IAA: Interference Aware Anticipatory Algorithm for Scheduling and Routing Periodic Real-time Streams in Wireless Sensor Networks

S. M. Shahriar Nirjon, John A. Stankovic and Kamin Whitehouse
Department of Computer Science, University of Virginia
{nirjon, stankovic, whitehouse}@cs.virginia.edu

***Abstract*—This paper provides a polynomial time heuristic for the real-time communication scheduling problem in multi-hop wireless sensor networks. Wireless networks add a new dimension to the real-time communication problem because of *interference*: a transmission cannot be scheduled on a radio link if another transmission is scheduled on any interfering link. The problem being NP-hard in nature, we propose a novel heuristic that comes into two parts: (1) a scheduler that uses a topological analysis of the network to *anticipate* the effects of radio interference in order to improve scheduling prioritization, (2) an iterative route update scheme that pushes apart interfering streams and spreads them out over the network to reduce interference and improve schedulability while meeting the deadline requirements. The whole algorithm runs in polynomial time of $O(N^3d)$, where N and d are the number of streams and maximum deadline respectively. We use a simulation-based study to demonstrate that this algorithm produces near-optimal schedules for approximately 10 packet streams in a 100 node network, where the optimal schedule can be computed. We also show that the overall algorithm is able to schedule as much as 47% more steams than simple heuristics that takes only deadline or interference into account. Of this improvement, $4\% - 26\%$ contribution comes from the iterative route update scheme.**

I. Introduction

Real-time communication is important for many industrial control applications such as process control and manufacturing automation: sensors, actuators, and controllers that are distributed throughout the industrial environment must be able to transfer information by a given deadline. Most industrial control applications rely on wired communication networks, such as those specified by the fieldbus standards like PROFIBUS [1] and CAN [2]. However, cabling is expensive, and can cost as much as $2000 per foot in highly controlled environments such as nuclear power plants [3]. Cabling systems do not scale easily to large geographic areas, and can be difficult to modify after initial deployment. Furthermore, cabling is vulnerable to the chemicals, vibrations, and moving mechanical parts that are present in industrial environments, which can reduce reliability and increase maintenance costs. Wireless communication is a cheaper and more scalable alternative to wired communication, and recent advancements in channel coding, spread spectrum, dynamic multi-hop routing, and cognitive radios [4] are making wireless communication increasingly robust to channel noise and environmental interference. However, real-time communication scheduling in multi-hop wireless networks is different than wired networks due to *interference*, which adds a new constraint to the scheduling problem: a transmission cannot be scheduled on a radio link if another transmission is scheduled on any interfering link.

This paper provides an analysis of and heuristic for the real-time communication scheduling problem in multi-hop WSN. We formally define the problem based on a general model of industrial control applications, in terms of a connectivity graph, an interference matrix, and a set of periodic packet streams with fixed source and destination. The descriptions of the streams and deadlines are usually known in such industrial plants since these are built and run for a long time to perform a specific task. After deploying our system, we can also experimentally measure and model the interference under such known and controlled operating conditions. However, like most scheduling problems, the problem is NP-hard in nature, and even after properly modeling the workload and interference, the problem of creating a schedule for large number of real-time packet streams is challenging since no optimum algorithm are practical for it.

Our algorithm is practical for use during the design and analysis phase of WSN deployment. The idea is to estimate the *supply* of time slots for a stream based on its deadline, period, and hop count, and estimate the *demand* of that stream based on its spatial overlap with other streams. Streams are then prioritized based on their demand relative to their supply. In essence, this heuristic uses a topological analysis to *anticipate* the effects of radio interference in order to improve the prioritization of streams. Routes taken by the streams are also important since this relates to the amount of interference among transmission links. To address this issue we provide a route update heuristic that is applied iteratively in a lock-step fashion with the scheduling algorithm. This whole algorithm runs in polynomial time of $O(N^3d)$, where N and d are the number of streams and maximum deadline respectively.

We use a simulation-based study to demonstrate that this supply and demand-based heuristic along with the route update algorithm is able to schedule as much as 47% more steams than heuristics that use only supply or demand. The route update scheme contributes a $4\% - 26\%$ increase in the schedulability. We also show that this heuristic produces near-optimal schedules for approximately 10 packet streams in a 100 node network which is comparable in size to many existing fieldbus standards that currently supports a maximum

of 64-128 devices.

The contributions of the paper include:

(1) A novel analysis of simultaneously scheduling multiple real-time streams in a multi-hop WSN, while taking interference into account.

(2) A heuristic-based offline scheduling algorithm that prioritized the streams based on the supply and demand of time slots and anticipates which streams are likely to meet the deadline and which are not.

(3) An iterative route update rule that selects the streams with excess supply of time slots and pushes them apart from other interfering streams to reduce interference and increase schedulability while meeting deadlines.

II. Background and Problem Definition

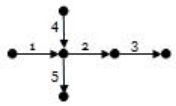
A. Models and Terminology

Network Model: The communication graph is a directed graph $G = (V, E)$, where $V = \{v_1, .., v_n\}$ is the set of nodes having a single radio interface and E is the set of possible links. Every node v_i has a transmission range t_i. The necessary condition for v_j to receive a message from v_i is $d_{i,j} \leq t_i$, where $d_{i,j}$ is the Euclidean distance between v_i and v_j. This means that not all sensor nodes v_j within the range of v_i has to form a link. Some links may not belong to E due to physical reasons like barrier or radio irregularities.

