Recorp: Receiver-Oriented Policies for Industrial Wireless Networks

Ryan Brummet, Octav Chipara, and Ted Herman
University of Iowa
{ryan-brummet, octav-chipara, ted-herman}@uiowa.edu

***Abstract*—The next generation of Industrial Internet-of-Things (IIoT) systems will require wireless solutions to connect sensors, actuators, and controllers as part of feedback-control loops over real-time flows. A key challenge in such networks is to provide predictable performance and adaptability to variations in link quality. We address this challenge by developing *REC*eiver *OR*iented *P*olicies (Recorp), which leverages the stability of IIoT workloads to build a solution that combines offline policy synthesis and run-time adaptation. Compared to schedules that service a single flow in a slot, Recorp policies share slots among multiple flows by assigning a coordinator and a set of candidate flows in the same slot. At run-time, the coordinator will dynamically execute one of the flows depending on what flows the coordinator has already received. The net effect of this strategy is that a node can dynamically repurpose the retransmissions remaining after receiving the data of an incoming flow to service other incoming flows opportunistically. Therefore, the flows that are executed in a slot can be adapted in response to the variable link conditions observed at run-time. Furthermore, Recorp also provides predictable performance: a policy meets the end-to-end reliability and deadline constraints of flows given probabilistic link qualities. When Recorp policies and schedules are configured to meet the same end-to-end reliability target of 99%, larger-scale multihop simulations show that across typical IIoT workloads, policies provided a median improvement of 1.63 to 2.44 times in real-time capacity as well as a median reduction of 1.45 to 2.43 times in worst-case latency.**

I. Introduction

Industrial Internet-of-Things (IIoT) systems are gaining rapid adoption in process control industries such as oil refineries, chemical plants, and factories. We are particularly interested in the next generation of smart factories that are expected to use sophisticated sensors (e.g., cameras, microphones) with higher data rates than current IIoT systems. Such systems need a versatile wireless alternative to wired networks to connect sensors, actuators, and controllers as part of feedback-control loops over multihop *real-time flows*. Since communication delays and packet losses may lead to control degradation or even control instability, wireless solutions must provide both reliable and real-time performance.

General wireless network protocols are not engineered for workloads typical of IIoT systems that involve recurrent combinations of real-time flows for significant time intervals. Knowing that a workload consisting of a set of flows will be stable for half an hour, justifies precomputing schedules to arbitrate media access. However, the challenge is to ensure schedules provide stable real-time and reliable communication for these prolonged intervals in harsh industrial environments such as those with sources of interference (e.g. [1], [2]). To avoid excessive engineering margins, nodes should be able to adapt their operation locally in response to the link conditions they experience at run-time. Finding the balance between what is precomputed and the degree of local adaptation is an open research question.

The state-of-the-art for wireless IIoT are standards such as WirelessHART [3] that use Time Slotted Channel Hopping (TSCH). TSCH combines Time Division Multiple Access (TDMA) and channel-hopping in a mesh network. TSCH networks employ a centralized network manager to collect topology information, compute routes and transmission schedules, and disseminate schedules to devices. Link dynamics are handled using a combination of retransmissions and channel hopping. Accordingly, schedules are built such that the links of a flow are assigned several retransmissions with each retransmission using a different channel. The number of retransmissions allocated is usually determined based on the worst-case quality of a link to tolerate significant variations in link quality without having to reconstruct the global schedule (which is expensive). The only run-time adaptation that nodes can perform is to cancel the retransmissions of a link if they have already received an acknowledgment. As a consequence, a significant number of slots remain unused when a packet is relayed successfully to the next hop before exhausting the link's allocated retransmissions. This limitation is inherent to scheduling approaches: scheduled retransmissions cannot be repurposed in response to the successes and failures observed locally at run-time without reconstructing the global schedule due to the need for global consensus.

