Schedulability Analysis under Graph Routing in WirelessHART Networks

Abusayeed Saifullah[†], Dolvara Gunatilaka[‡], Paras Tiwari[‡], Mo Sha[⋆], Chenyang Lu[‡], Bo Li[‡]
Chengjie Wu[‡], and Yixin Chen[‡]
Department of Computer Science, Missouri University of Science and Technology[†]
Department of Computer Science, State University of New York at Binghamton[⋆]
Cyber-Physical Systems Laboratory, Washington University in St. Louis[‡]

***Abstract*—Wireless sensor-actuator networks are gaining ground as the communication infrastructure for process monitoring and control. Industrial applications demand a high degree of reliability and real-time guarantees in communication. Because wireless communication is susceptible to transmission failures in industrial environments, industrial wireless standards such as WirelessHART adopt reliable graph routing to handle transmission failures through retransmissions and route diversity. While these mechanisms are critical for reliable communication, they introduce substantial challenges in analyzing the schedulability of real-time flows. This paper presents the first worst-case end-to-end delay analysis for periodic real-time flows under reliable graph routing. The proposed analysis can be used to quickly assess the schedulability of real-time flows with stringent requirements on both reliability and latency. We have evaluated our schedulability analysis against experimental results on a wireless testbed of 69 nodes as well as simulations. Both experimental results and simulations show that our delay bounds are safe and enable effective schedulability tests under reliable graph routing.**

I. Introduction

Wireless sensor-actuator networks (WSANs) are gaining ground as the communication infrastructure for industrial process monitoring and control systems. To support monitoring and control, a WSAN periodically delivers data from sensors to a controller and then delivers its control input data to the actuators through the multi-hop mesh network. Wireless control in process industries demands a high degree of reliability and real-time guarantees in communication [1]. Failures in wireless transmissions are prevalent in industrial environments due to channel noise, power failure, physical obstacle, multi-path fading, and interference from co-existing wireless systems. As a widely adopted industrial wireless standard, WirelessHART [2] introduces the *reliable graph routing* approach to handle transmission failures through retransmissions and route diversity.

Reliable graph routing [2] employs the following mechanisms to recover from transmission failures. A routing graph is constructed as a directed list of paths between two devices, thereby providing redundant routes for real-time flows between sensors and actuators. For transmission between a receiver and a sender, a time slot can be either *dedicated* (i.e., a time slot when at most one transmission is scheduled to a receiver) or *shared* (i.e., a time slot when multiple nodes may contend to send to a common receiver). For each flow, the network handles transmission failures by allocating a dedicated time slot for each node on a path from the source, followed by allocating a second dedicated slot on the same path for a retransmission, and then by allocating a third shared slot on a separate path for another retransmission [2]. While effective in achieving reliable communication, this fault-tolerant mechanism introduces significant challenges in worst-case delay analysis for real-time flows in a WSAN. A WirelessHART network is managed by a centralized network manager responsible for computing the routes and schedules for all field devices. As wireless conditions can change quickly in industrial environments, it must quickly assess the schedulability of real-time flows. If some flows cannot be guaranteed to meet their deadlines, the network manager can adapt to the overload through online admission control and reconfiguration. Fast online schedulability analysis is therefore important for WirelessHART networks to adapt to dynamic environments.

In this paper, we propose the first worst-case end-to-end delay analysis for periodic real-time flows under reliable graph routing. Specifically, we consider periodic real-time flows whose transmissions are scheduled based on fixed priority. In a *fixed priority scheduling* policy, all transmissions of a flow are scheduled based on the fixed priority of the flow. While delay analyses for single or independent routes have been proposed in the literature [3]–[6], an efficient delay analysis under reliable graph routing in a WSAN represents a challenging open problem. Since a routing graph for a flow can consist of an exponential (in number of nodes) number of routes between its source and destination, determining an effective delay bound for a flow by enumerating all of these paths is time consuming, making it unsuitable for WSANs subject to frequent changes to link and channel conditions in industrial environments. We address this challenge and propose an efficient end-to-end delay analysis *without* enumerating all the paths.

WirelessHART networks employ a multi-channel TDMA (Time Division Multiple Access) protocol. In a WirelessHART network, a flow may be delayed by higher priority flows due to (1) *channel contention* (when all channels are assigned to the higher priority flows in a time slot) or (2) *transmission conflicts* (when a transmission of the lower priority flow and a transmission of a higher priority flow involve a common node). We use an efficient method based on depth-first search to determine an upper bound of transmission conflict delay of each flow without enumerating all the paths. We then observe that, unlike single or independent routes, transmission conflict may increase channel contention in graph routing. Through an analysis of the worst-case scenario for channel contention in the presence of transmission conflict, we determine the worst-case end-to-end delay bounds of the flows. Moreover, we propose a probabilistic end-to-end delay analysis that provides tighter but probabilistic delay bounds for soft real-time flows that do not require absolute delay guarantees.

We have evaluated our schedulability analysis against experiments on a WirelessHART stack implemented on a 69-node wireless testbed. We have also performed trace-driven simulations on real network topologies. Both experiments and simulations show that our delay bounds are safe in practice and the probabilistic delay bounds represent safe upper bounds with probability ≥ 0.90. The worst-case and probabilistic bounds can be used in different application scenarios depending on the level of predictability required. Our analysis hence can be used for effective schedulability test and admission control of real-time flows under reliable graph routing.

Section II reviews related work. Section III describes the system model and graph routing mechanisms. Section IV provides an overview of fixed priority scheduling for real-time flows in a WSAN under graph routing. Section V presents the delay analysis under reliable graph routing. Section VI presents the probabilistic delay analysis. Section VII presents the experimental results. Section VIII concludes the paper.

II. Related Work

Real-time scheduling for wireless networks has been explored in many early [7] and recent works [8]–[17]. However, these works do not focus on efficient worst-case delay analysis in the network. Other works [4]–[6], [17], [18] have researched delay analysis in wireless sensor networks. These works focus on data collection through a routing tree [4], [5] and/or do not consider multiple channels [5], [6]. In contrast, we consider a WSAN based on multiple channels and reliable graph routing of WirelessHART. Besides, our analysis is targeted for real-time flows between sensors and actuators for process control purposes, and is not limited to data collection towards a sink.

Real-time scheduling for WSANs based on WirelessHART has received considerable attention in recent works [1], [3], [19]–[23]. The works presented in [22] and [23] address graph routing algorithm and localization, respectively, in WirelessHART networks. None of these concerns delay analysis. Our earlier work proposed delay analysis [3], [19]. As a first step in establishing a delay analysis for WSANs, this earlier effort is based on single-route routing instead of reliable graph routing, which are important for reliable communication in process control applications. We have also studied priority assignment policies in [20] and rate selection algorithms in [21] for real-time flows. Our work in [1], [24] also considered dynamic priority scheduling. However, *none of our earlier work considers delay analysis under reliable graph routing*.

This paper presents the first delay analysis for WSANs under reliable graph routing. Since industrial applications impose stringent requirements on *both* real-time performance and reliability in hash environments with frequent transmission failures, the delay analysis represents an important contribution to real-time scheduling for real-world WSANs. Efficient delay analysis is particularly useful for online admission control and adaptation (e.g., when network route, topology, or channel condition change) so that the network manager can quickly reassess the schedulability of the flows.