Link Model: A link between v_i and v_j is denoted by $l_{i,j}$. We assume successful link transmission in each time slot. To make this assumption practical we argue that the unsuccessful link transmission can be guarded against in a number of ways. For example, allowing multiple tries within the same time slot, using coding in upper layers, allowing multiple different paths between end nodes, assigning multiple time slots instead of just one slot, avoiding bad links altogether and so on. We also argue that real-time traffic inherently bear some redundancy and so some packet losses are easily absorbed. Any violation of this assumption may cause some packet losses but would never cause incorrect schedules.

Interference Model: To model the interference, we use a Boolean matrix I of the form: $I(l_{i,j}, l_{a,b}) = 1$, if $l_{i,j}$ has interference from $l_{a,b}$ and 0, otherwise. This matrix is capable of capturing various interference models such as hop based, range based, protocol model or thresholded SINR model [5]. For practical purpose this matrix can be estimated empirically, especially in an industrial process control plant where nodes are stationary and the environment is not open and the interference pattern is mostly deterministic. Such protocols exists in literature that are capable of detecting radio interference in WSN [6]. Moreover, to make this assumption true in practice, we can be more conservative. For example, choosing a conservative threshold in SINR model may ensures us interference free communication for the cost of less throughput [5]. But as long as the streams make their deadlines, we are not concerned about the throughput.

Stream: A stream σ is a 4- tuple, $\sigma = (s, d, p, dl)$ where $s, d \in V$ denote the source and the destination nodes, p denotes the period or interval when s fires and dl denotes the deadline of the packet relative to the instant when s fires. Period p is greater than or equal to dl. A *stream instance* is a stream with a start time st and specified by $\sigma = (s, d, p, dl, st)$. Because the periods of the streams are not equal, the scheduling table must schedule more instances of smaller streams than larger ones. Start time of consecutive two instances of a stream are p time units apart. The set of all stream instances is denoted by Σ. Hereafter, we use both the terms stream and stream instances to mean stream instances.



T1	T2	T3	T4
1	2	3, 4	5

Figure 1. An example with two streams: $1 \rightarrow 2 \rightarrow 3$ and $4 \rightarrow 5$ (left) and the schedule (right) is shown. Links 3 and 4 being non-interfering, they are scheduled in the same time slot T3. Both the streams meet the deadline of 4.

Routing Scheme: A routing scheme, $\pi = \{\pi_i\}$ is a set of routes. Under a given routing scheme, each stream σ_i has a fixed route π_i.

Schedule: A schedule is an assignment of links into time slots $[T_1, T_{max}]$. Since the streams are periodic, the value of $T_{max} = LCM(p_1, p_2, ..., p_{|\Sigma|})$ and we schedule T_{max}/p_i instances of the i-th stream within T_{max}. The k^{th} instance has the starting time $T_{1+(k-1)p}$. A *valid* schedule is one that does not violate the precedence of the links, deadline and interference constraints of any stream instance. An *optimum* schedule is one that is valid and schedules the maximum number of stream instances.

Schedulability: Schedulability is our performance metric. It has two forms. The first form is used when the workload is small and the optimum solution is computable within a reasonable time. The second one quantifies the performance of an algorithm when the optimum solution is not feasible to compute.

$$S_{op} = \frac{\text{No. of streams scheduled}}{\text{No. of streams in optimal scheduling}} \quad (1)$$

$$S_{st} = \frac{\text{No. of streams scheduled}}{\text{Total streams}} \quad (2)$$

Deadline Tightness: Deadline tightness is a measure of how tight the deadline is with respect to the average number of hops of all streams. This metric helps us to generate workloads with similar deadline constraint in different sizes of networks.

B. The Stream Scheduling Problem

An instance of the scheduling problem ρ is defined as a 4 tuple, $\rho = (G, \Sigma, \pi, I)$, where G is the communication graph, Σ is the set of stream instances, π is the routing scheme, and I is the interference matrix. Given ρ, our goal is to schedule the transmission links in such a way that all $\sigma \in \Sigma$ meet the deadlines while avoiding interference I. Figure 1 shows an example of scheduling problem and its solution. The communication graph has $|V| = 6$ and $E = \{1, 2, 3, 4, 5\}$. There are two streams: the first one follows the path $1 \rightarrow 2 \rightarrow 3$ and the second one follows the path $4 \rightarrow 5$. Interference matrix is such that links 1, 2, 4, and 5 interfere with one another. In this example, the period and

deadline of both the streams is 4. A schedule is shown on right.

To expose the nature of the problem, we make an analogy. If we assume a circular communication and interference range and assume that they are identical for any individual sensor, any single hop transmission can be thought of as a cylinder in spatial-temporal space with the transmission time as the cylinder height and interference range as the cylinder radius. An end-to-end transmission can then be pictured as a series of cylinders with the number of cylinders equals to the number of communication hops. By transforming end-to-end transmission into a geometric description, the original constraints of the problem are endowed with new meaning: the interference constraint implies that no two cylinders can overlap; the communication range constraint means that two adjacent cylinders cannot be far away from each other in the spatial plane; the deadline constraint means that the final cylinder should not exceed a given height; the precedence constraint means that cylinders can only be piled upwards. To schedule multiple streams now equals to be packing required cylinders in limited height, which can be interpreted to constrained cylinder-packing problem, a well-known NP-Hard problem.

