Interference-Aware Real-Time Flow Scheduling for Wireless Sensor Networks

Octav Chipara*, Chengjie Wu†, Chenyang Lu†, William Griswold*
* University of California San Diego
† Washington University in St. Louis

***Abstract*—With the emergence of wireless sensor networks, an enabling communication technology for distributed real-time systems, we face the critical challenge of meeting the end-to-end deadlines of real-time flows. This paper presents Real-time Flow Scheduling (RFS), a novel conflict-free real-time transmission scheduling approach for periodic real-time flows in wireless sensor networks. In contrast to existing transmission scheduling algorithms that ignore interference between transmissions or prevent spatial reuse within the same channel, RFS supports spatial reuse through a novel interference-aware transmission scheduling. While recent work on conflict-free transmission scheduling focused on specialized communication patterns such as queries and converge cast, RFS is designed for peer-to-peer real-time flows with arbitrary inter-flow interference. Moreover, RFS has three salient that make it particularly suitable for real-time systems: First, RFS includes a real-time schedulability analysis that accounts for interference between real-time flows. Second, RFS improves reliability by incorporating retransmissions in a flexible scheduling scheme. Finally, RFS enhances scalability by dividing the network into neighborhoods and provides real-time performance for flows crossing multiple neighborhoods through a novel application of the Release Guard protocol. RFS was evaluated through simulations based on the traces collected from an indoor wireless sensor network testbed. Compared to a traditional TDMA protocol, RFS reduces flow latencies by up to 2.5 times, while improving the real-time capacity by as much as 3.9 times.**

I. Introduction

Recent years have seen the adoption of wireless sensor networks (WSNs) as communication infrastructure for distributed real-time applications such as industrial process monitoring and control, structural health monitoring, and patient monitoring. These applications require real-time communication over multi-hop WSNs. While significant advances towards this goal have been made, existing solutions usually falls short in one (or more of) the following aspects:

- Existing results are often derived for *simplified communication workloads* such as converge-cast [1]–[3] or query services [4] where data is routed from sensors to a single base station. While this leads to elegant solutions, it also limits the applicability of these protocols. For example, in contrast to current centralized architectures in which all communication goes through one or a few gateways, to achieve higher scalability, the next generation of industrial process monitoring and control will require multiple control loops to be established between arbitrary sensors and actuators using real-time flows.
- Existing solutions often adopt *unrealistic interference* models or *ignore interference* [1]–[3]. For example, WirelessHART – a standard for sensor-actuator networks – prohibits concurrent packet transmissions within the same channel. As a result, the scale of existing WirelessHART networks is limited.
- Effective real-time solutions must also ensure *reliable* data delivery to end-points under variable workloads and network dynamics. This is difficult since packet retransmissions must be integrated effectively during scheduling and real-time analysis.
- The computation of transmissions schedules is often performed in a centralized fashion [1], [4]. This approach limits both system *scalability* as well as the capability of a system to adapt to workload and network *dynamics*.

In this paper, we adopt a flexible communication model in which *real-time flows* may be established between arbitrary sources and destinations. Packets pertaining to the same flow are transmitted periodically at known rates and deadlines, potentially over multiple hops. Flows are a flexible communication primitive that is familiar to network programmers who commonly reason about network systems in terms of flows that are established between sources and destinations.

Under this flexible model, in this paper we propose *Real-time Flow Scheduling* (RFS) a novel transmission scheduling technique for flows and associated schedulability analysis. Classical TDMA protocols address the demands of randomized workloads by constructing fixed schedules that are difficult to adapt in response to workload changes. In sharp contrast, RFS is optimized for scheduling flows by taking advantage of their *temporal properties* and *precedence constraints*. The precedence constraints of flows are the result of hop-by-hop forwarding that requires a sender to receive a packet before forwarding it. Moreover, RFS dispenses with the construction of fixed schedules and determines the transmissions that may be executed concurrently *dynamically*. In addition, RFS has the following salient features:

Real-time analysis for rich workloads: We derived an analysis for computing the response time of flows with static priorities. This analysis enables us to support real-time communication in a wider range of WSN systems. The analysis features a novel dynamic programming approach that bounds the maximum interference between flows even when they have arbitrary sources and destinations or inter-flow interference.

Interference-awareness: In contrast to schedulability algorithms and analysis for WirelessHART, RFS facilitates spatial reuse to support higher data rates. RFS features a generic interference model that allows for interference to be modeled both as graphs and based on Signal-to-Noise plus Interference (SNIR) measurements. Recently, a number of protocols for assessing interference have been proposed [5]–[7]. RFS may be integrated with these protocols to enable spatial reuse.

Reliability mechanism: In traditional TDMA protocols, retransmissions are not included in the schedule and are triggered by the link layer. In RFS, we incorporate retransmissions during the scheduling process to improve performance.

Distributed operation: RFS scales up by dividing the network in neighborhoods: the state maintained by a node is limited to its neighborhood. The real-time performance of flows that crosses multiple neighborhoods is ensured through the novel application of the Release Guard algorithm [8].

The remainder of the paper is organized as follows. In Section II, we present the related work. The flow and network models under which RFS operates are presented in Section III. A centralized version of RFS is presented in Section IV. The challenges of handling workload and network dynamics are discussed in Section V. We present the design of the distributed RFS is presented in Section VI. Simulation results based on traces collected from a WSN testbed are included in Section VII. Conclusions are presented in Section VIII.

II. Related Work

Real-time communication protocols proposed for WSNs can be categorized into contention-based or TDMA-based approaches. Contention-based protocols usually support real-time communication by manipulating the parameters of CSMA/CA such as the initial back-off, contention window, or sleep schedule [9]–[13]. Contention-based protocols may be inappropriate for real-time systems that require *predictable* performance due to the randomized back-off approach.

TDMA-based approaches are attractive for real-time communication because they may provide predictable performance. Several TDMA protocols that provide bounded communication latencies have been proposed. These protocols incorporate effective heuristics for reducing latencies and improving real-time performance; however, a majority of existing protocols do not support prioritization [3], [14]–[17] which is essential for real-time communication. An example of a protocol that supports prioritization is Implicit EDF [18]. Implicit EDF divides a network into cells operating on different frequencies to ensure that transmissions occurring in different cells do not conflict. The protocols supports message prioritization within cells but not across cells. In contrast, RFS supports prioritization for multi-hop flows.

The adoption of the WirelessHART standard has renewed the interest of the community in real-time communication for WSNs. A number of scheduling protocols have been proposed for effectively scheduling packet transmissions under the WirelessHART model [1], [19], [20]. These solutions adopt a centralized approach to the construction of transmission schedules and do not support spatial reuses. As a result, the scalability of such approaches is limited. RFS overcomes these limitations by supporting spatial reuse and providing mechanisms improving scalability.

Abdelzaher et. al. [21] proposed a sufficient condition for determining the schedulability of real-time flows in WSNs. The work provides insights into the theoretical limits of the real-time capacity of multi-hop WSNs. However, the paper assumes an ideal MAC and, as a result, the obtained results may not be applied in practice. More realistic results have been obtained in the simplified case when data is routed over a shared routing tree. Most prominently, network calculus is used in [2] to compute bounds on the latency of messages generated by sensors that form a cluster-tree topology. Similarly, analyses for WirelessHART have been proposed [1], [20], however they are derived under the simplifying assumption that no packet transmissions may occur concurrently. The real-time analysis presented in this paper handles the more general case when spatial reuse is possible and flows may be established between arbitrary end-points.

