i-MAC - A MAC that Learns

Krishna Kant Chintalapudi[*]
Microsoft Research India,
Bangalore, India

ABSTRACT

Traffic patterns in manufacturing machines exhibit strong temporal correlations due to the underlying repetitive nature of their operations. A MAC protocol can potentially learn these patterns and leverage them to efficiently schedule nodes' transmissions. Recently, with the advent of low power MEM based sensors, wireless sensing in these machines has gained prominence. Communication in control loops must cater to extremely low hard real-time latencies while embracing low-power design principles. In this paper, we present a novel MAC, *i*-MAC, a wireless MAC protocol that learns to expect and plan for traffic bursts and consequently coordinate node transmissions efficiently.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Algorithms, Design

1. INTRODUCTION

Manufacturing systems (*e.g.,* assembly lines, packaging machines *etc.*) are ubiquitous and form the cornerstone of perhaps every major industry around the world today. In countries such as the USA and Germany, where the manufacturing sector accounts for over a trillion dollars of their Gross Domestic Product (GDP), rely heavily on automated assembly lines that operate round the clock for several days, or even months, with limited or no human intervention.

Central to the operation of a modern day manufacturing machine is the communication system in its control loop that connects sensors *e.g.,* temperature, pressure, proximity sensors) and actuators (*e.g.,* drills, conveyers, robotic arms, welding units) to a central controller. Typical modern day assembly lines make use of wired communication solutions such as EtherCat[14] and Profibus [12] *etc.*.

Recent advances in low-power MEMs based sensors and low-power radio platforms (*e.g.,* CC2420 [22]) have spurred immense interest in low-power wireless sensing platforms. A typical modern day manufacturing machine has a large number of sensors (between 50-200) and enabling wireless sensing in control loops of machines has the potential to offer a number of benefits. First, a wireless solution promises cost reduction due to elimination of communication cables and expensive I/O hardware to and from these large number of sensors. Second, often machines have rapidly moving parts, and cables drawn from these parts are subject to repetitive stress resulting in frequent cable wear and tear based faults. A wireless solution can eliminate maintenance costs resulting from such faults. Third, It can reduce costs through its ease of installation, deployment and maintenance since it does not involve cumbersome installation procedures that require skilled labor.

Actuators (*e.g.,* drills, robotic arms) typically perform power intensive operations and are always tethered to a power source via cables. Since communication cables can typically be "bunched up" with power cables, the benefits from enabling wireless communication to actuators may not be "significant." *Thus, in this paper we shall focus on enabling wireless sensing in manufacturing machines.*

Given the highly competitive nature of the manufacturing industry, mechanisms that can result in even modest cost savings, can provide an edge. Despite the advantages, however, modern day manufacturing systems still shy away from employing wireless communication in their control loops. The reason for this is simple - *existing wireless standards do not satisfy the stringent communication requirements demanded by control loops in automated manufacturing.*

How a Manufacturing Machine Works

Most manufacturing machines are discrete event control systems. Sensors detect discrete events and notify a central controller. The central controller is a finite state machine, which updates its state based on the sensory notifications and induces the required actuation.

Figure 1 illustrates the operation of a section of an assembly line. This rather simple example has been specifically chosen to provide the reader an intuition regarding the functioning of typical machines. In (A) an infrared sensor (IR1) detects the arrival of the product on the conveyer and notifies the controller. Upon receiving the notification from IR1,

[*]This work was done at Robert Bosch Research and Technology Center, Palo Alto, CA

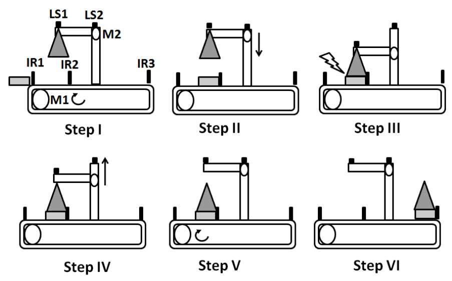


Figure 1: Assembly line operation example

the controller then starts the conveyer motor M1 to move the product to place it under the robotic arm. In (B), infrared sensor IR2 detects and notifies the controller of the arrival of the product under the robotic arm. Upon receiving notification from IR2, the controller immediately stops the conveyer and starts motor M2 of the robotic arm to move down with the part to be joined. In (C), as the surfaces of the two parts come into contact, the limit switch LS1 on the robotic arm is triggered and this is conveyed to the controller. The controller immediately stops the movement of the robotic arm and starts the gluing process. In (D), after the gluing, the controller starts the robotic arm motor to move upwards to its original position while the conveyer is started to move the product to the end. In (E), and (F) the limit switch LS2 is responsible for indicating that the robotic arm has reached the top while the infrared sensor IR3 senses the arrival of the product at the next stage.

Communication Requirements

There are three main requirements that need to be satisfied in order to make a wireless solution viable.

Hard Real Time Delay Bounds : Communication in control loops of manufacturing machines must cater to hard real time delay bounds *i.e.,* messages must reach their destination within a pre-specified delay bound. Messages that reach after the pre-specified delay bound may either lead to defective products or cause a serious disruption in the production process, leading to a need to flush the pipeline and restart. Such disruptions translate to significant monetary losses through reduction in production throughput.

To understand the need for hard real time delay bounds, consider a simple example. Suppose that in Figure 1 (B), the conveyer were moving at 1 m/sec. Let the delay between the time the product arrives at IR2 and the time the conveyer motor M1 stops be d ms. During this time, the conveyer would have moved d mm. This, in turn, means that the positioning error for the product would be d mm - in other words the parts would be joined d mm *off* the correct position. A maximum allowable positioning error bound then translates directly to a hard real time delay bound.

Ultra-low Communication Error Rates : The economic value of an automated manufacturing machine depends on its ability to manufacture large volumes and meet production demands. Machine operators will never accept a wireless solution that replaces the existing wired communication system at the cost of even minor losses in production throughput. The primary performance measure of a communication system in manufacturing machines is the *Communication Error Rate* (CER) - the probability that reliable message delivery latency exceeds the pre-specified hard real time delay bound. Consider a pipelined manufacturing machine that has 100 sensors and produces a product once every second. On an average, 100 sensors would send a message to the controller each second or about 8 million messages each day. 1ppm error then translates to about 8 errors per day per machine. Current day wired communication systems (*e.g.,* Profibus) provide CER in the range of 1ppm (10^{-6}) to 1ppb (10^{-9}) for hard real time delays in the range of 5-50ms. Any wireless solution must provide performance comparable to existing wired systems in ordered to be accepted by machine manufacturers.

Longevity : Frequent battery replacement in hundreds of wireless sensors in a machine can be a time-consuming, labor-intensive job and will offset all the gains obtained from a wireless communication system. In typical machines, sensors will be expected to operate without replacement for at least a few years. Consequently any wireless sensing solution in manufacturing machines must embrace low-power platforms (*e.g.,* low power transceivers) and protocol designs (allow for duty-cycling). A trivial power budget calculation will indicate that *the desired low-power radio platform must consume few ten milliwatts while transmitting/receiving. Further these platforms must have very small wakeup times (typically under 1ms) to enable fast duty-cycling.* Neither 802.11 based standards nor Bluetooth cater to these requirements [17],[9]. An example of a standard that does meet these requirements is IEEE 802.15.4.

What makes meeting these requirements hard?

A combination of three basic factors makes satisfying the above requirements a challenging problem.

Bandwidth Constraints of Low-Power Radios : Current day low-power radios typically do not provide high data rates similar to 802.11 based standards. For example, radios based on 802.15.4 such as the extremely popular CC2420 provide 250Kbps data rate. More recently radio low-power platforms with data rates of up to 2Mbps have started to become available. Often however, a higher data rate comes at the cost of decreased range and link quality.

Dense Node Placement : Most manufacturing machines span 2-10 mts along their longest dimension and house between 50-200 sensors within this small space. In other words, all the sensors are within one radio range of each other, leading to an extremely dense interference environment.

Bursty Traffic : Often in manufacturing machines, a single event might trigger several sensors at the same time. For example, a large number of proximity sensors may be used to ascertain orientation of a work-piece. In addition, manufacturing machines are typically pipelined and designed to process multiple products simultaneously. Consequently *traffic bursts*, in which several sensors attempt to notify the controller at the same time, are quite common in many machines. *In the event of a traffic burst, messages from all sensors must reach the controller within the pre-specified delay bound.* Since all these sensors are typically within each other's radio range, traffic bursts leading to packet collisions can dramatically undermine performance.

Can't we simply use TDMA?

