Spring 5-15-2019

High Performance Wireless Sensor-Actuator Networks for Industrial Internet of Things

Dolvara Gunatilaka
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST.LOUIS

School of Engineering & Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
Chenyang Lu, Chair
Shantanu Chakrabartty
Roch Guérin
Raj Jain
Ning Zhang

High Performance Wireless Sensor-Actuator Networks for Industrial Internet of Things

by

Dolvara Gunatilaka

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

May 2019
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

First, I would like to express my sincere gratitude to my advisor Prof. Chenyang Lu for his continuous guidance and encouragement throughout my years at WashU. His advice and guidance motivate me and allow me to grow as a research scientist and as a person.

My time at WashU was made enjoyable in large part due to my friends and lab mates. I am thankful to my fellow past and present CPS lab mates who have contributed greatly to both my professional and personal time: Mo Sha, Chengjie Wu, Abusayeed Saifullah, Jing Li, Yehan Ma, just to name a few. I also offer special thanks to my friends: Jidapa Kraisangka, Thanapon Noraset, Thitivatr PatanasakPinyo, Suthatip Jullamon, and Sun Pruksacholavit.

I gratefully acknowledge the financial support from Prof. Chenyang Lu through NSF grants, and from the Faculty of Information and Communication Technology, Mahidol University.

Most importantly, I would like to thank my family. I am deeply grateful to my mom and dad for their endless love and support. Without them, I would not be the person I am today.

Dolvara Gunatilaka

Washington University in Saint Louis

May 2019

Dedicated to my parents.

ABSTRACT OF THE DISSERTATION

High Performance Wireless Sensor-Actuator Networks for Industrial Internet of Things

by

Dolvara Gunatilaka

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2019

Professor Chenyang Lu, Chair

Wireless Sensor-Actuator Networks (WSANs) enable cost-effective communication for Industrial Internet of Things (IIoT). To achieve predictability and reliability demanded by industrial applications, industrial wireless standards (e.g., WirelessHART) incorporate a set of unique features such as a centralized management architecture, Time Slotted Channel Hopping (TSCH), and conservative channel selection. However, those features also incur significant degradation in performance, efficiency, and agility. To overcome these key limitations of existing industrial wireless technologies, this thesis work develops and empirically evaluates a suite of novel network protocols and algorithms.

The primary contributions of this thesis are four-fold. (1) We first build an experimental testbed realizing key features of the WirelessHART protocol stack, and perform a series of empirical studies to uncover the limitations and potential improvements of existing network features. (2) We then investigate the impacts of the industrial WSAN protocol's channel selection mechanism on routing and real-time performance, and present new channel and link selection strategies that improve route diversity and real-time performance. (3) To further enhance performance, we propose and design conservative channel reuse, a novel approach to support concurrent transmissions in a same wireless channel while maintaining a high

degree of reliability. (4) Lastly, to address the limitation of the centralized architecture in handling network dynamics, we develop *REACT*, a Reliable, Efficient, and Adaptive Control Plane for centralized network management. REACT is designed to reduce the latency and energy cost of network reconfiguration by incorporating a reconfiguration planner to reduce a rescheduling cost, and an update engine providing efficient and reliable mechanisms to support schedule reconfiguration. All the network protocols and algorithms developed in this thesis have been empirically evaluated on the wireless testbed. This thesis represents a step toward next-generation IIoT for industrial automation that demands high-performance and agile wireless communication.

Chapter 1

Introduction

As the industrial Internet of Things (IoT) is emerging, Wireless Sensor-Actuator Networks (WSANs) have been increasingly recognized as an important technology for process automation. Contrary to wired networks, which are often expensive to install and maintain, WSANs have been adopted by process industries because of their benefits in reducing cost for the deployment and maintenance of process monitoring and control systems [3], and their flexibility to accommodate new production process requirements. IEEE 802.15.4 based WSANs provide low-data rate and low-cost communication, which is suitable for industrial application where energy consumption and costs are crucial.

In contrast to the traditional Wireless Sensor Networks (WSNs) that require only best effort service, to preserve stability and guarantee control performance [69], industrial WSANs are inherently subject to real-time and reliability requirements while operating in harsh industrial environment,i.e., sensor data or control commands must be delivered to its destination by its deadline. Failing to meet such requirements may result in safety threats, production inefficiency, and financial loss.

While there are multiple industrial IoT standards such as WirelessHART [83], ISA100 [33], WINA [82], IETF 6TiSCH [31], our work focuses on the WirelessHART standard, a leading wireless standard for industrial process applications widely adopted in the industry [21, 36, 68]. To address the critical demands for timely and reliable communication, WirelessHART [83] incorporates a set of distinctive design choices. For instance, the network utilizes a Time Slotted Channel Hopping MAC (TSCH) [30], a TDMA-based protocol with other features such as channel hopping and network-wide blacklisting. TSCH can offer deterministic and collision-free communication. Moreover, WirelessHART employs the centralized management architecture that guarantees predictable communication.

Industrial WSANs have been gaining interest in the research community in recent years. Significant previous research focused on network algorithms and theoretical analyses for the WirelessHART network. These efforts aimed to improve the performance of different aspects of the WirelessHART protocol such as routing algorithms, scheduling algorithms, delay analysis, rate selection, etc. Comprehensive reviews of these works can be found in [42] and [49]. However, these works are mostly based on theoretical analyses and simulation studies. Conversely, there has been earlier work that implemented and evaluated custom real-time and TDMA-based WSN protocols experimentally [4, 50, 55].

A more recent direction is to exploit IP-based communication in industrial WSNs. As opposed to the WirelessHART protocol, 6TiSCH runs IPv6 over a TSCH network. An open-source implementation of 6TiSCH has been explored in [81]. 6TiSCH supports both centralized and distributed scheduling solutions. Previous research proposed scheduling algorithms [53, 77] for centralized 6TiSCH network, while several efforts [1, 2, 20] developed decentralized scheduling schemes to support network adaptation. Although, these decentralized approaches enhances network adaptivity, they are designed for best effort service. Alternatively, a Glossy-based network [23] offering efficient and fast network flooding by exploiting constructive

interference and capture effect, provides a promising solution to support real-time and reliable communication [90]. On the contrary, our work is tailored for industrial applications that demand reliable and real-time communication over the centralized TSCH-based networks, which is adopted by the WirelessHART standard.

Although different aspects of industrial WSANs have been studied in the aforementioned efforts, significant improvements in the performance of industrial WSANs are still needed to meet reliability and timing requirements. Our research aims to develop and experimentally evaluate new network protocols and algorithms to improve the performance, efficiency, and agility of industrial WSANs. The following is a summary of our contributions:

1. **WSAN Testbed:** To enable experimental studies and evaluations, we develop an experimental testbed realizing the key features of the WirelessHART protocol including (a) centralized network manager software supporting route and schedule generation and update, and (b) a protocol stack that implements WirelessHART's network features (e.g., TSCH MAC and and source/graph routings), and provides the capability to support network reconfiguration.

2. **Channel Selection:** To ensure reliability, a WSAN only uses links that are reliable in all non-blacklisted channels for transmissions. Hence, channel selection can impact the network topology, and as a result, using more channels degrades the routing performance. Therefore, we design channel and link selection algorithms that automated the channel selection process to balance channel diversity and route diversity, thereby enhancing route generation while maintaining reliability.

3. **Conservative Channel Reuse:** Because channel reuse is not allowed in a WSAN, the capacity of a WSAN is limited. To enhance real-time performance while maintaining reliability, we develop a conservative channel reuse approach that introduces channel

reuse only when necessary to meet the timing constraints of the WSAN traffic. The conservative approach is in contrast to the traditional channel reuse approach that is designed to maximize channel reuse. We also develop a policy to detect links with poor reliability due to channel reuse, so that these links can be rescheduled to enhance WSAN reliability.

4. **Adaptive Control Plane:** While centralized network management architecture provides deterministic and reliable communication, it impairs the network's ability to adapt in response to changing wireless environment. Hence, we propose REACT, a reliable, efficient, and adaptive control plane for WirelessHART to support network adaptation. To reduce the network reconfiguration cost, REACT features: (a) a reconfiguration planner offering flexible scheduling and routing algorithms and reactive update policies to reduce rescheduling cost, and (b) an update Engine providing efficient and reliable mechanisms to report link failures, and to disseminate updated schedules.

The rest of the dissertation is organized as follows. Chapter 2 introduces the background of the WirelessHART network and flow models. Chapter 3 describes our WirelessHART testbed implementation and empirical study. Chapter 4 presents our channel selection policies. Chapter 5 discusses our conservative channel reuse approach. Chapter 6 presents our efficient control-plane to support network adaptation, and Chapter 7 concludes this dissertation.

Chapter 2

Background

In this chapter, we describe the WirelessHART network and flow models.

2.1 Network Model

Our work considers WirelessHART, a widely adopted industrial wireless standard for process monitoring and control applications. Figure 2.1 shows the architecture of the WirelessHART network. The network comprises multiple field devices (sensors or actuators) forming a

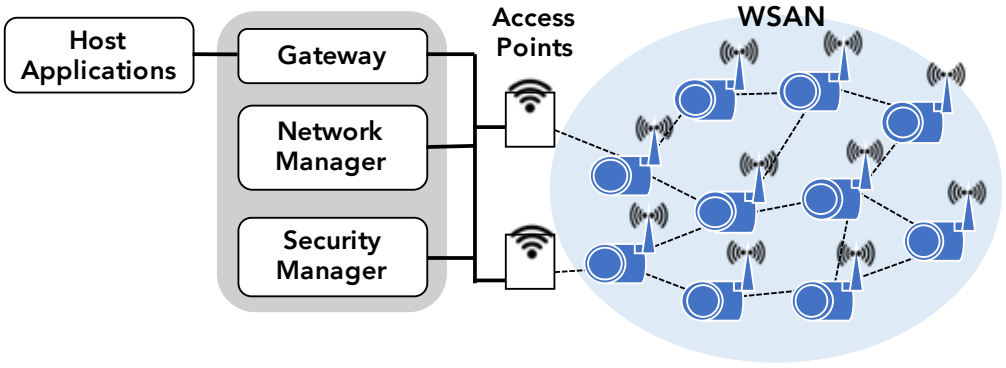
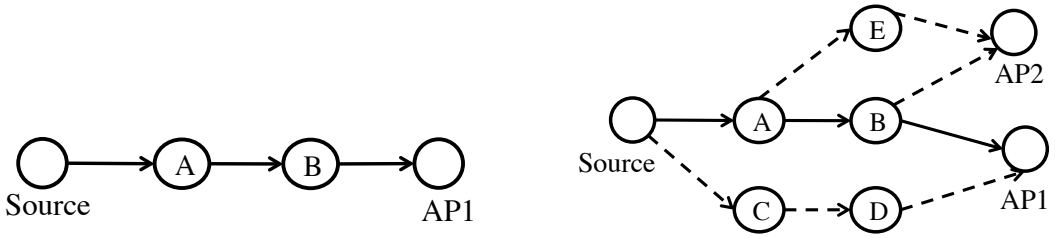


Figure 2.1: WirelessHART network architecture.

multi-hop wireless mesh network. Multiple access points connect the wireless network to the gateway and the managers. The gateway enables communication between wireless field devices and host applications (e.g, process controller). The network manager manages network operations, and the security manager handles security information and keys. To meet the stringent requirements on reliability and predictable real-time performance, WirelessHART adopts a specific sets of features including:

Centralized Manager: A WirelessHART network is supervised by a centralized network manager, which is responsible for managing and optimizing network operations. In particular, the network manager collects topology information, generates routes, and constructs a global schedule, which is then disseminates to field devices. The centralized approach enhances visibility and predictability of the network operations.

TSCH MAC: Time Slotted Channel Hopping technology has been implemented as a MAC protocol, and was introduced as part of the IEEE 802.15.4e standard in 2012 for the industrial process control and automation. WirelessHART employs the TSCH MAC that offers deterministic and collision-free communication. All nodes in a TSCH network are time synchronized, and time is divided into 10 ms time slots, which are sufficient to accommodate a packet transmission and its acknowledgment (ACK). A time slot can either be *dedicated* or *shared*. In a dedicated slot, only one transmission is allowed in each channel while in a shared slot, two senders compete for the channel for transmitting to a common receiver in a CSMA/CA fashion. TSCH operates on the 2.4 GHz ISM band and can use up to 16 channels defined in the IEEE 802.15.4 standard. To mitigate the impact of interference and enhance channel diversity, TSCH adopts a *channel hopping* mechanism where each device switches to a new channel in every slot.



(a) Source Routing (b) Graph Routing

Figure 2.2: Source and graph routing (solid lines represent primary paths and dashed lines represent backup paths).

Channel Blacklisting: To avoid using channels with poor quality, WirelessHART adopts a *network-wide* channel blacklisting approach that allows the network operator to manually blacklist channels with observed poor performance. This network-wide blacklisting policy ensures all channels used for communication are highly reliable and that all links used for transmissions use this same set of channels.

Source and Graph Routing: WirelessHART supports two routing approaches, named *source routing* and *graph routing*. Source routing provides a single route for each data flow from a source to a destination node. Graph routing consists of a single primary path and a backup path for each node on a primary path. As shown in Figure 2.2, for source routing, to send a packet to an access point AP1, a source node must transmit a packet through nodes A and B. Under graph routing, the packet may take backup routes to reach the access points if links on a primary path fail to deliver a packet. Compared to source routing, graph routing offers a higher degree of reliability through route diversity and transmission redundancy.

2.2 Flow Model

A WSAN is shared by a set of end-to-end flows $F = \{F_1, F_2, ...F_n\}$. For each flow $F_i \in F$, a source node S_i releases a packet at a periodic interval P_i. The packet must be delivered

to the destination Y_i through a route ϕ_i within the deadline D_i where $D_i \leq P_i$. A route ϕ_i consists of a sequence of links $l_{i1}, l_{i2}, ...l_{ik}$. A transmission over a link l_{ij} is denoted as t_{ij}. Therefore, each flow F_i is characterized by $< S_i, Y_i, D_i, P_i, \phi_i >$. A flow F_i is schedulable if all of its packets are scheduled to meet their deadlines. A set of flows F is schedulable only if all flows in F are schedulable.

