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 an 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.

Follow this and additional works at: http://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

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-aware real-time routing approach for WSANs. This approach leverages 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 III discusses the problem formulation. Section IV provides a brief review of the existing delay analyses, and Section V presents our real-time routing algorithms. Section VI evaluates our routing algorithms through simulations, then Section VII 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 the 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 \( \mathcal{F} = \{F_1, F_2, \ldots, F_n\} \). 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 \( \phi_k \), 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.

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.

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 $\Delta^i_k$. $\Delta^i_k$ is counted based on the routes of the two flows. $\Delta^i_k$ 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 $\kappa$ to denote the number of transmissions scheduled for each link. We use an example in Figure 1 to show how to count $\Delta^i_k$. $F_k$ 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 $\{(u,v),(v,x),(x,y),(y,z)\}$. For simplicity, assuming only one transmission is scheduled for each link, $\Delta^i_k$ 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

$$\Theta^i_k(t) = \lceil \frac{t}{T_i} \rceil \Delta^i_k,$$

where $T_i$ is the period of flow $F_i$, and $\Delta^i_k$ 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

$$\Theta_k(t) = \sum_{i<k} \lceil \frac{t}{T_i} \rceil \Delta^i_k.$$ 

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

$$\Theta^i_k(t) = \frac{D_k}{T_i} \Delta^i_k.$$ 

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

$$\Theta_k = \sum_{i<k} \frac{D_k}{T_i} \Delta^i_k.$$ 

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 $O(|\phi_k| \log |\phi_k|)$. The for loop from line 7 to line 11 has a complexity of $O(n|\phi_k| \log |\phi_k|)$. The for loop from line 2 to line 11 has a complexity of $O(n|\phi_k| + n|\phi_k| \log |\phi_k|)) = O(n^2|\phi_k| \log |\phi_k|)$. Because the length of any path $\phi_k$ 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 $O(n^2|V| \log |V|)$.

### 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 $\Theta_k = \sum_{i<k} \frac{D_k}{T_i} \Delta^i_k$, where $T_i$ is the period of a high-priority flow $F_i$, and $\Delta^i_k$ is the maximum conflict delay imposed by one packet of flow $F_i$. To be more specific, $\Delta^i_k$ 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, F)
2   Input: A graph G(V, E), A flow set
3     \( F = \{ F_1, F_2, \ldots, F_n \} \) ordered by priority
4     with \( F_k = (s_k, d_k, T_k, D_k) \)
5   Variable: link weight \( w \), link delay coefficient \( c \)
6   Output: A route \( \phi_k \) for each flow \( F_k \)
7   for each link \( (u,v) \in E \) do
8       \( w_{(u,v)} = 1; \)
9       \( c_{(u,v)} = 0; \)
10   for each flow \( F_k \) from \( F_1 \) to \( F_n \) do
11       if \( k > 1 \) then
12           for each link \( (u,v) \in E \) do
13               \( w_{(u,v)} = 1 + D_k \cdot c_{(u,v)}; \)
14           Find the shortest path \( \phi_k \) connecting \( s_k \) to \( d_k; \)
15           Assign \( \phi_k \) as flow \( F_k \)'s route;
16           for each link \( (u,v) \in E \) do
17               if \( (u,v) \) shares at least one node with \( F_k \)'s route \( R_k \) then
18                   \( c_{(u,v)} = c_{(u,v)} + \frac{1}{T_k}; \)
19   end if
20   end for
21 end for
22 end for
23 end function
```

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 \( D_k \cdot c_{(u,v)} \). After updating the link weights, we run Dijkstra’s algorithm [21] to find the path \( \phi_k \) with the smallest path weight. The algorithm terminates when the link with lowest priority is assigned with a route \( \phi_k \). 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 \( p \rightarrow b \rightarrow a \). 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 \( F_h \)'s route, we add an estimated conflict delay \( \frac{D_k}{T_k} = 4 \) 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 \( q \rightarrow e \rightarrow c \rightarrow a \) in this example. Note the path we found is different from the shortest path based on hop count \( q \rightarrow b \rightarrow a \).

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 \( O(|F|(|E| + |V| \log |V|)) \).

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

```plaintext
Function ICAR(G, F)
    Input: A graph G(V, E), A flow set 
          \( F = \{F_1, F_2, \ldots, F_n\} \) 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 \( \phi_k \) for each flow \( F_k \)
    \( changed = true; \)
    \( schedulable = false; \)
    for each flow \( F_k \in F \) 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
                            \( w_{(u,v)} = 1 + D_k \cdot \left( c_{(u,v)} - \frac{1}{T_k} \right); \)
                        else
                            \( w_{(u,v)} = 1 + D_k \cdot c_{(u,v)}; \)
        Find the shortest path \( \phi_{temp} \) 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 \( S_{(u,v)} \); 
                        \( c_{(u,v)} = c_{(u,v)} + \frac{1}{T_k}; \)
                        \( \phi_k = \phi_{temp}; \)
```

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 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 \( 2^{4 \times 7} \times 10 \) 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 ≤ EXP-MAX ≤ SIM ≤ 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 each flow $F_k$ is randomly generated in the range of $2^{4-7} \times 10$ 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.

Fig. 5. Acceptance Ratio in Simulation
Fig. 6. Acceptance Ratio in Analysis

Fig. 7. Delays in Simulation
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-aware real-time routing approach for WSANs. This approach leverages 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