III. IAA Scheduling and Routing

We at first discuss three scheduling heuristics: the first one is based on deadline, the second one is based on interference and the third one is our scheduling algorithm that combines these two. We then present our route update scheme that is applied after the scheduling is done to improve the workload for the next round of scheduling. We call the combination of these scheduling and routing heuristics the *IAA Algorithm*.

A. Laxity Heuristic

Laxity is a variable indicating how many time slots delay (waiting) a stream can tolerate during its remaining downstream transmissions. As we are incrementally building the schedule starting from time zero, the laxities of the streams are dynamically changing in this creation of the schedule as we proceed. Laxity of a partially scheduled stream at time slot t is defined as:

$$laxity(t) = start + dl - est(t) - \delta \times dist_{\pi}(x, d)$$

Here $est(t)$ stands for the earliest time slot (at and after t) where a link can be scheduled without interference, δ is the average transmission delay and $dist_{\pi}(x, d)$ is the remaining hops. The concept of est is motivated by the classic real-time scheduling works in [7] [8]. Based on this, the definition of $laxity$ is used in [9] to perform a real-time schedulability analysis which is the baseline that we compare our algorithm with. At any point of scheduling, the less the laxity of a stream the more its urgency is and if ever the laxity becomes negative, there is no way to schedule the rest of the stream anymore. The strength of this heuristic is that it is directly related to the remaining deadline of the stream and its performance is very good when streams have large deadlines. The addition of est in the formula of laxity improves its accuracy in choosing the right stream since by doing so it considers not only the remaining time and distance, but also other streams' behavior. One limitation of laxity is that it does not consider possible downstream (future) interference of current stream with other ones. Also the performance of the laxity heuristic is very poor when the deadline is very tight or streams are critically interfering with each other.

B. Spatial Overlap Heuristic

Given a routing scheme, streams may use the same link or links within the interference range creating spatial overlap. The more the overlap is the harder the workload is to schedule. We define *downstream spatial overlap* or simply *spatial overlap* as the number of different links of other streams a stream interferes with. While counting, we only consider those links that are yet to be scheduled. Now a scheduling algorithm might be designed that is based only on the downstream spatial overlap. The intuition behind this is that the stream that has the least downstream overlap causes less harm to other streams. Hence, scheduling such streams first would be a good idea. Alternatively, streams showing a large amount of downstream spatial overlap, if scheduled, is going to block many time slots that might be assigned to other streams. Filtering out such streams is therefore a good idea as well. Like laxity, downstream spatial overlap is also a dynamic heuristic. A stream with a high value of initial spatial overlap might become one with much less downstream spatial overlap as the algorithm proceeds and schedules all other links that were vulnerable to this in the beginning. The strength of this heuristic is that under very tight deadlines or high spatial overlap it performs better than the laxity heuristic. But its limitation is that, it does not consider deadline and so performs poorly when the deadline is loose and overlap is minimal.

C. Anticipatory Scheduling Heuristic

The idea behind anticipatory scheduling is to anticipate which streams are most likely to make their deadlines and which are not, based on the estimated supply and demand of time slots in the future. By future we mean the later iterations of our scheduling algorithm. Both the heuristics described earlier, laxity based and overlap based, inherently does some form of anticipation. For example, the laxity heuristic tries to give more priority to those streams whose deadlines are near. It delays decisions about the streams with larger laxity and filters out streams with negative laxity as they are going to miss the deadline anyway. On the other hand, the spatial overlap heuristic hypothesizes that streams having less spatial overlap with others are more likely to get through. It delays the decisions about the streams that have more overlaps with others and in the end, if not all streams are scheduled, it leaves out streams that were critically criss-crossed with other streams. Our proposed algorithm fuses both these heuristics, shapes their anticipatory natures and overcomes their limitations to yield a much better schedulability.

Laxity of a stream is the number of excess time slots after scheduling all the remaining links. These are the slots that

Algorithm 1 : IAAScheduler(Σ, π, I)

1: $Failed \leftarrow \phi, Marked \leftarrow \phi, Schedule \leftarrow \phi$
2: **while** $\Sigma \neq \phi$ **do**
3: **for** $i \leftarrow 1$ to $|\Sigma|$ **do**
4: $(D_i, S_i) \leftarrow IAAValues(\sigma_i, Schedule, I)$
5: **if** $S_i < 0$ **then**
6: $Failed \leftarrow Failed \cup \{\sigma_i\}$
7: $\Sigma \leftarrow \Sigma - \{\sigma_i\}$
8: **else if** $S_i > D_i$ **then**
9: $Marked \leftarrow Marked \cup \{\sigma_i\}$
10: $\Sigma \leftarrow \Sigma - \{\sigma_i\}$
11: **else**
12: $h_i \leftarrow \frac{D_i}{S_i + \epsilon}$
13: **end if**
14: **end for**
15: $\sigma \leftarrow$ Stream with the lowest h.
16: $Schedule \leftarrow$ Schedule next hop of σ at earliest slot.
17: $\Sigma \leftarrow \Sigma - \{\sigma\}$, if this is the last link of σ.
18: **end while**
19: **for** $i \leftarrow 1$ to $|Marked|$ **do**
20: $Schedule \leftarrow$ Schedule rest of the marked stream
21: **end for**
22: $Laxity \leftarrow S_i$ values of all streams.
23: **return** $(Schedule, Laxity)$

