Report Number: WUCSE-2015-005

2015-09-29

Conflict-Aware Real-Time Routing for Industrial Wireless Sensor-Actuator Networks

Authors: Chengjie Wu, Dolvara Gunatilaka, Mo Sha, and Chenyang Lu

Process industries are adopting wireless sensor-actuator networks (WSANs) as the communication infrastructure. WirelessHART is an open industrial standard for WSANs that have seen world-wide deployments. Real-time scheduling and delay analysis have been studied for WSAN extensively. End-to-end delay in WSANs highly depends on routing, which is still open problem. This paper presents the first real-time routing design for WSAN. We first discuss end-to-end delays of WSANs, then present our real-time routing design. We have implemented and experimented our routing designs on a wireless testbed of 69 nodes. Both experimental results and simulations show that our routing design can improve the real-time performance significantly.

Conflict-Aware Real-Time Routing for Industrial Wireless Sensor-Actuator Networks

Chengjie Wu, Dolvara Gunatilaka, Mo Sha, Chenyang Lu,
Cyber-Physical Systems Laboratory, Washington University in St. Louis

***Abstract*—As process industries start to adopt wireless sensor-actuator networks (WSANs) for control applications, it is crucial to achieve real-time communication in this emerging class of networks. Routing has significant impacts on end-to-end communication delays in WSANs. However, despite considerable research on real-time transmission scheduling and delay analysis for such networks, real-time routing remains an open question for WSANs. This paper presents a *conflict-ware real-time routing* approach for WSANs. This approach leverage a key observation that conflicts among transmissions sharing a common field device contribute significantly to communication delays in industrial WSANs such as WirelessHART networks. By incorporating conflict delays in the routing decisions, conflict-aware real-time routing algorithms allow a WSAN to accommodate more real-time flows while meeting their deadlines. Evaluation based on simulations and experiments on a real WSANs testbed show conflict-aware real-time routing can lead to up to three-fold improvement in real-time capacity of WSANs.**

I. Introduction

With the emergence of industrial standards such as WirelessHART [1] and ISA100.11a [2], process industries are adopting *Wireless Sensor-Actuator Networks (WSANs)* that enable sensors and actuators to communicate through low-power wireless mesh networks [3]. In recent years, we have seen world-wide deployment of WSANs. Technical reports [4] from the process industry show more than 1900 WirelessHART networks have been deployed around the world, with more than 3 billion operating hours in the field.

Feedback control loops in industrial environments impose stringent end-to-end delay requirements on data communication. To support a feedback control loop, the network periodically delivers data from sensors to a controller and then delivers control commands to the actuators within an end-to-end deadline. The effects of deadline misses in data communication may range from production inefficiency, equipment destruction to irreparable financial and environmental damages.

Previous works [5]–[7] demonstrate that the end-to-end delays of flows highly depend on routes. It is important to optimize routes to improve the real-time capacity of WSANs. Existing routing algorithms usually select routes with the minimum hop count, which introduces high transmission conflicts among different flows. Since high transmission conflicts cause long end-to-end delays, shortest paths usually lead to a low real-time capacity. This paper presents our real-time routing algorithms for WSANs. We incorporate conflict delays into our routing design and propose conflict-aware routing algorithms that allow WSANs to accommodate more real-time flows. Our conflict-aware routing algorithms reduce conflict delays of real-time flows so they can meet their deadline constraints. Our evaluation shows that our real-time routing algorithms can greatly improve the real-time capacity of the network.

The rest of the paper is organized as follows. Section II reviews the related work. Section IV discusses the problem formulation. Section V provides a brief review of the existing delay analyses, and Section VI presents our real-time routing algorithms. Section VII evaluates our routing algorithms through simulations, then Section VIII concludes the paper.

II. Related Work

WSANs have attracted much attention in the research community [5]–[15] recently. Previous works studied real-time transmission scheduling [5], [9], [15], communication delay analysis [6], [7], [14] and rate selection [10], [12]. All these works assume the routes of the flows are given, and do not provide any routing protocol. There has been increasing interest in developing routing algorithms for WSANs. For example, Han et al. [8] propose routing algorithms to build reliable routes based on hop count, but their algorithms do not consider real-time performance.

Real-time routing has been studied in the wireless sensor network community. Xu et al. [16] propose a Potential-based Real-Time Routing (PRTR) protocol that minimizes delay for real-time traffic. However, their end-to-end delay bounds are probabilistic based on network calculus theory, which is not applicable to WSANs that require strict delay bounds. SPEED [17] bounds the end-to-end communication delays by enforcing a uniform delivery velocity. MM-SPEED [18] extends SPEED to support different delivery velocities and levels of reliability. RPAR [19] achieves application-specified communication delays at low energy cost by dynamically adapting transmission power and routing decisions. However, SPEED, MM-SPEED, and RPAR all assume each device knows its location via GPS or other localization services, which is not always feasible in WSANs. Moreover, the stateless routing policies adopted by these algorithms can not provide end-to-end delay bounds. Despite existing results on the general problem of real-time routing, none of the aforementioned work can be applied to WSANs. To meet this open challenge in industrial WSANs, we investigate the problem of real-time routing in WSANs in this paper.

III. Network Model

We consider a network model based on the WirelessHART standard [1]. A WSAN consists of a gateway, multiple access points, and a set of field devices. The gateway is wired to the access points. The access points and network devices are all equipped with half-duplex radio transceivers compatible with the IEEE 802.15.4 physical layer. The gateway communicates with field devices, such as sensors or actuators, through the access points. The access points and the field devices form a wireless mesh network. We use the term network device to refer any device in the system, including the gateway, an access point or a field device.

The WSAN adopts a centralized network management approach, where the network manager (usually a software running in the gateway) manages all devices. The network manager gathers the network topology information from the network devices, and generates and disseminates the routes and transmission schedule to all network devices. This centralized network management architecture, adopted by the WirelessHART standard, enhances the predictability and visibility of network operations at the cost of scalability.

The WSAN adopts a Time Division Multiple Access (TDMA) MAC layer protocol on top of the IEEE 802.15.4 physical layer. All devices across the network are synchronized. Time is divided into 10 ms slots, and each time slot can accommodate one data packet transmission and its acknowledgment. The WSAN supports multi-channel communication using channels defined in the IEEE 802.15.4 standard. To avoid potential collision between concurrent transmissions in a same channel, only one transmission is scheduled on each channel across the whole network. While this conservative design reduces network throughput and scalability, it avoids interference between transmissions within the network and thereby enhances reliability and predictability, which are important for industrial applications.

IV. Problem Formulation

In this section, we discuss the problem formulation. We consider a WSAN with a set of real-time flows $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$. Each flow $F_k = (s_k, d_k, \phi_k, D_k, T_k)$ is characterized by a source s_k, a destination d_k, a source route ϕ_k[1], a relative deadline D_k, and a period T_k.