TDMA is perhaps the obvious choice when hard real time guarantees are desired; however, it does not scale very well for a large number of nodes. Consider a simple experiment using the CC2420 radio. Transmission of a typical sensor notification packet (comprising about 10-15 bytes including headers, data and trailers) will require about 800μsec [4]. A single TDMA slot, consisting of a packet transmission to the controller and an ACK from the controller will then be 1.6 ms long. The TDMA frame for a 100 sensor machine will thus be 160ms long (100 slots). Given that typical packet success rates in the wireless channel range 90% - 99.9%, 4-9 retransmissions may be required to guarantee a CER in the range of 1ppm-1ppb. Since a sensor has to wait for its turn in the next frame for a retransmission, 4-9 retransmissions translate to 4-9 frames (640-1600ms). This is clearly much greater than the desired 5-50ms. While recently some low-power radios have begin to offer up to raw data rates of 2Mbps, much higher rates are required to make TDMA based solutions viable for up to 200 nodes.

Existing Research in Low-Power MAC Protocols for Wireless Sensor Networks
Research efforts in the area of sensor networks have led to a large number of MAC protocols that are specifically targeted to increase the longevity of power constrained sensor nodes through aggressive duty-cycling techniques. One obvious problem that arises when sensor nodes are duty-cycled is that the receiver and transmitter must somehow coordinate so as to remain awake at the same time. B-MAC [18] tackles this problem by transmitting a long preamble prior to the packet. The duration of the preamble is longer than the duty-cycle period of the nodes. This allows the receiving nodes to sense the preamble and remain awake to receive the transmission. NanoMac [2] improves channel utilization for sense deployment scenarios by making use of RTS/CTS based mechanisms and novel algorithms for duty-cycling. S-MAC [23], T-MAC [5], P-MAC [24] rely on synchronizing the sleep and wakeup schedules of neighboring nodes and put the radios to sleep periodically (duty-cycling) while using mechanisms similar to RTS/CTS found in 802.11. WiseMAC [7], TRAMA [19] and μ-MAC [3] use spatial TDMA and np-CSMA, where nodes wake up based on schedules that are offset in order to avoid collisions. While f-MAC [21] provides collision free transmissions with guaranteed delay bounds over one-hop topologies, it does not scale to a 100 nodes. DMAC [16] is a spatial TDMA based MAC designed specifically for networks where sensors form a tree topology to transmit data to a base station. SIFT [13] uses a non-uniform probability distribution to pick transmission slots and exponentially adapts the transmission probabilities on noticing idle slots. PTDMA [8] and ZMAC [20] allow a seamless transition between TDMA and CSMA by assigning probabilistic ownerships to time slots that are adjusted based on the number of transmitters. RL-MAC [15] uses reinforcement learning to infer the traffic load conditions based on packet losses and adjusts the duty-cycle based on these estimates. *All of the above approaches are designed for generic multi-hop wireless sensor networks and are not specifically tailored to address the rather harsh requirements posed by manufacturing machines.*

Existing Approaches to enabling wireless sensing in manufacturing systems
Given the rather harsh set of requirements for manufacturing systems, some solutions ignore the power constraints and use high bandwidth wireless solutions hoping that the radios will be powered *e.g.,* Wireless Profibus [6]. Such solutions have found their place in certain niche applications. Some other solutions attempt to provide power wirelessly by using magnetic coupling *e.g.,* WISA [1]. These solutions significantly reduce the cost benefits due to their need for additional wireless power equipment. Other solutions such as Wireless HART [10] (based on 802.15.4 physical layer) uses TDMA based scheduling and cannot scale to dense deployments of 100s of nodes while adhering to hard real time latencies of a few ten ms.
Proprietary Low power Multi-radio Platforms : Solutions that have employed low-power radios have typically attacked the bandwidth bottleneck by employing low-power multi-radio platforms([1] [4] [11]). These solutions allow simultaneous reception and transmission over multiple channels, thereby increasing the bandwidth. Most of these solutions also rely heavily on platform specific low-level optimizations to maximize radio utilization [4].
Reliable MAC Protocols : Most existing systems([1] [11]) have attempted to use Freq-Time Division Multiple Access (FTDMA) based solutions, where each sensor is assigned a unique time-frequency slot. However, these solutions are undeniably inefficient given that not all sensors transmit data at the same time. More recently, Chintalapudi *et al.* [4] have evaluated and implemented contention based MAC that use multi-radio extensions to popular schemes such as ALOHA and Exponential Backoff. While these schemes perform better than FTDMA when traffic bursts are not large, they require setting certain MAC parameters that determine their efficiency. Since machines typically do not operate under constant conditions, it may not be practical to determine these parameters in advance.

Contribution made by this paper
In this paper we propose a fundamentally novel approach towards designing a MAC for manufacturing systems Ű namely, leveraging the repetitive nature inherent to almost all automated manufacturing systems. Manufacturing systems tend to be repetitive in the nature of their operations, consequently the communication traffic also tends to exhibit repetitive patterns. A MAC protocol that learns these patterns can potentially use them to schedule transmissions efficiently. In this paper we propose *i*-MAC, a MAC that leverages temporal communication patterns efficiently to avoid transmission collisions in the channel. *i*-MAC learns on the fly and continuously adapts itself to its host machine and its operating conditions. Consequently, it performs better than both traditional contention-free (*e.g.,* TDMA or FTDMA) and contention-based MAC protocols. To the best of our knowledge, we know of no such prior work that leverages the inherent repetitive nature of manufacturing machines.

Organization of the paper
Section 2 is geared to provide insights to the reader regarding the nature of communication traffic in manufacturing machines. This is followed by the description of the basic ideas behind *i*-MAC in sections 3 and 4. In Section 5 we provide the implementation details of *i*-MAC. Finally, in Section 6 we present the results of performance of *i*-MAC and compare them with existing schemes.

2. TRAFFIC PATTERNS IN MACHINES

An assembly line manufacturing machine generally comprises several stations, each designed to accomplish a part

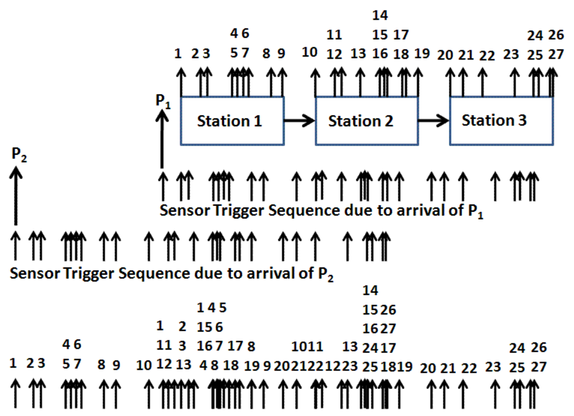

Station 1
Station 2
Station 3
P1
P2
Sensor Trigger Sequence due to arrival of P1
Sensor Trigger Sequence due to arrival of P2
Sensor Trigger Sequence due to arrival of P1 and P2

Figure 4: A traffic generation example

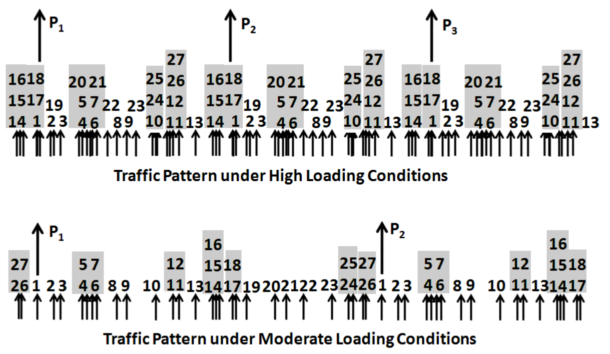

P1
P2
P3
Traffic Pattern under High Loading Conditions
Traffic Pattern under Moderate Loading Conditions

Figure 5: Recurring patterns exhibited at steady state.