A common technique to alleviate the limited flexibility of schedules is to *share* a slot and a channel (i.e., an entry henceforth). In shared entries, multiple transmissions are scheduled, and contention-based techniques are usually used at run-time to arbitrate channel access. Unfortunately, nearly all existing approaches can only improve network performance in a best-effort manner and, as a consequence, cannot support real-time traffic. In [4], we proposed a technique to support real-time communication when entries are shared only between the links of the same flow. Our experiments show that the amount of sharing enabled by this technique is limited, and, for some typical IIoT workloads, it even performs worse than schedules that do not use sharing. Therefore, the open research question is whether *multiple flows can share slots to improve network performance and agility without losing predictability* *for typical IIoT workloads.*

To answer this question, we propose ***REC**eiver **OR**iented Policies (Recorp)* – a new paradigm for building agile, reliable, and real-time wireless networks for IIoT systems. We exploit the typical characteristics of the industrial setting to obtain improvements in network capacity and latency while providing end-to-end reliability. Specifically, our approach has the following salient features: Since IIoT workloads tend to be stable, we compute offline Recorp policies and disseminate them to all nodes. Recorp policies assign multiple candidate flows that may be executed in the same slot and channel. At run-time, only one of the candidate flows will be executed depending on what flows a coordinator has already received. The benefit of this approach is that it allows flows to be dynamically executed in an entry depending on the successes and failures of transmissions observed at run-time. As a consequence, Recorp policies can support significantly higher real-time capacity than schedules.

II. System Model

We base our network model on WirelessHART as it is an open standard developed specifically for IIoT systems with stringent real-time and reliability requirements [3]. A network consists of a *gateway* and tens of *field devices*. A centralized *network manager* is responsible for synthesizing policies, evaluating their performance, and distributing them across the network. The field devices form a wireless mesh network that we model as a graph $G(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ and $\mathcal{E}$ represent the field devices and wireless links, respectively. WirelessHART may use either single-path source routing or multi-path graph routing. We will use source routing, assuming that there is a single shared routing tree. At the physical layer, WirelessHART adopts the 802.15.4 standard with up to 16 channels. In this paper, we focus on receiver-initiated communication, where a node requests data from a neighbor and receives a response within the same 10 ms slot.

We use *real-time flows* as a communication primitive. A real-time flow F_i is characterized by the following parameters: phase σ_i, period P_i, deadline D_i, end-to-end target reliability requirement T_i, path Γ_i, and static priority i where lower values have higher priority. The kth instance of flow F_i, $J_{i,k}$, is released at time $r_{i,k} = \phi_i + k * P_i$ and has an absolute deadline $d_{i,k} = r_{i,k} + D_i$. We assume $D_i \leq P_i$, which implies only one instance of a flow is released at a time. Consequently, to simplify the notation, we will use J_i to refer to the instance of flow F_i that is currently released. We will use $\mathcal{F}$ to indicate the set of flows in the network.

We use a simple probabilistic model where the quality of a link is a single parameter P_s that indicates the probability a pull is successful (i.e., including both the pull request and the response containing the data). Our results may be easily generalized to the case when each link has a different link quality probability.

A Recorp policy π is represented as a matrix whose rows indicate channels and columns indicate slots. We refer to a slot-channel pair as an entry. A policy is well-formed if it satisfies the following constraints: (1) Each node transmits or receives at most once in an entry to avoid intra-network interference. (2) The hop-by-hop packet forwarding precedence constraints are maintained such that senders receive packets before forwarding them. (3) Nodes do not perform consecutive transmissions using the same channel. (4) Each flow instance is delivered to its destination before its absolute deadline and meets its reliability constraint.

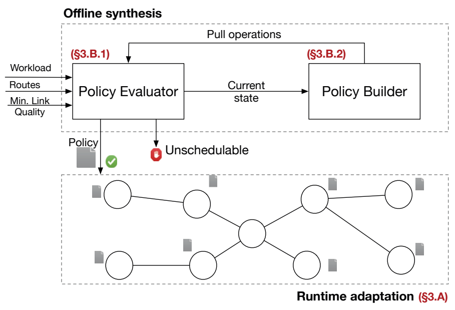


Fig. 1: Design of Recorp.