Chapter 3

Implementation and Empirical Studies of Industrial Wireless Sensor-Actuator Network Protocols

While industrial WSANs have received increasing attention in the research community recently, most published results to date have focused on the theoretical aspects and were evaluated based on simulations. There is a critical need for experimental research on this important class of WSANs. We developed an experimental testbed by implementing several key network protocols of WirelessHART, an open standard for WSANs that has been widely adopted in the process industries including TSCH at the MAC layer and reliable graph routing supporting path redundancy. We then performed a series of empirical studies showing that graph routing leads to significant improvement over source routing in terms of worst-case reliability, but at the cost of longer latency and higher energy consumption. It is therefore important to employ graph routing algorithms specifically designed to optimize latency and energy efficiency. Our

studies also suggest that channel hopping can mitigate the burstiness of transmission failures; a larger channel distance can reduce consecutive transmission failures over links sharing a common receiver. Based on these insights, we developed a novel channel hopping algorithm that utilizes far away channels for transmissions. Furthermore, it prevents links sharing the same destination from using channels with strong correlations. Our experimental results demonstrate that our algorithm can significantly improve network reliability and energy efficiency [1].

3.1 Introduction

Recently, there has been increasing interest in developing new network algorithms and analysis to support industrial applications. However, there remains a critical need for experimental testbeds to validate and evaluate network research on industrial WSANs. Without sufficient experimental evaluation, industry has shown a marked reluctance to embrace new solutions.

To meet the need for experimental research on WSANs, we have built an experimental testbed for studying and evaluating WSAN protocols. Our testbed supports a suite of key network protocols specific to the WirelessHART standard and a set of tools for managing wireless experiments. We then present a comparative study of the two routing approaches adopted by WirelessHART, namely source routing and graph routing, and an empirical study on the impact of channel hopping on the burstiness of transmission failures. Our studies have lead to two major insights on the development of resilient industrial WSANs:

[1] The contents in this chapter have appeared in:
M. Sha*, D. Gunatilaka*, C. Wu and C. Lu, Implementation and Experimentation of Industrial Wireless Sensor-Actuator Network Protocols, EWSN, February 2015. (*co-first author)
M. Sha*, D. Gunatilaka*, C. Wu and C. Lu, Empirical Study and Enhancements of Industrial Wireless Sensor-Actuator Network Protocols, IEEE Internet of Things Journal, 4(3): 696-704, June 2017. (*co-first author)

- Graph routing leads to significant improvement over source routing in term of worst-case reliability, at the cost of longer latency and higher energy consumption. It is therefore important to employ graph routing algorithms specifically designed to optimize latency and energy efficiency.
- Channel hopping can mitigate the burstiness of transmission failures; a larger channel distance can reduce consecutive transmission failures over links sharing a common receiver.

Based on these insights, we developed a novel channel hopping algorithm for graph routing that causes senders to utilize far-away channels between consecutive transmissions over the same link. It further prevents links sharing the same destination from using channels with strong correlations. Our experimental results demonstrate that our algorithm can significantly improve network reliability and energy efficiency.

The rest of the chapter is organized as follows. Section 3.2 reviews related work. Section 3.3.3 describes our implementation of the WirelessHART protocol. Section 3.4 presents our empirical studies. Section 3.5 presents and evaluates our channel hopping algorithm. and Section 3.6 concludes the chapter.

3.2 Related Work

There has been recent work that implemented and evaluated real-time WSN protocols experimentally. Recently, O'Donovan et al. [50] developed the GINSENG system, which uses wireless sensor networks to support mission-critical applications in industrial environments and shared their valuable experience during real-world deployments. Munir et al. [46] designed a scheduling algorithm that produces latency bounds of the real-time periodic streams and

accounts for both link bursts and interference. Pottner et al. [55] designed a scheduling algorithm to meet application requirements in terms of data delivery latency, reliability, and transmission power. While valuable insights can be drawn from the aforementioned efforts, the novelty of our work lies in its focus on key aspects of the WirelessHART standard, such as graph routing, that were not studied in earlier works. Our results are therefore complementary to earlier findings on other aspects of real-time WSANs.

There have been recent empirical studies that investigated the burstiness of transmission failures and 802.15.4 channel performance in various wireless environments and network settings. Srinivasan el al. [71, 72, 73] performed a series of link studies to quantify the burstiness of intra-link and inter-link performance on their office testbed. Sha et al. [65] performed a spectrum study in the 2.4 GHz band as well as a link study of IEEE 802.15.4 channels in residential environments. Hauer et al. [26] conducted a multi-channel measurement of Body Area Networks. Ortiz et al. [52] evaluated the multi-channel behavior of 802.15.4 networks in a machine room, a computer room, and an office testbed and found path diversity to be an effective strategy to ensure reliability. In contrast to these studies, our own study is specific to WirelessHART's key mechanisms such as graph routing and sequential slot based channel hopping. Therefore, our results are complementary to these earlier findings.

In recent years, there has been increasing interest in using channel hopping to enhance network reliability. Navda et al. [47] proposed a rapid channel hopping scheme to protect a network from jamming attacks. Le et al. [37] designed a control theory approach to dynamically allocate channels in a distributed manner. Sha et al. [64] designed an opportunistic channel hopping algorithm to avoid jammed channels. Industry standards such as Bluetooth's AFH [7] leveraged constant hopping in a pseudorandom fashion across channels to avoid persistent interference. In contrast to these works, our channel hopping algorithm CCS is designed to improve the reliability and energy efficiency of WirelessHART networks with several features

that distinguish itself from existing channel hopping approaches in WirelessHART and other networks. First, CCS enforces a specified channel distance between transmissions, thereby avoiding adjacent channels with strong correlations. Second, CCS combines link-based and receiver-based channel hopping, further enhancing its effectiveness in reducing bursty transmission failures. Finally, CCS is specifically tailored for WirelessHART protocols such as graph routing and per-slot channel hopping.

While our study focuses on WirelessHART, alternative industrial WSAN standards exist such as ISA-100.11a [33], WIA-PA [88], and the recently approved IEEE 802.15.4e MAC enhancement standard [29]. These standards share many common approaches and mechanisms. For example, IEEE 802.15.4e specifies a time-slotted channel hopping mode which combines time slotted access, multi-channel communication, and channel hopping to improve reliability and mitigate the effects of interference and multipath fading. Our insights and proposed algorithm therefore may be generalized to influence the implementation of WSANs based on these standards.

3.3 Implementation of a WirelessHART WSAN Testbed

In this section, we first describe our testbed architecture, then introduce our system architecture, and lastly present our WirelessHART protocol stack implementation.

3.3.1 Testbed Architecture

our WSAN testbed includes a four-tier hardware architecture that consists of field devices, microservers, a server, and clients. The field devices in the testbed are 55 TelosB motes, a widely used wireless embedded platform integrating a TI MSP430 microcontroller and a TI CC2420 radio compatible with the IEEE 802.15.4 standard. A subset of the field devices

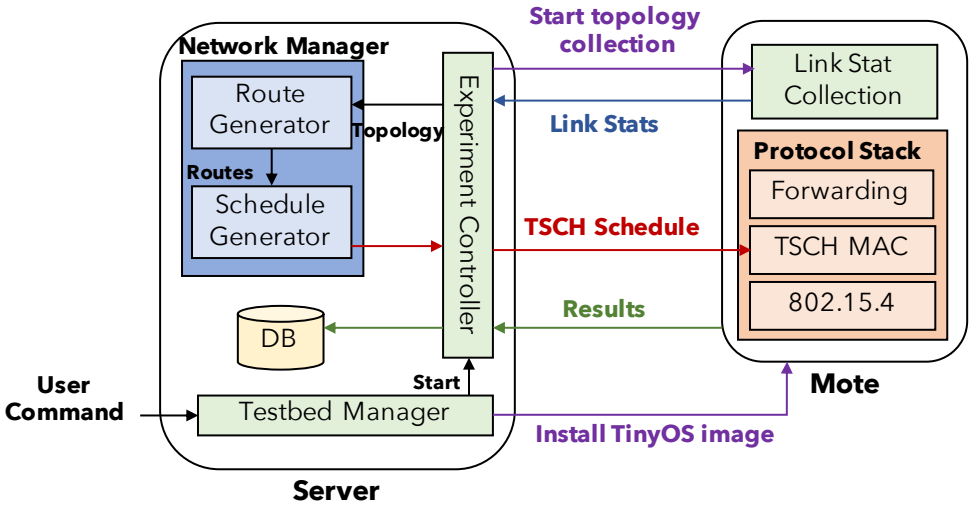


Figure 3.1: Testbed software architecture.

can be designated as access points in an experiment. The field devices and access points form a multi-hop wireless mesh network running WSAN protocols. A key capability of our testbed is a wired backplane network that can be used for managing wireless experiments and measurements without interfering with wireless communication. The backplane network consists of USB cables connecting the field devices and microservers, which are in turn connected to a server through the Ethernet. The microservers are responsible for forwarding network management traffic between the field devices and the server. The server executes network management processes, and serves as a gateway and runs the network manager of the WSAN. The clients are regular computers that users employ to manage their wireless experiments and collect data from the experiments through the server and the backbone network.

3.3.2 Testbed Software Architecture

Figure 3.1 depicts the software architecture for our WSAN testbed. Our WSAN system comprises the following components.

- Network manager: The network manager resides on the server. It implements a route and a schedule generators. The route generator is responsible for creating source routes or graph routes based on the collected network topology. We use Dijkstra's shortest-path algorithm to generate routes for source routing and follow the algorithm proposed in [25] to generate reliable graphs. The scheduler uses the rate monotonic scheduling algorithm [39] to construct transmission schedules.

- Protocol stack: The protocol stack runs on TinyOS 2.1.2 [78] and TelosB motes [76]. It is built on top of the IEEE 802.15.4 PHY layer, and consists of a TSCH MAC, a TDMA-based MAC with channel hopping, and a forwarding layer that forwards packets based on routes constructed by the network manager.

- Experimental management tools: The managements tools consist of (a) a link stat collection module running on a mote responsible for collecting link statistics, (b) an experiment controller responsible for managing wireless experiments (e.g., initiating topology collection, distributing a schedule, gathering experimental data), (c) a testbed manager allowing a user to install/erase TinyOS program onto/from motes and to initiate experiments.

These are the steps to run a testbed experiment. A user first installs the TinyOS program onto the mote, and starts the experiment controller. The controller issues commands to nodes to collect link statistics in all 16 channels. Motes report this topology information back to the network manager. The manager uses this information to compute routes for a flow set, and construct a global schedule, which is then disseminated to the motes. During an experiment, statistics regarding network behaviors are collected to the server and stored in the database. We note that the network management traffic including link statistics, a

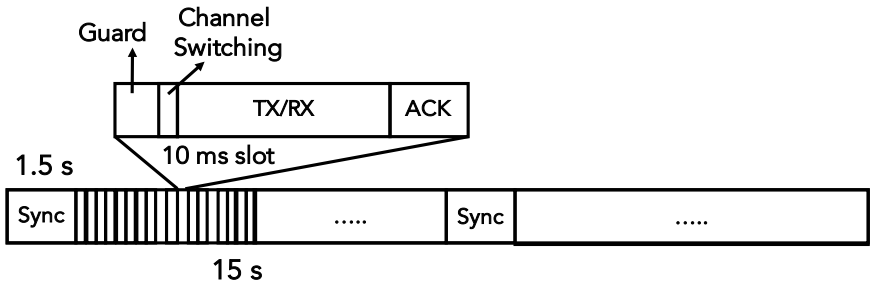


Figure 3.2: Time frame format of TSCH.

transmission schedule, and experimental data are transferred to/from the network manager through the wired back plane.

3.3.3 Protocol Stack

Our protocol stack adopts the CC2420x radio driver [12] as the radio core, which provides an open-source implementation of IEEE 802.15.4 physical layer in TinyOS operating over TI CC2420 radios. The CC2420x radio stack takes care of the low-level details of transmitting and receiving packets through the radio hardware. On top of the radio core, we have developed a TSCH MAC protocol, *RT-MAC*, which implements the key features of WirelessHART's MAC protocol. As shown in Figure 3.2, RT-MAC divides the time into 10 ms slots following the WirelessHART standards and reserves a **Sync** window (1.5 s) in every 1650 slots. Flooding Time Synchronization Protocol (FTSP) [44] is executed during the Sync window to synchronize the clocks of all wireless devices over the entire network. Our micro-benchmark experiment shows that an FTSP's time stamp packet can finish traversing of entire 55-node testbed within 500 ms. Therefore, RT-MAC configures the FTSP to flood three time stamps with 500 ms intervals over the network in each Sync window to adjust the local clocks of all devices to a global time source, which is the local time of the mote attached to the network manager. The time window following the Sync window consists of recurring superframes (a

series of time slots) and idle intervals. We reserve 2 ms of guard time in the beginning of each slot to accommodate the clock synchronization error and channel switching delay, since our micro-benchmark experiments show that more than 95% of field devices over the entire network can be synchronized with errors less than 2 ms, and channel switching takes only a few microseconds to write to the registers. The rest of the field devices may disconnect from the network due to larger clock synchronization errors, but they will be reconnected in the next Sync window after they catch the new time stamps generated by FTSP. RT-MAC supports both dedicated and shared slots. In a dedicated slot, only one sender is allowed to transmit, and the packet transmission occurs immediately after the guard time. In a shared slot, more than one sender can attempt to transmit, and these senders contend for the channel using CSMA/CA.

3.4 Empirical Study

We conduct our empirical studies on the WUSTL testbed. Following the practice of industrial deployment, the routing algorithms used in our study consider only reliable links with PRR higher than 80%. We use 8 data flows in our experiments. We run our experiments such that each flow can deliver at least 500 packets from its source to its destination. Figure 3.3 shows the network topology along, a set of flows used in our study, and locations of access points and field devices. The bigger yellow circles denote the access points that communicate with the network manager running on the server through the wired backbone network. The other circles and squares denote the field devices. The source and destination of a flow are represented as a circle and a square, respectively. The pair of source and destination of a same flow uses the same color. The period of each flow is randomly selected from the range of $2^{0\sim7}$ seconds, which falls within the common range of periods used in process industries. We

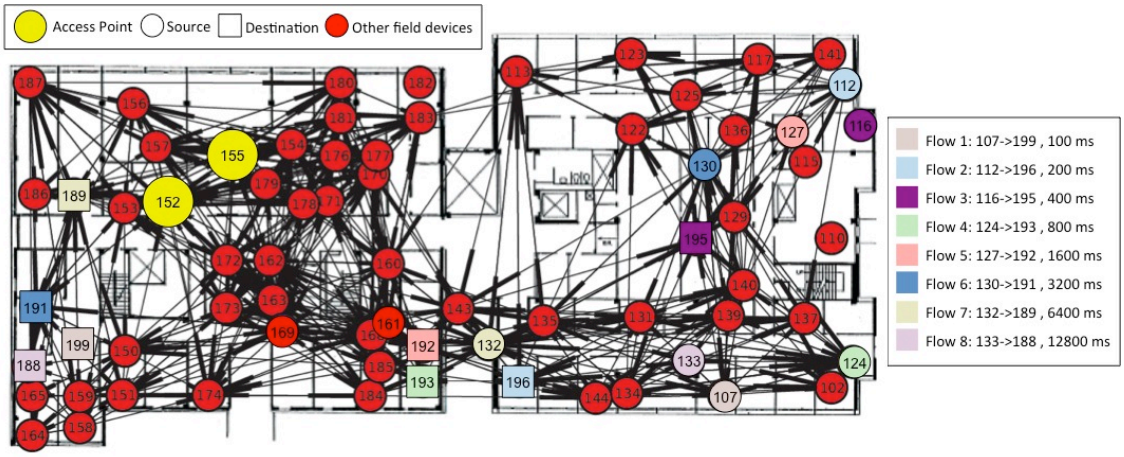


Figure 3.3: Testbed topology, a set of flows used in the experiment, and locations of access points, and sources and destinations of flows.

