9: end for
10: Return $F$ and $P = (p^{(1)}; \ldots; p^{(F)})$
found that optimal
From our earlier analysis (mentioned in Section 3.2) it was
the protocol execution, as adaptive estimates on
possibly larger number of channels to use without incur-
re-calculate
Number of Channels Per Slot
When the Alert protocol is deployed, we desire that only
a single value of $M$ be used. Changing the number of
channels per slot dynamically as number of messages $N$
remain to be read changes is difficult to implement in prac-
tice. Next, we describe our method to derive $M_{\text{opt}}$ given
in Procedure get\(M\). It begins with some initial guess of $N$,
$N_{\text{guess}}$, not necessarily close to the actual $N$, but not
a small value. The optimal number of channels $M_{\text{opt}}$
to use is calculated based on this $N_{\text{guess}}$. Because we do not
re-calculate $M_{\text{opt}}$, we need to ensure that the initial value
of $N_{\text{guess}}$ used gives us a good $M_{\text{opt}}$ to use for the rest of
the protocol execution, as adaptive estimates on $N$ changes.
From our earlier analysis (mentioned in Section 3.2) it was
found that optimal $M$ is quite insensitive to $N$.

The calculation of $M_{\text{opt}}$ uses the same method explained
in Section 3.2 but with a small difference. Assume $M_{\text{temp}}$
is the optimal value of $M$ for reading all messages with
minimum delay. But this value may not be good for getting
the first message through with minimum delay, for which
a larger number of channels might be better. So we intro-
duce a design factor $\beta, \beta \geq 0$ by which we look for
possibly larger number of channels to use without incur-
re-calculate

```
Algorithm 1 Adaptive Alert Protocol
1: Get $M_{\text{opt}}$ for some guess on number of nodes $N_{\text{guess}}$.
    $(M_{\text{opt}}) = \text{get}\(M(N_{\text{guess}})\).
2: Start with some initial value of $N = N_{\text{init}}$
3: Get some estimate on value of $Q$.
4: Get $P$, and expected number of frames needed, $F$,
based on current estimate $N$ and for $M_{\text{opt}}$,
    $(P, F) = \text{get}\(P(N, M_{\text{opt}})\)$
5: while message not read do
6:   for frame $t = 1$ to $F$ do
7:       for all slots in frame $t$ do
8:           Execute protocol for calculated $p(t)$ for frame $t$
9:           Terminate protocol if message read and reset
to initial state
10:   end for
11: end for
12: Increase estimate by a constant factor $N = N + C_{\text{incf}}$
13: Get $P$, and expected number of frames needed, $F$,
based on current $N$ for $M_{\text{opt}}$
    $(P, F) = \text{get}\(P(N, M_{\text{opt}})\)$
14: end while
```

and constant increase factor $C_{\text{incf}}$ ensures that we just need
to store $M_{\text{opt}}$ calculated for some constant $N_{\text{guess}}$, and $P$,
$F$ for all estimates $N$ which can only have the following
values: $\{N_{\text{init}}, N_{\text{init}} + C_{\text{incf}}, N_{\text{init}} + 2C_{\text{incf}}, \ldots\}$ up to
some maximum limit on the value of $N$ or memory capacity
available.

Procedure get\(M(N_{\text{guess}})\)
1: Use timing constants $\tau_1$ and $\tau_2$ of specific radio used
2: for each $M \leq M_{\text{max}}$ do
3:   $\tau_{\text{slot}}(M) = M\tau_1 + \tau_2$
4:   Get the expected number of frames needed, $F(M)$,
based on value of $N_{\text{guess}}$.
    $(P, F(M)) = \text{get}\(P(N_{\text{guess}}, M)\)$
5: end for
6: Get optimal $M$ for reading all messages:
    $M_{\text{temp}} = \text{arg\ min}_M F(M) \times \tau_{\text{slot}}(M)$
7: Get associated time to read all messages:
    $t_{\text{val}}^{\text{M_{\text{temp}}}} = F(M_{\text{temp}}) \times \tau_{\text{slot}}(M_{\text{temp}})$
8: Find largest $M$ within some factor $\beta$ of $t_{\text{val}}^{\text{M_{\text{temp}}}}$:
    $M_{\text{opt}} = \text{max}\{M \mid t_{\text{val}}^{\text{M_{\text{temp}}}} \leq (1 + \beta) t_{\text{val}}^{M_{\text{temp}}}\}$
9: Return $M_{\text{opt}}$

where $t_{\text{val}}^{M_{\text{temp}}}$ is the expected time to read all messages
using $M_{\text{temp}}$. Parameter $\beta$ allows us to control our optimiza-
tion criteria: $\beta = 0$ specifies selection of $M_{\text{opt}}$ that
optimizes the time to read all messages, while larger values
of $\beta$ increasingly look to optimize the time to send the first
message by considering usage of larger number of channels.