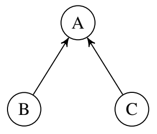
III. Design

Recorp is a practical and effective solution for IIoT applications that require predictable, real-time, and reliable communication in dynamic wireless environments (see Figure 1). Central to our approach are Recorp policies. The policy synthesis procedure runs on the network manager and has as inputs the workload, routing information, and the link quality of each link. If the synthesis procedure is successful, the constructed policy guarantees probabilistically that all flows will meet their real-time and reliability constraints as long as the likelihood of a successful transmission is P_s. The synthesis procedure fails when the workload is unschedulable, i.e., when a policy that meets the real-time and reliability constraints of all flows cannot be found. In this case, the workload must be reduced either manually by the developer or automatically using rate control mechanisms. If the synthesis procedure is successful, the manager disseminates the generated policy to all nodes.

The separation between offline synthesis and run-time adaptation is essential to building agile networks. The run-time adaptation is lightweight: when a node is a coordinator, it can execute one of several candidate flows without requiring global consensus. In contrast, policy synthesis is computationally expensive and ensures the global invariant no transmission conflicts occur due to the local decisions made by coordinators.

A. Recorp Policies and Their Run-time Adaptation

A Recorp policy is represented as a matrix whose rows indicate channels and columns indicate slots. The execution of a policy is cyclic, with nodes returning to the beginning of the policy upon reaching its end. A Recorp operation is assigned to an entry of the matrix, which specifies the slot when the operation will be executed and the channel that will be used. Each entry of the matrix may include at most



(a) Topology

	0	1	2	3
Spec	TX$_B(F_0)$	TX$_B(F_0)$	TX$_C(F_1)$	TX$_C(F_1)$
Trace 1	🟩 (BA)	🟩	🟩 (CA)	🟩
Trace 2	🟥 (BA)	🟥 x (BA) x	🟩 (CA)	🟩
Trace 3	🟩 (BA)	🟩	🟥 (CA)	🟥 x (CA) x

(b) Schedule

	0	1	2	3
Spec	PL$_A(F_0)$	PL$_A(F_0,F_1)$	PL$_A(F_0, F_1)$	PL$_A(F_1)$
Trace 1	🟩 (BA)	🟩 (CA)	🟩	🟩
Trace 2	🟥 (BA)	🟥 (BA)	🟩 (BA)	🟩 (CA)
Trace 3	🟩 (BA)	🟩 (CA)	🟥	🟥

(c) Recorp policy

Fig. 2: A schedule and policy for the topology shown in Figure 2a are constructed. At run-time schedules and policies behave differently depending on observed successes (green background) or failures (red background). The traces show how schedules and policies adapt run-time behavior in response to successes and failures. Notably, the schedule drops packets in traces 2 and 3 (indicated by white "x"-es) while the policy drops no packets.

one Recorp operation. An operation has two arguments: a *coordinator* and a *service list*. The coordinator is the node that executes the operation at run-time, and the service list includes the instances that *may* be executed in that slot and channel. At run-time, *only one* of the flows in the service list will be executed. Any node can become a coordinator, and the coordinators can change from slot to slot.

A coordinator executes a Recorp operation at run-time by considering the flows in the service list in priority order. For each such flow, the coordinator checks whether it has received the flow's packet. If the coordinator has already received the packet, it will proceed to consider the next flow in the service list. Otherwise, it will request the flow's packet from the coordinator's neighbor through which the flow is routed. Upon receiving the request, the neighbor may or may not have the packet (the latter case can happen when the packet was dropped at a previous hop). If the neighbor has the packet, it includes it in its response to the coordinator. Otherwise, the neighbor marks the packet as dropped in its response. The reception of either response will result in the coordinator marking the flow as successfully executed. If a response is not received, the flow remains unsuccessful. The invariant maintained by the execution of an operation is: *at most one instance from the service list is executed in a slot.*

To illustrate the differences between Recorp policies and schedules, consider the case of a star topology (see Figure 2a). In this example, two flows – F_0 and F_1 – are used to relay data from B and C to the sink A. Both flows release an instance in slot 0. WirelessHART requires the construction of a schedule with two transmissions for each flow (see Figure 2b). Three traces that differ in the pattern of packet losses observed at run-time are also included in the figure. The only run-time adaptation mechanism available in schedules is to cancel scheduled transmissions whose data has already been delivered. The notation TX$_B(F_0)$ indicates that B transmits F_0's packet to A. The synthesized Recorp policy is shown in Figure 2c and uses the notation PL$_A(F_0, F_1)$ to indicate a Recorp operation with A as the coordinator and $\{F_0, F_1\}$ as the service list.