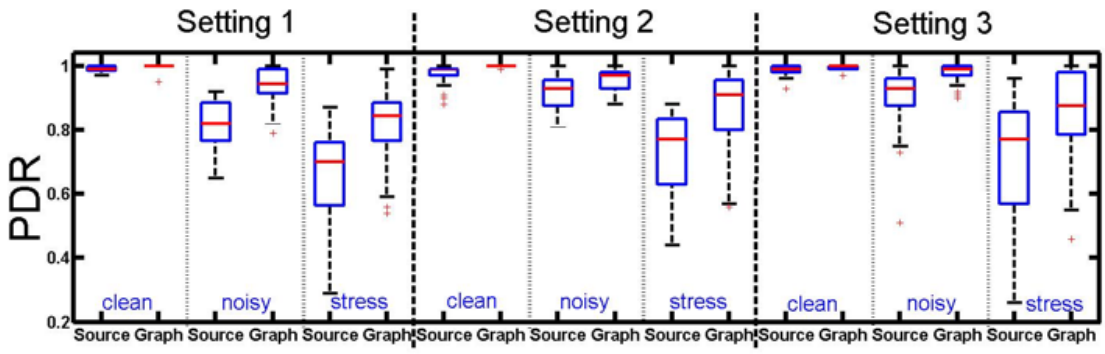


Figure 3.4: Box plot of the PDR of source routing and graph routing in the clean, noisy, and stress testing environments. Vertical lines delineate three different network configurations.

also repeat our experiments with two other network configurations by varying the location of access points, sources, and destinations.

3.4.1 Experimentation of Source and Graph Routing

We conduct a comparative study of the two alternative routing approaches adopted by WirelessHART, namely source routing and graph routing. Specifically, we investigate the tradeoff among reliability, latency, and energy consumption under the different routing approaches. We run two sets of experiments, one with the source routing and one with the

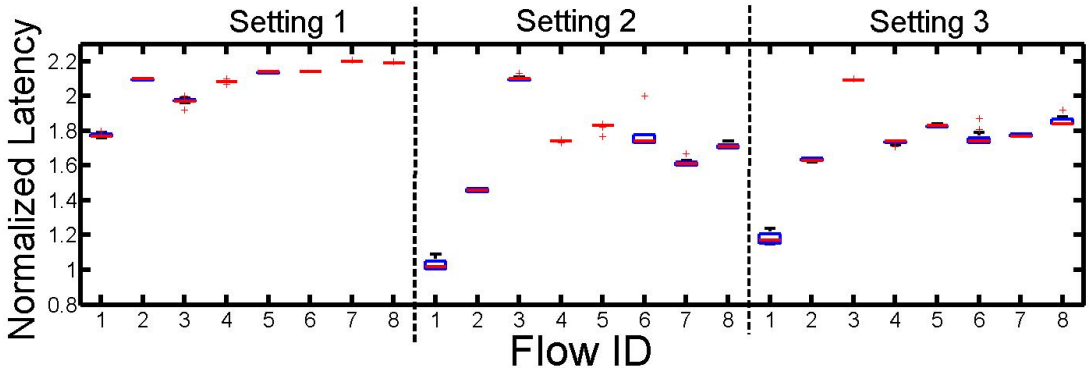


Figure 3.5: Box plot of the normalized latency of source routing and graph routing of each flow under graph routing over that under source routing.

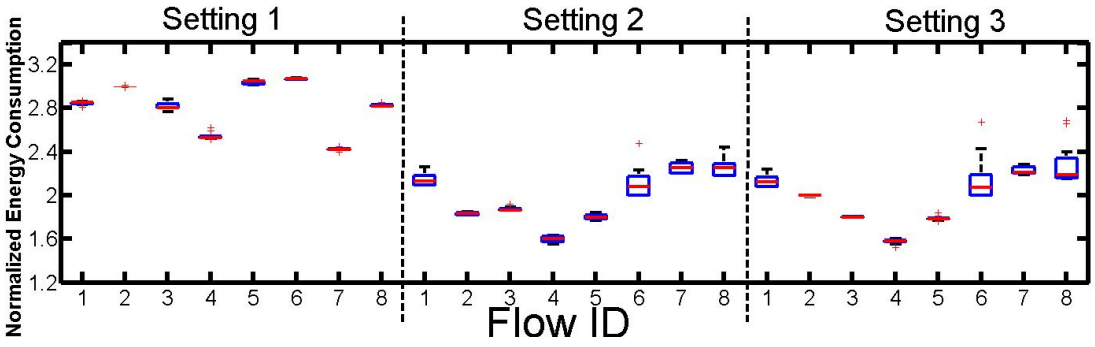


Figure 3.6: Box plot of the normalized energy consumption of source routing and graph routing of each flow under graph routing over that under source routing.

graph routing. We repeat the experiments under a clean environment, a noisy environment, and a stress testing environments

1. Clean: we blacklist the four 802.15.4 channels overlapping with our campus WiFi network and run the experiments on the remaining 802.15.4 channels.

2. Noisy: we run the experiments by configuring the network to use channels 16 to 19, which overlap with our campus WiFi network. Note that Co-existence of WirelessHART devices and WiFi is common in industrial deployments since WiFi is often used as backhauls to connect multiple WSANs.

3. Stress testing: we run the experiments with channels 16 to 19 under controlled interference, in the form of a laptop and an access point generating 1 Mbps UDP traffic over WiFi channel 6, which overlaps with 802.15.4 channels 16 to 19.

We use the packet delivery rate (PDR) as the metric for network reliability. The PDR of a flow is defined as the percentage of packets that are successfully delivered to their destination. Figure 3.4 compares the network reliability under source routing and graph routing in the three environments. As shown in Figure 3.4, under the first network configuration, compared to source routing, graph routing improves the median PDR by a margin of 1.0% (from 0.99 to 1.0), 15.9% (from 0.82 to 0.95), and 21.4% (from 0.70 to 0.85) in the clean, noisy, and stress testing environments, respectively. Graph routing shows similar improvement over source routing under the other two network configurations. More importantly, graph routing delivers a significant improvement in min PDR and achieves a smaller variation of PDR than source routing, which represents a significant advantage in industrial applications that demand predictable performance. The improvements in min PDR are 35.5% and 63.5% in noisy and stress testing, respectively. This result shows that graph routing is indeed more resilient to interference due to route diversity. However, as shown in Figure 3.5, route diversity incurs a

cost in term of latency, with graph routing suffering an average of 80% increase in end-to-end latency. We also estimate the energy consumption based on timestamps of radio activities and the radio's power consumption in each state. As Figure 3.6 shows, graph routing consumes an average of 130% more energy than source routing.

Observation 1*: Graph routing leads to significant improvement over source routing in term of worst-case reliability, at the cost of longer latency and higher energy consumption. It is therefore important to employ graph routing algorithms specifically designed to optimize latency and energy efficiency.*

3.4.2 Impact of Channel Hopping on Burstiness of Transmission Failures

As shown in previous studies [26, 46, 52, 65, 71, 72, 73] (and confirmed on our testbed), the burstiness of transmission failures significantly compromises the reliability and energy efficiency of WSANs. WirelessHART mitigates this issue by adopting spectrum diversity through sequential channel hopping in each time slot. Notably, while burstiness of transmissions in a same channel have been studied extensively, there have been few empirical studies of burstiness under channel hopping. To explore the impact of channel hopping, we run experiments on multiple links using 16 IEEE 802.15.4 channels under controlled interference generated by a WiFi access point and a laptop. We run the experiments with twenty different pairs of senders and receivers. In each run, each sender transmits 500 packets. If a transmission fails, a sender hops to the next channel and retransmits a packet. Previous study [65] showed that adjacent channels may suffer from bursty transmission failures due to significant correlation between adjacent channels. We hypothesize that increasing hopping distance can also improve the

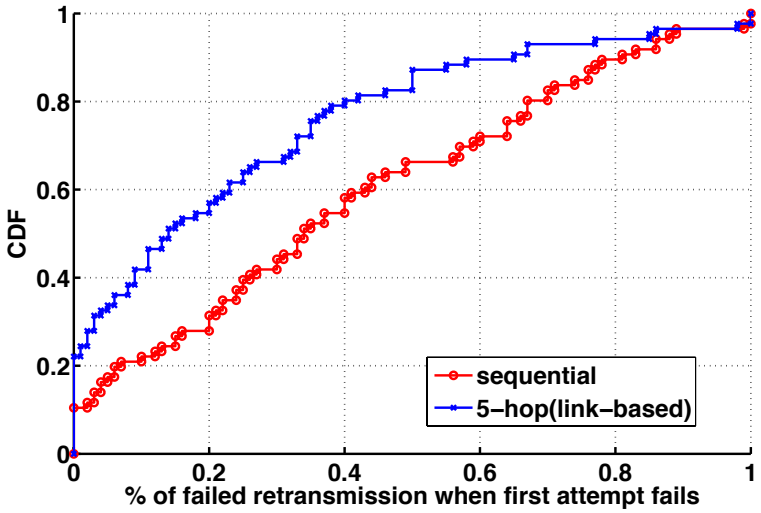


Figure 3.7: CDF of percentage of a failed retransmission when first transmission attempt fails.

reliability of WSANs. Therefore, we conduct another study by increasing to 5 the distance between the channels used for a transmission and its retransmission.

Figure 3.7 shows the cumulative distribution function (CDF) of the failure ratio of retransmissions following failed transmissions. Under sequential channel hopping, 33.5% of links have a 50% retransmission failure, while under a channel hopping distance of 5, only 13.5% of links have a 50% retransmission failure. To recover from a failed transmission, hopping over a large channel distance is more effective than sequential channel hopping. This may be due to the fact that interference often span multiple adjacent channels. For example, WiFi signals usually overlap with four channels of IEEE 802.15.4.

In addition to burstiness-of-transmission failures over a link, we also observe that links sharing the same receiver can also suffer from strong correlations of transmission failures. We apply channel hopping to links with a common receiver. For each setup, we pick 3 links that involve the same receiver. Each sender takes turns sending out a packet and switch to a channel

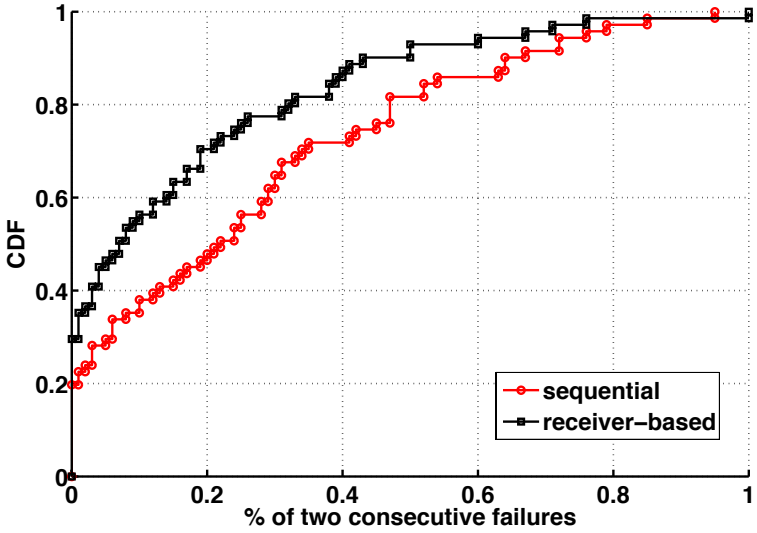


Figure 3.8: CDF of percentage of two consecutive transmission failures over links that share a common receiver.

that is one channel (sequential) or five channels away from the channel used by the previous sender. Each experiment lasts 500 rounds.

Figure 3.8 presents the CDF of the percentage of two consecutive transmission failures over links that share a common receiver. Again, increasing the channel distance used for channel hopping effectively reduces consecutive failures over links to a same receiver, and therefore mitigates the correlations of links to the same receiver.

Observation 2*: A larger channel distance can reduce consecutive transmission failures over links sharing a common receiver.* We have shown the limitation of sequential channel hopping over either the same link or links sharing a receiver. In practice, the effect of channel distance may vary in different wireless environments depending on the source of interference and on wireless conditions. The channel distance therefore should be treated as a tunable parameter that needs to be selected based on field testing and knowledge about existing interference and wireless environments.

3.5 Enhanced Channel Hopping for WirelessHART

In this section, we present and evaluate *Configurable Channel Stride (CCS)*, a novel channel hopping algorithm designed to improve the reliability and energy efficiency of WirelessHART networks. CCS has several salient features that distinguish itself from existing channel hopping approaches in WirelessHART and other networks. First, CCS enforces a specified channel distance between transmissions, thereby avoiding adjacent channels with strong correlations. Second, CCS combines link-based and receiver-based channel hopping, further enhancing its effectiveness in reducing bursty transmission failures. Finally, CCS is specifically tailored for WirelessHART protocols such as graph routing and per-slot channel hopping.

3.5.1 The Configurable Channel Stride Algorithm

In our empirical study in Section 3.4.2, we observed that consecutive retransmissions over a same link on adjacent channels cannot effectively eliminate transmission failures on primary routing paths due to strong channel correlation. We also observed a large number of consecutive failures when multiple senders transmit packets back to back using adjacent channels to a shared destination. To mitigates both per-link and per-receiver burstiness of failures, our algorithm combines two channel hopping approaches: (1) link-based channel hopping for links located on primary routing paths and (2) receiver-based channel hopping for links sharing the same destination.