III. System Model

A. Network Model

Because of the world-wide adoption of WirelessHART in process monitoring and control, we consider a WSAN based on the WirelessHART standard [2]. WirelessHART forms a multi-hop mesh network consisting of a Gateway, a set of field devices, and several access points. A centralized network manager and the controllers are connected to the *Gateway*. The *network manager* is responsible for managing the entire network such as routing and transmission scheduling. The *field devices* are wirelessly networked sensors and actuators. *Access points* are wired to the Gateway to provide redundant paths between the wireless network and the Gateway. The sensor devices periodically deliver sample data to the controllers (through the access points wired to the Gateway), and the control messages are then delivered to the actuators. The network manager creates the routes and schedules of transmissions.

To achieve high reliability, WirelessHART employs a number of mechanisms to handle transmission failures. Transmissions are scheduled based on a multi-channel TDMA protocol. Each time slot is of fixed length (10 ms), and each transmission needs one time slot. A transmission and its acknowledgement (ACK) are scheduled in the same slot using the same channel. For transmission between a receiver and its sender, a time slot can be either dedicated or shared for the link between the sender and the receiver, and the link is called a *dedicated link* or a *shared link*, respectively. In a time slot, when a link is used as a *dedicated link*, only one sender is allowed to transmit to the receiver. In a time slot, a *shared link* associated with a receiver indicates that multiple senders can attempt to send to the common receiver in that slot. The network uses the channels defined in IEEE 802.15.4, and adopts *channel hopping* in every time slot. Any excessively noisy channel is *blacklisted* not to be used. Each receiver uses a *distinct channel* for reception in any time slot. As a result, there are at most m successful transmissions in a time slot, where m is the total number of channels. This design decision prevents potential interference between concurrent transmissions in a dedicated slot and trades network throughput for a higher degree of predictability and reliability that is essential for industrial applications.

WirelessHART supports two types of routing approaches: source routing and graph routing. *Source routing* provides a single route for each flow. The delay analysis for source routing has been addressed in the literature [3]. The focus of this paper is to develop new delay analysis for graph routing that achieves a higher degree of reliability by providing multiple paths for each flow. In graph routing, a *routing graph* is a directed list of paths that connect two devices. Packets from all sensor nodes are routed to the Gateway through the *uplink graph*. For every actuator, there is a *downlink graph* from the Gateway through which the Gateway delivers control messages. The end-to-end communication between a source (sensor) and destination (actuator) pair happens in two phases. In the *sensing phase*, on one path from the source to the Gateway in the uplink graph, the scheduler allocates a dedicated slot for each device starting from the source, followed by allocating a second dedicated slot on the same path to handle a retransmission. The links on this path are dedicated links. Then, to offset failure of both transmissions along a primary link, the scheduler allocates a third shared slot on a separate path to handle another retry. The links on these paths are shared links. Then, in the *control phase*, using the same way, the dedicated links and shared links are scheduled in the downlink graph of the destination.

Each node is equipped with a *half-duplex* omnidirectional radio transceiver that cannot both transmit and receive at the same time and can receive from at most one sender at a time. Two or more transmissions that involve a common node are *conflicting*, and cannot be scheduled in the same dedicated slot. However, for the case of shared slot, the transmissions having the same receiver can be scheduled in the same slot. The senders that attempt to transmit in a shared slot contend for the channel using a CSMA/CA scheme.

B. Flow Model

A periodic end-to-end communication between a source (sensor) and a destination (actuator) is called a *flow*. We consider there are n real-time flows denoted by $F_1, F_2, \cdots, F_n$ in the network. The *source* and the *destination* of flow F_i are denoted by s_i and d_i, respectively. The subgraph of the uplink graph that s_i uses to deliver sensor data to the Gateway is denoted by UG_i. The downlink graph for d_i is denoted by DG_i. The graph consisting of UG_i and DG_i is the *routing graph* of F_i, and is denoted by G_i. The period and the deadline of flow F_i are denoted by T_i and D_i, respectively. Time slots are used as time units. We assume $D_i \le T_i, \forall i$.

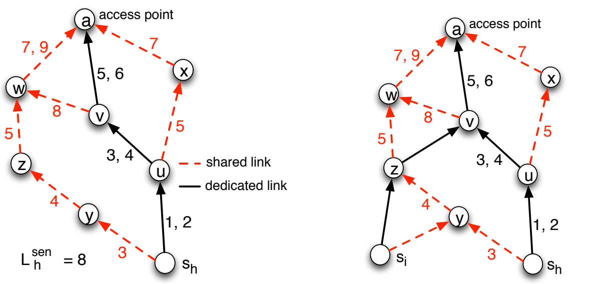
Each flow F_i, $1 \le i \le n$, has a fixed priority. Transmissions of a flow are scheduled based on its priority. In practice, flows may be prioritized based on deadlines, rates, or criticality. We assume that the priorities are already assigned using any algorithm, and that $F_1, F_2, \cdots, F_n$ are ordered by priorities. Flow F_h has higher priority than flow F_i if and only if $h < i$.

IV. Fixed Priority Scheduling

In this section, we provide an overview of the fixed priority transmission scheduling algorithm under reliable graph routing for which our delay analysis is developed. Due to its simplicity, fixed-priority scheduling is a commonly adopted policy in practice for real-time CPU scheduling, Control-Area Networks, and also for WirelessHART networks. In a *fixed priority scheduling policy*, each flow has a fixed priority, and its transmissions are scheduled based on this priority. The schedule is created by resolving the transmission conflicts and considering the limited number of channels. The complete schedule is split into superframes. A *superframe* is a series of time slots that repeat at a constant rate and represents the communication pattern of a set of flows.

We first describe how transmissions are scheduled using graph routing to account for failures. Figure 1(a) shows UG_h (the subgraph of the uplink graph used by F_h) for flow F_h. In the figure, the dedicated links used by F_h in the sensing phase are shown in solid lines while the dotted lines indicate the shared links used by F_h. Considering that F_h is not delayed by any other flow, the time slots in which a link is activated are shown beside the links (starting from slot 1). The first (starting from the source node s_h) dedicated link $s_h \rightarrow u$ is scheduled first at slot 1. Then to handle the transmission failure of slot 1, time slot 2 is also allocated for this link. Then, the next dedicated link $u \rightarrow v$ is allocated time slots 3 and 4. Similarly, the next dedicated link $v \rightarrow a$ is allocated time slots 5 and 6. Thus, if the first transmission (scheduled on slot 5) along $v \rightarrow a$ succeeds (given at least one transmission along $s_h \rightarrow u$ and at least one transmission along $u \rightarrow v$ succeeded), then the packet will reach the access point a in 5 time slots. If the first transmission (scheduled on slot 5) along $v \rightarrow a$ fails but the second one (scheduled on slot 6) along that link succeeds, then the packet will reach the access point a in 6 time slots. For every link starting from the source, to handle failure of both transmissions along the link, the scheduler again allocates a third shared slot on a separate path to handle another retry. There can be situations when the second transmission on a dedicated link, say $s_h \rightarrow u$, succeeds but the ACK gets lost. As a result, s_h retransmits the packet along the shared link $s_h \rightarrow y$ on the third slot (as s_h is unaware of the successful transmission on the dedicated link) while the packet at u is transmitted through the subsequent links. Thus a packet can be duplicated and delivered through multiple routes. We call this problem *ACK-lost problem*. To handle ACK-lost problem, we avoid conflicts among the duplicated packets while scheduling on a routing graph, except the case that transmissions along the shared links having the same receiver are allowed to schedule in the same slot. Thus, the links on paths $s_h \rightarrow y \rightarrow z \rightarrow w \rightarrow a$ are scheduled on slots 3, 4, 5, and 7. Then the links on path $u \rightarrow x \rightarrow a$ are scheduled on slots 5 and 7. Then the links on path $v \rightarrow w \rightarrow a$ are scheduled on slots 8 and 9. Thus the packet can take at most 9 slots to reach the access point (along $s_h \rightarrow u \rightarrow v \rightarrow w \rightarrow a$).