In prior work, we proposed RTQS [4] an approach for support real-time collection through queries. The fundamental difference between RFS and RTQS (and many of the previously discussed protocols) stems from the different communication models they adopt. RTQS requires data to be routed from sensors to a single base station over a shared routing tree. In contrast, RFS supports more general workloads resulting from concurrent real-time flows. The added flexibility of flows violates the design assumptions of RTQS preventing us from directly applying the previously developed techniques to real-time flow scheduling. The focus of this paper is to develop new transmission scheduling techniques and real-time schedulability analysis for real-time flows. In addition, RFS has three unique features. First, it adopts a generic interference model which is sufficiently general to enable spatial reuse. Second, it features a novel technique for ensuring reliability by incorporating packet retransmissions during the scheduling process. Finally, RFS supports real-time flows in large networks through the novel application of Release Guard.

III. System Model

A. Flow Model

RFS adopts real-time flows as a communication primitive. A flow i may be established between any source and destination. Packets pertaining to the same flow are transmitted periodically with a phase (i.e., start time) of ϕ_i and period P_i. A deadline D_i is associated with each flow. We refer to the phase, the period, and the deadline of a flow as its temporal properties. A static priority is associated with each flow and used to provide differentiated service. For a flow i, a new *flow instance* is released in the beginning of each period. We use $J_{i,v}$ to refer to the v^{th} instance of flow i which is released at $r_{i,v} = \phi_i + v \cdot P_i$. For brevity, in the remaining of the paper we will refer to a flow instance simply as an instance.

We do not require the programmer to specify all flows when the system is started. The workload may be changed by adding

new flows, removing existing flows, or changing the priority and the temporal properties of existing flows. RFS is designed to handle these operations efficiently.

B. Network Model

A key challenge to the design of TDMA protocols is to model interference accurately as to enable spatial reuse. Recently, a number of interference models and protocols for assessing interference have been proposed [5]–[7], [22]. Rather than adopting any specific interference model, we define a *generic interference* model that is sufficiently general to allow for interference relations to be expressed as graphs [7] or based on Signal-to-Noise plus Interference (SNIR) measurements [5], [6], [22]. As a result, we decouple the problem of modeling and assessing interference from the problems of scheduling real-time flows and analyzing their real-time performance. This decoupling has two advantages. First, our scheduling techniques and real-time analysis may be applied under a multitude of interference models highlighting the generality of the proposed methods. Second, as more accurate interference models are developed, our techniques may be integrated with them for improved performance.

The generic interference model defines an abstract interface that is used during scheduling and analysis to reason about interference. This interface supports two types of queries. The interference model may be used to determine if a set of concurrent transmissions conflict. A set of transmissions conflict if at least one of the receivers cannot correctly decode its packet due to interference. Additionally, we allow queries for determining if a pair of transmissions (l_1, l_2) will conflict when up to C arbitrary transmissions may be scheduled concurrently. Essentially, this allows us to bound the worst-case interference from a bounded number of transmissions on l_1 and l_2. Next, we detail how this interface may be instantiated when interference is expressed as graphs or based on SNIR.

Interference may be represented as a graph. Using this model, two transmissions $\overrightarrow{AB}$ and $\overrightarrow{CD}$ are *conflict-free* ($\overrightarrow{AB} \parallel \overrightarrow{CD}$) and may be scheduled concurrently if (1) A, B, C, and D are distinct and (2) $\overrightarrow{CB}$ is not an edge in the graph. Similarly, $\overrightarrow{CD} \parallel \overrightarrow{AB}$, if $\overrightarrow{AD}$ is not an edge in the graph. A set of transmissions is conflict-free if all pairs of transmissions are conflict-free. A limitation of the graph model is that interference is not cumulative: the conflict of two transmissions $\overrightarrow{AB}$ and $\overrightarrow{CD}$ does not depend on the number of concurrent transmissions. The examples presented in this paper use this model due to the ease of representing interference as graphs.

Recent empirical studies show that SNIR-based interference models that capture the cumulative nature of interference are more accurate [5], [6]. According to this model, a set of transmissions is conflict-free if the SNIR of all receivers exceeds a threshold. Using SNIR model it is also possible to determine if two transmissions remain conflict-free when up to C arbitrary transmissions occur concurrently.

Methods for assessing interference relations can be classified as active or passive. Active methods use active probes to collect signal strength measurements. These measurements may be used to construct interference graphs [7] or to predict interference by constructing a mapping between packet reception rate and SNIR [5], [6], [22]. A disadvantage of active methods is that they introduce significant overhead due to active probing. A recent protocol – Passive Interference Measurement (PIM) [5] – reduces this overhead through passive interference measurements while identifying interference with high accuracy. RFS may be integrated with any of the above protocols to assess interference among nodes. It is important to recognize that interference relations may change over time. According, we assume that interference relations are reassessed periodically.

IV. Protocol design

RFS divides the problem of real-time flow scheduling into two parts (see Figure 1). First, we consider the problem of scheduling the transmissions of a *single flow in isolation*. RFS will construct *plans* according to which all instances of a flow are executed. A plan is the sequence of transmissions required to deliver data from the flow's source to its destination over multiple hops. The planner accounts for unreliable links and enforces the precedence constraints introduced by multi-hop forwarding during the construction of plans.

Next, we consider the problem of scheduling *multiple flows concurrently*. RFS's dynamic scheduler executes multiple flows concurrently based on their temporal properties and the previously constructed plans. The scheduler dynamically determines the transmissions that will be executed in each slot such that no conflicting transmissions are scheduled in the same slot and prioritization among flows is provided.

The division of the problem in two parts has several intrinsic advantages: (1) RFS isolates the concerns of handling precedence constraints and link unreliability (handled by the planner) from the concerns of handling interference and providing prioritization (handled by the scheduler). (2) RFS separates the process of constructing plans from their dynamic execution allowing us to develop a computationally efficient scheduler. (3) RFS executes flows dynamically based on their temporal properties rather than constructing an explicit transmission schedule. Therefore, flows may be added/removed without reconstructing an explicit schedule.

RFS works as follows: (1) Any node may initiate the creation of a new flow that has it as a source. The node first checks whether an existing plan may be used to execute the new flow. As discussed in Section IV-A, it is often possible to reuse plans existing plans to execute new flows. When this is not possible, the planner initiates the construction of a plan for the new flow. (2) Next, admission control is performed on the source to determine whether the new flow may be added without any flows missing their deadlines. (3) At run-time, the scheduler dynamically executes flows based on their plans and temporal properties.

The remainder of the section is organized as follows. We start by considering the problem of constructing plans that account for unreliable links (see Section IV-A). Next, we present the design and analysis of the centralized RFS (see

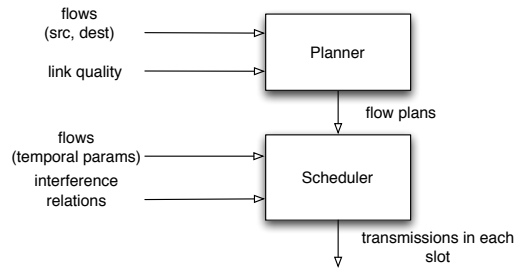


Fig. 1. RFS has two key components: a planner and a scheduler

Section IV-B). Mechanisms for scaling RFS to larger networks are presented in Section VI.