Algorithm 2 : IAARouting(G, Σ, Laxity)

1: $\Sigma \leftarrow$ Sort Σ in increasing order of Laxity
2: **for** $i \leftarrow 1$ to $|\Sigma|$ **do**
3: $l \leftarrow |\pi_i| + Laxity_i$
4: $\pi_i \leftarrow$ Min cost path of σ_i in G with length $\leq l$
5: $G \leftarrow$ Increase weight of all nodes on π_i in G by ϵ
6: **end for**
7: **return** π

it can offer to other interfering links. Laxity can be thought of as the future *supply* of time slots. Spatial overlap on the other hand is the count of links that are potential candidates for those slots and can be thought of as the future *demand* of slots. Now there are two cases: (1) $supply > demand$: It implies that the stream is going to meet the deadline, and (2) $supply < demand$: It implies that the stream will miss the deadline. But the catch is that, while laxity equals the supply of slots available to other streams, downstream spatial overlap is not in general equal to the demand. This is because, at one extreme, the deadline of all other interfering streams might be far away so that the actual demand for slots is zero, and at the other extreme all other interfering links might interfere with one another making the demand for slots as high as the total count of interfering links. Let us call these two cases: $min\ demand$ and $max\ demand$ respectively. We now have three scenarios: (1) $supply > max\ demand$: The stream will surely make its deadline. This stream is now marked and taken out of consideration till the end. Slot Assignment is done sequentially (avoiding interference) after all streams are either scheduled or failed or marked, (2) $supply < min\ demand$: The stream missed the deadline. It failed, (3) $min\ demand <$ supply $< max\ demand$: Scheduler chooses the stream with lowest value of $\frac{max\ demand}{supply}$ and schedules its next hop at the earliest possible time slot. The rationale behind choosing the stream with minimum value of $\frac{demand}{supply}$ is that it gives more priority to those streams that have either less downstream spatial overlap (demand) or more laxity (supply) or both. By this combination of laxity heuristic along with downstream spatial overlap, we can anticipate which stream is more likely to make the deadline. This heuristic also enables early detection of deadline misses and successes and by doing so it reduces the complexity and size of the search space.

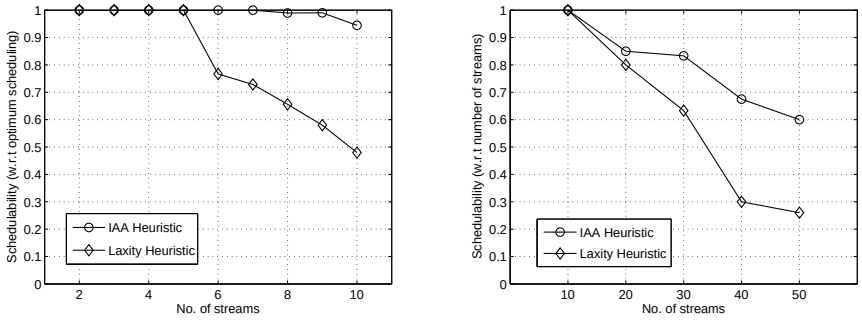
D. Route Update Algorithm

In our algorithm, although we have kept the scheduling and routing separate, these two are influenced and guided by each other. Scheduling is always done over a fixed routing scheme. If some streams do not get scheduled, based on the result of scheduling we adjust the routing scheme. This adjustment is guided by the laxities of all the streams. The intuition is that, streams that are fully scheduled, but still have good laxity, might be re-routed and still be able to make the deadline. The more the laxity, the longer a path it can take. In this way, streams spread out in a less congested part of the network and make the workload better for the next scheduling phase.

In our implementation, we start with the shortest paths for all the streams. After the scheduling is done, we order the streams based on their laxities at that moment. We then take the stream with the smallest laxity, route it through the shortest cost path and increase the weight of all the nodes on that path by a small amount. The same thing is done with all the streams in the order of their laxities. In this way, a stream avoids those paths that are already taken by previous ones and takes a less congested and probably a longer path. It may happen that by taking a longer path a stream becomes unschedulable. To avoid this situation, we restrict that the increase in path length be less than the laxity.