We assume that all flows are ordered by priorities. Flow F_i has a higher priority than flow F_j, if and only if $i < j$. In practice, priorities are assigned based on deadlines, periods, or the criticality of the real-time flows. In this work, we use the deadline-monotonic priority assignment policy [20], where flows with shorter deadlines are assigned with higher priorities.

[1]The WirelessHART standard supports two types of routing: source routing and graph routing. Source routing provides a single route for each flow, whereas graph routing provides multiple redundant routes in a routing graph and therefore enhances reliability through route diversity. Our routing algorithms currently assume source routing and can be easily extended to a model where each flow has multiple source routes and sends redundant packets through every route to enhance reliability. Supporting graph routing is part of our future work.

Under a fixed priority scheduling policy, the transmissions of the flows are scheduled in the following way. Starting from the highest priority flow F_1, the following procedure is repeated for every flow F_i in decreasing order of priority. For the current priority flow F_i, the network manager schedules its transmissions along its route (starting from the source) in the earliest available time slots and on available channels. A time slot is available if no conflicting transmission is already scheduled in that slot. In a WSAN, the complete schedule is divided into superframes. A superframe consists of transmissions in a series of time slots and represents the communication pattern of a group of devices. A superframe repeats itself when it completes all its transmissions.

The goal of our routing algorithm is to find routes for the flows so that every flow can meet its deadline. Shortest path algorithms based on hop count [8] are commonly adopted in practice in WSANs. However, as shown in our simulation results presented in this paper, the effectiveness of these algorithms is far from the optimal. Based on the insights from end-to-end delay analyses, we propose two heuristics to assign routes to meet real-time requirements.

V. Conflict Delay Analysis

In this section, we summarize the delay analysis for WSANs. Previous works have studied end-to-end communication delays in WSANs [6], [7]. Based on their analyses, a packet can be delayed for two reasons: conflict delay and contention delay. Due to the half-duplex radio, two transmissions conflict with each other if they share a node (sender or receiver). In this case, only one of them can be scheduled in the current time slot. Therefore, if a packet conflicts with another packet that has already been scheduled in the current time slot, it has to be postponed to a later time slot, resulting in *conflict delay*. As a WSAN does not allow concurrent transmissions in the same channel, each channel can accommodate only one transmission across the network in each time slot. If all channels are assigned to transmissions of other packets, a packet must be delayed to a later slot, resulting in *contention delay*.

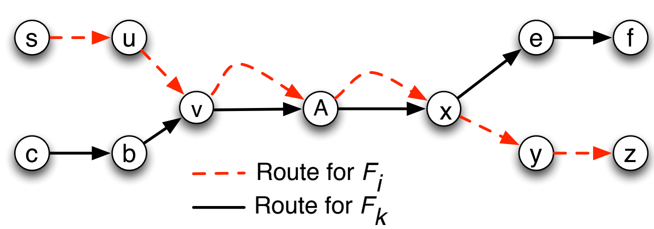


Fig. 1. An example showing conflict delay

From existing delay analyses [6], [7] as well as our simulations, conflict delay plays a significant role in the end-to-end delays of flows. Furthermore routing directly impacts conflict delays, whereas contention delays largely depend on the number of channels available. Therefore, in our routing design, we focus only on conflict delay. Saifullah et al.

proposed the Efficient Delay Analysis algorithm (EDA) in [6]. Here we briefly discuss their EDA algorithm and our approximation of EDA for our routing design.

We denote the maximum conflict delay that a package of flow F_k suffers from a package of flow F_i as Δ_k^i. Δ_k^i is counted based on the routes of the two flows. Δ_k^i equals the number of links in F_i's route that share nodes with F_k's route, times the number of transmissions scheduled on each link. We use κ to denote the number of transmissions scheduled for each link. We use an example in Figure 1 to show how to count Δ_k^i. F_k and F_i are two flows that share a part of their routes. Four links in F_i's route share nodes with F_k's route, which are $\{(u,v),(v,A),(A,x),(x,y)\}$. For simplicity, assuming only one transmission is scheduled for each link, Δ_k^i in this example equals 4.

Given a time interval of t slots, the number of packets of flow F_i that contribute to the delay of a packet of flow F_k during this time interval is upper bounded by $\lceil \frac{t}{T_i} \rceil$. As [6] shows, the worst-case conflict delay of a packet of flow F_k from all packets of flow F_i in a time window t can be bounded as

$$\Theta_k^i(t) = \lceil \frac{t}{T_i} \rceil \Delta_k^i, \quad (1)$$

where T_i is the period of flow F_i and Δ_k^i is the maximum conflict delay imposed by one packet of flow F_i.

By summarizing conflict delays from all flows with higher priorities than flow F_k, EDA proposes a upper bound of the conflict delay of flow F_k as

$$\Theta_k(t) = \sum_{i<k} \lceil \frac{t}{T_i} \rceil \Delta_k^i. \quad (2)$$

Based on Equation 2, EDA uses an iterative fixed-point algorithm to get the upper bound of F_k's conflict delay. However, the iterative fixed-point algorithm is too expensive for our routing algorithms since we will use the delay analysis as a basic component and call it extensively in our routing algorithm. Here, we propose an efficient approximation of EDA.

A packet of flow F_k can be delayed only within its lifetime D_k (the relative deadline of flow F_k). Instead of using an iterative fixed-point algorithm, we use the deadline of flow F_k as the length of time window. We further ignore the ceiling function and approximate the conflict delay that F_k can suffer from flow F_i as

$$\Theta_k^i = \frac{D_k}{T_i} \Delta_k^i. \quad (3)$$

By considering conflict delays from all flows, we approximate the conflict delay of flow F_k as:

$$\Theta_k = \sum_{i<k} \frac{D_k}{T_i} \Delta_k^i. \quad (4)$$

We present the pseudocode of our conflict delay analysis algorithm in Algorithm 1. Because each look up takes $log|\phi_k|$ in average, the for loop from line 9 to line 11 has a complexity of $O(|\phi_i| log|\phi_k|)$. The for loop from line 7 to 11 has a complexity of $O(n|\phi_i| log|\phi_k|)$. The for loop from line 2 to line 11 has a complexity of $O(n(|\phi_k| + n|\phi_i| log|\phi_k|)) = O(n^2|\phi_i| log|\phi_k|)$. Because the length of any path ϕ_k is no longer than $|V|$ (each node is visited only once given loop can be removed), then $|\phi_k| \leq |V|$. the complexity of our conflict delay analysis algorithm is $O(n^2|V| log|V|)$.