A. Plans

The plan of flow i is an ordered sequence of steps that contains the transmissions necessary to forward a packet from the source to the destination of flow i. A plan is a sequence of steps such that: (1) a single transmission is assigned in each step and (2) the order of transmissions respects the constraints of hop-by-hop forwarding. All instances of a flow are executed according the same plan. We use the following notations: Π_i denotes the plan of flow i, $\Pi_i[s]$ refers to the transmission assigned to step s of Π_i, and L_i is the plan's length. An example of a plan is shown in Fig. 3.

In the case when links are perfect, a plan is the routing path between the flow's source and destination. However, since all instances of a flow are executed according to the same plan, plans must be stable over time, otherwise plans would have to be reconstructed frequently. Unreliable links are usually handled through Automatic Repeat reQuest (ARQ). The ARQ mechanism automatically retransmits a packet that is unacknowledged up to a maximum number of retransmissions. Existing TDMA protocols do not coordinate their activity with the link layer. As a result, retransmitted packets are usually queued up for an additional TDMA frame until the sender is scheduled to transmit. This introduces significant delays when packets are retransmitted multiple times. An alternative is to increase the slot size to allow for retransmissions. However, since nodes are synchronized on slots boundaries, a TDMA protocol is forced to treat all links uniformly. Overestimating the number of retransmissions lowers throughput while underestimating it results in packet drops over low quality links.

RFS accounts for link unreliability by allowing a node to be assigned to multiple steps. In contrast to ARQ, we allow a maximum number of transmissions (MNT) to be specified per link. A number of link estimators evaluate the quality of a link using Expected Transmission Count (ETX) [23]. It is tempting to use ETX as an estimate of MNT. However, the ETX provided by the link layer estimates the average MNT. To ensure that plans remain stable over time, we are interested in estimating the worst-case MNT. The worst-case MNT may be estimated using Jacobson's algorithm [24]: Jacobson's algorithm computes both the average and standard deviation of ETX and then combines the two components.

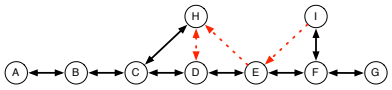


Fig. 2. Full and dashed lines denote communication interference edges

RFS may be integrated with existing routing mechanism to construct flows. The flow's source initiates the creation of a flow. As a route from the flow's source to its destination is constructed, each node along the path includes their MNT estimate. For example, a flow's route may involve three nodes A, B, C. When the MNT values for links (AB) and (BC) are 2 and 1, the flow's plan may reconstructed as: $(AB)(AB)(BC)$.

There are cases when it is possible to use reuse existing plans. Consider the case when a node A wants to establish a flow to B but there already is a flow that routes data from an arbitrary source through A to B. In this case, rather than constructing a new plan, A disseminates the part of the plan involving transmissions from A to B. We expect that this mechanism of reusing plans will effectively reduce of times new flows are constructed.

B. Centralized Scheduler

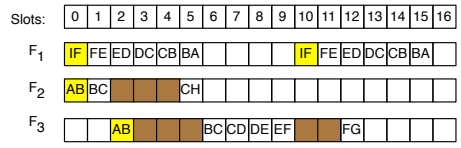
In this section, we consider the development of the centralized scheduler. Our goal is to devise a scheduler that supports high throughput and provides prioritization among flows. As a starting point for our solution, we will present a Greedy Scheduler (GS). Our analysis of GS's operation shows that it may prolong the response time of flows unnecessarily in some cases. To address this limitation, we present the RFS scheduler in Section IV-B3.

1) Greedy Scheduler (GS): At a high level, GS meets the requirements of high throughput and prioritization as follows. GS achieves high throughput by executing transmissions from multiple instances in the same slot when they do not conflict. If the execution of a low priority instance conflicts with the execution of a high priority instance, GS provides prioritization by suspending the low priority instance and executing the high priority instance.

A GS scheduler is deployed on each node. The GS scheduler maintains the following global state: the interference relations among all nodes, the flow plans, and flow parameters. All schedulers will independently perform the same transmissions in each slot if the above state is consistent. A discussion of how state is managed is deferred to Section V.

GS also maintains the following local state: a priority queue that contains all released instances and a per-instance counter, which indicates the step in the plan of an instance to be executed next. The local state does not need to be shared with other nodes. Let $J_{l,u}.step$ be the step counter of instance $J_{l,u}$. Instances in the priority queue are ordered according to their flow priorities. The release times and flow IDs are used to break ties among instances with the same priority.

GS determines a set of instances – the $exec$ set – that will be executed in each time slot s. The scheduler considers each instance J_l in the release queue in decreasing



Flow properties:

Flow	ϕ_i	P_i	D_i	π_i	Plan
F_1	0	10	10	1 (High)	(IF) (FE) (ED) (DC) (CB) (BA)
F_2	0	18	18	2 (Med)	(AB) (BC) (CH)
F_3	2	28	28	3 (Low)	(AB) (BC) (CD) (DE) (EF) (FG)

Fig. 3. Scheduling example according to GS. The light gray squares indicate the slots instances release times, the dark gray indicates slots in which a flow is suspended, the empty slots indicate no transmissions.

order of priority. J_l is added to the $exec$ set if it does not conflict with any of the higher priority instances that are already in the $exec$ set. This is accomplished by querying the generic interference model to determine if the transmissions $\Pi_l[J_l.step] \cup$ **transmissions**$(exec)$ are conflict-free. If no conflict is detected, J_l is added to the $exec$ set; otherwise, J_l is suspended. Multiple instances may be include in $exec$ when they do not conflict as to increase real-time capacity.

A key characteristic of GS is its operation during the following scenario. Consider the case when in a slot s three instances J_h, J_m, and J_l are released from a high priority flow h, medium priority flow m, and low priority flow l. Suppose that the following conflicts are present among the next-steps of the three instances: J_m conflicts with J_h, J_l conflicts with J_m, and J_l does *not* conflict with J_h. We call this algorithm greedy because – in an attempt to maximize throughput – it will execute J_h and J_l in the same slot since they do not conflict. However, our analysis shows that this greedy decision may increase the worst-case interference of flow m on l.

Consider the topology shown in Figure 2 in which three flows are established. Fig. 3 shows the transmissions executed in slots 0 – 16 by GS, when the three flows are executed according to the parameters summarized in the figure. In slot 0, instances from F_1 and F_2 are released. The scheduler determines that both instances may be added to the $exec$ set since the execution of the first steps in their respective plans (which involve transmissions $\overrightarrow{IF}$ and $\overrightarrow{AB}$) do not conflict. In slot 2, the scheduler adds the highest priority instance to the $exec$ set and then considers executing step 3 in the plan of the medium priority instance concurrently. However, this step involves transmission $\overrightarrow{CH}$, which conflicts with $\overrightarrow{ED}$ of the higher priority instance. To provide prioritization, the scheduler will execute only the higher priority instance. In slot 2, the first instance of F_3 is released. The first step in F_3's plan involves transmission $\overrightarrow{AB}$, which does not conflict with the transmission $\overrightarrow{ED}$, which has already been added to the $exec$ set. Accordingly, GS executes the two instances concurrently. This is an instance of the greedy choice we previously discussed. The scheduler continues to construct the schedule shown in the figure.