Under fixed priority scheduling, 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 current priority flow F_i, the network manager schedules its dedicated links and shared links on UG_i in its sensing phase on earliest available time slots and on available channels. It then schedules the dedicated links and shared links on DG_i in the control phase following the same way. A time slot is *available* if no



(a) UG_h for F_h (b) UG_i and UG_h

Fig. 1. Routing in the sensing phase of F_i and F_h (the numbers beside each link indicate the time slots allocated to the link.)

conflicting transmission is already scheduled in that slot except the case that transmissions along the shared links having the same receiver are allowed to schedule in the same slot. Thus a packet is scheduled on multiple paths along the routing graph. When there is no ACK-lost problem, a packet is delivered through one path in the routing graph. Otherwise, a packet can be duplicated and thus delivered through multiple paths.

Note that we do not propose any new algorithm for real-time transmission scheduling or any new fault tolerance mechanism for WirelessHART networks. Instead, the key contribution of our work is an efficient analysis for deriving the worst case delay bounds in a WSAN under graph routing, which is applicable for any existing fixed-priority scheduling policy for real-time flows in WSANs. The delay bound provided by our analysis is applicable only to the packets that are successfully delivered to the destination using existing graph routing mechanisms in WirelessHART.

V. Delay Analysis under Reliable Graph Routing

We first formulate the problem of worst-case delay analysis for real-time flows in a WSAN. We then present the delay analysis for any given fixed priority scheduling policy.

A. Problem Formulation

For each flow F_i, the sensor (s_i) periodically generates data at a period of T_i which has to be delivered to the Gateway (through an access point) in the sensing phase, and then the control message has to be delivered to the actuator (d_i) in the control phase. The total communication delay in two phases is called an *end-to-end delay* of F_i. The flows are called *schedulable* under a given fixed priority scheduling algorithm $\mathbb{A}$, if $\mathbb{A}$ is able to schedule the transmissions where no deadline will be missed. A schedulability test $\mathbb{S}$ is *sufficient* if any set of flows deemed schedulable by $\mathbb{S}$ is indeed schedulable. To determine the schedulability of a set of flows, it is sufficient to show that, for every flow, an upper bound of its worst case end-to-end delay is no greater than its deadline. Our objective is to determine an upper bound R_i of the end-to-end delay of each flow F_i. The end-to-end delay analysis will determine the flows to be *schedulable* if $R_i \le D_i, \quad \forall i$. Note that we derive upper bounds of communication delays in the network, and do not consider the time needed by the controller.

Note that creating a complete schedule for all flows requires an exponential time since the schedule has to be created up to the hyper-period of the flows. Specially, when the periods are not harmonic, the hyper-period can be extremely long making it computationally very expensive to determine the schedulability and the delays by creating a complete schedule. In contrast, the purpose of our analysis is to determine the schedulability of the flows very quickly in pseudo polynomial time *without* the need to create a complete schedule. Efficient delay analysis is particularly useful for online admission control and adaptation (e.g., when network route, topology, or channel condition change) so that the network manager can quickly reassess the schedulability of the flows and adjust workload or flow parameters (e.g., rates, priorities) accordingly. It hence is highly desirable in process monitoring and control applications that require real-time guarantees since various network dynamics affect the schedulability of the flows frequently requiring to reassess their schedulability.

In transmission scheduling, a lower priority flow may be delayed by higher priority flows due to (a) *transmission conflicts* (when a transmission of the lower priority flow and a transmission of a higher priority flow involve a common node) and (b) *channel contention* (when all channels are assigned to the transmissions of higher priority flows in a time slot). For each case, we first separately analyze how reliable graph routing in WSANs affect it. We then incorporate each component of the delays into one analysis that provides an upper bound of a flow's end-to-end delay under graph routing.

B. Transmission Conflict Delay under Graph Routing

First we analyze the delay that a flow can experience due to transmission conflicts only under graph routing. Whenever two transmissions conflict, the one that belongs to the lower priority flow needs to be delayed. The term 'delay' used in this subsection will refer to 'only transmission conflict delay'.

First we determine the conflict delay that one higher priority flow F_h may cause on a lower priority flow F_i. Under multi-path graph routing, a transmission of F_h along a link ℓ_h and a transmission of F_i along a link ℓ_i may be conflicting in 4 ways as follows when these two links involve a common node:

1) *Type 1:* ℓ_h is a dedicated link and ℓ_i is a shared or dedicated link.
2) *Type 2:* ℓ_h is a shared link and ℓ_i is a dedicated link.
3) *Type 3:* ℓ_h is a shared link and ℓ_i is a shared link, and the receiver nodes of the two links are different.
4) *Type 4:* ℓ_h is a shared link and ℓ_i is a shared link, and the receiver nodes of the two links are the same. In this case, the transmission of F_i is not delayed.

In the first 3 cases the transmission of F_i is delayed while for Type 4 conflict it will not be delayed. Therefore, the total delay caused by F_h on F_i depends on how their dedicated and shared links intersect in the routing graphs. Now we will first determine an upperbound of the conflict delay that one instance of a higher priority flow F_h may cause on F_i. In the next discussion we limit our attention only to F_h and F_i. Note that we measure delay in terms of number of slots.

In the routing graph G_i (consisting of UG_i and DG_i) of flow F_i which involves N_i nodes, there can be $O(N_i^2)$ directed end-to-end paths from its source s_i to destination d_i (calculated as the number of paths in UG$_i$ in the sensing phase times the number of paths in DG$_i$ in the control phase). If every node in the routing graph has to make the first two tries along a dedicated link and then to make a third retransmission along a shared link, then this number of paths can be 2^{N_i}. Thus if t denotes the time complexity (which is pseudo polynomial) for determining delay for a single path route, the complexity for a flow F_i becomes $O(t*2^{N_i})$. On the other hand, our method has time complexity of $O(t)$ flow F_i (as will be shown later). Among these end-to-end paths, the one that experiences the maximum conflict delay from F_h is called the *bottleneck path* with respect to F_h. The conflict delay caused by F_h along F_i's bottleneck path represents the upper bound of the conflict delay that F_h may cause on F_i. Let Δ_i^h be an *upper bound of conflict delay* that one instance of F_h may cause along the bottleneck path of F_i. We determine Δ_i^h in an efficient way *without requiring* to find the bottleneck path or without enumerating all end-to-end paths in G_i as described below.