E. Complexity

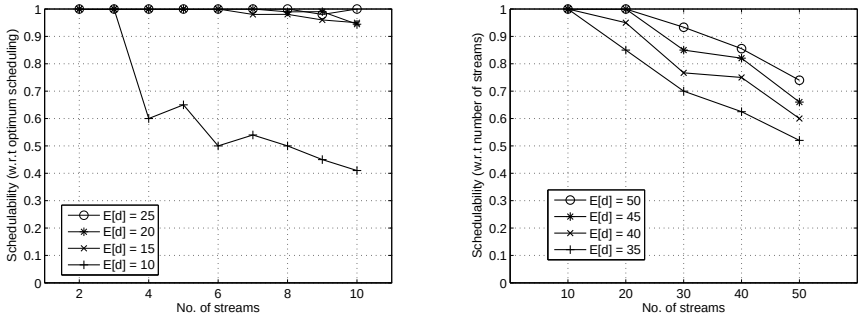
We assume that N, L, K, and d denote the number of streams, links, maximum path length and maximum deadline, respectively. The **for loop** from line 3 in Algorithm 1 schedules at least 1 link at each iteration, the **while loop** at line 2 runs $O(L)$. For computation at line 4, we pre-calculate D_i values and store them in a lookup table in $O(N^2K^2)$. Table entry stores the value of downstream overlap between σ_i and σ_j. Calculation of D_i and S_i then takes $O(N)$ and $O(Ld)$, respectively. Line 15 and 16 runs in $O(N) + O(Ld)$. Line 19 runs in $O(NLd)$. So the overall complexity is $O(N^2L + NL^2d + N^2K^2)$. Assuming L is $O(N)$, it becomes $O(N^3d)$. The route update procedure shown in Algorithm 2 sorts the streams into increasing order of laxities and re-routes their paths as described earlier. The complexity of this procedure is $O(NlgN) + O(N) \times [O(N^2) + O(L)]$. Assuming L is $O(N)$, it becomes $O(N^3)$. We apply these two algorithms



(a) For small number of streams we compare IAA and laxity heuristic with optimum scheduling. Schedulability of the laxity heuristics drops sharply, but IAA's performance is steady and near optimum.

(b) For large number of streams we compare the number of streams scheduled by IAA and laxity heuristics. IAA always schedules more streams than laxity heuristic.

Figure 2. Schedulability of IAA is always better than the laxity heuristic.



(a) For small number of streams, we compare IAA with the optimum scheduling. IAA is near optimum unless the deadline is severely tight.

(b) For large number of streams, we count the number of streams scheduled by IAA. Even in high loads with tight deadlines, we see a good schedulability.

Figure 3. Effect of varying expected deadline on IAA's performance.

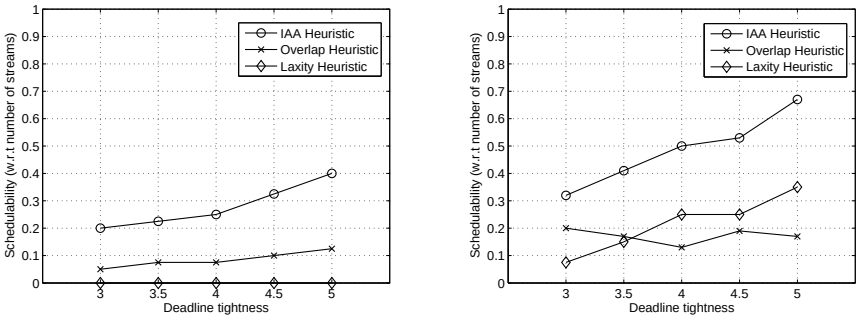
in lock step for 10 iterations. The overall complexity of IAA is thus $O(N^3 d)$.

IV. Experimental Setup

All algorithms are programmed in Matlab and run in a machine with 1.7GHz Pentium 4 CPU and 2GB memory. The performance metric is schedulability (Section II-A). To get a data point, we run an algorithm on 50 test samples and take the arithmetic mean. The 95% confidence interval for mean is within $\pm 1.05\%$. A random problem instance is characterized by a set of network configurations and workload properties, i.e. it defines network topology, communication and interference range, streams with sources, destinations, periods, and deadlines. Different streams' deadlines (and periods) either follow Poisson distribution or we use the 'deadline tightness'. Illegal values such as zero are discarded. To demonstrate the relative performance of the algorithms, we assume a square grid network topology with unit distance between two horizontal and vertical adjacent nodes. For the sake of simplicity, we assume a range based interference model and regulate that every sensor's radio range is 1.2 and its interference range is 2.5, and so two horizontal or vertical adjacent nodes can communicate while the diagonal two cannot. But we are not constrained by the choice of the model as long as it can be expressed in the form discussed in Section II-A.

V. Results

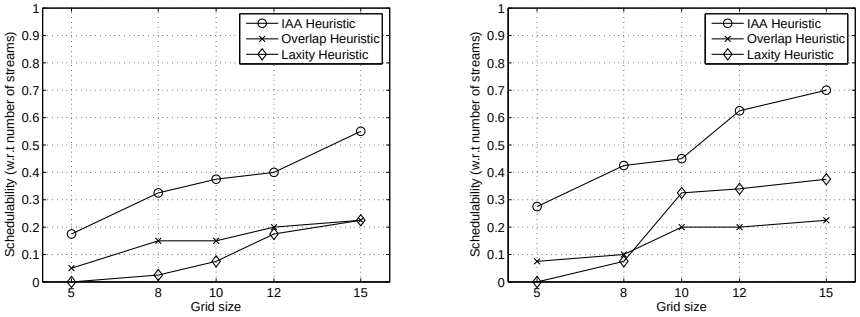
Figure 2 shows the relative performance of IAA heuristic and laxity heuristic. Laxity heuristic is taken as a baseline since like most heuristic based scheduling algorithms it uses the deadline as the key element whereas in our case, besides deadlines, we consider other issues of hop counts, spatial overlaps, and a supply and demand based anticipatory knowledge. We applied route updates for 10 iterations, starting from the shortest routes for all streams. The laxity heuristic is also applied on these updated workloads and its best performance is taken. Figure 2(a) shows the schedulability of these two heuristics with respect to optimum scheduling. The number of streams are limited to 10 so that we have an optimum solution to compare with. The expected deadline and period in this experiment is 20. We see that there is a sharp decline in the schedulability of the laxity based heuristic as the number of stream increases. But the IAA heuristic always performs better than the laxity based heuristic. The average relative performance of IAA with an optimum schedule for this workload is as high as 98.03%. For 10 streams, IAA is 47% closer to the optimum than the laxity heuristic is. Note that the Y-axis is not the percentage of streams scheduled, rather it shows how close they are to an optimum scheduling. Figure 2(b) shows the schedulability in term of fraction of the