2) Real-time Analysis: The schedulability analysis is performed to determine if a new flow may be admitted without missing any deadlines. We assume that flow priorities are static, deadlines do not exceed periods, and that the periods, deadlines, and response times are measured in slots.

To compute the response time R_l of flow l, we construct a recurrent equation similar to the one used in the response time analysis for processor scheduling. GS will preempt the execution of a lower priority instance when it conflicts with the execution of a higher priority instance. Let $I_{l,h}$ be the worst-case interference that an instance of flow l can suffer due to an instance of a higher priority flow h. Instances from flow h will interfere with l for at most $\lceil \frac{R_l}{P_h} \rceil$ times. Thus, the response time of flow l is:

$$R_l(n+1) = L_l + \sum_{h \in hp(l)} \left\lceil \frac{R_l(n)}{P_h} \right\rceil \cdot I_{l,h} \qquad (1)$$

where $hp(l)$ is the set of flows with higher or equal priority to l and L_l is the length of l's plan. This equation can be solved using the fixed-point algorithm used in [25].

The only difference between Equation 1 and that in [25] is that the classical response-time analysis includes the worst-case execution of a task whereas Equation 1 involves the pair-wise interference of flows. However, the similarity to classical response times is deceiving. The system we consider differs significantly from a single processor; in flow scheduling there are multiple radios that share a wireless medium with complicated interference relations. These unique features of flow scheduling are captured by the inter-flow interference term, which we show how to calculate next.

As an aid to the interference calculation, we introduce the concept of a conflict matrix (see Figure 4). A conflict matrix $C_{l,h}$ captures the conflicts between a high priority flow h and low priority flow l. For each step in the plan of the high priority flow h there is a column in the conflict matrix. Similarly, for each step in the plan of the low priority flow l there is a row in the conflict matrix. The ordering of the rows and columns is the same as the ordering of the steps in the plans of the two flows. The entries in the conflict matrix indicate whether a pair of steps from the two flows may conflict. It is important to note that other flows may be executing concurrently with flows l and h. Accordingly, the matrix captures the *worst-case* wireless interference that l and h may observe while being executed concurrently with any other flow. This information may be obtained from the generic interference model when the maximum number of transmissions per slot is bounded. GS may be modified to ensure that the number of transmissions in a slot is bounded.

In order to compute the worst-case interference, it is necessary to understand how GS executes the two instances based on their conflict matrix $C_{l,h}$. As an example, let us consider the conflict matrix in Fig. 4. Let c be the step counter of the higher priority instance from h, and r be the step counter of the lower priority instance from l. GS will execute the two instances by avoiding the conflicts present in the conflict matrix. As the instances are executed, the indices r and c are incremented to indicate the progress towards completing their execution. The execution of an instance completes when it reaches the boundary of the matrix.

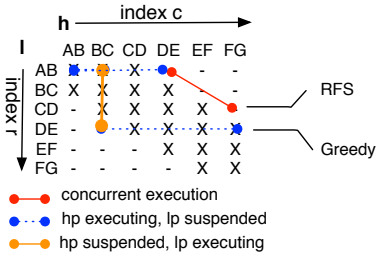


Fig. 4. Conflict matrix of two flows established between nodes A and F in topology shown in Fig. 2. An "x" indicates a conflict between a pair of transmissions and "-" indicates no conflict between transmissions.

compute-interf(C, r, c, $state$):
1: **if** $state[(r,c)] \neq \emptyset$: return $state[(r,c)]$
2: **if** r = num_rows **or** $c = num\_cols$: **return** 0
3: **if** $C[r,c]$ = 0:
4: $state[(r,c,1)]$= **compute-interf**(C, $r+1$, $c+1$, $state$) //case 1
5: $state[(r,c,3)]$= **compute-interf**(C, $r+1$, c, $state$) //case 3
6: $interf = \max([state[(r,c,1)], state(r,c,3)])$
7: **else:**
8: $state[(r,c,2)]$= 1 + **compute-interf**(C, r, $c+1$, $state$) //case 2
9: $state[(r,c,3)]$= **compute-interf**(C, $r+1$, c, $state$) //case 3
10: $interf = \max([state[(r,c,2)], state(r,c,3)])$
11: $state[(r,c)] = interf$
12: **return** $interf$

Fig. 5. Computing inter-flow interference using dynamic programming

Four cases may arise in the execution of the two instances. ***Case 1:*** The low priority and the high priority instances are executed concurrently. This occurs when there is no conflict between steps r and c of the two flows i.e., $C_{l,h}[r,c] = 0$. In this case, both r and c are incremented by one since both instance executed a step in their plan. The concurrent execution of instances may be visualized as lines parallel to the main diagonal as indicated by the red solid line (see Fig. 4).

Case 2: GS suspends the low priority instance to prioritize the execution of the higher priority instance. This case occurs when $C_{l,h}[r,c] = 1$. In this case, the indices r remains fixed while c is incremented by one as denoted by the horizontal dotted blue. In this case, the execution of h interfered with that of l for a slot.

Case 3: The low priority instance is executing while the higher priority instance is suspended. This case occurs only if the higher priority instance of h is suspended in order to give priority to another instance with even higher priority than h. Note that it is possible for this to occur regardless of the value of $C_{l,h}[r,c]$. In this case, r is incremented by one while the value of c remains the same. Executions of this type can be visualized as a vertical line.

Case 4: GS suspends both instances in which case the values of the two indices remain unchanged.

Based on the identified cases, an efficient algorithm for computing the worst-case interference may be devised (see Fig. 5). The algorithm takes as input the conflict matrix along with a start position in the conflict matrix as specified by indices r and c. The indicates r and c encode a possible phasing of flows l and h. Starting from position (r,c) the algorithm considers the execution of the instances according to cases 1 – 3. The worst-case interference may be computed by taking the maximum of the interferences of the three cases. It

scheduler(slot s**):**
1: $released = released \cup$ **released-in-slot**(s)
2: $exec = \emptyset$; $suspend = \emptyset$
3: **for each** $J_{l,u}$ **in** $released$**:**
4: **if interferes**($\Pi_l[J_{l,u}.step] \cup$ **transmissions**($exec$))**= True:**
5: $will\_exec$ **= False**
6: **else:**
6: $will\_exec$ **= True**
7: **for each** $J_{h,v}$ in $suspend$**:**
8: **if interferes**($\{\Pi_l[J_{l,u}.step], \Pi_h[J_{h,v}.step]\}$, C = N_s - 2)**:**
9: $will\_exec$ **= False; break**
10: **if** $will\_exec$**:** $exec = exec \cup \{J_{l,u}\}$
11: **else**: $suspend = suspend \cup \{J_{l,u}\}$
12: **if len**($exec$) **=** N_s**: break**

Fig. 6. The RFS scheduler

is important to note that each iteration in the algorithm, either r, or c, or both, are incremented by one. The **compute-interf** method terminates when r equals the number of rows or c equals the number of columns. To ensure that computations are not repeated, the **compute-interf** method stores the previously computed worst-case interference in the state variable.