Algorithm 1 shows the pseudo-code of our CCS algorithm. The input of our algorithm is both a desired channel hopping distance and a transmission schedule for the superframe, which is generated by the routing and scheduling algorithm, and which also specifies a set of transmissions scheduled for each time slot. Within a slot, transmissions are ordered according

Algorithm 1: Configurable Channel Stride

```
h: target channel distance per channel hop;
tx_ij: the j^th transmission assigned to time slot i;
channel_ij: the channel assigned to tx_ij;
flow_ij: the flow that tx_ij belongs to;
dest_ij: the receiver node of tx_ij;
channel_1st_tx: the channel assigned to the first transmission attempt;
ChannelPool_i: the set of available channels for time slot i;
for each time slot i within a superframe S do
    for each transmission tx_ij scheduled in the time slot i do
        if tx_ij is on a primary path then
            if tx_ij is the first attempt for a transmission then
                channel_ij ← first channel in ChannelPool_i;
                channel_1st_tx ← channel_ij;
            else                                        /* tx_ij is a retransmission */
                if there exists a channel c in ChannelPool_i that is at least h hop away from
                  channel_1st_tx then
                    channel_ij ← c;
                else
                    channel_ij ← channel in ChannelPool_i with a maximum channel distance
                      from channel_1st_tx;
        else                                            /* tx_ij is on a backup path */
            if there is no prior transmission to dest_ij from flow_ij then
                channel_ij ← first channel in ChannelPool_i;
            else
                if there exists a channel c in ChannelPool_i that is at least h hop away from
                  flow_ij's last transmission to dest_ij then
                    channel_ij ← c;
                else
                    channel_ij ← channel in ChannelPool_i with a maximum channel distance
                      from flow_ij's last transmission to dest_ij;
        Remove channel_ij from ChannelPool_i;
```

to their flow priority. For instance, tx_{ij} denotes the transmission j scheduled in slot i. $dest_{ij}$ denotes the receiver of tx_{ij}. $flow_{ij}$ denotes the flow tx_{ij} belongs to. $ChannelPool_i$ denotes a channel pool including all current available channels that can be scheduled in slot i. The output of the algorithm is the channel assigned to each tx_{ij}, denoted by $channel_{ij}$.

Our channel-hopping algorithm separates the channel used for a packet transmission from the one used for its retransmission such that the distance between them is at least h hop away. For instance, if the transmission tx_{ij} is the second transmission attempt (retransmission) over a link located on a primary path, a channel is chosen from the $ChannelPool_i$ such that it is at least h hop away from $channel_{1st\_tx}$, where $channel_{1st\_tx}$ is a channel assigned to the first transmission attempt over this link. Our receiver-based channel hopping requires that when a transmission tx_{ij} is on a backup path and there exists a prior transmission of $flow_{ij}$ to a receiving node $dest_{ij}$, then a selected channel must be at least h hop away from the channel used by $flow_{ij}$'s last transmission to $dest_{ij}$. In both cases, if such a channel does not exist, we assign tx_{ij} to use a channel in a $ChannelPool_i$ with a maximum spectral distance instead.

3.5.2 Evaluation

We ran our experiments under three different network configurations by varying the location of access points, sources, and destinations. Under each configuration, we performed four experimental runs: a first run (sequential) using the sequential channel hopping approach suggested by the WirelessHART standard [83], a second run (link-based) using our algorithm with only link-based channel hopping enabled, a third run (receiver-based) enabling only our receiver-based channel hopping approach, and a fourth run (link+receiver-based) enabling both our channel hopping approaches. We performed the experiments on our testbed under controlled interference (see stress testing setup in Section 3.2), use all 16 channels in 2.4 GHz, and set channel hopping distance h to 5. We note that our empirical study shows that the

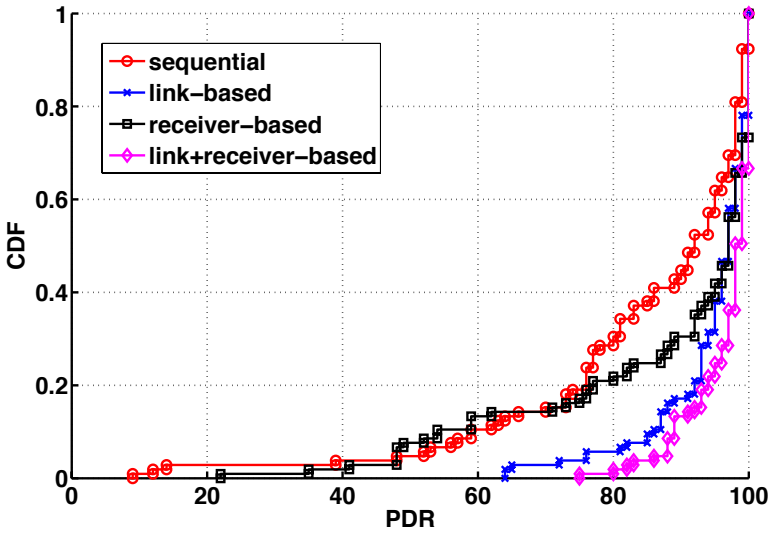


Figure 3.9: CDF of PDR under four channel assignment approaches.

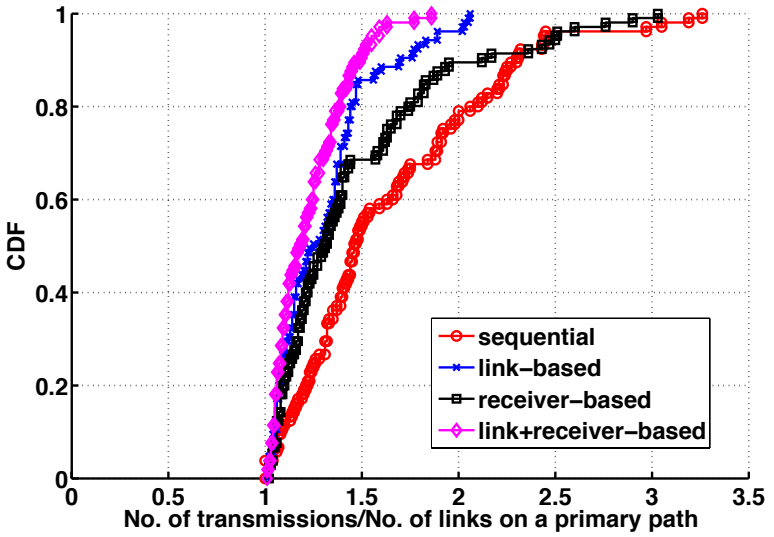


Figure 3.10: CDF of number of transmission attempts per number of links on a primary path under four channel assignment approaches.

probability of simultaneous channel failures drops off as channel distance increases to more than 3.

Figure 3.9 shows the CDF of the PDR for different channel assignment approaches. Each data point represents a percentage of flows with 100 generated packets that have a PDR less than or equal to x. Under sequential channel hopping, only 55.2% of flows achieve a PDR larger than 90%. However, under receiver-based, link-based, and integration (link+receiver-based) policy, 69.5%, 81.9%, and 85.1% of flows, respectively, attain a PDR larger than 90%. These results demonstrate the effectiveness of a larger channel hopping distance and the complementary benefits of link-based and receiver-based channel hopping. Overall the full CCS algorithm increased the number of reliable flows (with a PDR above 90% PDR) by 54% compared to the sequential channel hopping approach. Furthermore, CCS drastically improved the reliability of the flows, seeing the worst PDR among all the flows. Under sequential channel hopping, the least reliable flow experienced a PDR of only 9%. In contrast, under the full CCS algorithm, the least reliable flow still achieved a PDR of 75%. This shows that our CCS policy benefits industrial applications that demand a high degree of reliability and predictability.

Figure 3.10 presents a CDF of the number of transmission attempts per number of links on a primary path. With the standard channel assignment, 44.8% of flows requires more than $1.5X$ transmission attempts to achieve a desired PDR. Our receiver-based, link-based, and integration approaches are proved to be more efficient, with 31.0%, 14.0%, and 8.6% of flows requiring $1.5X$ transmission attempts, respectively. In a worst-case scenario, sequential, receiver-based, link-based, and integration policies yield at most $3.3X$, $3.0X$, $2.1X$, and $1.9X$ transmission attempts. Hence, our channel hopping policy provides a notable reduction in the number of transmission attempts of each flow to achieve a desired PDR, which indicates better link quality and can result in lower energy consumption.

3.6 Summary

Complementary to recent research on theoretical aspects of WSAN design, we have implemented a suite of network protocols of the WirelessHART standard in TinyOS and TelosB motes and then performed a series of empirical studies on WSAN protocol designs. We further developed a novel channel hopping algorithm that prevents consecutive transmissions from using channels with strong correlations on a common link or to a common receiver. Experimental results demonstrate that our algorithm can significantly improve network reliability and energy efficiency.

Chapter 4

Impacts of Channel Selection on Industrial Wireless Sensor-Actuator Networks

To meet stringent reliability requirements of industrial applications, industrial standards such as WirelessHART adopt TSCH as its MAC protocol. Since every link hops through all the channels used in TSCH, a straightforward policy to ensure reliability is to retain a link in the network topology only if it is reliable in all channels used. However, this policy has surprising side effects. While using more channels may enhance reliability due to channel diversity, more channels may also reduce the number of links and route diversity in the network topology. We empirically analyze the impact of channel selection on network topology, routing, and scheduling on a 52-node WSAN testbed. We observe inherent tradeoff between channel diversity and route diversity in channel selection, where using an excessive number of channels may negatively impact routing and scheduling. We propose novel channel and link selection

strategies to improve route diversity and network schedulability. Experimental results on two different testbeds show that our algorithms can drastically improve routing and scheduling of industrial WSANs [2].

4.1 Introduction

WirelessHART adopts a network-wide channel blacklisting approach in which all devices in the network blacklist the same set of channels. The network operator performs by measuring the wireless channel's condition and then manually blacklisting channels where a large number of external wireless signals were observed during the measurement. Unfortunately, this manual approach is labor intensive and error prone.

TSCH uses a channel hopping mechanism in which each link in the network hops through all channels that are not blacklisted, i.e., the channels used. To ensure reliability, a straightforward approach is to use only links that are highly reliable in all channels used for transmissions. Hence, channel selection can significantly impact the network topology. As a result, while using more channels can enhance reliability due to channel diversity and can increase the number of transmissions that may potentially be scheduled in a time slot, more channels may also lower route diversity in the network topology. Channel selection must therefore balance channel diversity and route diversity in a WirelessHART network.

To our knowledge, this work represents the first systematic study on channel selection for WirelessHART networks. We believe our findings and designs are applicable to other industrial WSAN standards based on TSCH networks. The contributions of this work are three-fold:

[2]The contents in this chapter have appeared in: D. Gunatilaka, M. Sha and C. Lu, Impacts of Channel Selection on Industrial Wireless Sensor-Actuator Networks, INFOCOM, May 2017.

- We empirically study the impact of channel selection on network topology, routing, and real-time performance based on the topology of a WSAN testbed. We find the performance of a WirelessHART network does not improve monotonically with an increasing number of channels used, due to the tradeoff between channel and route diversity.

- Based on the insights gathered from our studies, we propose a channel selection algorithm and a link selection strategy which can configure the network at deployment and/or runtime in response to operating dynamics. Our algorithms take the guesswork out of the channel blacklisting process through an automated selection of channels to balance channel diversity and route diversity.

- We evaluate our channel selection algorithms using a wireless testbed running a WirelessHART protocol stack. Experimental results show our channel selection algorithms can significantly improve the network's capability to meet the routing and scheduling demands of data flows, while requiring only moderate processing time.

The remainder of the chapter is organized as follows. Section 4.2 reviews related work. Section 4.3 presents our empirical studies. Sections 4.4 and 4.5 describe our channel selection algorithm and link selection strategy. Section 4.6 presents our evaluation, and Section 4.7 concludes the chapter.

4.2 Related Work

There has been significant research on industrial WSANs spanning transmission scheduling, routing algorithms and network protocols, assuming a given set of channels. Readers are referred to recent review articles for comprehensive surveys [42] [81]. However, the impact of

channel selection has not been studied in the context of industrial WSANs. Our work on channel selection is therefore complementary to previous work in this area.

802.15.4 radio link quality has been studied extensively in the context of WSNs. Sha et al. [65] performed empirical studies on real-world 2.4 GHz ISM band usage and channel conditions in residential environments. Ortiz et al. [52] evaluated the multi-channel behavior of 802.15.4 networks in a machine room, and an office testbed, and found route diversity to be an effective strategy to ensure reliability. Neither of these studies analyzed the impact of the number and the choice of channels on the route diversity of the network.

Multi-channel communication and channel allocation have been explored extensively in the wireless sensor network literature. For instance, Kim et al. [34] developed a multi-channel TDMA MAC protocol, and Raman et al. [59] designed a TDMA-based approach that uses channel hopping to improve network throughput. Wu et al. [85] and Vedantham et al. [80] proposed channel assignment schemes that partition a network into subgraphs and then assign channels to each subgraph. Tang et al. [75] and Le et al. [37] developed a multi-channel MAC protocol with dynamic channel allocation. Doddavenkatappa et al. [19] developed a channel selection and switching strategy that transforms intermediate quality links into good ones. Mobashir et al. [45] proposed a channel selection strategy in which only 3 channels far apart are used for communication. Chowdhury et al. [14] designed a scheme for packet scheduling and channel selection utilizing a carrier sensing mechanism. There also exists research on channel assignment for multi-radio multi-channel communication in the context of the IEEE 802.11-based wireless mesh networks, such as [5, 27, 38, 54, 57, 58, 74, 86].

These works focused on solving the problem of optimizing multi-channel communication assuming a given set of channels, while our work investigates complementary problem of how to select a good set of channels. In contrast to the existing work designed for best effort

service, our solution is tailored for industrial applications that demand reliable and real-time communication over the TSCH-based networks.

4.3 Empirical Study

In this section, we introduce our experimental setup and present our empirical studies on the impact of channel selection on network topology, routing, and scheduling.

4.3.1 Empirical Study Settings

To investigate the impacts of channel selection on WSAN performance, we collect topology information on all 16 IEEE 802.15.4 channels from the *WUSTL* testbed consisting of 52 TelosB motes spanning across two adjacent buildings. In our study, two nodes are designated as access points and the rest serve as field devices. We also repeat our experiments on the *Indriya* testbed deployed at the National University of Singapore [18]. The Indriya testbed consists of 86 TelosB motes spanning three floors of a school building.