To highlight several differences between policies and schedules, consider trace 2, where there are failures in slots 0 and 1. For this trace, the schedule included in Figure 2b cannot successfully deliver F_0's packet because it is allocated only a single retransmission. In contrast, the Recorp policy included in Figure 2c can successfully deliver F_0's packet. The policy includes F_0 in the service list of the operations in slots 0, 1, and 2. At run-time, F_0's transmission in slots 0 and 1 fail, but its packet will be delivered in slot 2. In slot 3, the policy successfully executes F_1. A similar scenario is included in trace 3 where F_1's packets cannot be delivered by schedules but are successfully delivered using a Recorp policy. Traces 2 and 3 highlight the flexibility of Recorp policies to improve reliability by dynamically reallocating retransmissions based on the success and failures observed at run-time.

B. Synthesizing Recorp Policies

At a high-level, the problem of synthesizing Recorp policies requires the construction of a Markov Chain (MC) to estimate the state of a network as Recorp operations are performed. A state encodes whether a node received the packets from all possible combinations of flows. A back of the envelope calculation indicates that without additional constraints, the state space is enormous – the worst-case number of states is $O(|\pi| \cdot 2^{|\mathcal{F}| \cdot |\mathcal{N}|})$, where $|\pi|$ is the length of the policy in slots.

The following three insights helped us develop a scalable approach to synthesizing Recorp policies. First, a key property of Recorp policies is that nodes operate independently. This property reduces the size of the state space significantly to $O(|\pi| \cdot |\mathcal{N}| \cdot 2^{|\mathcal{F}|})$. Second, our synthesis procedure is iterative, building the policy one slot at a time. As a result, it is sufficient to track only the states associated with each node in a given slot, thereby reducing the state size that is maintained by a factor of $|\pi|$ to $O(|\mathcal{N}| \cdot 2^{|\mathcal{F}|})$. Furthermore, we observe that given a fixed sequence of Recorp operations, only a small fraction of the possible states are reachable. Thus, states should be managed lazily to track only reachable states.

The policy synthesis procedure involves two key components – an evaluator and a builder (see Figure 1). The policy is built incrementally by sequentially determining the operations that will be executed in each slot as follows:

- The evaluator manages the state of each node by tracking the likelihood it received a packet for different combinations of flows. At the beginning of each slot, the evaluator releases new instances based on the phase and period of each flow. Upon releasing an instance, the first link of its flow becomes *active*.
- The builder is invoked with a list of released instances and their (single) active links as input. The builder identifies the Recorp operations that will be executed in the current slot, assigning no more than one operation per channel. Recorp operations are selected to maximize the number of flows executed while enforcing static priorities.
- The evaluator updates the likelihood of each network state to reflect the execution of the Recorp operations selected

by the builder. The end-to-end reliability requirement is mapped onto a local reliability requirement that must be met at each hop. The evaluator identifies the instances and their associated active links that meet their local reliability requirement. The states associated with each identified link are then removed and the next link on the path of the flow becomes the active link.

1) Recorp Evaluator: The evaluator keeps track of the system's states and their likelihood. Since nodes operate independently when executing Recorp policies, we manage the state of each node independently and identically. A state that is managed by a node R is a vector of size $|\mathcal{F}|$. The i^{th} element of the vector is the state of instance J_i. The state of an instance J_i may be success or failure, indicating whether R requested J_i's data and received a reply successfully. In the worst-case, a node R may need to store $2^{|\mathcal{F}|}$ states, one state for each possible combination of flows.