The problem of computing the worst-case interference exhibits optimal substructure since the result at position (r,c) is computed as the maximum interference computed at positions $(r+1,c)$, $(r+1,c+1)$, and $(r,c+1)$. The **compute-interf** method is a dynamic program that takes advantage of the structure of the interference between pairs of steps; the time complexity of the algorithm is O($L_l \times L_h$).

The **compute-interf** algorithm completes the real-time analysis of GS. After computing the worst-case interference between any pair of flows ($I_{l,h}$) the worst-case response time of flow l may be computed according to Eq. 1. The presented analysis advances the state-of-the art by bounding the response time of real-time flows in the general case when flows may be established between arbitrary end-points and interference is captured in a generic and realistic manner.

3) RFS Scheduler: During the real-time analysis of GS, we observed that its greedy choice may result in longer-than-necessary worst-case interference. Consider the execution of two instance J_l and J_h according to the trace that starts in the upper left corner of Fig. 4. As we trace the execution of the two instances by GS, we will be computing their interference $I_{l,h}$. GS starts by executing J_h for two steps while J_l is suspended. This contributes two slots towards $I_{l,h}$. For the subsequent three steps, GS preempts the execution of J_h (when it conflicts with an instance J_k that has higher priority than J_h) and greedily executes J_l according to our trace. When J_h is resumed, J_l is suspended until J_h completes its execution, due to the conflicts present in the fourth row of the matrix. This contributes 5 additional slots for a total of $I_{l,h} = 7$ slots.

Interestingly, if J_l had not been greedily executed while J_h was suspended, it would have suffered an interference of only 3 slots. This situation may be avoided in general by modifying how the GS algorithm behaves in case 3. Accordingly, we prohibit the execution of a lower priority instance whose next step is $I_l.step$ if there is a higher priority instance h such that the transmissions in the next step in each of their plans conflict (i.e., when $\Pi_l[I_l.step] \nparallel \Pi_h[I_h.step]$). With this change, **compute-interf** is updated to compute RFS's worst-

case interference by removing the greedy choice (line 5 and 9).

Figure 6 shows the pseudocode of the RFS scheduler. The scheduler uses the generic interference model through the **interferes** interface. Similar to GS, RFS maintains a $released$ queue containing all instances that are released. In each slot, RFS considers all released instances in order of their priority and determines the instances that will be executed (the $exec$ set) and the instances that will be suspended (the $suspend$ set). An instance $J_{l,u}$ whose execute step is $J_{l,u}.step$ will be added to the $exec$ set if three conditions are satisfied: (1) The transmission $\Pi_l[J_{l,u}.step]$ will not conflict with any of the transmissions of the instances previously added to the $exec$ set (line 4). (2) RFS avoids the greedy choice that leads to increased worst-case interference i.e., no instance $J_{h,v}$ in the $suspend$ set interferes with $J_{l,u}$ (line 8). (3) The number of transmissions per slot does not exceed N_s (line 12).

The time complexity of the operations performed per slot by RFS is $O(|released| \times |suspend|)$. In practice, we expect the time complexity to significantly lower since the size of the $exec$ set is constrained by N_s. The low computation overhead of RFS enables us to determine the transmissions in each slot dynamically even on resource constrained sensor nodes.

V. State Management

RFS requires nodes to maintain the following consistent state: the flow parameters, the flow plans, and the interference relations. Inconsistencies in this state would result in nodes making executing conflicting transmissions in a slot. The state is modeled as versioned vectors that may be created or updated dynamically. To avoid the possibility of creating consisting state, we partition the state such that a single node may modify an object while the other nodes may read it. Accordingly, the flow's source may creates and updates the flow's parameters. Flow plans are stored as multiple objects such that each sender involved in a plan may update its maximum number of transmissions and the next hop. This provide us with the flexibility of changing each step of the plan independently.

The state management of RFS needs to ensure that all nodes have a consistent state. To this end RFS reserves a fraction of the flows for state management. RFS leverages on existing gossip protocols to maintain a consistent state efficiently. Periodically, each node broadcasts beacons including the latest version of the objects it stores. A node receiving a beacon may detect if the sender has an object with an older version than it has. In that case, the node will transmit the newest version of the object. However, since beacons are broadcast, multiple nodes may try to reply. To avoid collisions at the receiver, a node will select a random delay before transmitting an update and cancels its transmission if it overhears an update being broadcast by its neighbor.

Network Dynamics: RFS handle network dynamics including variations in link quality and interference through two mechanisms. First, as previously discussed in Section IV-A, RFS includes packet retransmissions during the construction of plans. This mechanism allows RFS to recover fast to expected variations in link quality and interference. However, significant changes in network topology will infrequently require plans to be reconstructed and interference relations to be reevaluated. These events will result in objects updates which are disseminated through the state management protocol.

Variable Workload: RFS may adapt to workload changes involving the addition, removal, or changes in the temporal properties of flows with ease. In contrast to traditional TDMA protocols, these updates do not require the reconstruction of a fixed transmission schedule. All that is required is for the updated flow parameters to be disseminated to all nodes through the state maintenance protocol.

VI. Distributed Scheduler

The centralized scheduler may be used with reasonable memory and communication overheads for networks with diameters of 2 – 3 hops. To allow RFS to scale to larger networks, we distribute RFS as follows. The key abstraction of the distributed version of RFS is that of a *neighborhood*. By controlling the neighborhood size RFS effectively controls the memory and maintenance overhead: consistency must be preserved only within a neighborhood. However, the distributed RFS introduces three new challenges: (1) how to group nodes in a neighborhood, (2) how to ensure flows can be executed independently in each neighborhood without collisions between transmissions in adjacent neighborhoods, and (3) how to handle flows that cross multiple neighborhoods.

The grouping of nodes into neighborhoods occurs during a bootstrapping phase. The developer is responsible for identifying a number of nodes that will serve as network controllers. A controller initiates the formation of its neighborhood by constructing a routing tree having it as root. Each node is associated to the controller to which it has the shortest path.

In order to decouple the execution of flows among neighborhoods, we employ a two-level scheduling approach. To capture the potential conflicts of transmissions between neighborhoods, we construct an undirected neighborhood graph. The vertices of the graph are the neighborhoods. Edges are added to this graph such that if transmission $\overrightarrow{AB}$ from a neighborhood N_1 conflicts with a transmission $\overrightarrow{CD}$ from neighborhood N_2, an edge N_1N_2 is added. Traditional TDMA scheduling techniques may be used to assign each neighborhood to transmit in a non-conflicting time slot. In each time slot, neighborhoods colored with the same color are executed. Within a slot, the actual transmission is determined using RFS.

Of course, a flow can span multiple neighborhoods. For example a flow, $F(AB, BC)$ that spans two neighborhoods may be split into $F(AB)$ and $F(BC)$, where $F(AB)$ and $F(BC)$ may involve multiple hops. To minimize the memory and maintenance overheads, the distributed version of RFS executes flows $F(AB)$ and $F(BC)$ independently. However, the lack of synchronization between flows can lead to poor performance. For example, it is possible for flow $F(BC)$ to be released before flow $F(AB)$ delivers the packet to B. In this case, the packet will be queued at B (or even worse dropped) until the next release of $F(BC)$.