Algorithm 1: Conflict Delay Analysis

1 **Function** *CDA*($G, \mathcal{F}, \kappa$)
  Input : A graph $G(V,E)$, a flow set $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$ ordered by priority, where $F_k = (s_k, d_k, \phi_k, T_k, D_k)$
  Output : Conflict delays $\{\theta_1, \theta_2, \cdots, \theta_n\}$ for all flows
2   **for** ***each*** *flow* F_k ***from*** F_2 ***to*** F_n **do**
3     $S = \emptyset$;
4   **for** ***each*** *link* $(u,v) \in \phi_k$ **do**
5   insert u into S;
6   insert v into S;
7   **for** ***each*** *flow* F_i ***from*** F_1 ***to*** F_{k-1} **do**
8       $\Delta_k^i = 0$;
9   **for** ***each*** *link* $(u,v) \in \phi_i$ **do**
10   **if** $u \in S$ ***or*** $v \in S$ **then**
11          $\Delta_k^i = \Delta_k^i + \kappa$;
12 **for** ***each*** *flow* F_k ***from*** F_1 ***to*** F_n **do**
13    $\Theta_k = 0$;
14   **if** $k > 1$ **then**
15   **for** ***each*** *flow* F_i ***from*** F_1 ***to*** F_{k-1} **do**
16        $\Theta_k = \Theta_k + \frac{D_k}{T_i} \Delta_k^i$;

VI. Real-Time Routing

In WSANs, existing routing algorithms [8] usually take hop count as the metric when selecting routes. As a result, each flow will select a route with the minimum hop count. However, the shortest path does not necessarily lead to the smallest end-to-end delay. As previous delay analyses [6], [7] and our simulations presented in Section VII show, conflict delay plays an important role in the end-to-end delay. In this section, we take conflict delay into account in the routing decision and propose our real-time routing algorithms.

As we summarized in Section V, the conflict delay that a flow F_k experiences is approximated as $\Theta_k = \sum_{i<k} \frac{D_k}{T_i} \Delta_k^i$, where T_i is the period of a high-priority flow F_i, and Δ_k^i is the maximum conflict delay imposed by one packet of flow F_i. To be more specific, Δ_k^i is the number of transmissions of flow F_i that share nodes with flow F_k, which depends on the routes of flows F_i and F_k. In our real-time routing algorithms, we aim to reduce the conflict delay caused by high-priority flows under a deadline-monotonic priority assignment that assigns

higher priorities to flows with shorter deadlines. This policy can improve the number of flows meeting their deadlines, as shown in our simulation results in Section VII.

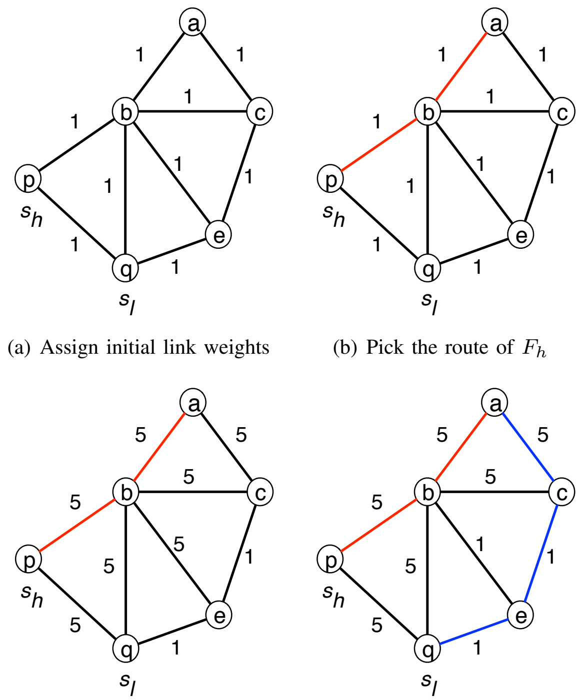
A. Conflict-Aware Routing

Algorithm 2: Conflict-Aware Routing

1 **Function** *CAR*($G, \mathcal{F}$)
  Input : A graph $G(V, E)$, A flow set $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$ ordered by priority with $F_k = (s_k, d_k, T_k, D_k)$
  Variable: link weight w, link delay coefficient c
  Output : A route ϕ_k for each flow F_k
2   **for** ***each*** *link* $(u, v) \in E$ **do**
3     $w_{(u,v)} = 1$;
4     $c_{(u,v)} = 0$;
5   **for** ***each*** *flow* F_k ***from*** F_1 ***to*** F_n **do**
6   **if** $k > 1$ **then**
7   **for** ***each*** *link* $(u, v) \in E$ **do**
8         $w_{(u,v)} = 1 + D_k \cdot c_{(u,v)}$;
9   Find the shortest path ϕ_k connecting s_k to d_k;
10   Assign ϕ_k as flow F_k's route;
11   **for** ***each*** *link* $(u, v) \in E$ **do**
12   **if** (u, v) *shares at least one node with* F_k*'s route* R_k **then**
13         $c_{(u,v)} = c_{(u,v)} + \frac{1}{T_k}$;

We discuss our Conflict-Aware Routing (CAR) algorithm, which pick routes with small conflict delays caused by high-priority flows. Our CAR algorithm runs as follows. We assign routes for flows following the priority order, from the highest to the lowest. For each flow F_k, we update the link weights based on routes of higher priority flows. If a link (u, v) shares at least one node with a higher priority flow F_i's route, its weight will be increased by $\frac{D_k}{T_i}$ based on Equation (3). After updating the link weights, we run Dijkstra's algorithm [21] to find the path ϕ_k with the smallest path weight. The algorithm terminates when the flow with lowest priority is assigned with a route ϕ_n. We present the pseudocode of our CAR algorithm in Algorithm 2.

Figure 2 shows an example of our CAR algorithm. In this example, we have two flows, F_h and F_l. Flow F_h has a higher priority than flow F_l. The flow F_h has a source p, a destination a, a period $1s$, and a deadline $1s$. The flow F_l has a source q, a destination a, a period $4s$, and a deadline $4s$. We use black lines to represent links in the network, red lines to represent the route of flow F_h, and blue lines to represent the route of flow F_l. In the first step (Figure 2(a)), we assign an initial link weight of 1 for each link in the topology. In the second step (Figure 2(b)), we run the shortest path algorithm to get F_h's route as $p \to b \to a$. In the second step (Figure 2(c)), we update the link weights based on flow F_h's route. If a link (u, v) shares at least one node with any link on flow



(a) Assign initial link weights (b) Pick the route of F_h

(c) Update link weights (d) Pick the route of F_l

Fig. 2. An example of the CAR algorithm. Red lines represent the route of flow F_h. Blue lines represent the route of flow F_l.

F_h's route, we add an estimated conflict delay $\frac{D_l}{T_h} = 4$ to the link weight, because each link in flow F_h's route will bring $\frac{D_l}{T_h} = 4$ conflict delay to flow F_l based on the delay analysis in Equation 3. In this example, links that could encounter conflict delay from flow F_h will have a link weight of 5. In the fourth step (Figure 2(d)), we find the shortest path from flow F_l's source q to its destination a, which is $q \to e \to c \to a$ in this example. Note the path we found is different from the shortest path based on hop count $q \to b \to a$.