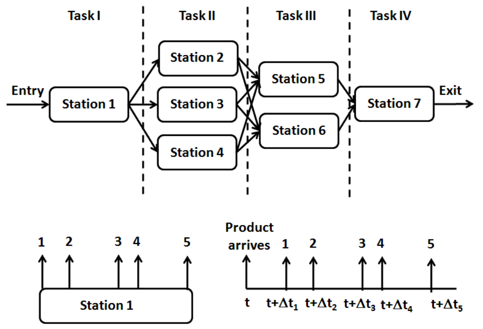


Figure 2: Example of a manufacturing machine

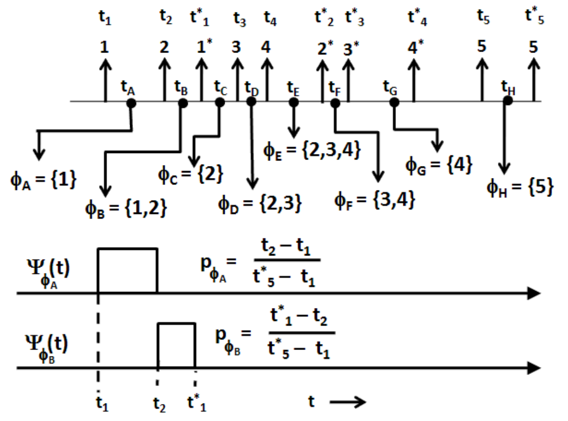


Figure 3: An illustration for the concept of burst sets

of the production process. A single station can typically process one product at any given time. After a station completes its task, the partially completed product is transferred to another station over conveyers or using robotic arms.

Often, a single machine is equipped with several copies of the same station to relieve bottlenecks in the manufacturing process (in case that particular station takes a much longer time than the rest). The example machine in Figure 2 has seven stations assigned to perform four different kinds of tasks on the product. The product enters at station 1, after which it may be transferred to stations 2, 3 or 4 for task II. The reason for having three identical stations is that task II takes thrice as long as the desired production rate. Similarly, there are two copies for task III, stations 5 and 6. Station 7 performs the final task on the product prior to its exit. Thus, a product may be routed through the stations via several different paths, *e.g.,* $< 1, 3, 5, 7 >$ or $< 1, 4, 6, 7 >$. The exact path that a product takes is determined by a scheduler at run-time, and may depend on several factors such as the current state of occupancy of the various stations, their current state of wear and tear.

Stations are designed to operate in a very deterministic manner (*e.g.,* a robotic wrapper is designed to take a fixed amount of time to complete packing a chocolate bar). When a product arrives at a station at time t, it results in a deterministic sequence of sensor events that occur at fixed offsets from t. Figure 2 illustrates an example where station 1 has 5 sensors where arrows depict sensor events. A product arrival at time t at station 1, generates a sequence of sensor events with sensor i triggering at $t + \Delta t_i$; Δt_i being a constant offset specific to the station and the type of product.

A Simple Example

Figure 4 depicts a simple example assembly line with three stations and a total of 27 sensors. Each product must pass through stations 1 to 3 in that order. The temporal sequence of sensor trigger events is depicted using small arrows at each station in Figure 4.

When product P_1 (Figure 4) is introduced at station 1, it will generate a sequence of sensor trigger (1-27) events as it passes through the three stations (Figure 4). The arrival of another product (P_2) at station 1 will generate a time-lagged version of the same sequence of sensor events generated by P_1 (Figure 4). The overall temporal sequence of sensor events generated due to the two products will be a superposition of their individual sequences (Figure 4), as each product independently generates events at different stations. During the steady state operation of a machine, as products keep arriving one after the other, the overall temporal sequence of sensor events will be the superposition of those generated by each product.

In many automated manufacturing machines, products arrive at fixed intervals, depending on production demands. For low production demands, the product inter-arrival times will be large and the machine is said to be *lightly loaded.* The minimum inter-arrival time is determined by the bottleneck station in the pipeline. A machine operating at the minimum inter-arrival time is said to be *fully loaded*.

During steady operation of a machine (machines often operate at contant load conditions for several hours), the sequence of sensor events will typically exhibit dominant recurring temporal patterns owing to the underlying repetitive nature of the process. The nature of these patterns will depend on the loading conditions. Figure 5 depicts the temporal sequences of sensor events generated during the operation of the machine for two different product arrival rates (loading conditions).

As seen in Figure 5, certain groups of sensors will have a tendency to detect events at about the same time, depending on the arrival rate of the products. For example, in the case of the highly loaded scenario, the sets of sensors $\{1, 17, 18\}$, $\{2, 19\}$, $\{4, 5, 6, 7, 20, 21\}$, $\{10, 24, 25\}$, $\{11, 12, 26, 27\}$ and $\{14, 15, 16\}$ detect events at around the same time. Such scenarios, where several sensors detect events around the same time, will lead to traffic bursts, as all these sensors will have event notification messages to transmit to the controller.

In case a machine has copies of identical stations, a scheduler at the controller decides at run time which path the product must take. The methods used by the scheduler are usually based on deterministic rules. As a result, even in machines where products can take multiple paths, the system often settles into a "rhythm" and exhibits recurring temporal patterns.

3. BURST SETS

In this section we introduce the notion of a burst set which is fundamental to the design of *i*-MAC. The *burst set* (represented by ϕ in the rest of the paper) is defined as - *the set of sensors in the machine that have pending data to transmit at the same time.*

Figure 3, illustrates the concept of a burst set through a simple example, which depicts the sensor activity within a small slice of time during a machine's operation. In Figure 3, sensor i detects an event at time t_i and has to notify this event to the controller. t_i^* represents the time when the event notification message from sensor i reaches the controller. The delay $t_i^* - t_i$ might include several components such as several retransmissions due to packet losses in the wireless channel or packet collisions, or delays such as back-offs induced by the underlying MAC. At any time $t_i < t < t_i^*$, the sensor i will have pending data to transmit to the controller.

In the time interval $t_1 < t_A < t_2$, there is only one sensor with pending data to transmit - sensor 1. The burst set seen in the machine at a time t_A is thus given by $\phi_A = \{1\}$. At a time $t_2 < t_B < t_1^*$, however, sensor 1 has not yet succeeded in transmitting its notification message to the controller and in addition, sensor 2 has data to transmit. Thus, two sensors (1 and 2) have data to transmit at time t_B and so the burst set seen by the machine at this time is $\phi_B = \{1, 2\}$. Similarly, at a time, $t_1^* < t_C < t_3$, sensor 1 has already succeeded and only sensor 2 has data to transmit, thus $\phi_C = \{2\}$.

Some burst sets may appear more frequently than others. To measure the relative frequency of occurrence of a burst set, *i*-MAC also maintains the *occurrence probability* p_ϕ for each observed burst set ϕ. p_ϕ reflects the chance that the machine sees the burst set ϕ given that one or more sensors have data to transmit. An intuitive way to compute p_ϕ is to compute the fraction of time when a certain burst set was seen by the machine. We illustrate this using a simple example in Figure 3. In Figure 3, let us assume for simplicity's sake that the communication pattern exactly repeats itself throughout the operation of the machine. The burst set ϕ_A is seen during the interval (t_1, t_2) within the total time interval of (t_1, t_5^*) thus $p_{\phi_A} = \frac{t_2 - t_1}{t_5^* - t_1}$. Similarly, since the burst set ϕ_B is seen during the interval (t_2, t_1^*), $p_{\phi_B} = \frac{t_1^* - t_2}{t_5^* - t_1}$. If L is the set of all observed burst sets probability of occurrence of a burst set ϕ computed at a time t can be mathematically expressed as,

$$p_\phi = \frac{\int_{-\infty}^{t} \Psi_\phi(\tau) d\tau}{\sum_{\forall \phi_i \in L} \int_{-\infty}^{t} \Psi_{\phi_i}(\tau) d\tau}. \tag{1}$$

In Eqn 1, $\Psi_\phi(\tau)$ is the *burst indicator function* (Figure 3) which attains a value 1 if the burst set ϕ is seen in the machine at a time τ, and remains 0 otherwise.

The main drawback of defining p_ϕ according to Eqn 1 is that it has infinite memory. In order to be adaptive, *i*-MAC must "forget" burst sets that have occurred far back in time. However, *i*-MAC should not "forget" during periods of prolonged inactivity when production has been temporarily paused. Thus, *i*-MAC uses a definition of p_ϕ that weights the more recent burst sets to a greater extent than those in the past while including only active periods in the computation.

$$p_\phi = \frac{\int_{-\infty}^{t} \Psi_\phi(t-\tau) e^{-\alpha(\gamma(t)-\gamma(\tau))} d\tau}{\sum_{\forall \phi_i \in L} \int_{-\infty}^{t} \Psi_{\phi_i}(t-\tau) e^{-\alpha(\gamma(t)-\gamma(\tau))} d\tau}. \tag{2}$$

$\gamma(t)$ in Eqn 2 is the total time during which at least one sensor had pending data and is given by,

$$\gamma(t) = \sum_{\forall \phi_i \in L} \int_{-\infty}^{t} \Psi_{\phi_i}(\tau) d\tau \tag{3}$$

During periods of inactivity $\gamma(t)$ does not increase at all since there will be no sensor with pending data; thus *i*-MAC does not "forget" during these periods.