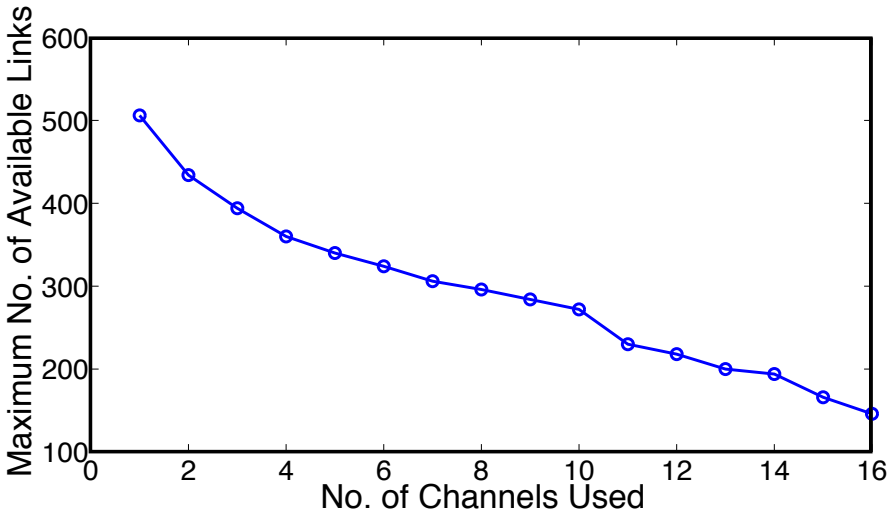
Channel Selection: We use the packet reception ratio (PRR) to estimate link reliability. PRR is defined as the fraction of transmitted packets successfully received by the receiver. Following a straightforward link selection policy, the WSAN only use bidirectional links whose PRRs are above a threshold in *all* the channels used and in both directions. This is because each link cycles through all the channels used through channel hopping. As a result, a link with a poor PRR in any of the channels used can incur packet loss at run time. Additionally, the link needs to be bidirectional, supporting reliable communication of both data packets and their acknowledgments.

To assess the impacts of channel selection on the performance of a WirelessHART network, we use a brute-force search algorithm to identify the set of channels that offers the maximum number of available links. Given k channels used, we compute all possible combinations of k channels from the 16 IEEE 802.15.4 channels and choose the set of channels that allows the most links to remain in the network topology. We set the PRR threshold PRR_t to 80% in this study.

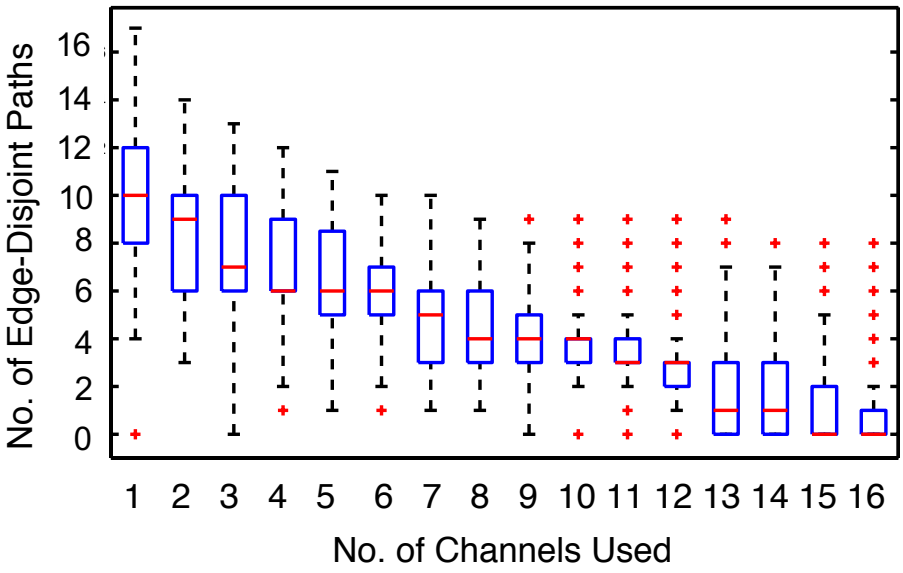
Routing and Scheduling: Consider a set of real-time periodic flows sharing a WirelessHART network, each of which delivers a packet from a source to a destination. The network manager finds the shortest path as the primary path of a flow for both source and graph routing. For backup paths of graph routing, we run the same shortest path algorithm to obtain the backup route from each node on a primary path to the destination. In addition, we also follow [84], which proposed a real-time and energy-efficient routing algorithms for WirelessHART networks, to generate source and graph routes. The network manager then schedules transmissions of each flow based on the rate monotonic policy [39], a fixed priority scheduling often adopted by industrial WSANs. Under the rate monotonic policy, a flow with a shorter period has a higher priority. The period of each flow is selected within the range $2^{0\sim7}$ seconds, which falls within the common range of periods used in process industries, and its deadline equals its period.

4.3.2 Impact on Routing

In this subsection, we first study the impact of the number of channels used on the network topology and routing. As Figure 4.1a shows, the number of available links *decreases* with an increasing number of channels used. More channels used result in fewer available links due to the stringent reliability requirement, in which the PRRs of these links must be no less than the desired threshold over all channels used. We further quantify the route diversity



(a) Maximum number of available links.



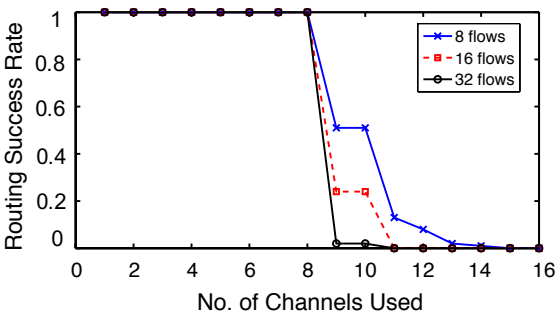
(b) Number of edge-disjoint paths.

Figure 4.1: Impact of the number of channels used on the network topology (WUSTL).

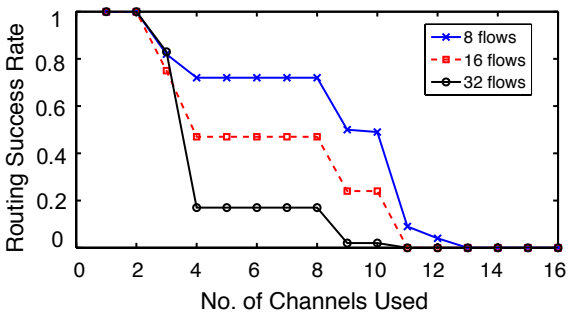
of a network by computing the number of edge-disjoint paths of a flow, i.e., the number of paths from a source node to a destination node that does not share any common edge. We generate 580 flows. Figure 4.1b presents box plots of the number of edge-disjoint paths per flow. Increasing the number of channels used degrades the route diversity in the resulting topology.

We then investigate how the number of channels used affects routing. Whether the network manager can generate a route successfully for a flow depends highly on the underlying network topology resulting from channel blacklisting. We stipulate that a source route is successfully generated for a flow set if at least one primary route exists for each flow in this flow set. A graph route is successfully generated if at least one primary route exists from the source to the destination of each flow in the flow set and at least one backup route exists from each node along the primary route to the destination. We generate 100 sets of flows by varying the locations of sources and destinations. If a route is successfully generated for all flows in a flow set, then route generation deems successful. The success rate of route generation is defined as the fraction of flow sets with successful routes.

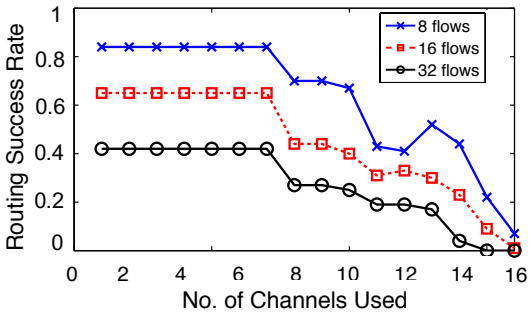
Figure 4.2 presents the success rates of route generation under source and graph routing based on both WUSTL and Indriya testbed topology. The routing success rate decreases with an increasing number of channels used due to its impact on network topology. For instance, when the number of flows is 8, the route generation success rate of source routing lower from 0.84 to 0.07 as the number of channels increases from 7 to 16 (Figure 4.2c). Similarly, with 8 flows, the success rate of graph routing drops from 0.84 to 0.08 when the number of channels used increases from 2 to 13 (Figure 4.2d). The results from both Indriya and WUSTL testbeds are consistent that is the route generation success rate decreases as the number of channels grows due to the reduction in path diversity. Additionally, the number



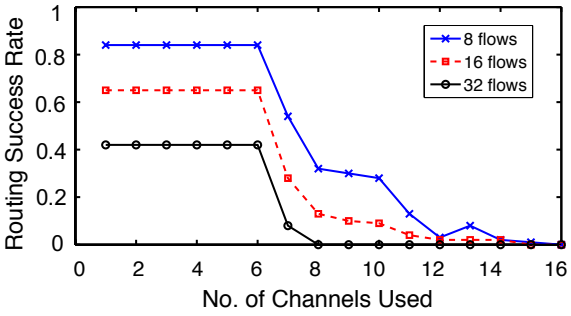
(a) Source Routing (WUSTL).



(b) Graph Routing (WUSTL).



(c) Source Routing (Indriya).



(d) Graph Routing (Indriya).

Figure 4.2: Route generation success rates, based on the shortest path algorithm.

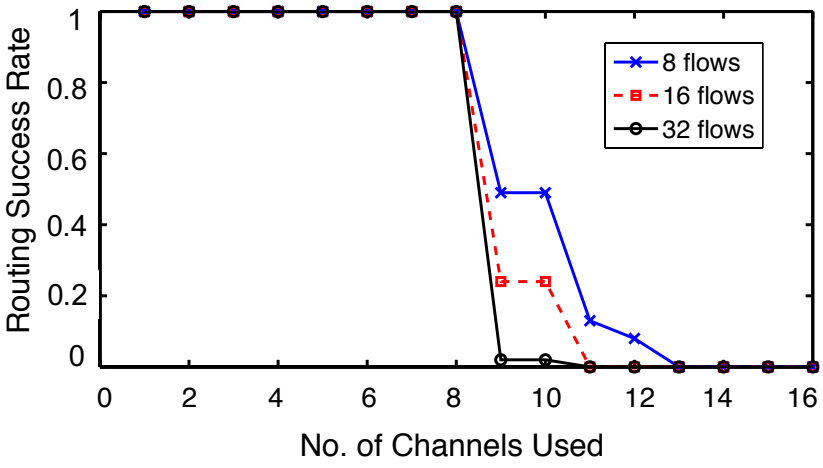


Figure 4.3: Impact of the number of channels used on routing based on the energy-efficient algorithm under WUSTL testbed.

of channels used has a particularly large impact on graph routing, given its dependency on route diversity.

We conduct another set experiment by running the energy-efficient algorithm. The results of this algorithm for both source and graph routing are almost identical to the results obtained from the shortest path algorithm. Figure 4.3 presents an example of the route generation success rate of graph routing based on the energy-efficient algorithm under the WUSTL testbed. Routes generated by the shortest path and the energy-efficient algorithms can be different due to different routing metrics, however both fail to generate a route when there is no path between two nodes.

Observation 1*: More channels used reduce the route diversity of a network, resulting in a lower success rate of route generation. Note that more channels used have the benefit of enhancing channel diversity. Our findings show that channel selection must strike a balance between channel and route diversity.*

4.3.3 Impact on Scheduling

In this subsection, we study how the number of channels used affects transmission scheduling. Once the routes of a flow set are successfully generated, the network manager computes a transmission schedule by assigning time slots and channel offsets to all transmissions of the flows. The transmission schedule generated by the network manager is called a *superframe* and is disseminated to field devices. At run time, field devices repeat the schedule when they reach the end of the superframe.

A flow is *schedulable* if all its packets will meet its deadline, and a set of flows is schedulable if all the flows are schedulable. Flows are scheduled based on priorities assigned by the rate

monotonic algorithm. The scheduling success rate is defined as the fraction of flow sets that are schedulable among flow sets with successful route generation.

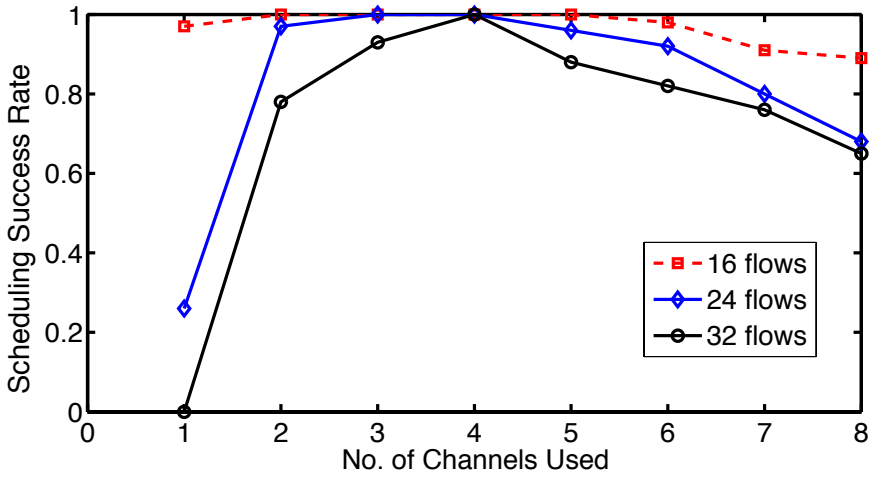
As Figure 4.4a shows, with 32 flows, the scheduling success rate increases when the number of channels used increases from 1 to 4. This is to be expected, because more channels allow more concurrent transmissions. Surprisingly, the trend reverses when the scheduling success rate *decreases* with more channels used. The results for 16 and 24 flows follow the same trend, but at a higher success rate due to lighter load. We observe the same trend in the worst-case packet delays of 32 flows, as shown in Figure 4.4b. The minimum delay occurs when the number of channels is 4. (When the number of channels exceeds 8 in our experiments, the routing success rate with 32 flows is very small. Thus, we do not have sufficient sets of flows to determine the scheduling success rate with more than 8 channels used.)