The builder supplies a set of Recorp operations that have been assigned to the current slot. The evaluator considers each of them and evaluates their impact on the state of the system. Let P be a mapping from each state to its probability. The builder construct a new mapping P' that includes the updated probabilities of the states after accounting for execution of the Recorp operations. Initially, the new probabilities of each state in P' are set to 0. According to its semantics, a Recorp operation that includes J_i in its service list will be executed in any $current$ state where the i^{th} entry of the state vector is a failure and all instances with higher priority than J_i in the service list have already succeeded. From such a state, there are two possible outgoing transitions depending on whether the Recorp operation is successful. First, if the execution of J_i fails, the system remains in the same state and the probability of the $current$ state in the next slot is lowered by $P_f = 1 - P_s$. Second, if the Recorp operation succeeds, the system transitions from the $current$ state to a $next$ state. The entries of the $current$ and the $next$ states are the same, except for the entry associated with the J_i element for which $next[J_i] =$ success. The updated likelihood of $next$, $P'[next]$, increases by the product of the likelihood of the current state $P[current]$ and P_s.

For each flow F_i, the user supplies the end-to-end target reliability T_i. To determine when to stop assigning retransmissions for an instance at a node, we map the end-to-end reliability onto a local minimum reliability L_i that must be achieved at each hop:

$$L_i = \left(T_i\right)^{\frac{1}{|\Gamma_i|}} \qquad (1)$$

where, $|\Gamma_i|$ is the length of F_i's path. After updating the state to reflect the execution of the Recorp operation, the local reliability at node R of an instance J_i is computed by summing the probability of all states in which the i^{th} entry is success. When the local reliability exceeds L_i, then node R has been assigned sufficient retransmissions for J_i to meet its local reliability and the next link of J_i becomes active.

Let us return to our running example of a star topology

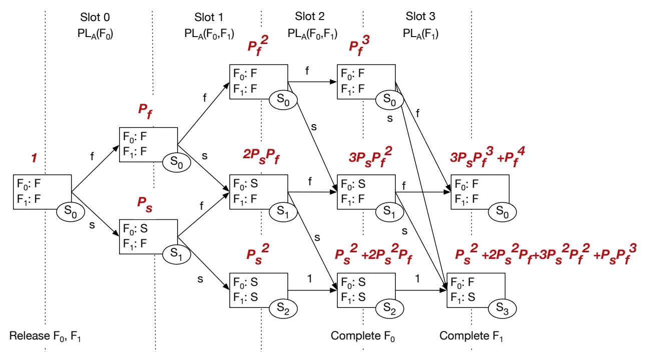


Fig. 3: State maintained by A while synthesizing the policy in Figure 2c (reproduced on top). Success and failures are denoted by s and f, respectively.

and consider how node A manages its state when the Recorp policy is executed. In Figure 3, we show the MC constructed to compute the end-to-end reliability of flows F_0 and F_1 by the evaluator. We remind the reader that the evaluator is incremental. Thus, it only maintains the likelihood of each state in a given slot and not the complete MC (which would limit the scalability). At the beginning of slot 0, F_0 and F_1 are released and a state $S_0 = [F_0 = \text{failure}, F_1 = \text{failure}]$ is created and has a probability of 1. Next, we account for the execution of PL$_A(F_0)$ in slot 0. If PL$_A(F_0)$ is successful, then the system transitions to the next state $S_1 = [F_0 = \text{success}, F_1 = \text{failure}]$. The probability of reaching S_1 at the end of slot 0 is $P[S_1] = P_s$. Otherwise, the system remains in S_0 and its likelihood decreases to $P[S_0] = P_f$. In the next slot, PL$_A(F_0, F_1)$ is executed. Consistent with the semantics of a Recorp operation, F_0 will be executed in state S_0 since $S_0[F_0] = \text{failure}$. If the operation is successful, the system transitions to state S_1; otherwise, the system stays in state S_0. In slot 1 and state S_1, F_0 is already successful and, as a consequence, F_1 will be executed. Depending on the outcome of F_1's transmission, the system may transition to either S_1 or a new state $S_2 = [F_0 = \text{success}, F_1 = \text{success}]$. We note that there are two possible ways to reach state S_1 at the end of slot 1. Therefore, the probability of reaching S_1 by the end of slot 1 is $P'[S_1] = P_sP[S_0] + P_fP[S_1] = 2P_sP_f$. The probability of reaching S_2 is $P'[S_2] = P_s^2$. The execution of the remaining Recorp operations produces the remainder of the MC shown in the figure.