Now we discuss the complexity of the CAR algorithm. We first check the complexity for each flow (one iteration within the for loop at lines 5-13). The complexity to update the link weights is $O(|E|)$. The complexity of the Dijkstra's algorithm is $O(|E|+|V|log|V|)$, and the complexity to update the delay coefficients is $O(|E|)$. Then the complexity of each flow is $O(|E| + |V|log|V|)$. Therefore, the complexity of our CAR algorithm is $O(|\mathcal{F}|(|E| + |V|log|V|))$.

B. Iterative Conflict-Aware Routing

By reducing the conflict delay of low priority flows, we can accommodate more flows while meeting their deadlines. However, CAR is based on flow priorities, and high priority flows are not aware of the routes of low priority flows. We further improve the real-time capacity by introducing an approach where high priority flows also take into account the routes of low priority flows. We introduce our Iterative Conflict-Aware Routing (ICAR) algorithm as Algorithm 3.

The ICAR algorithm terminates when no flows update their

Algorithm 3: Iterative Conflict-Aware Routing

Function *ICAR*($G, \mathcal{F}$)
  Input : A graph $G(V, E)$, A flow set $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$ ordered by priority with $F_k = (s_k, d_k, T_k, D_k)$
  Variable: link weight w, per link flow set S, link delay coefficient c
  Output : A route ϕ_k for each flow F_k
  $changed = true$;
  $schedulable = false$;
  for ***each*** *flow* $F_k \in \mathcal{F}$ **do**
    $\phi_k = \emptyset$;
  for ***each*** *link* $(u, v) \in E$ **do**
    $S_{(u,v)} = \emptyset$;
    $c_{(u,v)} = 0$;
  while $changed == true$ ***and*** $schedulable == false$ **do**
    $changed = false$;
    $schedulable = true$;
    for ***each*** *flow* F_k ***from*** F_1 ***to*** F_n **do**
      if $k > 1$ **then**
        for ***each*** *link* $(u, v) \in E$ **do**
          if $F_k \in S_{(u,v)}$ **then**
            $w_{(u,v)} = 1 + D_k \cdot (c_{(u,v)} - \frac{1}{T_k})$;
          else
            $w_{(u,v)} = 1 + D_k \cdot c_{(u,v)}$;
      Find the shortest path ϕ_{temp} connecting s_k to d_k;
      $schedulable_{temp} = EDA(\phi_{temp})$;
      if $\phi_k == \emptyset$ ***or*** $(\phi_{temp} \neq \phi_k$ ***and*** $schedulable_{temp} == true)$ **then**
        $routechanged = true$;
        $schedulable = schedulable_{temp}$;
      if $\phi_{temp} == \phi_k$ ***or*** $(\phi_{temp} \neq \phi_k$ ***and*** $schedulable_{temp} == false)$ **then**
        $routechanged = false$;
        $schedulable = EDA(\phi_k)$;
      if $routechanged == true$ **then**
        $changed = true$;
        for ***each*** *link* $(u, v) \in \phi_k$ **do**
          if $(u, v) \notin \phi_{temp}$ **then**
            Remove F_k from $S_{(u,v)}$;
            $c_{(u,v)} = c_{(u,v)} - \frac{1}{T_k}$;
        for ***each*** *link* $(u, v) \in \phi_{temp}$ **do**
          if $(u, v) \notin \phi_k$ **then**
            Insert F_k into $S_{(u,v)}$;
            $c_{(u,v)} = c_{(u,v)} + \frac{1}{T_k}$;
        $\phi_k = \phi_{temp}$;

routes in the last round or all flows are schedulable under EDA. Within each round, flows pick their routes one by one. For each flow F_k, the algorithm first updates link weights based on the routes of other flows. One difference between ICAR and CAR is that lower priority flows can also contribute to link weights for F_k. ICAR lets higher priority flows be aware of the routes of lower priority flows, and therefore reduces the overlapping of their routes. This leaves a bigger space for low priority flows and gives them a higher chance to find routes which are schedulable. If a new route R_{temp} is found, the algorithm will first check whether flow F_k with this new route is schedulable under EDA. If yes, this new route R_{temp} is assigned to F_k and flow F_k is indicated as schedulable. If not, flow F_k will use its old route R_k. If flow F_k is schedulable under EDA, we indicate it as schedulable; otherwise, F_k is unschedulable. The algorithm will enter into a new round if at least one flow is not schedulable and at least one flow has an updated route.

VII. Evaluation

We evaluate our real-time routing algorithms through both experiments on a physical WSAN testbed and simulations based on the WSAN testbed topology. We compare our Conflict-Aware Routing (CAR) algorithm and the Iterative Conflict-Aware Routing (ICAR) algorithm with the Shortest Path Routing (SP) algorithm. In SP, each flow uses breath-first search algorithm [21] to select a route with the minimum hop count.

A. Experiments on a WSAN Testbed

We evaluate our routing designs on an indoor WSAN testbed consisting of 63 TelosB motes, located on the fifth floors of two adjacent buildings. Figure 3 shows the topology of the WSAN testbed. We use motes 129 and 155 (green circles) as access points, which are physically connected to a root server (the gateway). The other motes are used as field devices (red circles). The network manager as a software runs on this root server. For each link in the testbed, we measure its *packet reception ratio (PRR)* by counting the number of received packets among 250 packets transmitted on the link. Following the practice of industrial deployment, we only add links with PRRs higher than 90% to the topology of the testbed. We implement a multi-channel TDMA MAC protocol on top of the IEEE 802.15.4 physical layer. Clocks of network devices across the entire network are synchronized using the Flooding Time Synchronization Protocol (FTSP) [22]. Time is divided into 10 ms slots.

We generate 8 flows in our experiment. We use 8 channels in this experiment. The period of each flow is picked up from the range of $2^{4\sim7} \times 10$ milliseconds. The length of the hyper-period is 128 milliseconds. The relative deadline of each flow equals to its period. All flows are schedulable based on our delay analyses. We run our experiments long enough such that each flow can deliver at least 100 packets.