We find that this phenomenon results from the tradeoff between two sources of delays in a WirelessHART network [42]: channel contention and transmission conflict:

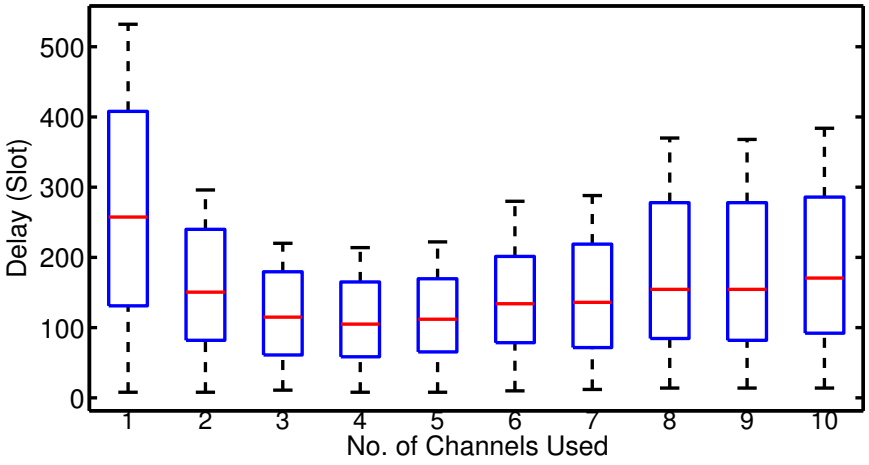
1. *Channel contention*: To avoid interference, WirelessHART allows only one transmission per channel in each time slot. If the number of transmissions scheduled in a time slot exceeds the number of channels, lower-priority transmissions must be delayed to later slots.

2. *Transmission conflict*: Due to the half-duplex nature of IEEE 802.15.4 radios, a node can participate in only one transmission or one reception in the same slot. Consequently, transmissions conflict if they involve a common node as either a sender or a receiver. Then, only one of the conflicting transmissions can be scheduled in that slot.

Figure 4.5a and Figure 4.5b present the number of transmission conflicts and the number of channel contentions in our previous experiment. Interestingly, while channel contention

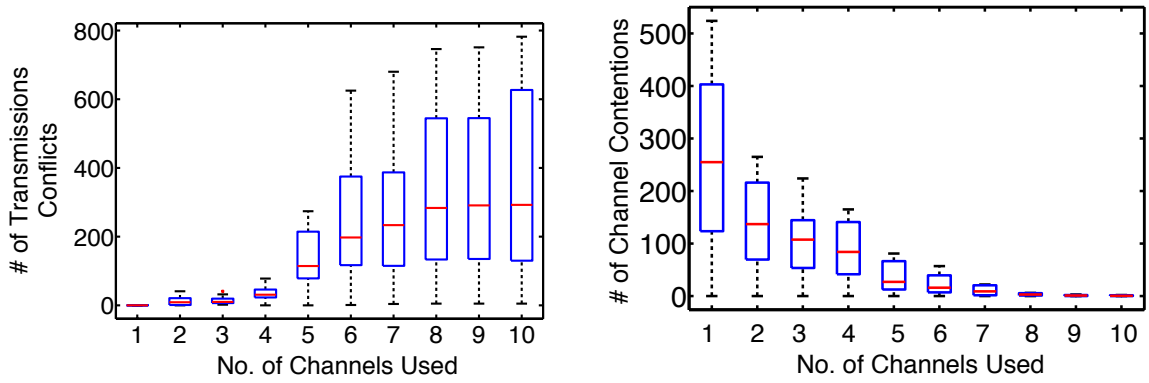


(a) Scheduling success rate under graph routing.



(b) Box plots of worst-case delays of 32 flows under graph routing.

Figure 4.4: Schedule generation success rates and worst-case delays.



(a) Transmission Conflict.

(b) Channel Contention.

Figure 4.5: Number of transmission conflicts and channel contentions of the 32-flow schedule under graph routing.

decreases with more channels used, transmission conflicts increase as more transmissions share nodes, due to reduced route diversity. The median value of transmission conflicts increases from 9 to 283 when the number of channels used increases from 2 to 8, while the median value of channel contentions decreases from 137 to 3 at the same time. We can also observe that after the number of channels reaches 7, there are almost no channel contention. Our results also suggest that transmission conflict dominates when the number of channels used is beyond a certain point, so adding more channels can be detrimental.

Observation 2*: Because both transmission conflict and channel contention affect scheduling, schedulability does not improve monotonically with more channels used; in fact, using more channels may degrade schedulability.*

4.4 Channel Selection

In Section 4.3, a brute-force approach is used to search for the set of channels that retains the maximum number of available links in the network topology. However, this approach is

impractical due to its high computational cost, especially because the network manager needs to adjust the selected channels at run time due to changes in wireless conditions. Moreover, maximizing the total number of links does not always lead to maximum route diversity because different nodes may not contribute equally to route diversity. For example, a node adjacent to fewer links may be more vulnerable to channel selection. If its few links are eliminated due to channel selection, the node may be removed from the network topology. To overcome these challenges, we introduce a new channel selection algorithm, which operates in three steps: (1) removing channels on which critical nodes are poorly connected to the network (**Channel Filtering**), (2) ranking the channels (**Channel Ranking**), and (3) finding the maximum number of channels that can be used to successfully generate the routes and schedule for a given set of flows. In contrast to the brute-force approach, our channel selection algorithm employs a new metric for ranking channels that considers the criticalities of nodes to the network topology.

We model a WSAN as a set of graphs $G = \{G_i(V, E_i) \mid 11 \leq i \leq 26\}$, where i is the channel index [3]. Each node $v \in V$ denotes a device, and each edge $e_i \in E_i$ denotes a bidirectional link with PRRs no lower than a given PRR threshold in both directions on channel i. We define the degree of v on channel i, $D_i(v)$, as the number of edges incident to v on channel i.

Channel Filtering: We first perform channel filtering to ensure that the channel selection retains all critical nodes, including the sources and destinations of flows and the access points connected to the gateway. A channel i is filtered out if there exists a critical node v whose node degree $D_i(v) < 3$ on this channel, as WirelessHART suggests that any device in a network should have at least three neighbors for route diversity [83].

[3] IEEE 802.15.4 standard specifies 16 channels (channel 11 $\sim$ 26) in the 2.4 GHz ISM band [28]

Channel Ranking (CR): For each node, a channel is preferable if the node has a higher degree under the channel. CR normalizes the degree of node v on channel i by dividing $D_i(v)$ by the maximum degree of v among all channels to prevent discrimination against nodes with fewer links. The normalized degree of v is denoted $D'_i(v)$. Furthermore, if a node has fewer good channels, it is more likely to be eliminated from the network after channel selection. Channel i is considered a *good* channel for the node v if the following two conditions are met:

- $D_i(v) > \overline{D_i(v)}$, the degree of node v is larger than the average degree of all nodes on the channel i.
- $D_i(v) > 3$, the node v has more than 3 neighbors.

The number of good channels of a node v is denoted by $n_c(v)$. To avoid eliminating nodes with few good channels in channel selection, we include $n_c(v)$ in the channel score as follows:

$$score(i) = \sum_v D'_i(v)/n_c(v) \tag{4.1}$$

The output of CR is an ordered list of channels $c_1, c_2, ...c_n$ sorted in decreasing order of $score(i)$. A channel with a higher rank potentially enhances route diversity compared to a channel with a lower rank.

Finding the Maximum Number of Channels Used: Because channel hopping enhances reliability through channel diversity, our channel selection approach searches for the *maximum* number of channels that allows routes to be successfully generated and all flows to meet their deadlines. Given a ranked list of n channels generated by the CR, we select the top k (initialized to $k = n$) channels from the ranked list, and then run routing and scheduling algorithms. If the route and schedule generation fail, the algorithm decreases k by 1 and

reruns the routing and scheduling algorithms until it finds k that can successfully generate routes and a schedule for the flow set.

4.5 Link Selection

To meet the reliability requirement of Industrial WSANs, we blacklist links whose PRR is below a threshold in any of the channels used. The choice of the threshold involves a tradeoff between network reliability and route diversity, and a higher threshold may improve reliability at the cost of route diversity. Figure 4.6 shows the impact of different PRR thresholds on the success rate of route generation. In comparison to a 90% PRR threshold, an 80% PRR threshold allows both source and graph routes to achieve a higher route generation success rate. Although a higher PRR threshold leads to a lower success rate, employing a lower PRR threshold may degrade the overall network reliability.

In contrast to traditional link selection based on a *single* PRR threshold, we propose a novel *Channel Pairing (CP)* strategy to maintain reliability while improving route diversity. CP takes advantage of the redundant transmissions offered by WirelessHART, which dictate that both the source and the graph routing perform a second transmission (retransmission) if the first attempt (transmission) fails when delivering a packet. Therefore, instead of using a single PRR threshold to ensure that selected links are reliable across all channels used, CP divides the channels used into two sets and applies two different PRR thresholds to these two sets of channels during link selection. A high PRR threshold is applied to the first set of channels that are dedicated to the first transmissions, while a lower PRR threshold is applied to the other set of channels used for retransmissions. Using more reliable channels for transmissions can reduce the chance of retransmissions, while allowing retransmissions to use

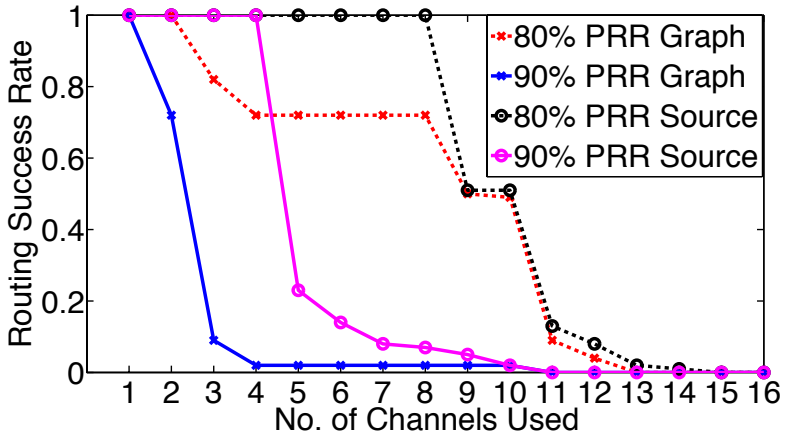


Figure 4.6: Success rates of source and graph route generation with an 80% and a 90% PRR threshold.

less reliable channels helps preserve links and route diversity. CP ensures overall reliability by taking both transmissions into account.

4.5.1 Channel Pairing Algorithm (CP)

The input of CP is a ranked list of k channels obtained from the channel selection algorithm, C_r, and $P_{success}$, which is the required probability of successfully sending a packet after a transmission and a retransmission. In addition, CP allows two PRR thresholds, $PRR1_t$ and $PRR2_t$, for link selection on channels used for transmissions and retransmission, respectively, where $PRR1_t > PRR2_t$.

The outputs of CP are two sets of channels, $C_1 = \{c_{1,1}, c_{1,2}...c_{1,x}\}$ and $C_2 = \{c_{2,1}, c_{2,2}...c_{2,x}\}$, where $|C_1| = |C_2| = x$ and $x = \left\lfloor \frac{k}{2} \right\rfloor$. (If the number of channels used, k, is odd, CP defines c_{back}, which can be used by backup transmissions of graph routing. c_{back} is the middle channels in C_r. Every link in the output set of links L must also have a PRR no lower than $PRR1_t$ on c_{back}.) CP schedules the first transmission on more reliable channels in C_1 and retransmissions on channels in C_2. CP also outputs a set of links, L, selected for use in communication. For

every link $l \in L$, if its first transmission attempt is on channel $c_{1,i}$, then its retransmission must happen on channel $c_{2,i}$. The PRR of a link l on a channel c is defined as $PRR(l_c)$. Every link $l \in L$ must satisfy the following *link quality constraints*:

1. $PRR(l_{c_{1,i}}) \geq PRR1_t$, $\forall c_{1,i} \in C1$
2. $PRR(l_{c_{2,i}}) \geq PRR2_t$, $\forall c_{2,i} \in C2$
3. $1 - (1 - PRR(l_{c_{1,i}}))(1 - PRR(l_{c_{2,i}})) \geq P_{success}$ $\forall c_{1,i} \in C1$ and $c_{2,i} \in C2$ (The two transmissions are independent due to channel hopping.)

Constraint (3) allows a communication over a link to meet the reliability requirement ($P_{success}$) through the transmission and its retransmissions. Constraint (1) ensures that the links have a high PRR in the first transmission and hence reduces the likelihood of the retransmission and saves energy consumption by communication devices. Constraint (2) allows the retransmission to utilize a lower quality link when allowed by constraint (3).

CP works in three steps. It first splits the channels in C_r into two sets. $C1$ contains the top x channels in C_r, and C_2 has the remaining channels. CP considers a set of links L_1, in which each link $l \in L_1$ meets constraint (1). It then computes an average PRR of all links in $L1$ for each channel $c_{1,i} \in C1$ and sorts the channels in $C1$ in increasing order of the average PRRs.

In the second step, for each channel $c_{1,i}$ in a sorted $C1$, CP pairs it with a channel $c_{2,i}$ in $C2$. The selected $c_{2,i}$ pairing with $c_{1,i}$ must maximize the number of links meeting all three link quality constraints. $c_{2,i}$ must also be at least h hops away from $c_{1,i}$. If no such channel exists, $c_{2,i}$ is the channel with the maximum hopping distance from $c_{1,i}$. Hopping to a nearby channel may not be effective in improving reliability due to the significant correlation between transmission failures among adjacent channels [65].

Lastly, let the set of valid links of a pair of channels $c_{1,i}$ and $c_{2,i}$ be $L_{(c_{1,i},c_{2,i})}$. Then, the final output is a set of links $L_{(c_{1,1},c_{2,1})} \cap L_{(c_{1,2},c_{2,2})}, ... \cap L_{(c_{1,x},c_{2,x})}$, where every selected link meets the link quality constraint in every channel pair.

4.5.2 Channel Scheduling for CP

In this section, we describe how to adjust a scheduler to support channel pairing. In WirelessHART networks, the network manager generates one or more superframes, where each superframe comprises a transmission schedule that is repeated in a cyclic fashion. To avoid interference, each time slot can accommodate at most k concurrent transmissions, where k is the number of channels used. At run-time, each sender and receiver pair switches to the same channel to communicate. The standard calculates the channel hopping sequence based on the following formula:

$$LogicalChannel = (ASN + choffset)\%k \tag{4.2}$$

where ASN (Absolute Slot Number) denotes the cumulated slot number since the network starts and k is the number of channels used. The channel offset $choffset$ is assigned by the scheduler during transmission schedule computation ($0 \leq choffset \leq k-1$). It guarantees that no transmissions in the same slot use the same channel. The sender and receiver then map a logical channel on to a physical channel using a common mapping table stored in a node.