In Eqn 2 burst sets lose their relevance in an exponentially decaying manner. While in principle, other weighting functions could be used, our choice in Eqn 2 is motivated by the fact that it allows for an algorithm - the BSUA algorithm (described in Figure 6) - that is amenable to simple incremental updates.

The choice of the *memory parameter* α is a tradeoff between the accuracy of the estimated values of occurrence probabilities p_ϕ versus the quickness in the adaptability of *i*-MAC. If α is too small, *i*-MAC will be slow to adapt, whereas if α is too large it will forget too quickly for the statistics (occurrence probabilities) to be accurate. *i*-MAC uses an adaptive α that is proportional to rate of product arrival. The rationale being that when products arrive faster, not only is the need for rapid adaptation greater, but statistics also tend to converge faster as more sensor events occur within a short time. For this, *i*-MAC maintains an estimate of the average inter-arrival time $\hat{T}$ by noting the intervals between two consecutive times when the same sensor event is detected across several sensors. α is chosen as, $\alpha = \frac{log(\chi)}{log(q\hat{T})}$. This ensures that an event corresponding to a product that arrived roughly q products ago will be given a weight of χ. In our implementation of *i*-MAC, we chose $q = 1000$ and

```
1: $\phi^* = \{\}$, $T = 0$, $L = \{\}$
2: Read next event triple $< t,id,type >$
3: if $\phi^* \neq \{\}$ then
4:     $\Delta t = t - t_{last}$, $\tau = 1 - e^{-\alpha T}$
5:     $T = T + \Delta t$, $\eta = e^{-\alpha \Delta t}$
6:     for all $\phi \in L$ do
7:         $p_\phi = \frac{\eta p_\phi \tau}{1-e^{-\alpha T}}$
8:     end for
9: end if
10: $t_{last} = t$
11: if $type = SENSOR\_TRIGGER\_EVENT$ then
12:     $\phi^* = \phi^* \cup \{id\}$
13: else
14:     $\phi^* = \phi^* - \{id\}$
15: end if
16: if $\phi^* \in L$ then
17:     $p_{\phi^*} = p_{\phi^*} + \frac{(1-\eta)}{1-e^{-\alpha T}}$
18: else
19:     $L = L \cup \phi^*$
20:     $p_{\phi^*} = \frac{1-\eta}{1-e^{-\alpha T}}$
21: end if
22: goto Line 2
```

Figure 6: The BSUA Algorithm

$\chi = 0.01$[1].

Burst Set Update Algorithm (BSUA)

In *i*-MAC, the base-station always maintains a list of frequently observed burst sets and their probabilities. The event notification packets in *i*-MAC (Figure 11) carry the time when the event was detected (the Trigger Time field) in the sensor data packet and the ID of the sensor that detected the event. Two events are inferred from the receipt of a single packet at the base-station - a *sensor trigger event* corresponding to the trigger time (t_i) read from the packet, and a *successful notification event* corresponding to the time of receipt of the packet (t_i^*) at the base-station[2].

i-MAC uses BSUA to update and maintain the burst set list at the base-station. Each event in BSUA is represented by the triple $< t, id, type >$. Here t is the time of occurrence of the event (either sensor trigger event or successful notification event), *id* is the sensor id of the sensor responsible for the events and *type* the type of event. The output of BSUA is a list L of burst sets and their probabilities of occurrence computed using Eqn 2. Figure 6 depicts the BSUA algorithm.

As events are presented to BSUA in increasing order of times, ϕ^* maintains the current burst set seen by the machine based on the sequence of events presented so far. Upon seeing a sensor trigger event from sensor i, BSUA adds i to the set of active sensors ϕ^* (Line 12, Figure 6). When it sees a successful notification event from i, BSUA removes i from ϕ^* (Line 14, Figure 6). BSUA maintains only a list of observed burst sets with two or more elements ($|\phi^*| > 1$), since transmission collisions can only occur if more than one sensor has data to transmit at the same time. Upon receiving each new packet, BSUA updates the occurrence probability of every burst set in L based on Eqn 2. If a new burst set with greater than two elements ($\phi^* \notin L$) is seen, BSUA adds it to the list (line 22, Figure 6).

The events need to be presented to BSUA in increasing order of their event times. However, packets from sensors may not necessarily arrive in order. To present the events in order, upon receiving a packet, the corresponding events are buffered in increasing order of their times of occurrence into an event buffer (Figure 12) and then presented to BSUA. Events must be buffered for a time greater than the prespecified hard real time deadline to ensure correctness of BSUA. In our implementation we buffered for 500ms.

Given n sensors there can be 2^n possible burst sets. In practice however, we always found that the number of burst sets observed is limited to a few hundred in number. To avoid a possible explosion in the number of burst sets, in our implementation we used three methods. First, burst sets with negligible occurrence probabilities (under 10^{-5}) were discarded. Second, in many cases burst sets are subsets of each other *i.e.,* $\phi_1 \subset \phi_2$. In such cases ϕ_1 may be removed by simply incrementing p_{ϕ_2} by p_{ϕ_1}. Thus, in our list of burst sets we ensured that no burst set was a subset of another in the list. Finally, we enforced an upper limit of 10,000 burst sets in the list by dropping the burst sets with the lowest ourrence probability.

4. COLLISION AVOIDANCE IN *i*-MAC

If two sensors belong to the same burst set ϕ with a high value of p_ϕ, it is likely that these sensors will attempt to transmit to the controller at the same time. Similarly, if two sensors do not belong to any of the burst sets, then these sensors will most likely never attempt to transmit at the same time. *i*-MAC uses this basic fact to avoid potential packet collisions.

i-MAC is a slotted protocol like TDMA (or FTDMA), where each sensor is assigned a fixed slot in which to transmit to the base-station. *i*-MAC differs from TDMA, in that multiple sensors may be assigned to the same slot. In *i*-MAC, sensors that have a high probability of transmitting together are assigned different transmission slots. However, sensors that have a no or extremely low likelihood of transmitting at the same time may be assigned to the same slot.

An Example

To illustrate this idea, we shall continue to build on the example in Figure 3. Consider for simplicity's sake that the machine has only five sensors and that the communication pattern shown in Figure 7 exactly repeats itself periodically. In the list of burst sets generated in our example in Figure 3, there are four burst sets with cardinality greater than one - $\{1, 2\}$,$\{2, 3\}$,$\{2, 3, 4\}$ and $\{3, 4\}$. These are the only scenarios where a transmission collision can occur. As shown in Figure 7 we can schedule sensors 1, 3 to transmit in slot 1, sensors 2, 5 in slot 2, and sensor 4 in slot 3. Such a schedule requires only 3 slots unlike 5 slots needed for TDMA and yet makes sure that no two sensors that belong to the same burst set are assigned to the same slot, thus precluding any possibility of packet collisions.

More formally, *i*-MAC attempts to find a *Sensor Slot Assignment* (SSA) - a function $f(x) = y$ that maps sensor ids (x) to slot numbers (y) so as to minimize the number of slots

[1]The particular choice was motivated by the fact that usually, production machines operate under the same operating conditions for prolonged periods of time, several hours or even days.

[2]In *i*-MAC all sensors are time-synchronized to the base-station and have the same notion of time.

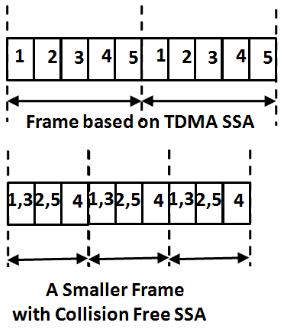


Figure 7: A Collision free SSA in *i*-MAC

(s) while ensuring that the expected number of collisions in any given slot is extremely small.

Optimal SSA Problem
Given an SSA f, let θ_k be the set of sensors that were assigned to a certain slot k (in our example $\theta_1 = \{1,3\}$). Given the list of burst sets $L = \{\phi_1, \phi_2, \cdots\}$ and their occurrence probabilities, the expected number of transmission collisions e_k in a slot k are given by,

$$e_k = \sum_{\phi \in L} p_\phi \Upsilon\left(|\phi \cap \theta_k|\right). \tag{4}$$

Here,

$$\Upsilon(x) = \begin{matrix} 0 & if & x \leq 1 \\ x & if & x > 1 \end{matrix} \tag{5}$$

i-MAC attempts to find an SSA f that minimizes the number of slots s while ensuring that the expected number of colliding sensors e_k within each slot k is less than a pre-specified threshold ϵ i.e., $e_k < \epsilon \forall k = 1, \cdots, s$. We refer to this problem as the *Optimal SSA (OSSA)* problem in the rest of the paper. The OSSA problem is NP-Hard as the corresponding decision problem can be shown to belong to class NP-Complete by reducing it to the well known graph coloring problem. We do not provide the proof in this paper due to space limitations. Consequently, for solving the OSSA problem, *i*-MAC resorts to using a heuristic.