In Figure 4, we compare delays from the experimental results with delay analyses as well as simulation. We compare four delays for each flow: minimum delay in experiments

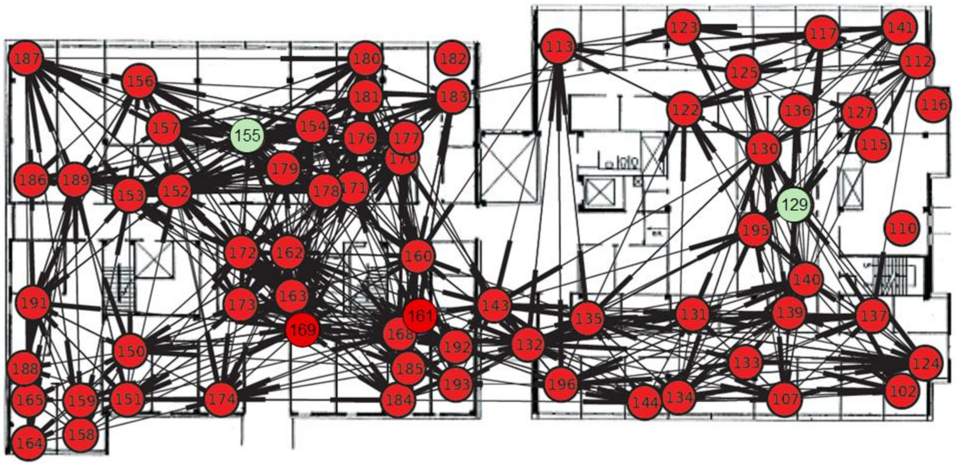
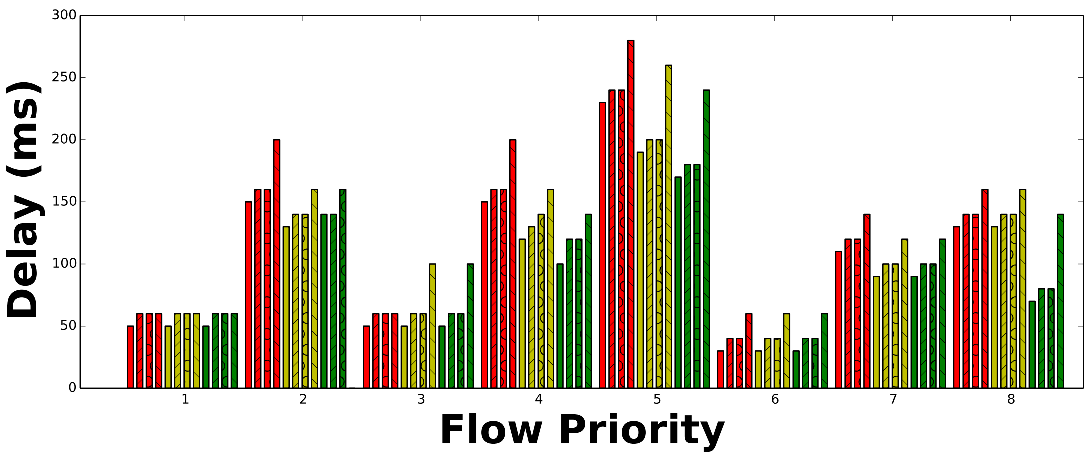
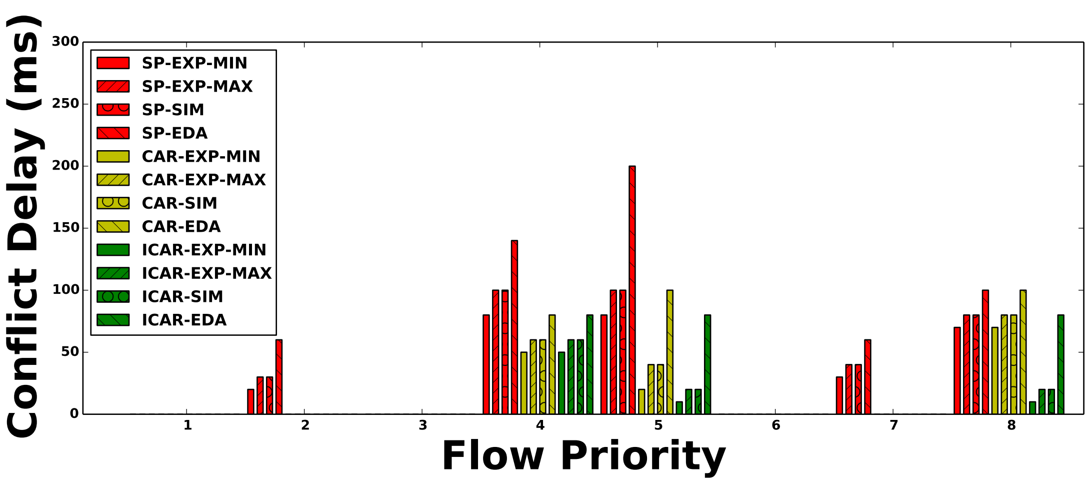


Fig. 3. Topology of the WSAN Testbed



(a) End-to-end delay



(b) Conflict delay

Fig. 4. Delays

(EXP-MIN), maximum delay in experiments (EXP-MAX), maximum delay in simulation (SIM), and the estimated delay in EDA [6]. We evaluate both the end-to-end delays and the conflict delays. To save space, Figure 4(a) shares the same legend with Figure 4(b).

First of all, the results show for both the end-to-end delay and conflict delay, every flow has the four delays follow the following order: EXP-MIN $\leq$ EXP-MAX $\leq$ SIM $\leq$ EDA.

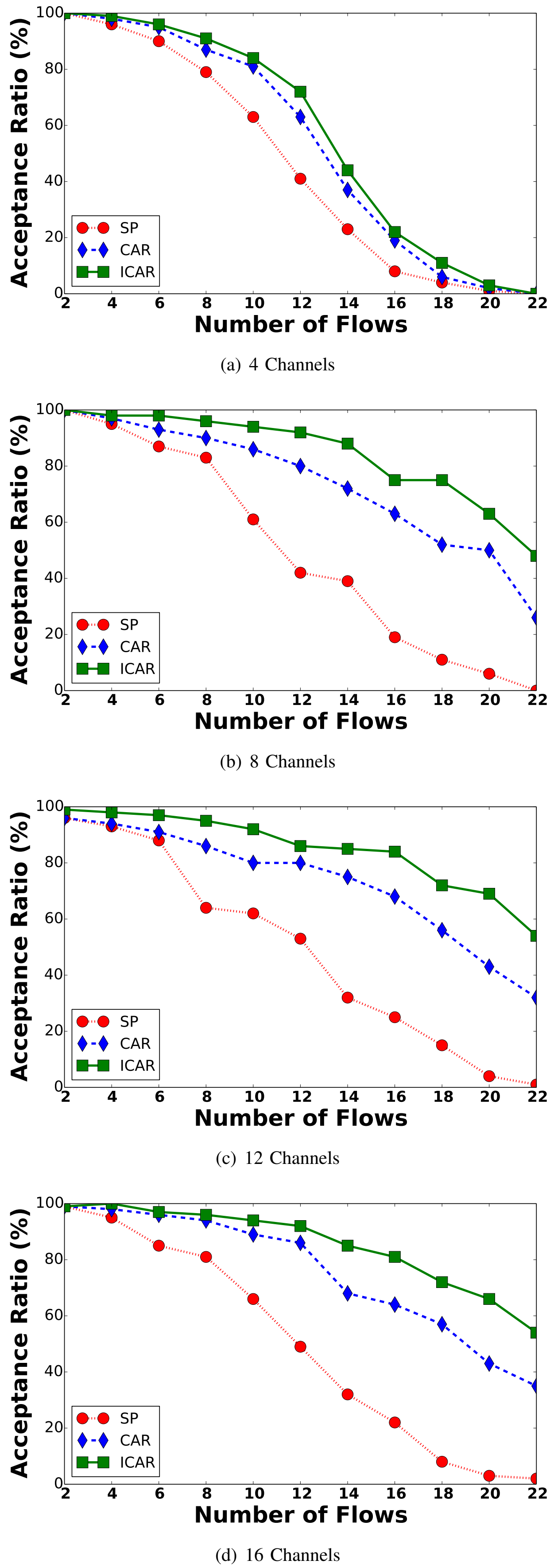
This shows that simulation and delay analysis are safe upper bounds of the actual delays. In addition, SIM is consistently higher than EXP-MAX, which indicates our simulations can generate test cases with worse delays than those observed on the testbed.