Algorithm 1: Finding conflict delay on F_i caused by F_h

```
Procedure FindConflict($UG_i$, $r$)        /* $r$ is a node in UG$_i$ */
  for each node $u$ in $UG_i$ do
    status($u$):=$undiscovered$; $\lambda_i^h(u) := 0$;
  end
  DFSearch($r$);                        /* start search at node $r$ */
  return $\lambda_i^h(r)$;                  /* $\lambda_i^h$ in subtree rooted at $r$ */
end Procedure

Procedure DFSearch($r$)
  status($r$):=$discovered$;      /* node $r$ is now discovered */
  for each $v \in children(r, UG_i)$ do
    if status($v$)=undiscovered; then DFSearch($v$);
  end
  $\lambda_i^h(r) := \max\{\lambda_i^h(v) | v \in children(r, UG_i)\}$;
  $x(r)$ := new conflict delay on $F_i$ by $F_h$ observed at node $r$;
  $\lambda_i^h(r) := \lambda_i^h(r) + x(r)$;
end Procedure
```

Let us call the bottleneck path (with respect to F_h) in UG$_i$ the *bottleneck sensing path* of F_i. Let an upper bound of conflict delay caused by F_h on F_i's bottleneck **sen**sing path be $\lambda_i^{h,\text{sen}}$. A value of $\lambda_i^{h,\text{sen}}$ can be efficiently calculated without enumerating all paths in UG$_i$ as explained below. Let us consider a particular path p in UG$_i$. The total number of transmissions of (one instance of) F_h that may have Type 1, 2, or 3 conflict on p represents a value of conflict delay along p caused by one instance of F_h. To illustrate this, in Figure 1(b), with flow F_h, we also show UG$_i$ for flow F_i. The figure shows links $s_i \to z$, $z \to v$, and $v \to a$ as dedicated links in UG$_i$ while the corresponding shared links are $s_i \to y$, $z \to w$, and $v \to w$, respectively. In Figure 1(b), F_h has 9 transmissions that may cause delay along $p = s_i \to z \to w \to a$ of F_i. (Note that this is the delay along p considering links $s_i \to z$, $z \to v$, $z \to w$, and $w \to a$ of F_i. Link $z \to v$ is considered because $z \to w$ is scheduled only after scheduling $z \to v$.) Now the path in UG$_i$ whose delay (calculated using the above method) is maximum is the bottleneck sensing path, and its delay represents $\lambda_i^{h,\text{sen}}$. Such a value of $\lambda_i^{h,\text{sen}}$ is determined quickly by exploring each link on UG$_i$ once based on a depth-first search on UG$_i$. The method is shown as Algorithm 1, and $\lambda_i^{h,\text{sen}}$ is determined by calling

$$\lambda_i^{h,\text{sen}} = \text{FindConflict(UG}_i,\ s_i);$$

In Algorithm 1, we use $children(u, UG_i)$ to denote the set of nodes to which node u transmits in UG_i for flow F_i. (For example, in Figure 1(a), node s_h has children u and y.) The search starts at node s_i. In this method, when the search backtracks at a node u, we use $\lambda_i^h(u)$ to denote the maximum conflict delay along a path among all the paths in the subtree (induced by depth first search) rooted at u. The value of $\lambda_i^h(u)$ is calculated by taking the maximum of the values from u's children and then by adding the new conflict delay that we observe at node u, when the search finishes node u. Note that we do not need to execute Algorithm 1 for every distinct F_h to determine $\lambda_i^{h,\text{sen}}$. Instead, we need to execute Algorithm 1 only once for all $h < i$ to determine $\lambda_i^{h,\text{sen}}$ for flow F_i, making our approach highly efficient.

Similarly, let $\lambda_i^{h,\text{con}}$ be the conflict delay along the bottleneck **con**trol path. The value of $\lambda_i^{h,\text{con}}$ is determined using Algorithm 1 on DG$_i$ starting the search at an access point a, i.e., by calling

$$\lambda_i^{h,\text{con}} = \text{FindConflict(DG}_i,\ a);$$

Thus, our method has time complexity of $O(|G_i| + t) = O(t)$ (where t denotes the time complexity for determining delay for a single path route and is pseudo polynomial) for a flow F_i where $O(|G_i|)$ is the time complexity of Algorithm 1 (considering its execution on both UG$_i$ and DG$_i$), with $|G_i|$ indicating the total number of links and nodes in G_i.

Lemma 1 provides a bound of Δ_i^h.

Lemma 1: For a higher priority flow F_h and a lower priority flow F_i, $\Delta_i^h \leq \lambda_i^{h,\text{sen}} + \lambda_i^{h,\text{con}}$.

Proof: Since the control phase of F_i starts after its sensing phase is complete, the bottleneck path between s_i and d_i consists of its bottleneck sensing path and the bottleneck control path. Hence, $\lambda_i^{h,\text{sen}} + \lambda_i^{h,\text{con}}$ is an upper bound of conflict delay caused by one instance of F_h along F_i's bottleneck path. ■

Note that Δ_i^h is an upper bound of delay that one instance of F_h can cause along F_i's bottleneck path. Now we will upperbound the total delay caused by all instances of F_h. In considering the delay caused by multiple instances, we observe that at the time when a transmission on a directed path p in G_i conflicts with some transmission of F_h, the preceding transmissions on p are already scheduled. These already scheduled transmissions on p are no more subject to delay by the subsequent instances of F_h. For example, in Figure 1(b) let us consider the path $s_i \to y \to z \to v \to w \to a$ in UG_i of F_i. If some instance of F_h conflicts and causes delay on F_i's transmission along $v \to w$, the next instance of F_h must not delay F_i's transmissions along links $s_i \to y$, $y \to z$, and $z \to v$ on this path since these are already scheduled. Thus only the transmissions that are not yet scheduled along path p will be considered for conflict delay by the subsequent instances of F_h. These observations lead to Lemma 2, and then to Theorem 3 to upperbound the total delay (due to transmission conflict) caused on F_i by all instances of F_h.

Lemma 2: Let us consider any two instances of a higher priority flow F_h such that each causes conflict delay on a directed path p in G_i of a lower priority flow F_i in a time interval. Then, there is at most one common transmission on p that can be delayed by both instances.

Proof: Let these two instances of F_h be denoted by $F_{h,1}$ and $F_{h,2}$, where $F_{h,1}$ is released before $F_{h,2}$. Suppose to the contrary, both of these instances cause delay on two transmissions, say τ_j and τ_r, on directed path p of F_i. Without loss of generality, we assume that τ_j precedes τ_r on p. $F_{h,1}$ causes delay on τ_r because τ_r is ready to be scheduled. This implies that τ_j has already been scheduled. Hence, $F_{h,2}$ which releases after $F_{h,1}$ cannot cause any delay on τ_j, thereby contradicting our assumption. ■

By Lemma 2, for any two instances of F_h, any directed path in G_i of F_i has at most one transmission on which both instances can cause delay. Let the link on G_i that may have maximum conflict delay of Type 1, 2, or 3 with F_h be called the *bottleneck link* of F_i (with respect to F_h). That is, a transmission of F_i along this link may face the highest conflict with F_h. Let δ_i^h denote the *maximum conflict delay* along the bottleneck link. (For example, considering only UG$_i$ in Figure 1(b), we can see that $\delta_i^h = 7$, since a link of F_i can have conflict with at most 7 transmissions of F_h. Here, $z \to v$ is F_i's bottleneck link.) In the worst case, the transmission along the bottleneck link of F_i (with respect to F_h) can be delayed by multiple instances of F_h. Hence, the value of δ_i^h plays a major role in determining the worst case delay caused by F_h on F_i as shown in Theorem 3.