(a) 5x5 grid, 40 streams: The laxity heuristic performs severely poor in a small grid. The overlap heuristic maintains a steady performance in this case.

(b) 10x10 grid, 40 streams: The laxity heuristic performs poor under tighter deadlines, but as deadline looses it catches up with the overlap heuristic.

Figure 4. The laxity heuristic is more sensitive to deadline tightness than the overlap heuristic.



(a) Deadline tightness of 3, 40 streams: The laxity heuristic lags behind the overlap heuristic. But it shows an upward trend as grid size increases.

(b) Deadline tightness of 4, 40 streams: The laxity heuristic is better than the overlap heuristic in larger grids. But it suffers when the grid size is smaller.

Figure 5. The laxity is better when we have larger grids (less overlaps) and loose deadlines.

total streams scheduled for large number of streams. Expected deadline is kept at 40 for all cases so that the curve shows the trend in change of schedulability with increasing streams and increasing tightness of deadline. We see that, with the increase of the number of streams, the schedulability decreases for both. But IAA always retains its superiority over the laxity heuristic. Under high load of 50 streams, the IAA algorithm schedules 35% more streams than the laxity heuristic.

VI. Analysis

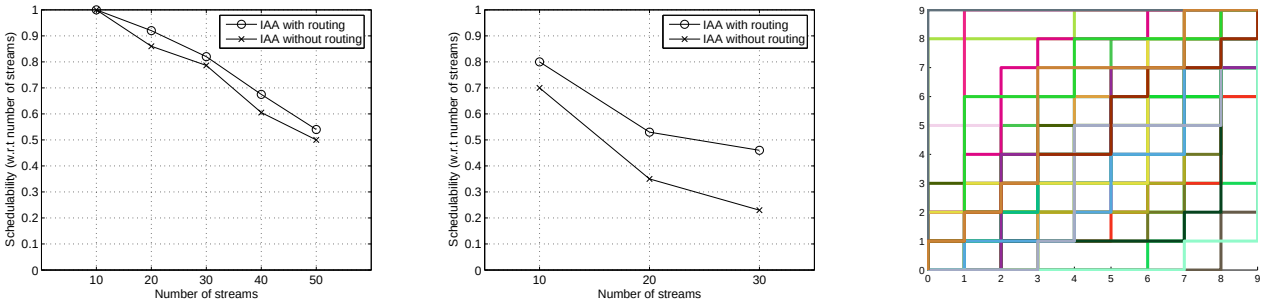
A. Effect of Expected Deadline

Figure 3(a) shows the schedulability of IAA with respect to an optimum scheduling for different deadlines. The purpose of this experiment is to demonstrate that IAA is not sensitive to the tightness of deadline unless the deadline is severely tight. For example, the top three lines of this figure stands for expected deadline values of 25, 20, and 15, respectively. Schedulability of IAA in these cases are almost similar to the optimum. In some cases, e.g. for 10 streams, we see some performance degradation of IAA, but still we get approximately 95% schedulability. A deadline of 10 is a case when IAA fails to perform well. The reason is that according to the supply and demand principle of IAA, deadline being too tight the supply of remaining time slots is way too small than the demand of slots afterwords. The priorities of all the streams becomes excessively high and IAA fails to pick the right stream to schedule. Figure 3(b) shows the trend in change in schedulability of IAA against both the number of streams and deadlines. The number of streams scheduled decreases with decreasing deadline or increasing number of streams. It is expected that for the same deadline, the more the number of streams the harder it is to schedule them. Streams in these cases are highly interfering with each other and thus the demand for time slots increases relative to supply of time slots. A similar effect is seen when deadlines are decreased for the same number of streams.

B. Performance Gain from the Combination of Laxity and Spatial Overlap Heuristic

Figure 4 and Figure 5 compare the schedulability of IAA with its two major components: the laxity and the spatial overlap heuristics. To demonstrate the relative performance of these two components and show the combined effect, we perform two sets of experiments. First, we vary the deadline tightness while keeping the grid size fixed and second, we vary the grid size while keeping the deadline tightness fixed. In these experiments, we do not apply the routing update so that we can demonstrate the performance gain of IAA from the combination of the two components only.

In Figure 4, we vary the deadline tightness from very tight (tightness of 3) to loose enough (tightness of 5) value in increments of 0.5. We keep the number of streams fixed at



(a) Random workload: As streams are already spread out all over the grid, performance improvement due to route update is small.