Figure 4(a) compares end-to-end delays of flows based on different routing algorithms: SP, CAR, and ICAR. The results show CAR and ICAR can reduce the end-to-end delays compared with SP. Furthermore, ICAR can further reduce the delays for flows with low priorities. Given we have enough channels in this experiment, there is no contention delay. We further compare conflict delays of flows in Figure 4(b). Clearly, CAR and ICAR can reduce the conflict delays of flows. For example, flow 7 has conflict delays in SP routing. However, its conflict delays in CAR and ICAR routings are zero. By reducing conflict delays, CAR and ICAR can reduce the end-to-end delays of flows.

B. Simulations

Besides the testbed experiments, we also test our routing algorithms through simulations on testbed topology. The simulator uses the same routing and scheduling design used on our testbed experiments and is written in C++. All simulations are performed on a MacBook Pro laptop with 2.4 GHz Intel Core 2 Duo processor. To show the impact of the number of channels, we test our algorithms under different number of channels (4, 8, 12, and 16) in our simulation. We test our routing designs on different numbers of flows by increasing the numbers of source and destination pairs from 2 to 22. The period T_k of the each flow F_k is randomly generated in the range of $2^{4\sim7} \times 10$ milliseconds. The relative deadline D_k of every flow F_k is equal to its period. For each flow set, we generate 100 test cases and simulate them on testbed topologies.

We first compare the acceptance ratios of CAR, ICAR and SP in Figure 5. SP always has the lowest acceptance ratio. Both CAR and ICAR have much higher acceptance ratios than SP when the network has at least 8 channels. ICAR has a higher acceptance ratio than CAR, which shows the benefit of letting flows with higher priorities be aware of the routes of lower priority flows. The performance of our real-time routing algorithms improves when the number of channels increases. Because when the network has very few channels, contention delay is the main part of end-to-end delay. However, when the network has more channels, the conflict delay becomes the dominant part of the end-to-end delay. Compared to SP, CAR and ICAR can improve the acceptance ratio by 239% and 350% in average with 16 channels, respectively. We