***i*-MAC's heuristic for OSSA**
i-MAC's Sensor Slot Assignment (ISSA) algorithm starts by hoping that one slot ($s = 1$) will be sufficient to find an SSA that satisfies the constraints $e_k < \epsilon \forall k = 1, \cdots, s$. It then uses the *Maximum Collision First Sensor Slot Assignment* (MCF-SSA) algorithm (described shortly in this section) to determine an SSA. The MCF-SSA algorithm returns a *FAILURE* if it is unable to find a viable SSA. Upon failure, ISSA increments the value of s and attempts to find an SSA until it is successful. The algorithm pseudo-code is provided below:

```
1: ISSA($L, \{p_{\phi_1}, p_{\phi_2}, \cdots\}, n, \epsilon$)
2: $s = 1$
3: while MCF-SSA($L, \{p_{\phi_1}, p_{\phi_2}, \cdots\}, s, n, \epsilon$) returns FAILURE do
4:     $s = s + 1$
5: end while
```

The MCF - SSA Algorithm
Given a list of burst sets $L = \{\phi_1, \phi_2, \cdots\}$ and their occurrence probabilities, the number of slots s, the number of sensors n, and the threshold ϵ, the MCF heuristic attempts to determine an SSA that satisfies the constraints $e_k < \epsilon, \forall k = 1, \cdots, s$. It returns a failure in case it is unable to find such an assignment.

```
1: MCF-SSA ($L, \{p_{\phi_1}, p_{\phi_2}, \cdots\}, s, n, \epsilon$)
2: for $i = 1$ to $n$ do
3:     Compute $c_i$ using Eqn 6
4: end for
5: Sort the sensors in decreasing order of $c_i$. Let this order
   of the sensor ids be $O =< o_1, o_2, \cdots, o_n >$.
6: $f(o_1) = 1$, $\theta_1 = \{o_1\}, \theta_i = \{\} \forall i = 2, \cdots, s$. Assign
   sensor $o_1$ to slot 1.
7: for $i = 2$ to $n$ do
8:     for $j = 1$ to $s$ do
9:         $\theta_j = \theta_j \cup \{o_i\}$ (assign sensor $o_i$ to slot $j$.)
10:        Compute $e_j$ using Eqn 4
11:        $\theta_j = \theta_j - \{o_i\}$ (remove sensor $o_i$ from slot $j$).
12:    end for
13:    Find the slot $k$ that offers the minimum expected
       number of collisions $k = \arg\min_j e_j$. In case several
       different values of $k$ have the same minimum value,
       one among them is uniformly randomly chosen.
14:    if $e_k > \epsilon$ then
15:        return FAILURE
16:    else
17:        Assign sensor $o_i$ to slot $k$. $f(o_i) = k$, $\theta_k = \theta_k \cup \{o_i\}$.
18:    end if
19: end for
```

Figure 8: The MCF-SSA Algorithm

The MCF-SSA algorithm is provided in Figure 8. MCF-ISSA starts by computing *collision index* c_i for each sensor i (Lines 1-3 in Figure 8) defined as,

$$c_i = \sum_{\forall \phi \in L | i \in \phi} p_\phi \Upsilon(|\phi|). \tag{6}$$

The collision index measures the expected number of sensor transmissions that a transmission from sensor i would collide with, if all sensors were assigned the same slot. The essential idea in the MCF heuristic is to first assign slots to those sensors whose transmissions are most likely to collide with others'. The sensors are thus sorted in the decreasing order of their collision indices (line 4 Figure 8) and considered for slot assignment sequentially in this order. Each sensor under consideration is assigned a slot that minimizes the expected number of collisions with already assigned sensors (lines 6 to 18). In case no slots can be found with expected number of collisions (with already assigned sensors) less than the desired threshold ϵ, a failure is returned.

The base-station periodically runs the ISSA algorithm on the current list of burst sets to determine an SSA. If this newly found SSA provides a "significant" improvement over the existing SSA, the base-station disseminates the new SSA to all the sensors. In our implementation, a new SSA is deemed a significant improvement over the existing SSA only if at least one of two criteria are met. First, the old SSA is no longer a valid solution to the current list of burst sets (it violates one or more constraints). Second, the number of time slots for the new SSA is smaller than that for the old SSA.

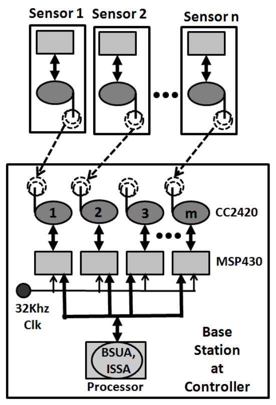


Figure 9: The architecture of the base-station used in our implementation of *i*-MAC

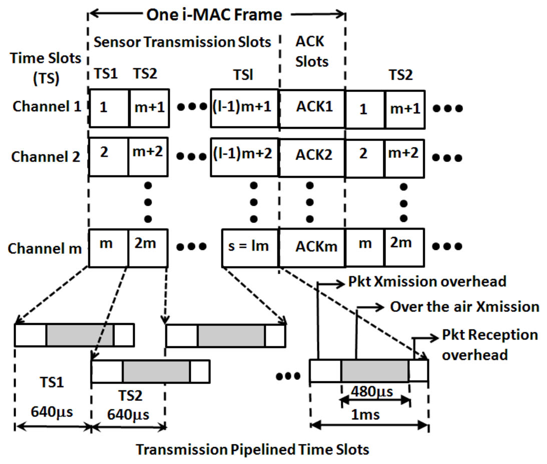


Figure 10: The operation of *i*-MAC

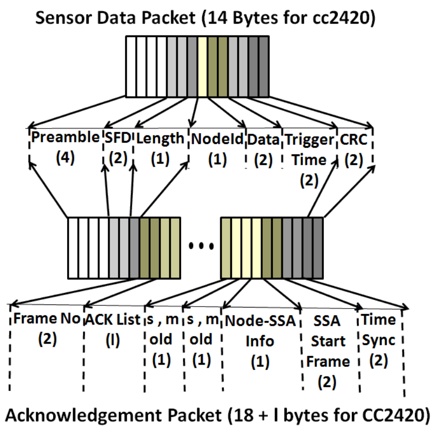


Figure 11: Packet formats in *i*-MAC

5. *i*-MAC- THE DETAILS

As depicted in Figure 10, communication between the sensors and the controller in *i*-MAC is organized into frames (similar to TDMA) that repeat. Each *i*-MAC frame comprises several *sensor transmission slots* in which sensors transmit event notification messages to the controller (base-station), and *acknowledgement slots* in which the controller transmits an acknowledgement packet to notify the reception of packets in the current frame.

i-MAC allows for the base-station to have multiple radios, each operating on a different channel. In figure 10, the base-station is equipped with m radios, so that it can simultaneously receive packets from m sensors. The total number of sensor transmission slots $s = lm$, where l is the number of time-slots within a frame. In the acknowledgement slots, the base-station transmits m acknowledgement packets simultaneously. An acknowledgement packet transmitted in channel i contains a list of all the sensors from which packets were successfully received in the current frame in channel i (see Figure 11). The sensor persistently retransmits in each successive frame until it receives an acknowledgement packet with its ID included in the ACK-list. The acknowledgement packets also serve as *time-synchronization beacons* to keep the sensors synchronized to the base-station and carry a two byte time-stamp (Figure 11)[3].

We have implemented *i*-MAC on a home-built prototype platform that uses CC2420 transceivers. In our implementation, the base-station is a home-built platform with four CC2420 transceivers, each controlled as a slave by an MSP430 micro-controller of SPI interfaces as shown in Figure 9. The BSUA and the ISSA algorithms were implemented on a desktop computer,[4] which communicated with the MSP430 micro-controllers driving the transceivers over UART.

Upon receiving notification packets, the base-station continuously updates the list of burst sets using BSUA (Section 3). This list is then used to generate an SSA using the ISSA algorithm described in section 4. If the new SSA is a significant improvement over the old SSA (section 4), the base-station distributes the new SSA to all the sensors. This is depicted in Figure 12. In typical manufacturing systems, each sensor node is responsible for a unique set of events. Consequently, in our implementation, the field node id (in Figure 11) actually contained an event id from which the receiver can deduce the id of the transmitting node.