5 Protocol Evaluation
In this section we evaluate the feasibility and perform-
ance of Alert through an implementation on commodity
hardware and also simulations. We begin with our imple-
mentations focusing on the feasibility of Alert.

Feasibility of Alert
Alert protocol is implemented on Bosch CC2420-based wireless
nodes. The Bosch boards use Chipcon/TI CC2420
radio which is an IEEE 802.15.4 compliant transceiver op-
rating at 2.4GHz band with Direct Sequence Spread Spec-
trum (DSSS) O-QPSK modulation and 250Kbps data rate.
An external power amplifier (max transmit power 10mW)
is used to improve the communication range.

In the first experiment set up we deployed 15 senders in
our office in Palo Alto, CA, as shown in Figure 5. The re-
ceiver (base station) is in communication range of all nodes
and it kept them in sync by sending periodic beacon mes-
sages. Every second, all the nodes sent a message simulta-
neously using Alert protocol with the following fixed prob-
ability distribution: $p = (0.05, 0.063, 0.092, 0.182, 0.613)$. The receiver measured the number of time slots to receive
the first message and number of time slots to collect all the
15 messages. Each time slot is 8ms long.

Figure 7 shows the measured distribution of the number of
time slots (for both the first and all messages). We see
that the Alert protocol is performing better in real deploy-
ment (the experiment setup) than what the analysis predicts.
The calculations show that on average we need 24.82 time slots to collect from all 15 nodes, but our experiment gives an average of 17.60 time slots. This improvement in performance is mainly due to Capture effect. When two nodes are sending simultaneously, in our analysis we assume that there will be a failure, but in many cases the receiver can correctly decode one of the packets while treating the signal from the other sender as noise. Since the CC2420 radio employs spread spectrum techniques, it can tolerate higher level of interference and this helps increase the chance of capture effect. Note that Alert, by reducing the number of contending nodes at higher priority channels, increases the likelihood of capture effect.

To validate our analysis we reduced the chance of the capture effect by repeating the experiment with a different set up. In the second set up, all the 15 nodes were placed close to each other and close to the receiver on a table. The network configuration is shown in Figure 6. Since all nodes are close to one another, the received power at the receiver from all senders is high and equal. This reduces the chance of capture effect. The results are shown in Figure 7. We see that the measurements distribution with the second setup matches very closely to what the analysis predicts.

**Simulation Setup**

Using simulations enables us to evaluate the performance of Alert with a large number of nodes with messages to send i.e. for scalability, and also compare against other protocols. We compare Alert with two other contention based MAC protocols - Sift [2] and Slotted Aloha (S-Aloha) [4]. Sift was chosen because it is a CSMA based protocol (unlike Alert) and previously shown to do better than variable contention window protocols like 802.11 for the target application scenario (refer Section 6 for more details). S-Aloha is a simple protocol, allowing each time slot to be very small, possibly providing advantages in reducing time required to read messages. We believe comparisons of Alert with these two protocols covers a wide design space for MAC protocols for the target application. We further discuss the infeasibility of other possible MAC protocols (e.g., TDMA) in Section 6.

For the evaluations, we wrote a simulator in Matlab. The important abstraction was the concept of time across different protocols from a implementation perspective. The interference was modeled as pointed out in Section 3. All protocols send messages to the receiver (or base station) in fixed time slots. The size of a time slot for each protocol is different (but of fixed size) based on how it is used. The timing of all three protocols as used in our simulations are shown in Figure 8. In this figure, $t_1$ is the guard time plus the rx/tx switching time, $t_{Skew}$ is the maximum clock skew, $t_3$ is the channel sensing time, $t_3$ is the channel switching

---

9The 802.15.4 protocol also has a fixed slot structure with both a contention access period and a contention free period within each frame [5].
time, and $t_4$ is the total time to exchange a packet and ack. Based on the implementation of Alert and measurement on the CC2420 radio, the values used for these constants are: $t_1 = 0.5 \text{ ms}$, $t_2 = 0.1 \text{ ms}$, $t_3 = 0.3 \text{ ms}$, $t_4 = 2.5 \text{ ms}$.