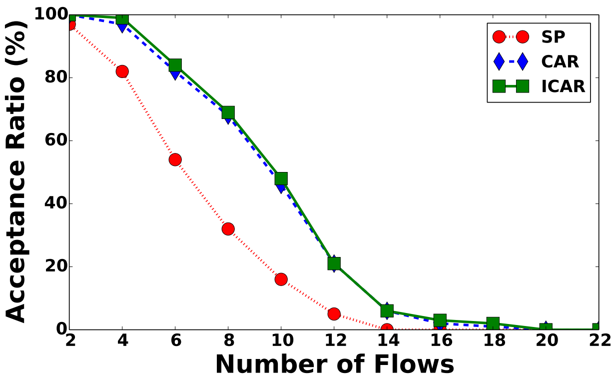
(a) 4 Channels

(b) 8 Channels

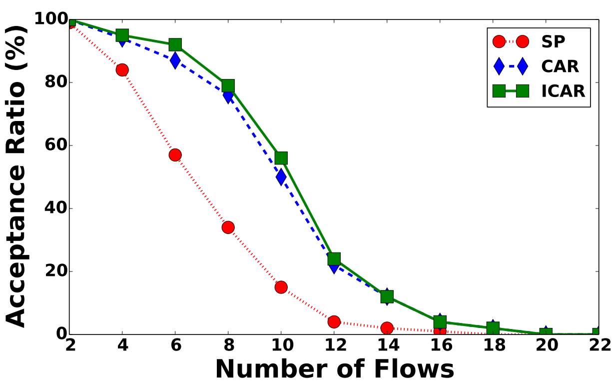
(c) 12 Channels

(d) 16 Channels

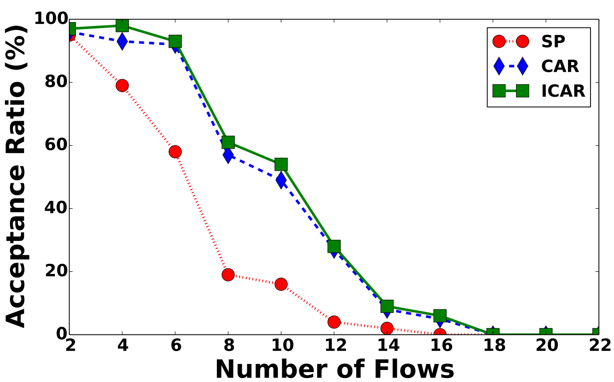
Fig. 5. Acceptance Ratio in Simulation



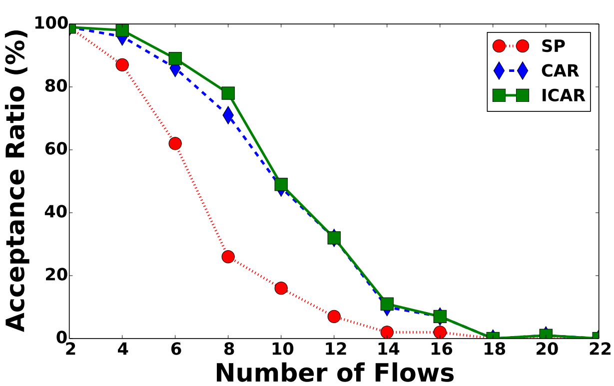
(a) 4 Channels



(b) 8 Channels

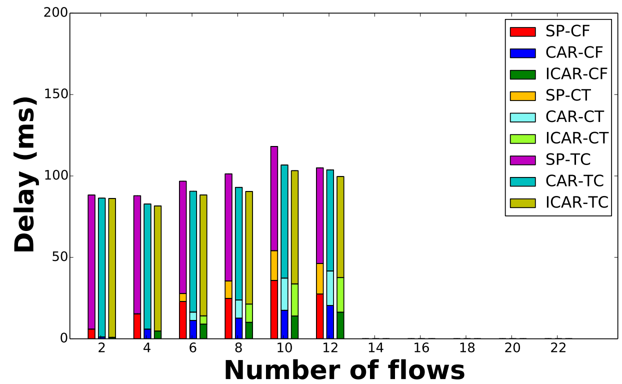


(c) 12 Channels

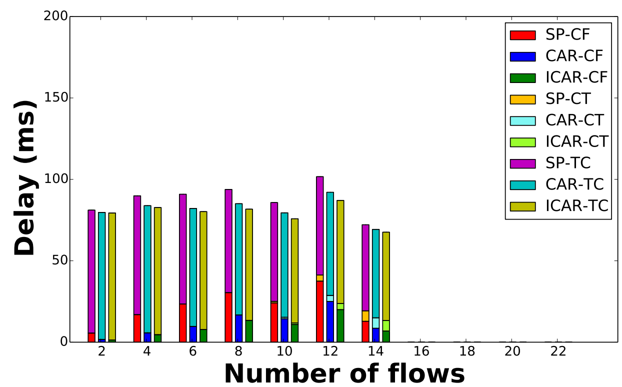


(d) 16 Channels

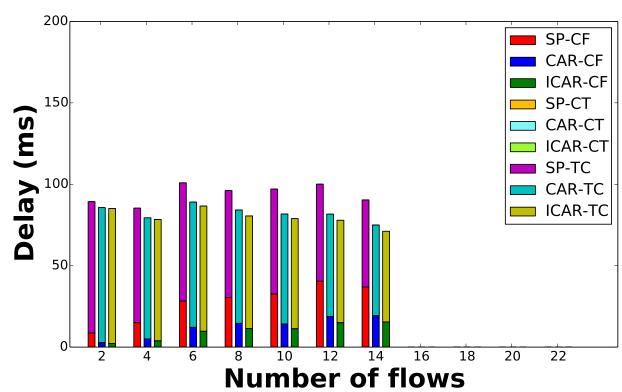
Fig. 6. Acceptance Ratio in Analysis



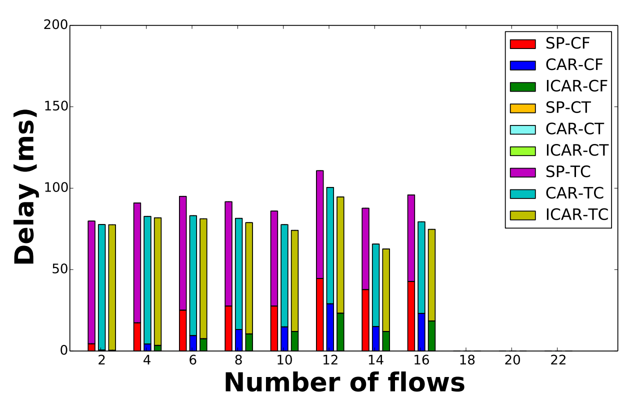
(a) 4 Channels



(b) 8 Channels

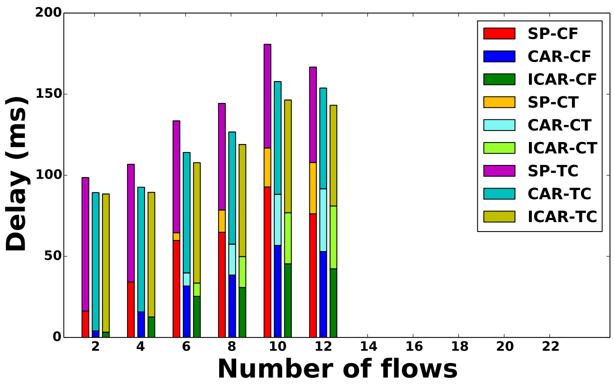


(c) 12 Channels

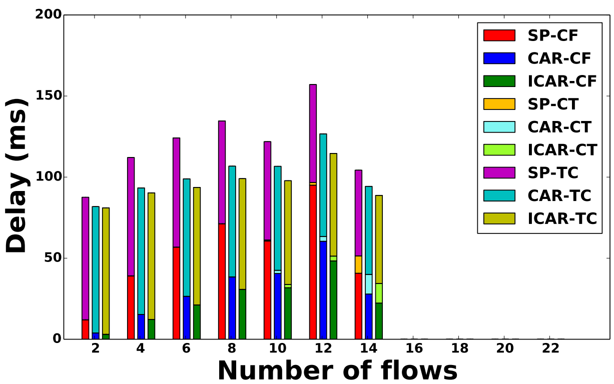


(d) 16 Channels

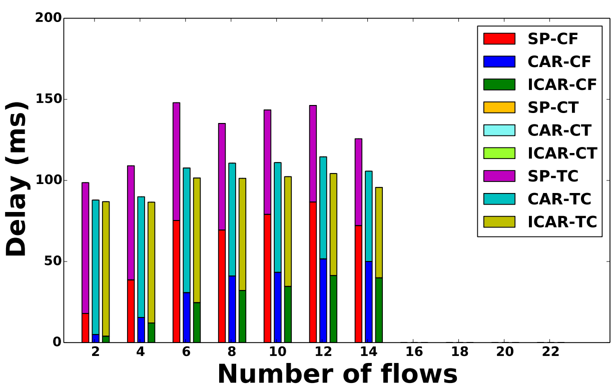
Fig. 7. Delays in Simulation



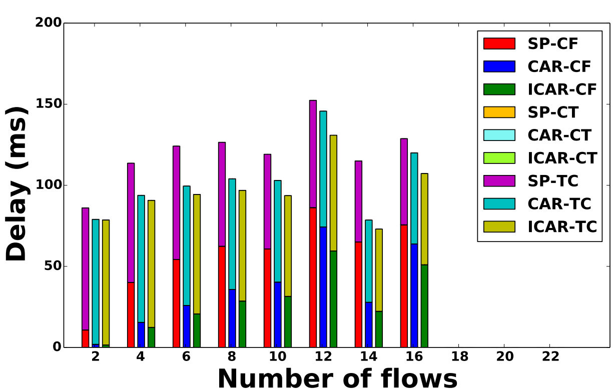
(a) 4 Channels



(b) 8 Channels



(c) 12 Channels



(d) 16 Channels

Fig. 8. Delays in Analysis

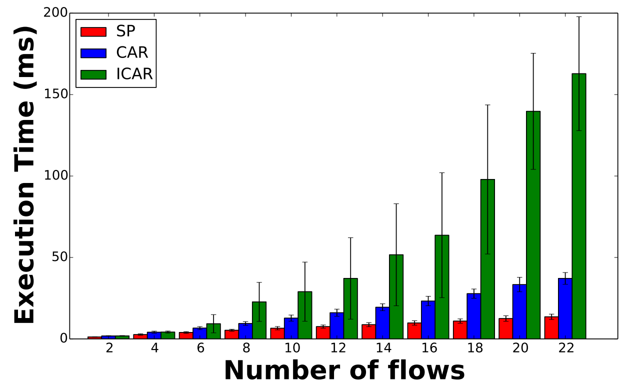


Fig. 9. Execution Time

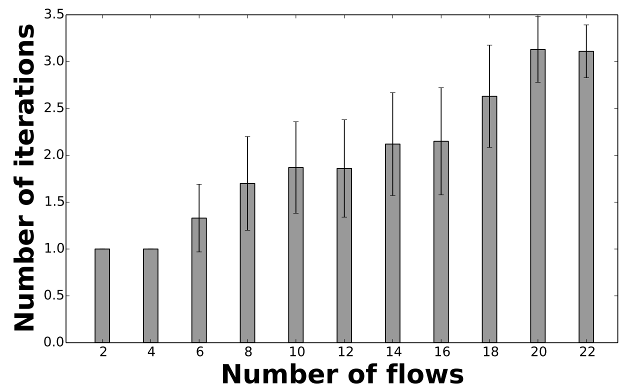


Fig. 10. Number of Iterations

further compare the acceptance ratios of CAR, ICAR and SP on efficient delay analysis [6] in Figure 6. Our simulation results show CAR and ICAR have higher acceptance ratio in delay analysis compared to SP. Because the delay analysis is pessimistic compared to simulation, acceptance ratios in delay analyses (Figure 6) are lower than simulation (Figure 5).

We further compare end-to-end delays of CAR, ICAR, and SP in Figure 7. Here we draw the average delays of all 100 test cases. We use CF to stand for conflict delay, CT for contention delay, and TC for transmission count (number of transmissions scheduled on the route). When the number of channels is small (4 or 8), the contention delays can be important part of the end-to-end delays. However, when the network has 12 channels, the contention delays are zero, and conflict delays dominate since then. Although CAR and ICAR may lead to routes with longer hop count, their end-to-end delays are smaller than SP in average. Because CAR and ICAR have fewer conflict delays than SP in all cases. The end-to-end delays in delay analysis [6] show the same trend in Figure 8.

We compare the execution time of SP, CAR, and ICAR when there are 10 channels in Figure 9. The execution time increases as the number of flows increases in all three algorithms. The execution time of three routing algorithms follows this order: SP<CAR<ICAR. SP has the lowest execution time since it uses the breadth-first search algorithm. ICAR has a

higher execution time than CAR because it is an iterative algorithm. The execution time of ICAR is less than 200 ms when the number of flows is 22, which is acceptable in real-world operations. We also show the number of iterations in Figure 10. The number of iterations increases as the number of flows increases. Even for 22 flows, the maximum number of iterations is 4 in our simulations, which is relatively small when considering the size of the network.

VIII. Conclusion

As process industries start to adopt wireless sensor-actuator networks (WSANs) for control applications, it is crucial to achieve real-time communication in this emerging class of networks. Routing has significant impacts on end-to-end communication delays in WSANs. However, despite considerable research on real-time transmission scheduling and delay analysis for such networks, real-time routing remains an open question for WSANs. This paper presents a *conflict-ware real-time routing* approach for WSANs. This approach leverage a key observation that conflicts among transmissions sharing a common field device contribute significantly to communication delays in industrial WSANs such as WirelessHART networks. By incorporating conflict delays in the routing decisions, conflict-aware real-time routing algorithms allow a WSAN to accommodate more real-time flows while meeting their deadlines. Evaluation based on simulations and experiments on a real WSANs testbed show conflict-aware real-time routing can lead to up to three-fold improvement in real-time capacity of WSANs.