In a straightforward link selection policy, where a link must be equally reliable on all channels used, a transmission can use any of these channels. However, CP needs to guarantee that, given two sets of channels C_1 and C_2, if the first transmission attempt over a link uses $c_{1,i} \in C_1$, then its retransmission must use $c_{2,i} \in C_2$. Therefore, we impose a channel

assignment constraint when scheduling a transmission. For transmissions on a primary path, if the first transmission is assigned to a slot number s_1 and channel offset $choffset_1$, then, for its retransmission, the scheduler needs to find a slot s_2 and a channel offset $choffset_2$ such that

$$(s_1 + choffset_1)\%x = (s_2 + choffset_2)\%x \tag{4.3}$$

A channel offset in each slot is in the range $[0, x-1]$, where $x = |C_1| = |C_2|$. The constraint ensures that a transmission and its retransmission always use the logical channel pair $c_{1,i}$ and $c_{2,i}$, where $0 \leq i \leq x-1$. In addition, for the backup transmissions of graph routing, they can use c_{back} or any available channel from C_1 and C_2 since CP relies on the two transmission attempts on a primary path to achieve the desired reliability. c_{back} is preferable to a channel from C_1 and C_2 because it is not used by transmissions on the primary path.

4.6 Evaluation

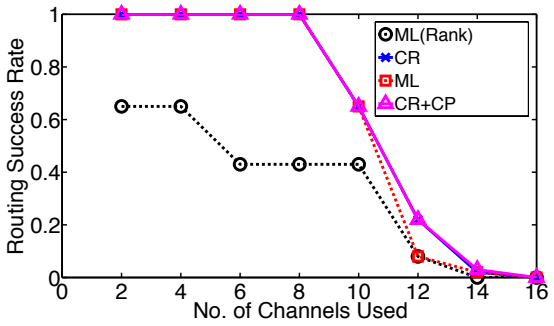
We evaluate our approach in four aspects: (1) routing, (2) scheduling, (3) execution time, and (4) network reliability. To demonstrate the generality of our approach, we evaluate the routing performance using three real network topologies of different sizes and locations: (1) Full Testbed - a 52-node WUSTL testbed deployed in two connected buildings; (2) Half Testbed - part of the WUSTL testbed including only 32 nodes in one of the buildings; (3) Indriya - an open-access testbed deployed on three floors at the National University of Singapore [18], consisting of 86 nodes. We compare our solutions CR alone and CR+CP against two baseline approaches: (1) ML, searching for the set of channels used that maximizes the number of available links by computing all channel combinations; (2) ML(Rank), ranking channels based on the number of available links and then selecting the k highest ranked channels as the channels to be used.

4.6.1 Success Rate of Route Generation

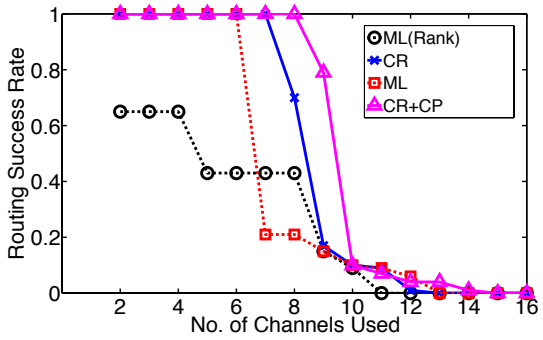
To evaluate the effectiveness of our algorithms in enhancing route generation, we run experiments with 100 flow sets, each of which contains eight flows that have different sources and destinations. We generate routes based on the shortest path algorithm. Because an industrial WSAN usually employs reliable links, we set the PRR threshold to 90% for ML, ML(Rank), and CR. Given a transmission and a retransmission using a uniform PRR threshold of 90%, the overall reliability of a link is 99%. We hence set the $P_{success} = 99\%$ for CP to achieve the same level of reliability. $PRR1_t$ and $PRR2_t$ are set to 90% and 70%, respectively.

Figure 4.7c and Figure 4.7d show the route generation success rates under source and graph routing on the Full Testbed. A more flexible link selection strategy allows CR+CP to significantly outperform ML(Rank), CR, and ML, especially under graph routing, which relies heavily on route diversity. Figure 4.7a and Figure 4.7b show similar results in which CR+CP again outperforms the two baselines on the smaller but better connected Half Testbed. CR+CP on both testbeds significantly improves the route generation success rate when the number of channels used is high (between 6 - 10), thereby allowing the network to maintain more channel diversity.

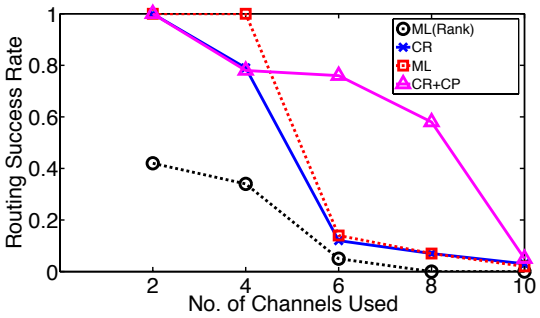
As Figure 4.7e and Figures 4.7f show, on the Indriya testbed, CR+CP likewise achieves a higher success rate of route generation than the two baselines for both source and graph routings. CR+CP offers less improvement in the route generation success rate on the Indriya testbed than on the other two testbeds due to weaker network connectivity. It can also be observed that CR outperforms ML(Rank) on all testbeds for both source and graph routings, due to a more effective ranking criteria. CR also achieves a route generation success rate comparable to ML, but with less complexity. Although CR adopts a better heuristic than ML, CR still has a drawback when compared to ML that explores all combinations of channels.



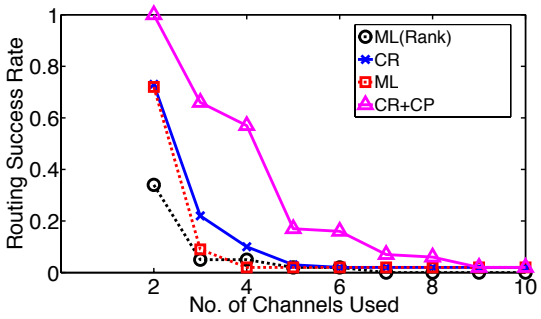
(a) Half Testbed (source routing).



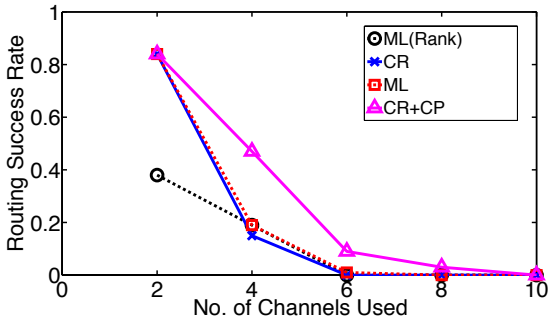
(b) Half Testbed (graph routing).



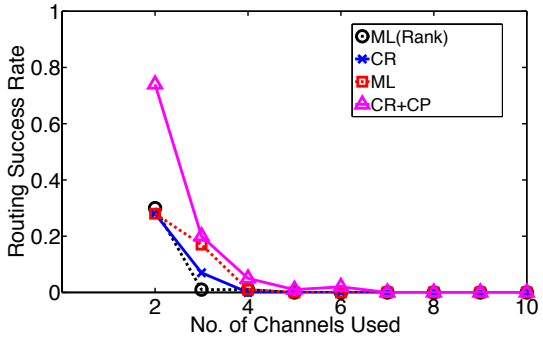
(c) Full Testbed (source routing).



(d) Full Testbed (graph routing).



(e) Indriya Testbed (source routing).



(f) Indriya Testbed (graph routing).

Figure 4.7: Success rates of route generation under source and graph routing routing.

Table 4.1: Channel Selection Algorithm Execution Time (in ms).

Approach	Half Testbed	Full Testbed	Indriya
ML(Rank)	26.6	59.6	190.0
CR	55.7	121.4	324.9
ML	1297.2	2133.8	4676.5
CR+CP	83.5	156.5	420.5

4.6.2 Algorithm Execution Time

We measure the execution time of our algorithms on a MacBook Pro laptop with a 2.7 GHz Intel Core i7. The channel selection process includes finding the best set of channels for each k ($1 \leq k \leq 16$) channels used, and obtaining the output topology containing links that satisfy the reliability requirement. Table 4.1 presents the execution time (in milliseconds) of ML(Rank), CR, ML, and CR+CP on the Half Testbed, the Full Testbed, and the Indriya testbed. (Because the WirelessHART standard [83] suggests the network with a single gateway should comprise at most 80 field devices, the Indriya testbed represents a WirelessHART network with the maximum size.) CR incurs an approximately 2X higher execution time than ML(Rank) in all testbeds due to a more sophisticated ranking criterion. CR+CP consumes up to 3.1X and 1.5X longer execution time than ML(Rank) and CR because it introduces additional computations for pairing channels. ML finds the best set of channels through a brute-force search on all channel combinations, hence it incurs 15.5X, 13.6X, and 11.1X longer execution times than CR+CP on the Half Testbed, the Full Testbed, and the Indriya testbed, respectively.

We then find the maximum number of channels that can be used to generate a schedule. The number of flows significantly affects the execution time of this process. Table 4.2 compares the search time (in milliseconds) consumed by CR+CP and ML on the Indriya testbed.

Table 4.2: Search and Schedule Generation Time (in ms).

Approach	8 flows	16 flows	32 flows
ML	38.47	63.75	125.8
CR+CP	42.07	66.18	120.2

(We limit the number of flows to 32 due to the network size. Almost 80% of the nodes are designated as either a sender or a receiver.) As the number of flows increases, the execution time also grows under both CR+CP and ML. With the same number of flows, CR+CP and ML require similar amounts of execution time. CR+CP is more efficient than ML since it requires less overall execution time, but it achieves a higher route generation success rate.

4.6.3 Network Performance

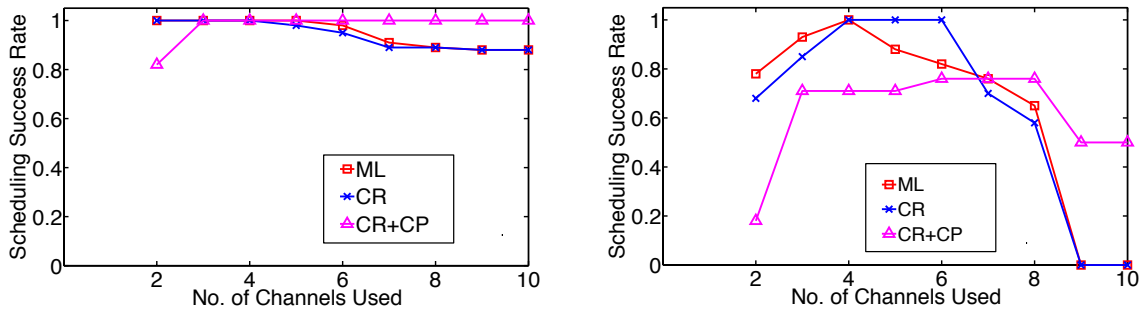
Because CR+CP improves the success rate of route generation by allowing links to use less reliable channels for packet retransmissions, it may degrade the performance of the network. In this set of experiments, we compare the performance of CR+CP against ML, which employs a single PRR link selection ensuring all links are reliable across all channels used. We perform experiments on the Full Testbed to evaluate our approaches in term of the network reliability, the total number of transmissions of each schedule, and the number of transmission attempts per hop. The testbed runs the implementation of key network features of the WirelessHART standard, including TSCH MAC and a routing layer that supports both source and graph routings on TinyOS 2.1.2 and TelosB motes (see Section 3.3.3). For each channel selection approach, we run experiments under graph routing with three different network settings. Each setting contains eight distinct flows and two access points. Each experiment is run for 100 rounds using eight IEEE 802.15.4 channels. Both ML and CR+CP follow formula (4.2) in Section 4.5.2 when computing a channel hopping sequence.

Table 4.3: Network performance comparison under graph routing

Set	Approach	PDR	#Trans / hop	#Trans	#Hops / flow
1	ML(B)	99.25%	1.01	49.56	6.13
	CR+CP	99.88%	1.01	49.43	6.00
2	ML(B)	99.75%	1.02	70.82	8.63
	CR+CP	98.75%	1.05	57.61	6.88
3	ML(B)	99.00%	1.03	44.13	5.38
	CR+CP	98.38%	1.08	42.37	4.88

We use the end-to-end packet delivery ratio (PDR) as the metric to quantify network reliability. As Table 4.3 shows, CR+CP achieves an average PDR lower than that of ML by approximately 1% in all settings, except under setting 1, where CR+CP surpasses ML by 1%. The slight decrease in the average PDR of CR+CP is in exchange for a proportionally much-higher success rate of route generation.

To determine the number of transmissions incurred by CR+CP, we measure the average number of transmission per hop. Graph routing allows two transmission attempts over a link on a primary path, and an additional transmission on a backup link. Table 4.3 shows that in setting 1, CR+CP requires the same average number of transmissions per hop as ML. Nevertheless, CR+CP performs slightly worse than ML, as observed in setting 2 and 3, where CR+CP has 3% and 5% increases in the average number of transmissions per hop compared to ML. We further compute the average total number of transmissions needed for all flows to deliver packets from their sources to destinations, as shown in Table 4.3. As ML requires fewer transmissions per hop than CR+CP, it should require fewer transmissions per schedule. However, the results demonstrate that CR+CP achieves a smaller total number of transmissions in all settings than ML, especially in setting 2, where CR+CP needs 22.9% fewer transmissions than ML. This is due to the fact that CR+CP enhances route diversity and increases the chance of the routing algorithm obtaining a shorter flow path. For instance,



(a) 16 flows. (b) 32 flows.

Figure 4.8: Schedule generation success rates under graph routing on Full Testbed.

in setting 2, CR+CP reduces the average number of hops per flow by 20.2% over ML. Although CR+CP incurs more transmission attempts per hop than ML because of the use of weaker channels for retransmissions, it requires fewer transmissions per schedule than ML and maintains high reliability.

4.6.4 Success Rate of Schedule Generation

Because using more channels may have a negative effect on the scheduling success rate as the number of channels reaches a certain point, we investigate whether a similar trend can be observed under CR and CR+CP. We compare the scheduling success rate of our approaches against ML on the Full Testbed topology. In this set of experiments, we lower the PRR thresholds of ML and CR to 80%. For CP, $P_{success}$ is set to 96% and $PRR1_t$ and $PRR2_t$ are 80% and 70%, respectively. (To evaluate the success rate of schedule generation, the network needs to contain a large number of flows. However, increasing the number of flows under a high PRR threshold, e.g., 90%, significantly decreases the route generation success rate. Hence, there are not enough sets of flows to determine the schedule generation success rate.) We run the experiments with 100 sets of flows.