Theorem 3: In a time interval of t slots, the worst case conflict delay caused by a higher priority flow F_h on a lower priority flow F_i is upper bounded by

$$\Delta_i^h + \Big(\Big\lfloor \frac{t}{T_h} \Big\rfloor - 1\Big).\delta_i^h + \min\Big(\delta_i^h, t \bmod T_h\Big)$$

Proof: There are at most $\lceil \frac{t}{T_h} \rceil$ instances of F_h in a time interval of t slots. We consider a particular directed path p in G_i of F_i. Let the set of transmissions of F_h which cause conflict delay along p be denoted by Γ. When one instance $F_{h,1}$ of F_h causes conflict delay on p, a subset Γ_1 of Γ causes delay on p. Now consider a second instance $F_{h,2}$ of F_h. For $F_{h,2}$, another subset Γ_2 of Γ causes delay on p. When all subsets $\Gamma_1, \Gamma_2, \cdots, \Gamma_{\lceil \frac{t}{T_h} \rceil}$ are mutually disjoint, by the definition of Δ_i^h, the conflict delay caused by Γ on p is at most Δ_i^h. Hence, the total conflict delay caused by all $\Gamma_1, \Gamma_2, \cdots, \Gamma_{\lceil \frac{t}{T_h} \rceil}$ in this case is at most Δ_i^h. That is, the total conflict delay on p caused by F_h is at most Δ_i^h.

Now let us consider the case when the subsets $\Gamma_1, \Gamma_2, \cdots, \Gamma_{\lceil \frac{t}{T_h} \rceil}$ are not mutually disjoint, i.e., there is at least one pair Γ_j, Γ_k such that $\Gamma_j \cap \Gamma_k \neq \emptyset$, where $1 \leq j, k \leq \lceil \frac{t}{T_h} \rceil$. Let the total delay caused by all instances of F_h on p in such case be $\Delta_i^h + Z_i^h$, i.e., the delay is higher than Δ_i^h by Z_i^h time slots. The additional delay (beyond Δ_i^h) happens because the transmissions that are common between Γ_i and Γ_j cause both instances of F_h to create delay along p. By Lemma 2, for any two instances of F_h, p has at most one transmission on which both instances can cause delay. If there is no transmission of p that is delayed by both the k-th instance and the $(k{+}1)$-th instance of F_h, then no transmission of p is delayed by both the k-th instance and the q-th instance of F_h, for any $q > (k + 1)$, where $1 \leq k < \lceil \frac{t}{T_h} \rceil$. Thus, Z_i^h is maximum when for each pair of consecutive instances (say, the k-th instance and $k + 1$-th instance, for each k, $1 \leq k < \lceil \frac{t}{T_h} \rceil$) of F_h, there is a transmission of p that is delayed by both instances. Hence, at most $\lceil \frac{t}{T_h} \rceil - 1$ instances contribute to this additional delay Z_i^h, each instance causing some additional delay on a transmission. Since one instance of F_h can cause delay on a transmission of p at most by δ_i^h slots, $Z_i^h \leq (\lceil \frac{t}{T_h} \rceil - 1)\delta_i^h$. Since the last instance may finish after the considered time window of t slots, the delay caused by it is at most $\min(\delta_i^h, t \bmod T_h)$ slots. Taking this into consideration, $Z_i^h \leq (\lfloor \frac{t}{T_h} \rfloor - 1)\delta_i^h + \min(\delta_i^h, t \bmod T_h)$. Thus, the total delay caused on p by all instances of F_h is at most

$$\Delta_i^h + Z_i^h \leq \Delta_i^h + (\Big\lfloor \frac{t}{T_h} \Big\rfloor - 1).\delta_i^h + \min(\delta_i^h, t \bmod T_h)$$

Since the above bound is true for any path in G_i (of F_i), it is true for the bottleneck path in G_i. Since the conflict delay along the bottleneck path represents the conflict delay caused on F_i by F_h, the theorem follows. ■

From Theorem 3, now an upper bound of the total delay that flow F_i can experience from all higher priority flows due to transmission conflicts during a time interval of t slots is calculated as follows.

$$\sum_{h<i} \Bigg(\Delta_i^h + \Big(\Big\lfloor \frac{t}{T_h} \Big\rfloor - 1\Big).\delta_i^h + \min\Big(\delta_i^h, t \bmod T_h\Big)\Bigg) \quad (1)$$

C. Channel Contention Delay under Graph Routing

In this section, we analyze the channel contention delay caused by one higher priority flow F_h to a lower priority one F_i under reliable graph routing. First we analyze the delay without considering channel hopping. Later, we will analyze the effect of channel hopping.

Let E_h and S_h denote the total number of dedicated links and total number of shared links of flow F_h, respectively. Since every dedicated link is scheduled on 2 dedicated slots, there are $2E_h + S_h$ assignments of channels for flow F_h.

Note that a packet is scheduled on multiple paths in its routing graph for fault tolerance. While a natural approach to analyzing channel contention delay of a flow under this scenario is to consider it as a parallel task, we observe that the scheduling on routing graphs experiences only a little parallelism making it more closer to sequential task scheduling due to the following two problems.

ACK-lost problem. Assuming no packet duplication, we could schedule the link $w \to a$ for delivery through paths $s_h \to y \to z \to w \to a$ on slot 6, ignoring the fact that link $v \to a$ is already scheduled on slot 6 because the packet will be delivered through one path only (Figure 1(a)). But, in presence of ACK-lost problem, to avoid conflict among the duplicate packets (of the same packet), we cannot schedule link $w \to a$ on slot 6. Thus $v \to a$ and $w \to a$ are scheduled sequentially, on slot 6 and slot 7, respectively.

Impact of transmission conflict on channel contention delay. Channel contention delay and transmission conflict delay are often correlated. Specifically, channel contention delay can increase when a flow experiences transmission conflict delay. Let us consider links $z \to w$ and $u \to x$ (in G_h) that can be scheduled on slot 5 when there are no other higher priority flow (Figure 1). In the presence of higher priority flows, if any of transmissions $z \to w$ and $u \to x$ in F_h is delayed, for example by 1 slot, due to transmission conflict with a higher priority flow, while the other can happen at slot 5, then these two transmissions have to be scheduled sequentially (instead of scheduling in parallel). Therefore, even though scheduling of F_h has some parallelism, in the worst case in presence of transmission conflict, it can cause channel contention delay on its lower priority flows like a flow that happens like a sequential task with execution requirements of $2E_h + S_h$ slots.