(b) Workload with same initial route: The schedulability of IAA is significantly improved by rerouting the streams.

(c) Streams having the same route at the beginning, spread out all over the grid after 10 iterations of route updates.

Figure 6. Effect of applying route update on IAA.

40. The workload is random as before and the routing scheme is the shortest path for all streams and the average path length is 7.56 hops. We see that the laxity heuristic performs the worst (almost zero) when the workload is highly congested with lots of streams in a small sized grid (Figure 4(a)). This is because, the laxity heuristic does not consider downstream overlap and depends mainly on deadline which is kept similar for all streams. The spatial overlap heuristic being aware of overlaps, schedules at least some streams that have less overlap with others. In a larger sized grid (Figure 4(b)), the laxity heuristic shows an upward trend in schedulability with a decrease in deadline tightness. It performs bad (only 6%) when the deadline is very tight (value of 3). This is because, when all the streams have similarly tighter deadline and their hop counts are also not so different, laxity fails to prioritize them correctly. In this case, it switches priority very fast among streams, schedules many links, but in the end fails to get many streams scheduled. Spatial overlap in those cases is a better estimate of priority as it does not depend on deadline and maintains a steady performance (avg. 18%). When deadlines are loose, the laxity heuristic schedules more links from the same stream before the priority changes and eventually schedules more streams than it does with tighter deadline case. The IAA algorithm performs better than both the two since in the case of tighter deadlines, it gains performance from downstream spatial overlap and when deadlines are not so tight a combination of laxity and downstream spatial overlap direct its choices to get more streams scheduled.

In Figure 5, we vary the amount of overlap among the streams to see its effect on laxity and spatial overlap. We do so by varying the size of the grid from 5×5 to 15×15 while keeping the number of streams fixed at 40. We see that when the deadline is tighter (Figure 5(a)) or when the grid size is smaller (in Figure 5(b)), the spatial overlap heuristic is dominating over the laxity heuristic. This is already evident from our previous discussion of Figure 4. Although, laxity lags behind the spatial overlap heuristic in these cases, we see that as the grid size increases it catches up with the spatial overlap heuristic. This is because, although the deadline here is similarly tight for all the streams, in a larger grid, the average path length of the streams being larger, streams get more breathing space in terms of deadline (recall that the deadline tightness is defined in terms of the average path length of all the streams). In smaller grids with fixed deadline tightness, we get smaller deadlines for all streams, making the scheduling job harder for laxity heuristic.

Using our supply and demand analogy we comment that the spatial overlap heuristic always gives high priority to streams that are less demanding. It does not account for the future supply of time slots. On the other hand, the laxity heuristic gives high priority to the streams for which the future supply is less. It does not account for the future demand of time slots. IAA makes a balance between the supply and demand and gives high priority to the stream for which is the ratio between the future demand and the future supply of time slots is smaller. In this way, IAA is far more accurate in picking the suitable stream for scheduling than only laxity or only spatial overlap based heuristics.

C. Performance gain from route update

A further increase in schedulability is achieved by updating the routes iteratively. Figure 6 shows the relative performance of IAA with and without the application of route updating. In Figure 6(a), workloads are generated randomly and initial routes are the shortest paths from source to destination nodes. We measure the schedulability with this initial routing scheme and then apply route update rule for 10 iterations and measure the schedulability again. We see a $4\% - 7\%$ increase in schedulability when routes are updated using our algorithm. This small increase in schedulability may seem not so significant, but we argue that for large number of streams we do not have optimum schedule at hand, so any increase in schedulability that takes us closer to the optimum is thus important. The reason of such a gain is that for the same workload, a change in routes brings changes in both the spatial overlap and laxity. Our routing update algorithm iteratively searches for the routing scheme that lowers the spatial overlap while keeping the laxity as good as possible. In this way we get a better workload at later iterations and hence the increase in performance. The gain from our route update algorithm largely depends on the initial workload. When the workload is random and streams spread all over the grid, a change in

routes may not be beneficial. But when the initial routing is such that most streams follow the same route or highly interfering routes, it is indeed possible to gain much better improvement in schedulability by changing the routes. To see the power of route update algorithm we apply it to special kind of workload where all the streams start from the same source (at $(0, 0)$) and end at the same destination (at $(9, 9)$) in a 10×10 grid. We measure the schedulability of IAA on this workload before and after route update and get the result shown in Figure 6(b). We see a $10\% - 26\%$ increase in schedulability with route updates this time. Figure 6(c) shows changed routes after 10 iterations. We see that updating the routes iteratively has guided the streams to spread out nicely all over the grid, making the workload suitable for IAA to schedule better.

VII. Related Works

There are a number of scheduling algorithms for stream transmission. Previous performance analysis [10] shows that DSR [11] outperforms other ID-based protocols in terms of packet delivery ratio, but it does not consider time constraints. RAP [12] uses a velocity monotonic algorithm that takes into account both time and distance and outperforms DSR in terms of end-to-end deadline miss ratio, but is suitable for streams having a single sink and no guarantee is provided to individual streams. Also, RAP needs support from MAC layer. SPEED [13] maintains a desired delivery speed across the sensor network by a combination of feedback control and non-deterministic geographic forwarding. It outperforms AODF [14], GF [15], and DSR in terms of end-to-end deadline miss ratio. But it is not suitable for arbitrary streams; rather it supports streams having a single source or single destination.