References

[1] "WirelessHART specification," 2007. http://www.hartcomm2.org.

[2] "ISA100: Wireless Systems for Automation." https://www.isa.org/isa100/.

[3] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, and M. Nixon, "WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control," in *RTAS '08*.

[4] "Emerson's wirelesshart report." http://www2.emersonprocess.com/en-us/plantweb/wireless/pages/wirelesshomepage-flash.aspx.

[5] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-Time Scheduling for WirelessHART Networks," in *RTSS'10*.

[6] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end delay analysis for fixed priority scheduling in WirelessHART networks," in *RTAS'11*.

[7] C. Wu, M. Sha, D. Gunatilaka, A. Saifullah, C. Lu, and Y. Chen, "Analysis of EDF Scheduling for Wireless Sensor-Actuator Networks," in *IEEE/ACM Symposium on Quality of Service (IWQoS'14)*, May 2014.

[8] S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "Reliable and Real-time Communication in Industrial Wireless Mesh Networks," in *RTAS'11*.

[9] O. Chipara, C. Wu, C. Lu, and W. Griswold, "Interference-Aware Real-Time Flow Scheduling for Wireless Sensor Networks," in *ECRTS'11*.

[10] A. Saifullah, C. Wu, P. Tiwari, Y. Xu, Y. Fu, C. Lu, and Y. Chen, "Near Optimal Rate Selection for Wireless Control Systems," in *RTAS'12*.

[11] B. Li, Z. Sun, K. Mechitov, C. Lu, S. Dyke, G. Agha, and B. Spencer, "Realistic case studies of wireless structural control," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'13)*, April 2013.

[12] A. Saifullah, C. Wu, P. B. Tiwari, Y. Xu, Y. Fu, C. Lu, and Y. Chen, "Near optimal rate selection for wireless control systems," *ACM Transactions on Embedded Computing Systems (TECS'14)*, April 2014.

[13] M. Sha, D. Gunatilaka, C. Wu, and C. Lu, "Implementation and Experimentation of Industrial Wireless Sensor-Actuator Network Protocols," in *The 12th European Conference on Wireless Sensor Networks (EWSN'15)*, February 2015.

[14] A. Saifullah, D. Gunatilaka, P. Tiwari, M. Sha, C. Lu, B. Li, C. Wu, and Y. Chen, "Schedulability Analysis under Graph Routing for WirelessHART Networks," in *RTSS'15*.

[15] B. Li, L. Nie, C. Wu, and H. G. C. Lu, "Incorporating Emergency Alarms in Reliable Wireless Process Control," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'15)*, April 2015.

[16] Y. Xu, F. Ren, T. He, C. Lin, C. Chen, and S. K. Das, "Real-time routing in wireless sensor networks: A potential field approach," *ACM Transactions on Sensor Networks*, vol. 9, pp. 35:1–35:24, June 2013.

[17] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks," in *ICDCS '03*.

[18] E. Felemban, C.-G. Lee, and E. Ekici, "MMSPEED: Multipath Multi-SPEED Protocol for QoS Guarantee of Reliability and Timeliness in Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 6, pp. 738–754, 2006.

[19] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic, and T. Abdelzaher, "Real-time Power-Aware Routing in Sensor Networks," in *14th IEEE International Workshop on Quality of Service (IWQoS'06)*, pp. 83–92, June 2006.

[20] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*. Addison Wesley, 3rd ed., 2001.

[21] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd ed., 2001.

[22] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The Flooding Time Synchronization Protocol," in *SenSys'04*.