[3]In our implementation, nodes were synchronized up to within 64μsec of each other

[4]Standard processors used in controllers today are mini-computers which, we believe, are quite capable of running the ISSA and BSUA algorithms.

Transmission Pipelining in *i*-MAC

For efficient channel utilization the transmission slots in *i*-MAC are *transmission pipelined* [4]. The time interval between initiating transmission at the transmitter and the end of packet reception of the receiver was about 1ms. However, the total time during which bits were actually transmitted over the air was only 480μsec. The remaining channel idle period comprised 150μsec of radio over head (including time to transmit bits between MSP430 and CC2420), 192μsec of radio calibration, 120 μsec of packet processing overhead at receiver and a guard band of 64μsec to allow for time-synchronization errors. To avoid this idle time, consecutive transmission slots were arranged so that the next node would initiate transmission so as to offset the channel's idle period[5]. This is depicted in Figure 10.

Frequency-Transceiver Hopping

To avoid temporary fades in the wireless channel, our implementation of *i*-MAC uses *frequency-transceiver* hopping *i.e.,* in any two successive retransmissions (successive frames), a sensor nodes transmits on a different physical channel and to a different physical transceiver (among the m available transceivers). This is achieved by using a *channel mapping function* (CMF) known to all the sensors. The CMF translates a slot (assigned by SSA) to a physical channel based on the current frame number and the node id. Consequently, knowing the current frame number and the slot assigned to it by SSA, a node can determine which physical channel to transmit on. In our implementation, the physical channel used by a sensor was skipped by four channels between each successive frame. Each of the transceivers at the BS, on the other hand skipped five channels between each successive frame. This ensured that a sensor node transmitted to a

[5]These measurements were done using an oscilloscope.

different physical transceiver in every retransmission.

SSA Distribution in *i*-MAC
The design of SSA distribution scheme in *i*-MAC has to fulfill two crucial goals. First, the updated schedules must be distributed so as to waste minimum energy for the sensors. Second, at any given time all nodes must be using the same schedule.

To achieve these goals, each frame in *i*-MAC has a frame counter f which increments for every frame and wraps after $20n$. The base-station transmits SSA info for sensor i in the 10 consecutive frames (to ensure reliable reception of the SSA information) that satisfy $10*(i-1) \leq mod(f, 10*n) < 10*i-1$ in the Node-SSA field (Figure 11). Sensor nodes thus wake up in the frames intended for them to update their SSA information as well as to correct possible time-synchronization errors due to skew. Since the number of slots s might change, the ACK carries and s, m field 11. This scheme allows the sensors to duty-cycle and save power.

In the wake of an SSA change, each packet also contains an SSA-start frame number (SSA-SFF)(Figure 11) that indicates the frame number at which the sensors can start using the new schedule. Until this frame number, the sensors must continue using the old schedule. The SSA-SFF is chosen to be n frame away from the frame at which the schedule was updated. This ensure that all the nodes have received the new SSA before the SSA-SFF.

Persistent Collision Syndrome (PCS)
When load is suddenly increased in a machine, the traffic patterns will change. While *i*-MAC will eventually learn and begin adapting itself to the new conditions, during the transition period when the list of burst sets still retains memory from the low load conditions, the existing SSA might fall short of avoiding all the collsions. Suppose for example, that a machine was lightly loaded and sensors 1,15 were never triggered at the same time and thus were assigned the same time-frequency slot for transmission. Now suppose the machine was suddenly fully loaded and under these conditions sensors 1 and 15 frequently trigger together. During the short period when the old SSA is followed, sensors 1 and 15 will persistently keep trying to transmit in the same slot and consequently never succeed in their transmissions. We shall refer to this as the *Persistent Collision Syndrome* (PCS). PCS can also arise under normal operating conditions, since certain very rarely occurring bursts may not have been captured in the list of burst sets.

Solution To PCS
To circumvent PCS, each sensor keeps track of the number of times a particular transmission has failed. If a particular transmission fails more than a certain number of times in consecutive frames (say 3 consecutive frames), *i*-MAC assumes a PCS and attempts to avoid it by switching temporarily to a *Randomized Transmission Mode* (RTM). In RTM, instead of using its assigned slot, it uniformly randomly chooses one of the next available b time-frequency slots (these may span across multiple frames) and transmits in this slot. The value b should be a conservative value to accommodate large bursts, but not too conservative to reduce the throughput significantly. *i*-MAC maintains the size of the largest observed burst and uses this as the value of b.

The Starting Problem
When a machine starts afresh after a halt, *i*-MAC has not yet compiled the list of burst sets and thus has not computed an SSA. During this phase, *i*-MAC transmits in the randomized transmission mode for a few product cycles until the new SSA is computed and adopted by the various sensors. For this phase we chose b to be 10% of the total number of sensors of the machine.

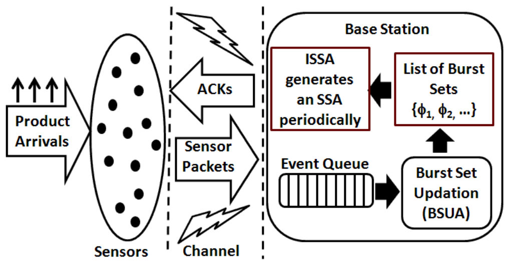
6. RESULTS

Since *i*-MAC adapts to its host machine, a meaningful evaluation requires testing *i*-MAC on several different machines. Gaining access to real manufacturing machines to conduct experiments proves to be extremely difficult for two reasons. First, typical manufacturing environments run on aggressive deadlines and obtaining an idle machine for performing experiments is usually not easy. Second, these machines are extremely expensive and not easy to procure. While *i*-MAC has been implemented, consequently, in this paper, for a meaningful evaluation of *i*-MAC, we relied on simulations. However, we used actual measurements from our implementation to seed the various parameters (such as slot timings *etc.*) into the simulations.

While there are several accurate simulators to test the timing and performance of specific assembly lines that are used by designers, to the best of our knowledge, there is no established generic model for traffic in assembly lines. Further, to the best of our knowledge, there are no public databases that provide traffic traces for assembly lines. We used a proprietary simulator to generate random assembly lines in order to test *i*-MAC. The simulator was built based on inputs collected from several machine users. These inputs were in many cases approximate and of verbal nature rather than precise. For example, "there are few large sized machines (150 sensors or more) and most machines have between 60-100 sensors" or "large number of automated machines have product arrival rates in the range of 1-4sec at full load while larger machines with small robotic parts have arrival rates of 5-10 sec and some might have up to 60 seconds," *etc.* Figure 13 depicts the distribution of number of sensors per machine from 50,000 assembly lines generated from our simulator and Figure 14 depicts the distribution of production cycle times (the inter-arrival times of products at full load).

Based on the studies performed for WISA [1], there is almost no electromagnetic interference due to the operation of machines in the 2.4Ghz region in factory floors. The packet loss model in our simulator was rather simple - the channel is assigned a packet success rate (PSR) and packets are dropped with a probability 1-PSR. Given that communicating nodes frequency hop in every frame in *i*-MAC there no correlation between two successive transmissions, this model tends to be not too far to reality. Further, even if one channel has a low PSR, typically another will have a higher PSR. Choosing the same low PSR for all channels provides us with a worst case performance.

An Example Run
In this section we aim to provide the reader an intuition into the learning behavior of *i*-MAC using a machine with 165 sensors and seven stations. The machine has a production cycle time of 2 seconds. We equip the machine's base-station with a single CC2420 radio $m = 1$. The machine is inten-



Product Arrivals
Sensors
ACKs
Sensor Packets
Channel
Base Station
ISSA generates an SSA periodically
List of Burst Sets {ϕ1, ϕ2, …}
Event Queue
Burst Set Updation (BSUA)

Figure 12: Operation of *i*-MAC

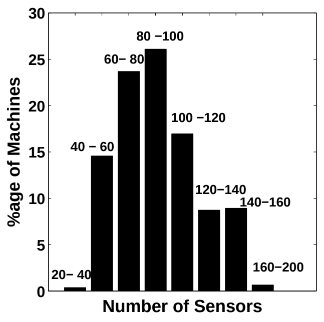


Figure 13: Distribution of number of sensors.