2) Recorp Builder: The builder determines the Recorp operations that will be assigned in a slot. In each slot, the evaluator supplies the builder with a list of released flows and their associated active links. The builder then determines a set of Recorp operations that maximizes the number of flows that are executed in the slot while enforcing the priorities of flows. It is important to note that the builder does not use the state probabilities maintained by the evaluator since this information is not necessary given the optimization objective.

The optimization problem can be formulated as an Integer Linear Program (ILP). The ILP includes three types of variables. For each node R ($R \in \mathcal{N}$), the variable N_R ($N_R \in \{0, 1\}$) indicates whether R is the coordinator of a

Recorp operation. For each released instance J_i, the variable I_i ($I_i \in \{0,1\}$) indicates whether its associated active link will be added to a service list. Finally, variable $C_{R,ch}$ ($C_{R,ch} \in \{0,1\}$) indicates whether R will use channel ch to communicate. The ILP solution is converted into a set of Recorp operations as follows: for each node R such that $N_R = 1$, we add a Recorp operation that has R as the coordinator and a service list with all instances J_i where $I_i = 1$ and R is the receiver of the active link of J_i. The Recorp operation is assigned to the entry in the matrix for the current slot and the channel ch for which $C_{R,ch} = 1$.

A well-formed policy must ensure that no transmission conflicts will be introduced at run-time. Consider a Recorp operation that has R as a coordinator and services instance J_i. Let (SR) be the active link of J_i, where $S = src(J_i)$ and $R = dst(J_i)$. If J_i will be assigned in the current slot (i.e., $I_i = 1$), then S cannot be a coordinator for any other instance since this would require S to transmit and receive in the same slot. We enforce this using the following constraint:

$$N_S \leq (1 - I_i) \quad \forall I_i \in \mathcal{A}: \quad S = src(J_i) \tag{2}$$

A similar constraint must also be included for the receiver R. If node R is not a coordinator (i.e., $N_R = 0$), then J_i cannot be assigned and $I_i = 0$. Conversely, if R is selected as a coordinator, instance J_i may (or may not) be assigned (i.e., $I_i \leq N_R = 1$) depending on the objective of the optimization, which we will discuss later in this section. These aspects are captured by the following constraint:

$$I_i \leq N_R \quad \forall I_i \in \mathcal{A}: \quad R = dst(J_i) \tag{3}$$

The above constraints avoid all transmission conflicts with one exception. Consider the case when two instances J_i and J_j share the same sender but have different receivers. An assignment that respects constraints 2 and 3 is for both instances to be assigned in the current slot (i.e., $I_i = I_j = 1$). However, this would result in a conflict, since the common sender can only transmit one packet in a slot. To avoid this situation, we introduce the following constraint:

$$I_i + I_j - 1 \leq N_S \tag{4}$$
$$\forall I_i, I_j \in \mathcal{A}: \quad S = src(J_i) = src(J_j) \ \ \& \ \ dst(J_i) \neq dst(J_j)$$

The next set of constraints ensures that each Recorp operation is assigned a unique channel. We accomplish this by introducing $C_{R,ch}$ to indicate whether coordinator R uses channel ch ($ch = 1 \dots K$), where K is the number of channels. The selection of channels is subject to the constraints:

$$\sum_{R \in \mathcal{V}} C_{R,ch} \leq 1 \quad \forall ch \in 1 \dots K \tag{5}$$

$$\sum_{ch=1}^{K} C_{R,ch} = N_R \tag{6}$$

We require that coordinators switch channels between Recorp operations to ensure independence between transmissions. We enforce this property by introducing additional constraints to prevent coordinators from using the same channel. To enforce the prioritization of instances, we set the optimization objective to be:

$$\sum_{i=0}^{i<|\mathcal{A}|} 2^{|\mathcal{A}|-i} * I_i \tag{7}$$

The objective function ensures that a flow F_i will be assigned over lower priority flows unless there is a higher priority flow with a conflict with F_i.