Based on the above observations, the analysis for upper bounding the channel contention delay reduces to that for a set of flows where each flow F_i has the worst-case time requirement of e_i slots through a single path route, where $e_i = 2E_i + S_i$. Hence, we leverage our result in [3] whose analysis was given for flows with single-path routes to find the channel contention delay caused by F_h on F_i. Using that result, in any time interval of x slots, there are at most $m-1$ higher priority flows each flow F_h among which can cause at most $I_i^h(x)$ delay on F_i as expressed below

$$I_i^h(x,e_i) = \min\Bigg(x - e_i + 1,\ \left\lfloor \frac{x-e_h}{T_h} \right\rfloor e_h + e_h + \min\Big(e_h - 1, \max\big((x-e_h) \bmod T_h - (T_h - R_h), 0\big)\Big)\Bigg)$$

where R_h is the worst-case end-to-end delay of F_h. The delay caused by each other higher priority flow F_h on F_i is at most

$$J_i^h(x,e_i) = \min\Big(x - e_i + 1,\ \left\lfloor \frac{x}{T_h} \right\rfloor e_h + \min\big(x \bmod T_h,\ e_h\big)\Big)$$

Thus, considering a total of m channels, an upper bound $\Omega_i(x)$ of the channel contention delay caused by all higher priority flows on F_i in any time interval of x slots is derived as follows.

$$\Omega_i(x,e_i) = \left\lfloor \frac{1}{m}\Big(Z_i(x,e_i) + \sum_{h<i} J_i^h(x,e_i)\Big) \right\rfloor \tag{2}$$

with $Z_i(x,e_i)$ being the sum of the $\min(i-1, m-1)$ largest values of the differences $I_i^h(x,e_i) - J_i^h(x,e_i)$ among the higher priority flows $F_h,\ h < i$.

Effect of Channel Hopping. To every transmission, the scheduler assigns a *channel offset* between 0 and $m-1$ instead of an actual channel, where m is the total number of channels. All devices in the network maintain an identical list of available channels. At any time slot t, a channel offset c (i.e., $1, 2, \cdots, m-1$) maps to a channel that is different from the channel used in slot $t-1$ as follows.

$$\text{channel} = (c + t) \bmod m \tag{3}$$

Both the sender and the receiver of the corresponding transmission link switches to the new channel. As can be seen from Equation 3, at every time slot any 2 different channel offsets always map to 2 different channels. The scheduler assigns at most one channel offset to a link at any time which maps to different physical channels in different time slots, keeping the total number of available channels at m always, and scheduling each link on at most one channel at any time. Hence, channel hopping does not have effect on channel contention delay.

D. End-to-End Delay Bound

Now both types of delays are incorporated together to develop an upper bound of the end-to-end delay of every flow. This is done for every flow in decreasing order of priority starting with the highest priority flow. Theorem 4 provides an upper bound R_i of end-to-end delay for every flow F_i.

Considering no delay from higher priority flows, let the *worst-case time requirement* of F_h in the **sen**sing phase be denoted by L_h^{sen}. For example, in Figure 1(a), $L_h^{\text{sen}} = 9$ slots (as described in Section IV). A similar scheduling is followed in the control phase also. Similarly, considering no delay from higher priority flows, let the *worst-case time requirement* of F_h in the **con**trol phase be denoted by L_h^{con}. Thus, considering no delay from higher priority flows, the *time requirement through a critical path* denoted by L_i, of flow F_i is

$$L_i = L_i^{\text{sen}} + L_i^{\text{con}} \tag{4}$$

Theorem 4: Let x_i^* be the minimum value of $x \geq L_i$ that solves Equation 5 using a fixed-point algorithm.

$$x = \Omega_i(x,e_i) + L_i \tag{5}$$

Then the end-to-end delay bound R_i of flow F_i is the minimum value of $t \geq x_i^*$ that solves Equation 6 using a fixed-point algorithm.

$$t = x_i^* + \sum_{h<i}\Big(\Delta_i^h + \Big(\left\lfloor \frac{t}{T_h} \right\rfloor - 1\Big).\delta_i^h + \min\Big(\delta_i^h, t \bmod T_h\Big)\Big) \tag{6}$$

Proof: According to Equation 2, x_i^* is calculated considering R_h (i.e., the end-to-end delay bound of F_h considering both channel contention delay and conflict delay) of each higher priority flow F_h. According to Equation 2, $\Omega_i(x, L_i)$ is the channel contention delay caused by all higher priority flows on F_i in any time interval of x slots. Hence x_i^* is the bound of the end-to-end delay of F_i when it suffers only from channel contention delay caused by higher priority flows (and no conflict delay). Equation 1 provides the bound of transmission conflict delay of F_i. Hence, adding this value to x_i^* must be an upper bound of F_i's end-to-end delay under both channel contention and transmission conflict. ■

Thus we can determine R_i for every flow F_i in decreasing order of priority starting with the highest priority flow using Theorem 4. In solving Equations 5 and 6, if x or t exceeds D_i, then F_i is decided to be "unschedulable".

Time complexity. Since both x in Equation 5 and t in Equation 6 can reach a value of at most D_i using a fixed point algorithm, the worst-case time required to determine R_i can be $O(D_i)$. Given n flows with a maximum deadline of $D_{\max}$, the required time for the delay analysis of all the flows is $n*O(D_{\max})$ which implies a pseudo polynomial time complexity. Note that this time is usually much less than that required for computing all superframes, as the latter would take the least common multiple of all periods which can be extremely large when the periods are non-harmonic. Moreover, as discussed in Subsection V-B, our delay analysis does not require the enumeration of all the possible paths through a routing graph. Therefore, the proposed analysis is an efficient approach for determining the schedulability of real-time flows in process monitoring and control applications.

VI. A Probabilistic End-to-End Delay Analysis

Graph routing provides a very conservative approach to scheduling transmissions in a WirelessHART network. In the scheduling used in the previous sections, there is a synchronization at the access points in the sense that the scheduling in the downlink graph of a flow (the control phase) is started after all links in its uplink subgraph are scheduled. However, there is high probability that a packet will be delivered through the dedicated path only because each link on the path is dedicated and scheduled twice. Therefore, whenever the gateway receives a sensor packet through the dedicated link, the corresponding control message can be calculated and delivered through the downlink graph's dedicated link in the next available slot avoiding synchronization at the access points. The corresponding retry on the shared slot can be scheduled only after all links on the uplink subgraph of the flow are scheduled. The advantage of such a scheduling policy is that the actual end-to-end delay in most cases will be substantially shorter since a packet follows the dedicated links in most cases. Under this scheduling, we can determine a probabilistic delay bound that is tighter than the bound derived in the last section but represents a bound with high probability. These bounds rely on statistical independence among the links.

Considering the dedicated route has E_i links, and p_k as the probability of a successful transmission along link k, the probability of being successful upon 2 transmissions through link k is $1-(1-p_k)^2$. Therefore, the probability that a packet will be delivered through the dedicated links is

$$\prod_{k=1}^{E_i}\left(1-(1-p_k)^2\right) \tag{7}$$

Let, in G_i, the path consisting of all dedicated links be called *dedicated path*. Let ${\Delta'}_i^h$ denote the total number of transmissions of (one instance of) F_h that share a node on the dedicated path of F_i. Similarly, let ${\delta'}_i^h$ denote the maximum conflict delay caused by one instance of F_h on the bottleneck link on F_i's dedicated path (i.e., a link on F_i's dedicated path can share a node with at most ${\delta'}_i^h$ transmissions of F_h). Corollary 1 now follows from Theorem 4.