This problem is an instance of the classical phase synchronization problem. To address this problem, a dynamic traffic shaper based on the Release Guard (RG) protocol [8] can synchronize the execution of flows across multiple neighborhoods. Intuitively, the dynamic traffic shaper modifies the phasing of flow $F(BC)$ to reflect the worst-case response time of $F(AB)$ across its neighborhood. The dynamic traffic shaper operates according to the following two rules. When the instance of $F(BC)$ is executed, if the packet from $F(AB)$ is already in B's queue, then the next instance of $F(BC)$ will be executed at $r_{i,u+1}(BC) = r_{i,u}(BC) + P_F$. In contrast, if the packet from $F(AB)$ arrives after $F(BC)$ is released, the next instance of $F(BC)$ will be executed at $r_{i,u+1}(BC) = now + P_F$, where now denotes the current slot. This case is called a phase shift.

A traditional TDMA protocol would have to reconstruct its explicit schedule in order to account for a phase shift. In contrast, RFS has the advantage of its dynamic executing flows based on their temporal properties. Accordingly, in order for RFS to accommodate a phase shift, it suffices to update the nodes within the neighborhood experiencing the shift. A phase shift may be implemented efficiently in RFS: the updated phase must be disseminated to all nodes within the neighborhood using RFS's state maintenance protocol. Note that phase updates do not require us to rerun the schedulability analysis since they do not affect flow periods or deadlines.

Adapting the analysis of the centralized algorithm for the distributed version is straightforward. First, RFS adapts the RG protocol for its use. The modifications of the RG do not affect its most important analytical property: the end-to-end response time of a flow is the sum of the response times of each subflow. Accordingly, the response time of an end-to-end flow R_i is $R_i = \sum_{AB \in subflows(i)} R_i(AB)$. The response time of each subflow $R_i(AB)$ is computed based on the real-time analysis presented in Section IV-B2. To account for the two-level scheduling, the obtained results are multiplied by the number of colors necessary to color the neighborhood graph.

VII. Simulations

To create a realistic simulation environment, we developed a discrete-event simulator that operates based on traces collected from an indoor WSN testbed. The testbed is deployed in Bryan Hall at Washington University in St Louis. The testbed consists of 43 TelosB motes each equipped with a Chipcon CC2420 radio compliant with the IEEE 802.15.4 standard. The traces were obtained by having each node in the testbed take turns broadcasting a sequence of 50 packets with a transmission power of 0 dBm. All nodes operated on a single channel (channel 26 of IEEE 802.15.4). While the application transmits packets as soon as possible, the MAC layer applied for each transmission a randomized back-off uniformly distributed in the interval [10 ms, 170 ms]. The batch of 50 packets takes 4.5 s on average to transmit. The remainder of the nodes recorded the Received Signal Strength (RSS) of the packets they receive. The short delay between the transmissions of packet pertaining to the same batch allows us to capture the short-term variability of RSS. Collecting four traces over three consecutive days captured the long-term variability.

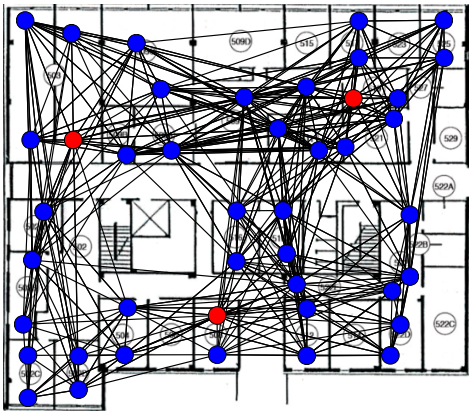
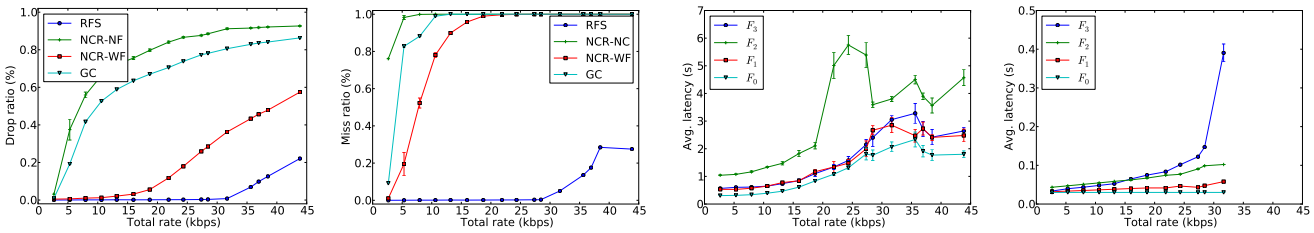


Fig. 7. Communication links for the considered topology. Sinks for scenario 2 are colored in red.

Received Signal Strength traces collected from the 43-node testbed are used to configure the simulations. All simulation results are obtained from the same topology. Figure 8 shows the locations of the testbed nodes at Bryan Hall (a 34m $\times$ 30m area). The network topology used in the simulations is based on RSS traces collected from the testbed. We determine the communication and interference links between nodes as follows. A node A may communicate with a node B if node B's RSS average during A's transmissions exceeds a threshold of -85 dBm. Prior empirical studies have shown that links with RSS above this threshold typically have high packet reception rates [26]. Interference links are determined similarly to the Radio Interference Detection (RID) protocol [7]. RID models interference as a graph that is constructed as follows. To determine whether the transmissions of other nodes can interference with a communication link $\overrightarrow{AB}$, RID calculates the Signal to Noise Plus Interference Ratio (SNIR) at node B for each set of n senders ($n = 3$ in our setup) assuming they transmit simultaneously as A transmits to B. For each set of senders $I(B)$, RID computes the SNIR at B when A and the set of senders $I(B)$ transmit simultaneously. The RSS of a link is computed as the average of the four 50 packet batches collected from the testbed. The RSS of missing packets is overestimated to equal the receiver sensibility of CC2420 (-90 dBm). If the computed SNIR is below a threshold a link from each node in $I(B)$ to B is added as an interference link. The SNIR threshold was set to 5 dB consistent with empirical studies that showed meeting this a threshold is usually sufficient for correctly decoding packets in the presence of interference [5], [22]. All RSS traces were collected when the testbed uses a single channel (channel 26 of IEEE 802.15.4).

We configured the simulator to settings similar to WirelessHART: a slot size of 10 ms and packets of 133 bytes. A single packet is transmitted in each slot. Each data point is the average of five runs; the 90% confidence intervals are also plotted. The deadline of all flows was set equal to their periods; the priority of a flow is assigned using Rate Monotonic Scheduling.



(a) Drop ratio (b) Miss ratio (c) Avg. latency of NCR-WF (d) Avg. latency of RFS

Fig. 8. Performance comparison under Scenario 1

For performance comparison, we implemented two baselines: NCR [27] and a TDMA protocol based on graph coloring. We ran two versions of NCR: NCR-NF, which splits bandwidth among interfering nodes evenly, and NCR-WF, which splits bandwidth proportionally to the workload of each node[1]. The graph-coloring TDMA (GC) uses a greedy coloring strategy in which nodes are sorted according to their degree in the interference graph and then assigned non-conflicting colors within their two-hop neighborhood. We chose this greedy coloring heuristic because it is known to have a bounded worst-case performance. These protocols do not provide prioritization and do not take advantage of the properties of flows. Their performance is characteristic of existing TDMA protocols.