In the S-Aloha Protocol, each node tries to send its message in a slot with probability $1/N$ until it finally succeeds in doing so, where $N$ is the current estimate of number of messages. We let the S-Aloha protocol use the same methodology of adapting $N$ as Alert, and chose an initial value and increment factors that gave best results; this was initial $N = 10$ with additive increments $C_{incf} = 50$. For S-Aloha, a slot duration consists of the guard time plus RX/TX switching time $t_1$, a single adjustment for clock skew $t_{Skew}$ and the time to exchange a Packet and Ack.

Sift uses a fixed contention window (CW) size, and relies on a geometric probability distribution with which nodes pick a backoff slot for transmission. Once a node counts down to its chosen backoff slot, and is the only one that has chosen that slot, it completes the packet and ack exchange with the receiver and all nodes move onto the beginning of the next protocol time slot. For simplicity we do not implement RTS/CTS with Sift and do not consider the hidden terminal effect\textsuperscript{10} in our evaluations. The Sift slot duration consists of the guard time $t_1$, the length of each backoff slot which is the sum of adjustment for clock skew and the channel sensing time, $t_{Skew} + t_2$, and the time to exchange a packet and ack, $t_4$.

For Alert, a slot duration consists of the guard time $t_1$, adjustment for possible clock skew $t_{Skew}$ both before and after the sampling time by receiver, multiple copies of time to sample channel plus channel switching time, $t_2 + t_3$, and the time to exchange a packet and ack, $t_4$. In our simulations, $N_{\text{init}}$ was taken as 10 with additive increments $C_{incf} = 50$. The value of $M$ was calculated for $N_{\text{guess}} = 50$ and $N_{\text{thr}} = 10$. The value of $\beta$ was set to 0.1 which provided a good balance between minimizing delay of collecting first message and collecting all messages.

Two levels of time synchronization were considered; tight and loose. Note that these are relative terms that are used to convey the compensation required for expected clock skew within slots. The tight synchronization allows smaller compensation times to be used with all protocols, but can prove to be a heavy burden on the higher level protocol that is responsible for it. Tight synchronization would require all nodes to participate in the synchronization protocol more frequently and would consume a lot of energy, even when the nodes have no messages to send. Thus, for applications targeting rare events, tight synchronization may not be feasible and a 'looser' form of synchronization may be more desirable. We use the values $t_{Skew} = 0.7$ or $0.2 \text{ ms}$ for loose and tight synchronization respectively\textsuperscript{11}.

To get a sense of the effectiveness of the adaptive version of Alert, we also show a plot of the expected number of slots required to read all messages when exact value of $N$ is known throughout the protocol execution. This is calculated theoretically using (7) for a known $N$. The scheme is termed $ExOptAlert$. Each data plot shown is the average of 150 runs and 95% confidence intervals are shown for plots that show time required to read all messages. The maximum number of nodes was set at 100 which is a reasonably large number for a one hop centralized network.

**Comparisons through simulations**

Figure 9 shows the comparison\textsuperscript{12} between all 4 schemes for the average time required to send the first message. It can be seen that Alert manages to send the first message out far earlier than Sift and S-Aloha and is quite close to its optimal expected performance $ExOptAlert$. The chosen channel probabilities of Alert allow the first message to go through in the initial few slots. The same happens for Sift, but more backoff slots in its contention window mean it takes more time to send the first message even though it may be successful in the first time slot. A smaller contention window for Sift could be useful here, but that could have a negative impact on success probability of a single slot and, hence the delay to send all messages. S-Aloha seems to have the most delay since the random slots picked by nodes to send may not be the initial slots, or if they are, may not be successful due to collisions. The small slot time does not seem to have helped S-Aloha in this case.

Figure 10 shows the comparisons for all schemes to read all messages when $Q = 0.95$ (interference level of 5%). For the tight synchronization case we see that Alert does slightly

\textsuperscript{10}We will mention how that consideration would effect the comparison between the protocols when we present our results.

\textsuperscript{11}Note that, for tight synchronization, the values used ($t_{Skew} = 0.2, t_2 = 0.1 \text{ ms}$) gives roughly the specified backoff slot duration of 0.32 ms in the IEEE 802.15.4 standard [5].