Corollary 1: Let x_i^* be the minimum value of $x \geq 2E_i$ that solves Equation 8 using a fixed-point algorithm.

$$x = \Omega_i(x, 2E_i) + 2E_i \tag{8}$$

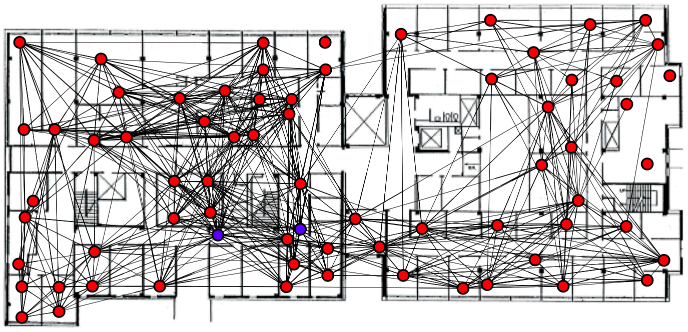


Fig. 2. Testbed topology (access points are blue colored)

Then the minimum value of $t \geq x_i^*$ that solves Equation 9 is the worst-case end-to-end delay bound of flow F_i with a probability of at least $\prod_{k=1}^{E_i}\left(1-(1-p_k)^2\right)$.

$$t = x_i^* + \sum_{h<i}\left({\Delta'}_i^h + \left(\left\lfloor \frac{t}{T_h}\right\rfloor - 1\right).{\delta'}_i^h + \min\left({\delta'}_i^h, t \bmod T_h\right)\right) \tag{9}$$

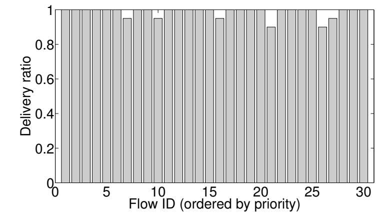
Proof: By Equation 2, $\Omega_i(x, 2E_i)$ represents the channel contention delay on the dedicated path of F_i. Following Theorem 4, the minimum value of $t \geq x_i^*$ that solves Equation 9 is the worst case delay bound for the dedicated route. The proof follows since a packet has $\prod_{k=1}^{E_i}\left(1-(1-p_k)^2\right)$ probability of being delivered through the dedicated route. ■

VII. Experiment

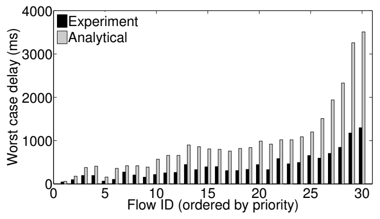
A. Testbed Experiment

1) Implementation: We evaluate our delay analysis on an indoor wireless testbed deployed in two buildings at Washington University [25]. The testbed consists of 69 TelosB motes, each equipped with Chipcon CC2420 radios compliant with the IEEE 802.15.4 standard. We implement a WirelessHART protocol stack on TinyOS 2.1.2 [26] and TelosB. Our protocol stack consists of a multi-channel TDMA MAC protocol with channel hopping and a routing layer supporting both source routing and graph routing. The uplink and downlink graphs are generated using the graph routing algorithms presented in [22]. Time is divided into 10 ms slots and clocks are synchronized across the entire network using the Flooding Time Synchronization Protocol (FTSP) [27]. The details of our protocol stack implementation are available in [28].

2) Experimental Setup: The senders scheduled in a shared slot follow CSMA/CA mechanism for transmission within that slot of the TDMA schedule. We setup the parameters of the CSMA/CA mechanism in a shared slot as follows to ensure that a node that acquires the channel has sufficient time to complete the transmission within that slot (a slot consists of 10ms). There are 2 backoff intervals and 3 backoff values: *initialBackoff*, *minimumBackoff*, and *congestionBackoff*. In the beginning of a shared slot, a node first makes an initial backoff to avoid capturing a channel from the other nodes who also are scheduled in that shared slot. The first backoff interval is random in the range [*minimumBackoff, initialBackoff*]. Once this backoff interval elapses, a node performs CCA. If the



(a) Delivery ratio



(b) Delay

Fig. 3. Delay and reliability on testbed

channel is free it transmits. If the channel is busy, a node will pick another backoff period randomly from the range [*minimumBackoff, congestionBackoff*]. After the second backoff interval elapses, it will immediately transmit if the channel is free (checked upon CCA). If the channel is sensed busy, the node immediately stops, and no further attempt is made to transmit in that slot. We set *initialBackoff* = 2240 μs, *minimumBackoff* = 320 μs, and *congestionBackoff* = 960 μs.

We use IEEE 802.15.4 channels 15, 16, 19, and 20 in our experiments. For each link in the testbed, we measured its packet reception ratio (PRR) by counting the number of received packets among 250 packets transmitted on the link. Following common practice of industrial deployment, we only consider links with PRR higher than 90% on every channel to determine the testbed topology. Figure 2 is a topology of the testbed showing the node positions on the two buildings' floor plan. We use two nodes (colored in blue in the figure) as access points, which are physically connected to a root server (Gateway). The Network Manager runs on this root server. The other motes work as field devices.

We experiment by generating 30 flows on our testbed. The period of each flow is randomly selected from the range of $[10*2^6, 10*2^{10}]$ms. The relative deadline of each flow equals to its period. All flows are schedulable based on our delay analyses. Priorities of the flows are assigned based on the Deadline Monotonic (DM) policy. DM assigns priorities to flows according to their relative deadlines; the flow with the shortest deadline being assigned the highest priority.

3) Results: We run our experiments long enough so that each superframe is run for at least 20 cycles. We evaluate our proposed approaches in terms of reliability and delay. We use delivery ratio to measure reliability. The *delivery ratio* of a flow is defined as the fraction of the packets generated by the flow that are successfully delivered to destination. Since there exists no prior work on delay analysis under reliable graph routing, we compare our analytical delay bounds with the maximum delay observed in experiments on the testbed.

Figure 3 shows our results. Figure 3(a) shows the delivery ratios of all 30 flows. There are 2 flows each with a delivery ratio of 0.90, and 4 flows each with a delivery ratio of 0.95, while every other flow has a delivery ratio of 1. This result demonstrates the effectiveness of graph routing in achieving reliable communication over WSANs. Figure 3(b) plots the maximum end-to-end delay observed in our experiments and the end-to-end delay bounds derived through our delay analysis. As the figure shows, our analytical delay bounds are no less than the experimental maximum delays, demonstrating that our delay analysis provides safe upper bounds of the actual delays. The ratio of the analytical delay bound to the maximum delay observed in experiments is at most 2.79. Note that the experiments may not have experienced the worst case scenarios and hence the maximum delays observed in experiments may not represent the actual worst case delays.

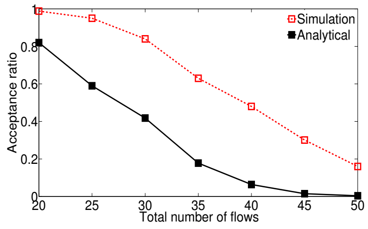


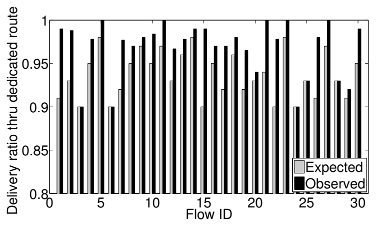
Fig. 4. Acceptance ratio based on the worst case delay analysis

B. Simulations

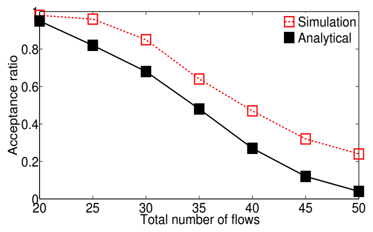
For a more extensive evaluation, we now use the same testbed topology and evaluate the results in simulations. We generate flows by randomly selecting sources and destinations, and simulate their schedules in these topologies. Two nodes in the topology are selected as access points. The uplink and downlink graphs are generated using the same graph routing algorithms as the one we used in testbed experiment. The periods of the flows are randomly generated in the range $[10 * 2^5, 10 * 2^{13}]$ms. The deadlines are equal to periods. Priorities of the flows are assigned based on the DM policy. In all cases, we use 12 channels for scheduling. We evaluate our analysis in terms of *Acceptance ratio* defined as the fraction of the total number of test cases that are deemed schedulable.