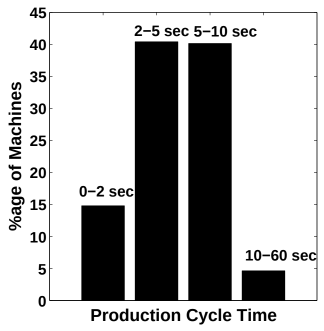


Figure 14: Distribution of production cycle times.

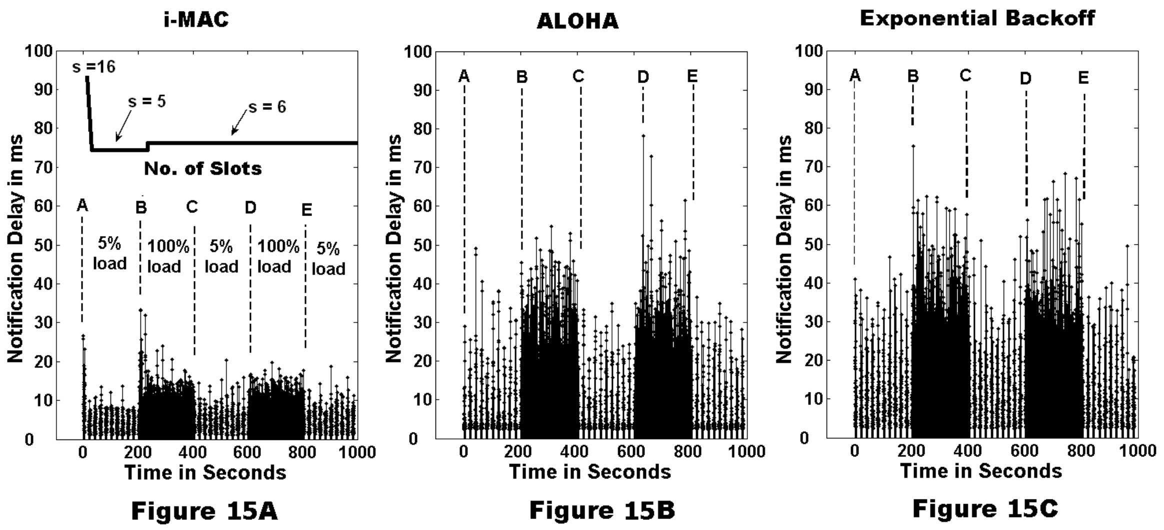


Figure 15: How *i*-MAC learns and adapts to traffic conditions.

tionally subject to extreme changes in load conditions in order to stress-test the learning behavior of *i*-MAC. To provide a reference for comparison we ran p-persistent slotted ALOHA and exponential backoff (EB) based MAC protocols on the same machine with identical product traffic. To give ALOHA and EB the maximum benefit of doubt, through experiments we determined the parameters (p for ALOHA and maximum backoff window for EB) that minimized their average delay. We found that for ALOHA, $p = 0.2$ and for EB a backoff window of 8 were the best values. Figures 18A, B and C depict the delays experienced by each of the event notification packets over the first 1,000 seconds of a machine's operation using *i*-MAC, ALOHA and EB respectively. The PSR was fixed at 90% for this experiment[6].

Point A : When the first product arrives *i*-MAC has no information regarding the nature of traffic in the machine. Consequently it starts in the *Randomized Transmission Mode* (RTM) with 16 time-slots (10% of the sensors) as discussed in section 5. The maximum notification delays are around 28ms for the packets resulting from the arrival of the first product. Peak delays seen in ALOHA are similar and in the range of around 30ms while those seen by EB are slightly higher in the range of 40 ms.

Interval A and B : In the interval A to B products arrive at one tenth the full load. By the time the second product arrives, *i*-MAC learns from the traffic generated by the first product by updating the list of burst sets and generates an SSA with only 5 time-slots (Figure 18A). Starting from the second product, *i*-MAC exits RTM when all the sensors adopt this new SSA. Consequently, the delays during the rest of interval A-B have now dramatically reduced to less than 12ms as *i*-MAC has "learned" and adapted to the machine's traffic. Delays for ALOHA and EB however, vary between 30 and 40ms throughout this interval.

At point B : Here the load on the machine is suddenly increased to 100% (products arrive 10 times faster that in the interval A-B). This changes the nature of traffic significantly; however, some part of what *i*-MAC has learned is still valid (many of the burst sets are still valid). The old SSA only partially succeeds in avoiding collisions among sensor node transmissions. Consequently, once again at point B, the notification delays spike to around 35ms.

Interval B to C : Within a few seconds of being subject to the new load, *i*-MAC once again learns the new traffic con-

[6]This was a pessimistic choice, as in all our measurements for distances less than 10mts, we found PSR to be higher than 90%.

ditions by updating its list of burst sets. This new SSA has 6 time-slots instead of 5 (see Figure 18A). It then generates a new SSA and distributes it among all the sensors.
Once the new SSA is adopted, the notification delays decrease and remain below 25ms during the rest of interval B-C. The delays seen in ALOHA and EB however increase and lie between 40 and 60ms.
Interval C to D : The load is reduced back to 10% during the interval C-D. The delays experienced by *i*-MAC in the interval A-B are slightly (about 2ms on average) higher on average than those in the interval C-D. This is because the new SSA has 6 slots in this interval geared towards accommodating the possibility of products arriving at full load (*i*-MAC learns quickly but forgets slowly since recent events have exponentially higher weight). Delays seen by EB and ALOHA revert back to those seen in interval A to B.
Interval D to E : Once again the load is suddenly switched to 100%, however, this time there is no spike in the notification delays at point D similar to that seen at point A. This is because *i*-MAC learned from the previous fully loaded interval B-C and adapted its SSA accordingly with 6 slots. Delays for EB and ALOHA remain the same as in the interval B to C.

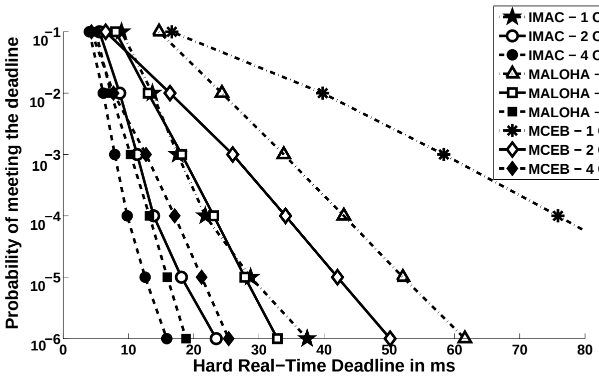


Figure 16: Performance of *i*-MAC

6.1 Performance of *i*-MAC

In this section we compare the performance of *i*-MAC with other MAC protocols proposed in literature. We generated 1000 different random machines from our simulator. Each machine was simulated under fully loaded conditions for 10^8 events. CER was computed as the fraction of packets that succeeded before a given deadline. The graph provides the average delay incurred in achieving various CER values across all the machines. The channel PSR was set to 90% in all simulations. As mentioned before, this choice of PSR was significantly pessimistic compared to our measurements in a real factory floor that gave us a PSR of about 99% was observed for non-line of sight distances of 3-10 mts (typical size of a machine)

We compare the performance of *i*-MAC with three other multi-radio MAC protocols, namely, FTDMA, Multi-channel Exponential Backoff (MCEB) and T-MALOHA.
MCEB [4] is a multi-channel extension of the exponential backoff scheme where the base-station is equipped with multiple radios and can receive simultaneously over different channels from various sensors. A crucial parameter determining the performance of MCEB is the maximum back-off window. To provide the maximum benefit of doubt for MCEB we tried several different maximum window sizes and chose the one corresponding to the best results namely 16.
T-MALOHA [4] uses a pre-determined fixed window size of time-frequency slots unlike MCEB. This window is chosen to be larger than (ideally equal to) the largest possible traffic burst in the machine. In our simulations we found that a window size of 16 gave T-MALOHA the maximum benefit. For seeding the slot widths for each of these protocols we used the measurement data used in [4].

Figure 16 depicts the CER (y-axis), versus different hard real-time deadline (x-axis) and number of radios used for each of the four protocols. As seen in Figure 16, *i*-MAC performs "significantly" better than all the other existing MAC protocols. Delays offered by *i*-MAC are less than half that offered by MCEB or FTDMA for the same probability of communication error and more than 30% less than that offered by T-MALOHA. This is largely due to the efficient SSA in *i*-MAC.