\textsuperscript{12}Confidence intervals are not shown for this plot to allow a close up snapshot of the schemes other than S-Aloha.
better than Sift. Note that this result does not take into account additional procedures like RTS-CTS which Sift might need to employ to handle hidden terminal collisions. Alert being a non-carrier sense protocol does not suffer from such issues. When loose time synchronization is used, the difference between Alert and Sift increases; in fact, S-Aloha does better than Sift now due to the much smaller slot structure it uses. When a higher level of interference \((Q = 0.8)\) is taken into consideration Sift does better than Alert for the tight synchronization case because the latter has a higher possibility to be affected due to its use of multiple frequency channels (see Figure 10). The possibility of such high levels of interference \((Q = 0.8 = 20\%\) interference) is, however, very unlikely. In practice, Alert switches the frequency channels it uses periodically so that an interference source on some channel does not affect performance for long.

6 Related Work

The related MAC protocols for wireless sensor networks can be mainly classified into contention free, contention based and energy saving protocols. Contention free protocols are mainly the ones based on Time-Division Multiple Access (TDMA) where slots are assigned to each node by the base station and each node sends its message (if it has one) only during its assigned slot (e.g., GUARD [6]). Such TDMA based protocols perform very poorly when the number of nodes contending is unknown or keeps varying. For the specific application targeted in this work, only a subset of nodes may have events to report. This makes assigning slots to all nodes undesirable as it leads to a significant increase in delay to receive first message. For example, consider the case of only one node having a message to report. The average delay incurred to collect this message would be half the TDMA cycle with the worst case delay being the whole cycle. TDMA schemes are useful in cases where most of the nodes have events to report, a rare case for our scenario. Other contention free approaches, e.g., Frequency-Division Multiple Access (FDMA) face similar limitations as outlined above.

Contention based protocols can be bifurcated into Carrier Sense Multiple Access (CSMA) based or non-CSMA based. The IEEE 802.11 and 802.15.4 protocols are examples of CSMA protocols with the latter designed specifically for applications catered to by wireless sensor networks [5, 7]. They use a variable sized contention window whose size is adjusted at each node based on the success of the node in sending its message, with each node picking a slot in this window using a uniform probability distribution. These protocols do a good job in handling scenarios with small number of nodes but do not handle a large number of simultaneous messages well. The Sift protocol was designed to overcome these deficiencies for WSN applications.
which need to handle such large number of event driven spatially co-related messages. Sift is also CSMA based but uses a fixed size contention window. Nodes pick slots from a geometric probability distribution such that only a few nodes contend for the first few slots, and thus handles a large number of messages easily. Protocols based on Aloha on the other hand do not sense the channel before transmission and rely on each node picking a slot to transmit on randomly, with the probability of transmission depending on number of messages contending [4]. When this number is not known, these protocols do not adapt well. In general, CSMA based protocols outperform Aloha based protocols when the propagation time between nodes is small enough to make carrier sense useful. Alert is similar to Sift in that it uses a similar non-uniform distribution to control contention among nodes, but is non-CSMA based. Alert separates message transmissions across different frequency channels with this distribution while Sift does it over different time slots. This allows Alert to be free of hidden terminal issues while Sift is susceptible\(^\text{13}\). Alert extends this distribution to handle different interference levels as well. While Sift is also optimized for unknown number of messages, the adaptiveness of Alert allows it to perform well even when the number of contending messages is small.

The topic of energy saving MAC protocols has been well researched for wireless sensor networks in the last few years primarily due to the limited energy supply in these devices [8,9,13]. Our work focuses on applications for which latency is the primary concern and the goal is to let all urgent messages reach a receiver node as soon as possible. Energy is a concern as well, but the rare occurrence of messages in these applications allows the MAC protocol to focus solely on the latency aspects. With such applications, energy efficiency should be built into more common tasks carried out by each node like time synchronization. Some researchers have focused on reducing latency in the collection of messages at a sink node from source nodes through duty cycling approaches (e.g., [10,12]). These types of low-latency protocols solve a fundamentally different problem which is to sleep as often as possible while ensuring messages reach the sink as fast as possible (with latencies in the order of seconds instead of milliseconds). Our work focuses on event driven message generation where the receiver is always awaiting messages. The transmitters try to send messages when they have one, and the only latency that needs to be reduced is the one created due to contention with other nodes trying to send messages at the same time.

7 Conclusions

We presented Alert, a MAC protocol to collect rare event driven messages from multiple wireless sensor nodes with low latency. The feasibility and effectiveness of the protocol were demonstrated through both an implementation as well as extensive simulation based comparisons with other protocols.

References


\(^{13}\)Aside from the performance perspective, operations performed at the sender side in Alert are comparatively much simpler than Sift or any other CSMA-based protocol. So fewer resources (less memory and less computational power) are required for implementation of Alert at individual nodes. This makes Alert a more cost-effective solution.