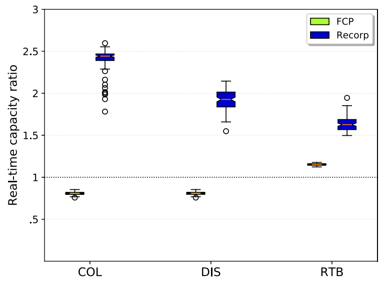
IV. Experiments

Our experiments demonstrate the efficacy of Recorp to support higher performance and agility than traditional scheduling approaches. We focus on the next generation of smart factories that are expected to use sophisticated sensors requiring higher data rates than current IIoT systems. Specifically, we are interested in answering the question of whether Recorp improves the real-time capacity of IIoT workloads.

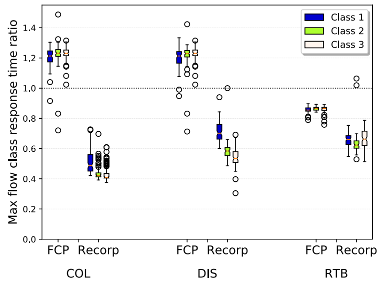
We compare Recorp policies against two baselines. First, we compare against scheduling (Sched) approaches that allocate a fixed number of transmissions per link. To provide a fair comparison between schedules and policies, we construct schedules using the same ILP formulation as Recorp policies with the additional constraint that the size of the service list is one. In this way, we ensure that Recorp and Sched differ only in one aspect: Recorp can share an entry among multiple flows and dynamically adapt which flow is executed at run-time. In contrast, under Sched, only one transmission is assigned in an entry. Second, we compare against the Flow Centric Policy (FCP) [4], which allows entry sharing only among the links of *a single flow*, whereas Recorp can share entries *across multiple flows*. Both Sched and FCP utilize sender-initiated transmissions while Recorp utilizes receiver-initiated pulls.

All simulations are performed on a 41-node, 6-hop diameter topology obtained from a testbed deployed at Washington University in St. Louis. We use $P_s = 70\%$ and used all sixteen 802.15.4 channels. The number of retransmissions used by Recorp, Sched, and FCP is configured to achieve a 99% end-to-end reliability for all flows. The period and deadline are equal, and the phases are 0 in all workloads. Flow priorities are assigned such that flows with shorter deadlines have higher priority. To break ties, flows with longer routes are assigned higher priority. Remaining ties are broken arbitrarily.

To provide compare Recorp, Sched, and FCP, we consider data collection workloads (COL), data dissemination workloads, and route through the base station workloads (RTB) where the base must be a node in each flow's route. The results presented in this section are obtained from 100 simulation runs per workload type. In all runs, the node closest to the center of the topology is selected as the base station. In each run, we generate 50 flows randomly generated according to the workload type. Each flow is assigned at random to one of three flow classes whose periods and deadlines maintain a 1:2:5 ratio. For example, if Class 1 has a period of 100 ms, then Class 2 has a period of 200 ms, and Class 3 has a period of 500 ms. We refer to the period of Class 1 as the base period. In a run, the base period of the flows is decreased



(a) Real-time capacity improvement ratio



(b) Max response time per flow class

Fig. 4: Simulations in a 41-node multi-hop network.

until the workload is unschedulable. The results of a run are obtained for the smallest base period for which the workload is schedulable.

We compute the real-time capacity ratio provided by Recorp and FCP over that of Sched. Ratios above one show improvements in capacity; conversely, ratios below one show reductions. Figure 4a plots the distribution of these ratios for each workload. FCP provides a median improvement of 1.15 times over Sched only for RTB. For the other workloads where the base station is the source/destination, FCP has worse performance since sharing within a flow reduces only the utilization of the intermediary nodes on a flow's path, but not on the source and destination nodes. In contrast, Recorp outperforms both Sched and FCP by providing a median improvement in the real-time capacity of 2.44, 1.93, and 1.63 times over Sched for the data collection, dissemination, and route through the base station scenarios, respectively.