We compare the performance of the protocols based on four metrics: *drop ratio*, *miss ratio*, *average flow latency*, and *maximum flow latency*. The drop ratio is the number of packets received at the destination out of the transmitted packets. The miss ratio is the number of packets which were either dropped or missed the deadline out of the transmitted packet. We define the network capacity to be the maximum throughput that a protocol supports without dropping packets. Similarly, the real-time capacity of a protocol is the maximum throughput that a protocol supports without missing deadlines.

The simulation results focus on two scenarios, motivated by our interest in supporting real-time communication for industrial monitoring and control: a scenario where interfering flows established between sensors and actuators and a data collection scenario where data is collected from multiple sensors to base stations. The simulation results highlight three aspects of RFS. First, RFS significantly outperforms the baselines along the considered metrics. Second, the data indicates that the schedulability analysis presented in Section IV-B2 is correct and the bounds are tight. Third, we evaluate the impact of dividing the nodes into neighborhoods to facilitate scalability.

A. Peer-to-peer Flows Scenario

The first scenario is motivated by the need to support real-time communication between sensors and actuators over multiple hops. We establish four flows (F_0, F_1, F_2, F_3) connecting the nodes located on the corners of the topology that intersect in the middle of the topology. The rates of the flows F_0:F_1:F_2:F_3 have the ratio 1:1.5:2.2:4.3. The workload is varied by increasing the rates of the flows while maintaining the same ratios between the flow rates. This resulted in periods (and deadlines) for F_0 in the range of [50 ms, 840 ms]. The rate of a flow is the product of the flows rate (in Hz) and the packet size (in bits)[2]. The total rate is the sum of the rates of all the flows.

Figure 8(a) plots the drop ratio as the rate of the flows is increased. All curves follow a similar pattern: they start at zero, remain at zero until the network capacity of the protocol is exceeded, and then they increase sharply. GC and NCR-NF had the lowest network capacity (i.e., started dropping packets first). This is a result of the fact that both protocols allocate bandwidth equally to interfering nodes, even when a node does not have any packets to transmit. During the graph coloring, GC required 32 colors allowing for a maximum bandwidth of:

$$\frac{1}{32} \cdot 100 \texttt{ slot/s} \cdot 1 \texttt{ pkt/slot} \cdot 1064 \texttt{ bits/pkt} = 3.325 \texttt{ kbps}$$

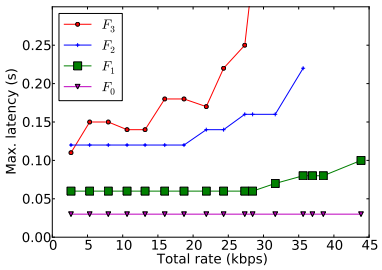
Since NCR-NF approximates the behavior of GC, it achieves comparable performance. In contrast, NCR-WF allocates slots based on the bandwidth requirements of nodes, allocating no slots to nodes that are not generating or forwarding data. As a result, NCR-WF supported a total flow rate of 13.46 kbps. These results highlight the importance of allocating slots proportionally to the bandwidth requirements of a node.

RFS achieved significantly higher performance than the baselines. This show the importance of accounting for flow properties during scheduling. Whereas the baselines started dropping packets when the total rate was 13.46 kbps, RFS did not drop packets until the total rate became 31.78 kbps, a 2.36 times increase in network capacity.

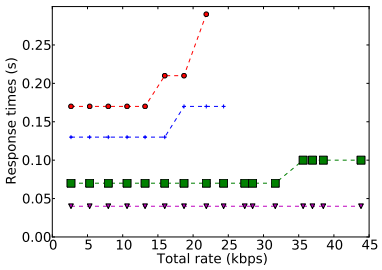
The difference between RFS and the baselines is even more pronounced when we consider the miss-ratio metric (see Figure 8(b)). The baselines miss packets even when the total rate is relatively low. Two factors contributed to this result. First, the baselines have long flow latencies, as indicated by Figure 8(c). This is because, on average, a packet waits for half the frame before it is forwarded to the next hop, since the TDMA schedule was constructed without accounting precedence constraints introduced by hop-by-hop forwarding. In contrast, RFS uses this information, effectively aligning the transmission of packets across multiple hops, leading to latencies below 0.4 s when the total load is below RFS's 31.78 kbps capacity (see Figure 8(d)). Event at the lowest

[1]In NCR-WF, nodes that do not generate or forward data will not be allocated any slots.

[2]Note that in the simulator, a single packet is transmitted in each slot.

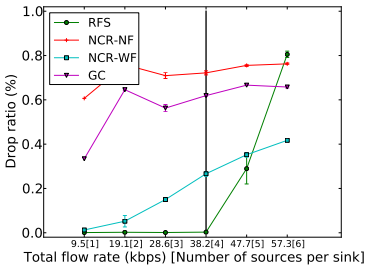


(a) Max. latency of RFS

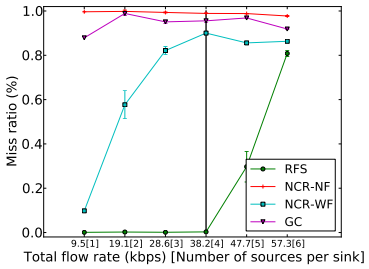


(b) Analytical response times

Fig. 9. Validation of schedulability analysis for Scenario 1



(a) Drop ratio



(b) Miss ratio

Fig. 10. Performance comparison under Scenario 2

tested rate, NCR-WF has latencies as long as 1 seconds, a factor of 2.5 difference. Second, the baselines do not provide any prioritization between flows, whereas RFS provides differentiated flow latencies.

We performed the schedulability analysis for all the simulated settings, which revealed that the analysis correctly identified unschedulable workloads. To evaluate the tightness of the schedulability analysis we plotted the maximum latency per flow and their response times in Figure 9(a) and Figure 9(b). A few things are worth highlighting. The theoretical real-time capacity is 21.9 kbps, 23.29% less than the empirically observed real-time capacity of 28.55 kbps (when packets start missing their respective deadlines). Note that even if we do not operate the system beyond 21.9 kbps, RFS still provides 1.62 times higher network capacity than NCR-WF, the best performing baseline.

B. Data Collection Scenario

The second scenario is motivated by our interest in supporting the real-time collection of sensor data. Three nodes were selected as base stations (see Figure 7) to which sensor data was forwarded. We varied the workload by increasing the number of sources that forwarded data. The sources were selected as follows. When the number of sources is 1-2, the sources were selected randomly from nodes that are 1 hop away from the base station. When the number of sources was increased to be between 3-4 and 5-6, the sources were selected randomly out of the nodes that are 2 and 3 hops from each base station, respectively. To model sensors with different rates, three flows per source node were established, with periods (and deadlines) of 640 ms, 1020 ms, and 2230 ms, respectively.

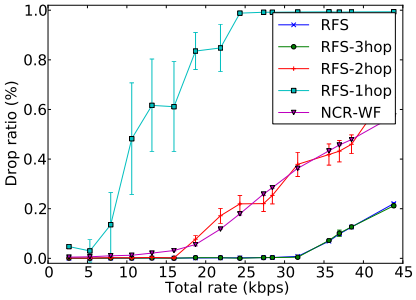
Figure 10 plots the performance of the protocols in terms of drop and miss ratios. As the workload is increased, the protocols dropped or missed an increasing number of packets. Similar to the previous setup, NCR-NF and GC achieved the worst performance on both metrics. While NCR-WF performed the best out of the baselines, it still dropped and missed a significant number of packets, even in the case when a single source generated 9.5 kbps. RFS significantly outperformed the baselines. RFS was able to support four sources generating 38.2 kbps without dropping packets or missing deadlines, an improvement of 3.97 times.