RI-EDF [16] is a MAC layer protocol that provides a real-time guarantee by utilizing the rules of EDF to derive a network schedule. But in this work the network has to be fully linked, i.e. every node is within transmission range of every other node. Another MAC layer protocol, as described in [17], provides soft real-time and bandwidth guarantees by avoiding packet collisions. It uses multiple channels which may not be possible for all sensors. And also this protocol assumes that the wireless link is not affected by jamming or EMI which means that the only way a transmission can fail is because of interference from other streams.

Abdelzaher et. al. [18] proposed a sufficient condition for the schedulability of real-time streams on a wireless sensor network. This work provides important insight into the limitations of achieving guaranteed delays in WSN. In our work we provide an actual scheduling heuristic that creates schedules for every link in the WSN. [19] provides timeliness guarantees to multi-hop streams by explicitly avoiding collisions and scheduling messages based on per-hop timeliness constraints in real-time robotic sensor applications. But it is not suitable for large scale networks. Another transmission scheduling algorithm is presented in [20] that performs better than TDMA based protocols in terms of both real-time performance and throughput. But this algorithm is mainly for scheduling queries from the base station and their responses. So there is a single destination for the streams here.

VIII. Conclusions

In this paper we presented a heuristic based static offline scheduling algorithm for periodic real-time streams in WSN. Our algorithm is practical for use during the design and analysis phase of WSN deployment. In environments like industrial process control plants, we can measure the interference pattern, identify the workload and then use our algorithm to produce a schedule to provide real-time guarantees. Existing fieldbus standards (e.g. PROFIBUS, CAN) that supports a maximum of $64 - 128$ devices can be replaced with WSN devices and our algorithm can be used to generate near-optimal schedule for them. In order to support very large networks with thousands of devices, a hierarchical scheduling approach may be applied which we consider in our future work.

References

[1] U. Jecht, W. Stripf, and P. Wenzel, "Profibus: Open solution for the world of automation," *The Industrial Information Technology Handbook*, 2005.

[2] G. Cena and A. Valenzano, "Operating principles and features of CAN," *The Industrial Information Technology Handbook*, 2005.

[3] G. Karayannis, "Standards-based wireless networking alternatives," *Sensors Magazine*, November 2003.

[4] J. Mitola III, "Cognitive radio," Ph.D. dissertation, KTH, Stockholm, Sweden, March 2000.

[5] R. Maheshwari, S. Jain, and S. R. Das, "A measurement study of interference modeling and scheduling in low-power wireless networks," in *SenSys*, 2008, pp. 141–154.

[6] G. Zhou, T. He, J. Stankovic, and Abdelzaher, "RID: Radio interference detection in wsn," in *Infocom*, 2005.

[7] W. Zhao, K. Ramamritham, and J. Stankovic, "Scheduling tasks with resource requirements in hard real-time systems," *IEEE Transactions on Software Engineering*, vol. 13, pp. 564–577, 1987.

[8] W. Zhao, K. Ramamritham, and J. A. Stanovic, "Preemptive scheduling under time and resource constraints," *IEEE Trans. Comput.*, vol. 36, no. 8, pp. 949–960, 1987.

[9] V. Prasad, T. Yan, P. Jayach, Z. Li, S. H. Son, J. A. Stankovic, J. Hansson, and T. Abdelzaher, "ANDES: an analysis-based design tool for wireless sensor networks," in *RTSS*, 2007.

[10] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Mobicom*, October 1998.

[11] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*. Kluwer Academic Publishers, 1996, pp. 153–181.

[12] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, , and T. He, "RAP: a real-time communication architecture for large-scale wireless sensor networks," in *RTAS '02*, September 2002, pp. 55–66.

[13] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher, "Speed: A stateless protocol for real-time communication in sensor networks," in *ICDCS '03*, May 2003, pp. 46–55.

[14] C. E. Perkins and E. M. Royer, "Ad-hoc on demand distance vector routing," in *WMCSA '99*, February 1999, pp. 90–100.

[15] B. Karp and H. T. Kung, "Gpsr: Greedy perimeter stateless routing for wireless networks," in *MobiCom 2000*, August 2000, pp. 243–254.

[16] T. L. Crenshaw, S. Hoke, A. Tirumala, and M. Caccamo, "Robust implicit edf: A wireless mac protocol for collaborative real-time systems," *ACM TECS*, vol. 6, no. 4, p. 28, September 2007.

[17] B. D. Bui, R. Pellizzoni, M. Caccamo, C. F. Cheah, and A. Tzakis, "Soft real-time chains for multi-hop wireless ad-hoc networks," in *RTAS*, April 2007, pp. 69–80.

[18] T. Abdelzaher, S. Prabh, and R. Kiran, "On real-time capacity limits of ad hoc wireless sensor networks," in *RTSS*, December 2004.

[19] H. Li, P. Shenoy, and K. Ramamritham, "Scheduling messages with deadlines in multi-hop real-time sensor networks," in *RTAS*, 2005.

[20] O. Chipara, C. Lu, and G.-C. Roman, "Real-time query scheduling for wireless sensor networks," in *RTSS '07*, December 2007, pp. 389–399.