Figure 4b shows the distribution of the max response time of each flow class over Sched. Ratios above one show increases in max response time; conversely, ratios below one show reductions. Consistent with the above experiments, FCP performs better than Sched only for the RTB scenario. Recorp significantly reduces the response time for all classes under all workloads. Recorp provides a median reduction in maximum response time in the range of 0.41 – 0.69 times, depending on the flow class and workload type. These results indicate *Recorp policies can significantly improve real-time capacity and max response times for common IIoT workloads.*

V. Related Work

Researchers have considered various approaches to combining CSMA and TDMA into hybrid protocols, ultimately sacrificing either flexibility or predictability. A common approach to combine CSMA and TDMA is to have each protocol run in different slots. This approach is adopted in industrial standards such as WirelessHART [3] and ISA100.11a [5]. However, predictable performance cannot be provided for the traffic carried in CSMA slots. Another alternative is to dynamically reuse slots (e.g., [6]) or transmit high-priority traffic (e.g., [7]) by selecting primary and secondary slot owners. In this approach, slot owners are given preference to transmit and send data using a short initial back-off. If a slot owner does not have any data to transmit, other nodes may contend for its use after some additional delay. A generalization of this scheme is prioritized MACs that divide a slot into sub-slots to provide different levels of priority [8]. However, none of these protocols provide analytical bounds on their performance. In contrast to the above approaches that involve carrier sensing, our policies rely on receiver-initiated polling and the local state of the nodes to adapt. We expect policies to be less brittle in practice than solutions that use carrier sense as they do not require tight time synchronization for adaptation.

VI. Conclusions

Recorp is a practical and effective solution for IIoT applications that require predictable, real-time, and reliable communication in dynamic wireless environments. We leverage the stability of IIoT workloads and the increased resources of wireless nodes to build a solution that combines offline policy construction and run-time adaptation. A Recorp policy assigns a Recorp operation to each slot and channel, which specifies a coordinator that will arbitrate channel access and a list of flows that may be serviced. At run-time, the coordinator dynamically executes the flows in the service list from which it has not received a packet. The advantage of Recorp is that nodes can locally reallocate the retransmissions of flows in response to transmission failures, as a result, provide higher performance than scheduling approaches.

The synthesis of policies required us to address two key challenges: handling the state explosion problem and providing predictable performance as the quality of links varies. We developed a practical approach to synthesize policies iteratively. In each slot, the builder employs an ILP program to determine the Recorp operations that will be performed in the current slot. Based on the selected operations, the evaluator updates the network state using each link's quality to determine which flows have met their reliability constraints. We guarantee that a constructed Recorp policy will meet a user-specified reliability and deadline constraint for each flow given that each link has a constant success likelihood. Our results indicate that due to their increased agility, Recorp policies can significantly improve real-time capacity (median 1.63 – 2.44) and reduce worst-case response time (median 1.45 – 2.43) while meeting a specified end-to-end reliability.

Acknowledgements

This work is funded in part by NSF under CNS-1750155.

References

[1] R. Candell, C. A. Remley, J. T. Quimby, D. R. Novotny, A. E. Curtin, P. B. Papazian, G. H. Koepke, J. E. Diener, and M. T. Hany, "Industrial wireless systems: Radio propagation measurements," *Technical Note (NIST TN)-1951*, 2017.

[2] K. Ferens, L. Woo, and W. Kinsner, "Performance of ZigBee networks in the presence of broadband electromagnetic noise," in *CCECE*, 2009.

[3] "Wirelesshart." [Online]. Available: https://fieldcommgroup.org/

[4] R. Brummet, D. Gunatilaka, D. Vyas, O. Chipara, and C. Lu, "A flexible retransmission policy for industrial wireless sensor actuator networks," in *ICII*, 2018.

[5] Isa100.11a. [Online]. Available: https://www.isa.org/isa100/

[6] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-mac: a hybrid mac for wireless sensor networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 16, 2008.

[7] B. Li, L. Nie, C. Wu, H. Gonzalez, and C. Lu, "Incorporating emergency alarms in reliable wireless process control," in *ICCPS*, 2015.

[8] W. Shen, T. Zhang, F. Barac, and M. Gidlund, "Prioritymac: A priority-enhanced mac protocol for critical traffic in industrial wireless sensor and actuator networks," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 824–835, 2013.