Longevity : The CC2420 radio draws 17.4mA,19.7mA and 0.426mA in transmit, receive and idles modes [22]. Further each radio wakeup in itself costs about 7.5μASec [4]. There are three main sources of power consumption for the sensors running *i*-MAC. First, the energy consumed by the sensor itself, second, the energy consumed for transmitting event notifications and finally the energy consumed in listening to ACKs for SSA information and time-synchronization.
Power Consumed due to Events : If the average number of retransmissions required by *i*-MAC is r, then for each event occurrence, the sensor consumes energy required for transmitting r times and listening to the ACK r times. The radio is put in idle mode between re-transmissions and powered off between events. Consequently it must be woken up once for each event occurrence. Each transmission lasts about 800 μsec and each ACK reception lasts about 1ms, the total energy consumed in the process is given by $r(17.4 \times 0.8 + 19.7) + 7.5$ μAHr. If events occur at a rate of β, the total average consumed will be $(r(17.4 \times 0.8 + 19.7) + 7.5)\,\beta$.
Power Consumed Listening to ACKs : Given a clock-skew of MSP430 is around 20-30ppm and the real time clock has a minimum resolution of about 32μsec (at 32Khz), the clocks need to be corrected roughly once a second to maintain perfect synchronization with an error of $\pm$ 32μsec. However, correction every 500ms provides for sufficient over-provisioning. Listening to an ACK costs about 19.7 μAHr. A sensor may not succeed in listening to the ACK in the very first attempt and will on an average require $\frac{1}{PSR}$ attempts. Given that PSR usually ranges between 90% -99%, the average energy consumption to listen for a single ACK is about 20μAHr. While a sensor typically listens to an ACK twice in one second, it does so while trying to receive and ACK response for its event notification message Ű this occurs at a rate β. Further since the radio is powered down, it needs to be woken up each time to listen to an ACK. Thus, the average energy consumed listening to ACKs is given by $(2 - \beta)(7.5 + 20)$.
Power Consumed by the Sensor : This depends on the specific sensor and can be comparable to, or even higher than, the power consumption of the radio. In our analysis, however, we consider only the power consumed by *i*-MAC.

In our simulations we found that that the average number of re-transmission for *i*-MAC was slightly higher than $\frac{1}{PSR}$ and close to about 1.17. Using high end LiH batteries that have a capacity of 2900mAHr we can compute the longevity of the sensors as 6 yrs for β of 1 per second and 8 yrs for $\beta < 0.1$. Power consumption by the sensor may cut these numbers by half or even more.

7. PRACTICAL CONSIDERATIONS

In this section we discuss some practical considerations that arise when deploying *i*-MAC in real environments.

Interference from neighboring machines : A typical factory floor is designed to hold a large number of machines. Consequently, a machine is often surrounded by other machines within a distance of a few meters. As a result wireless communication among neighboring machines may interfere with each other. These effects can however be significantly mitigated through careful *frequency planning* to ensure that neighboring machines use non-overlapping set of channels, coupled with carefully chosen *transmission power settings* to avoid over-extending the range significantly beyond the machine's boundaries.

Effect of WiFi : Since WiFi operates in the same frequency range as 802.15.4, its presence in the factory floor can potentially degrade the performance of *i*-MAC. In our experience however, we found that the persistent retransmissions and frequency-hopping are usually sufficient to mitigate the effects due to WiFi. Several factories today forbid the use of WiFi inside factories to avoid interference with the operation of their machines. Such regulations may not be unreasonable for dedicated manufacturing environments.

8. CONCLUSION

Automated manufacturing machines perform repetitive tasks and consequently, repetitive communication patterns are induced in the control loops of these machines. In this paper we presented a novel MAC protocol,*i*-MAC, that learns collision patterns among transmitting nodes, and leverages it to efficiently coordinate transmissions. *i*-MAC performs significantly better than existing low-power wireless MAC protocols designed for sensing in manufacturing machines. We believe that learning mechanisms developed in this paper are generic and can be applied to other systems that exhibit repetitive patterns.

9. REFERENCES

[1] ABB-WISA. http://www.eit.uni-kl.de/litz/WISA.pdf.

[2] Ansari, J., Riihijaveri, J., Mahonen, P., and Haapola, J. Implementation and Performance Evaluation of nanoMAC : A Low-Power MAC Solution for Higher Density Wireless Sensor Networks. *International Journal of Sensor Networks 2* (July 2007), 341–349.

[3] Barroso, A., Roedig, U., and Sreenan, C. J. uMAC: An Energy-Efficient Medium Access Control for Wireless Sensor Networks. In *Proceedings of the 2nd IEEE European Workshop on Wireless Sensor Networks (EWSN2005)* (Istanbul, Turkey, February 2005), IEEE Computer Society Press.

[4] Chintalapudi, K., and Venkatraman, L. On the Design of MAC Protocols for Low Latency Hard Real TimeDiscrete Control Applications Over 802.15.4 Hardware. In *International Conference on Information Processing in Sensor Networks (IPSN)-SPOTS* (2008), pp. 356–367.

[5] Dam, T. V., and Langendoen, K. An Adaptive Energy Efficient MAC Protocol for Wireless Sensor Networks. In *The First ACM Conference on Embedded Networked Sensor Systems (Sensys'03)* (Los Angeles, CA, USA, Novemeber 2003).

[6] ELPRO-Technologies. http://www.elprotech.com/.

[7] Enz, C. C., El-Hoiydi, A., Decotignie, J. D., and Peiris, V. WiseNET: An Ultralow-Power Wireless Network Solution. *IEEE Computer 37*, 8 (November 2004).

[8] Ephremides, A., and Mowafi, O. A. Analysis of a Hybrid Access Scheme for Buffered Users - Probabilistic Time Division. *IEEE Transactions on Software Engineering SE-8*, 1 (Jan 1982), 52–61.

[9] Feeny, L. M., Nilsson, M., Aia, M., and Min, J. Investigaing the power consuption of a wireless network interface in an ad-hoc networking environment. In *IEEE INFOCOM* (2001).

[10] Foundation, H. C. http://www.hartcomm.org/.

[11] H. J Korber, H. W., and Scholl, G. Modular Wireless Rreal-Time Sensor/Actuator Network for Factory Automation Applications. *IEEE Transactions on Industrial Informatics 3*, 2 (May 2007), 111–119.

[12] International, P. . P. http://www.profibus.com/.

[13] Jamieson, J., Balakrishnan, H., and Straser, Y. C. T. Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks. Tech. Rep. 893, MIT Laboratory for Computer Science, May 2003.

[14] Janssen, D., and Buttner, H. Real-time Ethernet: theEtherCAT solution. *Computing and Control Engineering 15* (2004), 16–21.

[15] Liu, Z., and Elhanany, I. RL-MAC : A Reinforcement Learnning Based MAC Protocol for Wireless Sensor Networks. *International Journal of Sensor Networks 1* (September 2006), 117–124.

[16] Lu, G., Krishnamachari, B., and Raghavendra, C. S. An adaptive Energy-Efficient and Low-Latency MAC for Data Gatherting in Wireless Sensor Networks. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (April 2004).

[17] Negri, L., Beutel, J., and Dyer, M. The Power Consumption of Bluetooth Scatternets. In *Consumer Communications and Networking Conference* (Jan 2006).

[18] Polastre, J., Hill, J., and Culler, D. Versatile Low Power Medium Access for Wireless Sensor Networks. In *The Second ACM Conference on Embedded Networked Sensor Systems (Sensys'04)* (Baltimore, MD, USA, Novemeber 2004).

[19] Rajendran, V., Obraczka, K., and Garcia-Luna-Aceves, J. J. Energy-Efficient, Collision-Free Medium Access Contorl for Wireless Sensor Networks. In *The First ACM Conference on Embedded Networked Sensor Systems (Sensys'03)* (Los Angeles, CA, USA, Novemeber 2003).

[20] Rhee, I., Warrier, A., Aia, M., and Min, J. Z-MAC: a Hybrid MAC for Wireless Sensor Networks. In *The Third ACM Conference on Embedded Networked Sensor Systems (Sensys'05)* (San Diego, CA, USA, Novemeber 2005).

[21] Roedig, U., Barroso, A., and Sreenan, C. J. f-MAC: A Deterministic Media Access Control Protocol Without Time Synchronization. In *European Workshop on Wireless Sensor Networks (EWSN2006)* (Zurich, Switzerland, February 2006).

[22] TI-CC2420. http://focus.ti.com/docs/prod/folders/print/cc2420.html.

[23] Ye, W., Heidemann, J., and Estrin, D. Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks. *IEEE/ACM Transactions on Networking 12*, 3 (June 2004), 493–506.

[24] Zheng, T., RadhaKrishnan, S., and Sarangan, V. PMac : An Adaptive Energy Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05)* (2005).