The point at which the workload of the second scenario becomes unschedulable is plotted as vertical lines in Figure 10. The schedulability analysis correctly identified the workloads were unschedulable, since no packets missed their deadline when the number of sources was 1 – 4. Our schedulability analysis determined that the system with an aggregated workload of 38.2 kbps was schedulable.

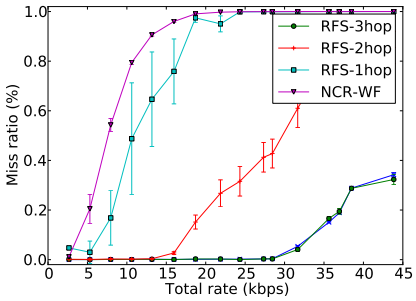
C. Evaluation of Distributed RFS

The last experiment investigates the impact of the neighborhood diameter on the performance of the distributed RFS. We decided to evaluate the performance of the distributed RFS under the first scenario, since it contains longer flows. Accordingly, we divided RFS in neighborhoods of diameter one, two, and three hops. The diameter of the network is four hops and the results from the centralized algorithm are obtained when all nodes belong to a single neighborhood.

Figure 11 plots the impact of increasing the neighborhood size on the drop ratio and miss ratio of RFS. RFS using three hop information achieved performance similar to the centralized scheduler. This is due to the fact that the network diameter is four hops, and in this case almost all nodes are included in a single neighborhood. The RFS-2hop achieved



(a) Drop ratio



(b) Miss ratio

Fig. 11. Impact of the neighborhood diameter on the distributed protocol

comparable performance with NCR-WF in terms of drop ratio. However, the benefit of RFS is that it achieves a higher real-time capacity by providing prioritization and employing Release Guard to synchronize flows across neighborhoods. In contrast, NCR-WF performance in terms of miss ratio was even worse than RFS-1hop, highlighting the importance of the mechanisms that we developed.

VIII. Conclusions

RFS is a novel transmission scheduling approach for executing prioritized real-time flows in WSNs. RFS is designed to address several key limitations of existing solutions: simplified workload models, unrealistic interference models, variable link quality, and limited scalability. Most prominently, RFS features a novel response time analysis that accounts for arbitrary inter-flow interferenceand is derived under a general workload model in which flows may be established between arbitrary sources and destinations. RFS scales by limiting the scope of the state maintained by a node to its neighborhood. The real-time performance of flows crossing multiple neighborhoods is ensured through the novel application of Release Guard. Simulation results based on traces collected from an actual wireless sensor network testbed show that RFS reduces flow latency by 2.5 times and provides improvements real-time capacity as large as 3.9 times compared to classical TDMA protocols. The results also suggest that the real-time capacity determined using the schedulability analysis is only 30% lower than the empirical real-time capacity.

Acknowledgment

This work is support in part by NIH/NLM grant number R01-LM009522. Additional funding was provided NSF under grants CNS-0448554 (CAREER), CNS-1035773 (CPS), and CNS-0708460 (CRI).

References

[1] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end delay analysis for fixed priority scheduling in wirelesshart networks," in *RTAS*, 2010.
[2] P. Jurcik, A. Koubâa, R. Severino, M. Alves, and E. Tovar, "Dimensioning and worst-case analysis of cluster-tree sensor networks," *ACM Transaction on Sensor Networks*, vol. 7, no. 2, pp. 1–47, 2010.
[3] P. Suriyachai, J. Brown, and U. Roedig, "Time-critical data delivery in wireless sensor networks," in *Distributed Computing in Sensor Systems*.
[4] O. Chipara, C. Lu, and G.-C. Roman, "Real-time query scheduling for wireless sensor networks," in *RTSS*, 2007.
[5] S. Liu, G. Xing, H. Zhang, J. Wang, J. Huang, M. Sha, and L. Huang, "Passive interference measurement in wireless sensor networks," in *ICNP*, 2010.
[6] R. Maheshwari, S. Jain, and S. R. Das, "A measurement study of interference modeling and scheduling in low-power wireless networks," in *SenSys*, 2008.
[7] G. Zhou, T. He, J. A. Stankovic, and T. F. Abdelzaher, "RID: radio interference detection in wireless sensor networks," in *INFOCOM*, 2005.
[8] J. Sun and J. Liu, "Synchronization protocols in distributed real-time systems," in *ICDCS*, 1996.
[9] Y. Ge, J. C. Hou, and S. Choi, "An analytic study of tuning systems parameters in IEEE 802.11e enhanced distributed channel access," *Computer Networks*, vol. 51, no. 8, pp. 1955–1980, 2007.
[10] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Distributed multi-hop scheduling and medium access with delay and throughput constraints," in *MobiCom*, 2001.
[11] G.-S. Ahn, A. T. Campbell, A. Veres, and L.-H. Sun, "Swan: service differentiation in stateless wireless ad hoc networks," in *INFOCOM '02*.
[12] 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*, 2007.
[13] Y. Gu, T. He, M. Lin, and J. Xu., "Spatiotemporal delay control for low duty-cycle sensor networks." in *RTSS*, 2009.
[14] A. Koubaa, M. Alves, and E. Tovar, "Modeling and worst-case dimensioning of cluster-tree wireless sensor networks," in *RTSS*, 2006.
[15] A. Rowe, R. Mangharam, and R. Rajkumar, "RT-Link: a global time-synchronized link protocol for sensor networks," *Ad Hoc Networks*, vol. 6, no. 8, pp. 1201–1220, 2008.
[16] S. M. S. Nirjon, J. A. Stankovic, and K. Whitehouse, "IAA: interference aware anticipatory algorithm for scheduling and routing periodic real-time streams in wireless sensor networks," in *INSS*, 2010.
[17] A. Koubaa, M. Alves, and E. Tovar, "i-GAME: an implicit GTS allocation mechanism in IEEE 802.15.4 for time-sensitive wireless sensor networks," in *ECRTS*, 2006.
[18] M. Caccamo, L. Y. Zhang, L. Sha, and G. Buttazzo, "An implicit prioritized access protocol for wireless sensor networks," in *RTSS*, 2002.
[19] H. Zhang, P. Soldati, and M. Johansson, "Optimal link scheduling and channel assignment for convergecast in linear WirelessHART networks," in *WiOPT*, 2009.
[20] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-time scheduling for wirelesshart networks," in *RTSS*, 2010.
[21] T. F. Abdelzaher, S. Prabh, and R. Kiran, "On real-time capacity limits of multihop wireless sensor networks," in *RTSS*, 2004.
[22] G. X. Mo Sha, G. Zhou, S. Liu, , and X. Wang, "C-mac: Model-driven concurrent medium access control for wireless sensor networks," in *INFOCOM*, 2009.
[23] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *SenSys*, 2009, pp. 1–14.
[24] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*, 1988.
[25] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, 1993.
[26] K. Srinivasan and P. Levis, "Rssi is under appreciated," in *Proceedings of EmNets*, 2006.
[27] L. Bao and J. J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *MobiCom*, 2001.