1) Worst Case Delay Analysis: Since there exists no prior work on delay analysis under reliable graph routing, we analyze the effectiveness of our analysis by simulating the complete schedule of transmissions of all flows released within the hyper-period. In all figures in this subsection, "Simulation" indicates the fraction of test cases that have no deadline misses in the simulations, and represents conservative upper bounds of acceptance ratios because we did not simulate all possible arrival patterns of the flows; "Analytical" indicates the acceptance ratio based on our delay analysis.

Figure 4 shows the acceptance ratios for 1000 test cases under varying number of flows. For 20 flows, 986 test cases are schedulable through simulations while our analysis has determined 818 cases as schedulable showing an acceptance ratio of 0.818. The ratios decrease with the increase in the number of flows. The gap between the analytical acceptance ratio and that based on simulations stems from the pessimism of our analysis which provides safe upper bounds. Next we evaluate the probabilistic delay analysis in reducing pessimism.



(a) Delivery ratio through dedicated links



(b) Acceptance ratio

Fig. 5. Performance of probabilistic delay analysis

2) Probabilistic Delay Analysis: Again the testbed topology is used in our simulations for the probabilistic delay analysis, and the measured PRR of each link is used as the probability of a successful transmission along the link.

For a test case of 30 flows, we simulate the schedule in 1000 runs where a link's failure or success is determined probabilistically based on its PRR. Figure 5(a) shows the fraction (labeled "Observed") of the packets that are delivered through dedicated routes. In the figure "Expected" indicates the expected fraction of the packets to be delivered through dedicated routes which is the analytical probability calculated based on the measured PRR using Equation 7. The results show that more than 90% of the packets are delivered through their dedicated routes, and this value is close to what the analytical probability in Equation 7 indicates. Thus the probabilistic bound delay bound corresponds to a 90 percentile delay bound. Figure 5(b) shows the acceptance ratios under the probabilistic delay bound. It shows the probabilistic analysis can accept at least 80% of the test cases that are schedulable according to simulations for up to 35 flows. The results suggest that our probabilistic analysis can effectively reduce the pessimism of analytical delay bounds, and hence represents an effective alternative for delay analysis of soft real-time flows for which probabilistic delay bounds are sufficient. For example, this analysis may be used for non-critical applications.

VIII. Conclusion

Industrial wireless sensor-actuator networks must support reliable and real-time communication in hash environments. Industrial wireless standards such as WirelessHART adopt a reliable graph routing approach to handle transmission failures through retransmissions and route diversity. These mechanisms introduce substantial challenges in analyzing the schedulability of real-time flows. We have presented the first worst-case delay analysis under reliable graph routing. We have also proposed a probabilistic delay analysis that provides delay bounds with high probability. Experiments based on a wireless testbed of 69 nodes and simulations show that our analytical delay bounds are safe, and can be used as an effective schedulability test for real-time flows under reliable graph routing.

Acknowledgment

This work was supported by NSF through grant CNS-1320921 (NeTS).

References

[1] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in *RTSS '10*.
[2] "WirelessHART," 2007, http://www.hartcomm2.org.
[3] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end delay analysis for fixed priority scheduling in WirelessHART networks," in *RTAS '11*.
[4] J. B. Schmitt and U. Roedig, "Sensor network calculus - A framework for worst case analysis," in *DCOSS '05*.
[5] O. Chipara, C. Lu, and G.-C. Roman, "Real-time query scheduling for wireless sensor networks," in *RTSS '07*.
[6] T. F. Abdelzaher, S. Prabh, and R. Kiran, "On real-time capacity limits of multihop wireless sensor networks," in *RTSS '04*.
[7] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.
[8] H. Li, P. Shenoy, and K. Ramamritham, "Scheduling messages with deadlines in multi-hop real-time sensor networks," in *RTAS '05*.
[9] X. Wang, X. Wang, X. Fu, G. Xing, and N. Jha, "Flow-based real-time communication in multichannel wireless sensor networks," in *EWSN '09*.
[10] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Distributed multi-hop scheduling and medium access with delay and throughput constraints," in *MobiCom '01*.
[11] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He, "RAP: A real-time communication architecture for large-scale wireless sensor networks," in *RTAS '02*.
[12] K. Karenos and V. Kalogeraki, "Real-time traffic management in sensor networks," in *RTSS '06*.
[13] T. He, B. M. Blum, Q. Cao, J. A. Stankovic, S. H. Son, and T. F. Abdelzaher, "Robust and timely communication over highly dynamic sensor networks," *Real-Time Syst.*, vol. 37, no. 3, 2007.
[14] T. W. Carley, M. A. Ba, R. Barua, and D. B. Stewart, "Contention-free periodic message scheduler medium access control in wireless sensor/actuator networks," in *RTSS '03*.
[15] K. Liu, N. Abu-Ghazaleh, and K.-D. Kang, "JiTS: Just-in-time scheduling for real-time sensor data dissemination," in *PERCOM '06*.
[16] Y. Gu, T. He, M. Lin, and J. Xu, "Spatiotemporal delay control for low-duty-cycle sensor networks," in *RTSS '09*.
[17] N. Pereira, B. Andersson, E. Tovar, and A. Rowe, "Static-priority scheduling over wireless networks with multiple broadcast domains," in *RTSS '07*.
[18] A. Saifullah, S. Sankar, J. Liu, C. Lu, B. Priyantha, and R. Chandra, "CapNet: A real-time wireless management network for data center power capping," in *RTSS '14*, 2014.
[19] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-end communication delay analysis in industrial wireless networks," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1361–1374, 2014.
[20] ——, "Priority assignment for real-time flows in WirelessHART networks," in *ECRTS '11*.
[21] 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*.
[22] S. Han, X. Zhu, and A. K. Mok, "Reliable and real-time communication in industrial wireless mesh networks," in *RTAS '11*.
[23] X. Zhu, P.-C. Huang, S. Han, A. Mok, D. Chen, and M. Nixon, "RoamingHART: A collaborative localization system on WirelessHART," in *RTAS '12*.
[24] C. Wu, M. Sha, D. Gunatalika, A. Saifullah, C. Lu, and Y. Chen, "Analysis of EDF scheduling for wireless sensor-actuator networks," in *IWQoS '14*.
[25] Wireless sensor network testbed, http://mobilab.wustl.edu/testbed.
[26] "TinyOS Community Forum," http://www.tinyos.net/.
[27] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04*.
[28] M. Sha, D. Gunatilaka, C. Wu, and C. Lu, "Implementation and experimentation of industrial wireless sensor-actuator network protocols," in *EWSN '15*.