As shown in Figure 4.8a and Figure 4.8b, CR obtains a scheduling success rate comparable to ML under both 16 and 32 flows. With 32 flows, CR+CP leads to a lower schedule generation success rate than ML when the number of channels is within [2,6]. While CR+CP potentially reduces the number of transmission conflicts through route diversity, it also introduces a channel scheduling constraint that causes transmissions of flows to be delayed and consequently miss their deadlines. In contrast, when the number of channels exceeds 6, CR+CP outperforms ML for both 16 and 32 flows because CR+CP can fully take advantage of route diversity since it has enough channels to use. Note that when the number of channels reaches 9, both CR and ML can no longer successfully generate routes; therefore, their scheduling success rates drop to 0. CR+CP improves the success rate of schedule generation when there are more channels used, hence it enhances channel diversity. The number of channels used also has a similar impact on the success rate of schedule generation under CR and CR+CP. CR and CR+CP achieve the maximum success rate of schedule generation when the number of channels is within [4,6] and [6,9] respectively. Judicious selection of the number of channels is important to meet the timing constraints of WSANs. By exploring different numbers of channels automatically, our approaches can configure the channels efficiently.

4.7 Summary

In this chapter, we investigate the tradeoff between channel and route diversity, based on empirical studies on testbeds. Our findings reveal that while using more channels means more channel diversity, a large number of channels may reduce route diversity, with negative impacts on routing and scheduling. As a result, using more channels is not always desirable in industrial WSANs. Based on these insights, we present novel channel and link selection algorithms that automatically select channels to facilitate routing and scheduling. Experimental results show

that our solution significantly improves the success rates of routing and scheduling, while maintaining network reliability.

Chapter 5

Conservative Channel Reuse in Real-Time Industrial Wireless Sensor-Actuator Networks

Industrial applications impose stringent requirements in reliability and real-time performance on WSANs. To enhance reliability, industrial standards, such as WirelessHART, embrace Time Slotted Channel Hopping (TSCH) that integrates channel hopping and TDMA at the MAC layer. Within a network governed by a same gateway, WirelessHART prohibits *channel reuse*, i.e., concurrent transmissions in the same channel, to avoid interference between concurrent transmissions. Preventing channel reuse however negatively affects real-time performance. To meet the demand for both reliability and real-time performance by industrial applications, we propose a *conservative* channel reuse approach designed to enhance the real-time performance while limiting its impact on reliability in WSANs. In contrast to traditional channel reuse designed to optimize performance at the cost of reliability,

our conservative approach introduces channel reuse only when needed to meet the timing constraints of flows. Finally, we present an algorithm to detect reliability degradation caused by channel reuse so that channels can be reassigned to further improve reliability. Experimental results based on two physical testbeds show that our approach significantly improves real-time performance while maintaining a high degree of reliability [4].

5.1 Introduction

In contrast to traditional wireless sensor networks that require only best-effort service, industrial To meet such stringent requirements, WirelessHART [83], incorporate a set of salient features. To achieve reliable communication, channel reuse is prohibited within a network based on the same gateway, i.e., in the same time slot only one transmission is allowed in each channel. However, this policy can severely affect the real-time performance and the capacity of the network.

While channel reuse has been explored in wireless sensor networks, traditional approaches are unsuitable for industrial WSANs with stringent reliability requirements. Earlier solutions either deal with interference between concurrent transmissions in a best-effort manner or schedule transmissions based on estimated interference. In practice, estimating interference incurs significant overhead and errors, especially in the presence of temporal variations in real-world environments. Moreover, traditional approaches are usually designed to *maximize* channel reuse. This *aggressive* approach to channel reuse, combined with an inaccurate interference estimate, may cause unacceptable degradation in network reliability for industrial applications.

[4]The contents in this chapter have appeared in: D. Gunatilaka and C. Lu, Conservative Channel Reuse in Real-Time Industrial Wireless Sensor-Actuator Networks, ICDCS, July 2018.

This chapter explores channel reuse for industrial WSANs that demand both real-time performance and reliability. The contributions of this work are four-fold.

- We propose a *conservative* channel reuse approach that introduces channel reuse only when necessary to meet the timing constraints of existing data flows.
- We design *Reuse Conservatively (RC)*, a conservative channel reuse algorithm that incorporates channel reuse to meet the real-time performance constraints while mitigating its impact on reliability.
- We develop a classifier to detect links with poor reliability due to channel reuse so that these links can be rescheduled to enhance network reliability.
- We present experimental results based on two testbeds that demonstrate RC significantly improves real-time performance when compared to the standard WirelessHART approach without channel reuse, while maintaining higher reliability than aggressive channel reuse.

The rest of the chapter is organized as follows. Section 5.2 reviews related work. Section 5.3 introduces the problem. Section 5.4 details the channel reuse algorithm. Section 5.5 describes the classifier to detect reliability degradation due to channel reuse. Section 5.6 presents the experimental results. Section 5.7 summarizes the chapter.

5.2 Related Work

There have been extensive studies on interference measurements in the past years. Maheshwari et al. [43] studied different interference models including hop-based, range-based, SINR-based, and PRR-SINR models. Liu et al. [40] modeled interference based on the PRR-SINR

relationship. Qiu et al. [56] used a measurement-based model to estimate throughput, and Zhou et al. [89] proposed a radio interference detection protocol based on a thresholded-SINR model. Although SINR-based models have been shown to be more realistic than other approaches [67], estimating interference based on such models involves multiple measurements (e.g., receiver sensitivity, noise, aggregated interferences, etc.), which, in practice, incurs significant overhead and complexity. Hence, we opt for mitigating interference by increasing the hop distance between concurrent transmissions on the same channel (Other protocols adopting the hop-based model include [6, 70].). More importantly, we reduce the impact of channel reuse on reliability by introducing channel reuse only when needed to meet the real-time performance requirements of existing flows.

Several TDMA protocols leverage channel reuse to enhance network capacity. Gronkvist et al. [24] and Brar et al. [8] proposed scheduling policies based on spatial reuse TDMA protocol to maximize network throughput. The works presented in [11, 13, 48, 60] designed real-time scheduling strategies, which incorporate channel reuse, to optimize network throughput and latency. In contrast, our approach conservatively adopts channel reuse to avoid its impact on network reliability. Notably, since WSAN traffic loads are known by the centralized network manager in industrial settings, we therefore only need enough channel reuse to meet the real-time requirements of existing workloads.

There also exist scheduling algorithms customized for TSCH networks that support channel reuse. Palettella et al. [53] proposed TASA, a centralized traffic-aware scheduling algorithm that improves network latency and power efficiency. Accettura et al. [1] developed DeTAS, a decentralized scheduling protocol for the 6TiSCH [31] network. In contrast to our work, both algorithms *always* reuse channels when transmissions are estimated not to interfere with each other, while our approach introduces channel reuse only when needed to meet the real-time requirements of the workload. Duquennoy et al. [20] presented Orchestra, an

autonomous scheduling solution for 6TisCH networks. Transmissions may be mapped to the same slot on the same channel, and hence they must contend for a channel. While Orchestra incurs channel reuse in a best-effort manner, our approach manages channel reuse to reduce interferences by (1) introducing channel reuse conservatively and (2) avoiding interference by judiciously scheduling channel reuse for concurrent transmissions.

A recent direction in wireless sensor networks is to exploit synchronous transmissions. The Glossy-based approach [23] provides a promising alternative to TSCH-based networks. It offers efficient and fast network flooding by exploiting constructive interference and capture effect allowing packets to be received successfully in the presence of concurrent transmissions. Zimmerling et al. [90], to the best of our knowledge, was the first to combine Glossy-based Low-Power Wireless Bus [22] and real-time scheduling. Our solution is based on TSCH that has been adopted by the industry. Our channel reuse approach also relies on the capture effect to allow concurrent transmissions on a channel, and therefore it represents a step to incorporate capture effects into industrial standards such as WirelessHART.

5.3 Problem Description

In this section, we first describe fixed-priority scheduling, and define the communication and channel reuse graphs of a network. Finally, we set the objectives of our channel reuse algorithm.

5.3.1 Fixed Priority Scheduling

Our work follows the flow model describe in Section 2.2 where a WSAN is shared by a set of real-time flows $F = \{F_1, F_2, ...F_n\}$, and each flow is characterized by a source S_i, a destination Y_i, a route ϕ_i, a period P_i, and a deadline D_i. A flow F_i is schedulable if all of its packets are

scheduled to meet their deadlines. A set of flows F is schedulable only if all flows in F are schedulable.

In a *fixed priority scheduling policy*, each flow F_i is assigned a fixed priority. A flow F_i has higher priority than a flow F_k if $i < k$. Priorities are commonly assigned based on deadlines or periods. The fixed priority scheduler schedules transmissions of higher priority flows before transmissions belonging to lower priority flows. It allocates the earliest *feasible* slot to each transmission. A feasible slot is subject to no *transmission conflict* and each channel is restricted by *channel constraint*. Two transmissions *conflict* if they share a common node, either a sender or a receiver. This is because of the half-duplex nature of the IEEE 802.15.4 radios. Within the same network, the WirelessHART protocol does not support channel reuse. Hence, following the standard, the number of transmissions in each slot must not exceed the number of channels used. However, our approach extends WirelessHART by allowing multiple transmissions to share the same channel in the same slot.

5.3.2 Communication Graph vs Channel Reuse Graph

We consider two types of graphs in our network: (1) the communication graph that is used for constructing routes from a source to a destination for each flow and (2) the channel reuse graph that is for estimating interferences when channel reuse is adopted.

A communication graph $G_c(V, E)$ consists of a set of network devices V, and a set of links E. We use the Packet Reception Ratio (PRR), a ratio of successfully received packets to the total number of transmitted packets, to assess link quality. Let M be a set of channels used and let $PRR(\vec{xy})_i$ be the PRR of a link $\vec{xy}$ on channel $i \in M$. A bidirectional edge uv is in E if $PRR(\vec{uv})_i \geq PRR_t$ and $PRR(\vec{vu})_i \geq PRR_t$ for all channels $i \in M$ where PRR_t is a link selection threshold. An edge must be bidirectional to support a transmission and its

acknowledgement. Due to channel hopping, links hop through all channels so they must be reliable in all channels used.

Our channel reuse policy adopts a hop-based interference model in which concurrent transmissions are allowed if a sender of a transmission is at least k hops away from a receiver of a concurrent transmission. Therefore, we construct a channel reuse graph, which determines the channel reuse hop count between nodes. We represent the channel reuse graph as graph $G_R(V, E)$, where V is a set of network devices and E is a set of edges between two devices. Due to the challenge in directly measuring interference, we define an edge in the channel reuse based on its PRR, which is already collected by the network manager in order to construct the communication graph in the WirelessHART network. Specifically, a bidirectional edge uv exists in E if for ***any*** channel $i \in M$, $PRR(\vec{uv})_i > 0$ or $PRR(\vec{vu})_i > 0$. Note that we consider any channel here because each link cycles through all channels through channel hopping. As the hop count between u and v increases, interference is likely to diminish. We denote a minimum channel reuse hop count between any node u and v as ρ.

5.3.3 Objectives of Channel Reuse Policy

Given a real-time WSAN and a set of flows, the objective of our channel reuse algorithm is to introduce concurrent, channel-sharing transmissions such that all flows will meet the deadlines. Note that, if no channel reuse is needed to meet the deadlines, no concurrent transmissions will be scheduled on the same channel. Channel reuse occurs only when needed to make the deadlines. To further mitigate interference caused by channel reuse, our secondary objectives are to (1) increase the hop distance between transmissions sharing a channel and (2) reduce the number of concurrent transmissions per channel. The larger the hop distance between concurrent transmissions, the more likely for both transmissions to succeed due to capture effect. Scheduling more transmissions on the same channel can make transmission failures

more likely because interferences are cumulative [40, 43]. Therefore, it is desirable to schedule fewer concurrent transmissions on each channel.

5.4 Channel Reuse Algorithm

In this section, we first introduce the channel reuse constraints, the concept of flow laxity, and then provide a detailed description of our channel reuse algorithm.

5.4.1 Channel Reuse Constraints

The channel reuse constraints set up conditions that a transmission must satisfy in order to be assigned a specific time slot and channel offset by a scheduler. The maximum number of slots in a schedule is determined by the hyper-period of a set of flows F, which is the least common multiple of the periods of all the flows. Let $|M|$ be the number of channels used. A channel offset is in the range $[0, |M| - 1]$. Let $H(F_i)$ be a set of flows with higher priority than F_i. At slot s, T_s denotes a set of scheduled transmissions of $H(F_i)$. For each channel offset associated with slot s, $T_{sc} \subset T_s$ is a set of scheduled transmissions assigned to channel offset c. A transmission t_{ij} can be scheduled in s, if t_{ij} does not conflict with other transmissions in T_s (see Section 5.3.1). If two transmissions are assigned to a same slot and channel, they must be at least ρ hops apart on $G_R(V, E)$. Therefore, any transmission t_{ij} of F_i can be allocated a slot s and a channel offset c if the following *channel reuse constraints* are satisfied:

1. *Transmission conflict*: $t_{ij} = uv$ does not conflict with $xy \in T_s$: $u \neq x$ and $u \neq y$ and $v \neq x$ and $v \neq y$, for all $xy \in T_s$.

2. *Channel constraints*:

(a) If channel reuse is not allowed ($\rho = \infty$): channel offset c must not have been assigned to any transmission, $T_{sc} = \emptyset$.

(b) If channel reuse is allowed ($\rho < \infty$): given $t_{ij} = uv$ and $xy \in T_{sc}$: u must be at least ρ hops from all y and all x must be at least ρ hops from v.

5.4.2 Flow Laxity

A flow laxity helps anticipate if a packet belonging to a flow can be delivered to meet its deadline or not. It indicates how much delay a packet can tolerate. The higher the flow laxity, the more a transmission can be delayed to later slots. The concept of laxity has been used in [48] and [63] as a heuristic for real-time scheduling. Our algorithm uses flow laxity to assess if channel reuse is needed or if the current channel reuse hop distance will allow a flow to meet its deadline or not. Given a slot s that meets the *channel reuse constraints* for t_{ij}, a set of remaining transmissions of a flow F_i after t_{ij} is denoted as $T_{post,t_{ij}}$. A *flow laxity* is defined as the number of remaining time slots minus the number of slots required to schedule transmissions in $T_{post,t_{ij}}$. Let d_i be a slot corresponding to F_i's deadline D_i. For each transmission $t \in T_{post,t_{ij}}$, we estimate the number slots in an interval $[s+1, d_i]$ that contains transmissions conflicting with t denoted as q^t_{s+1,d_i}. Hence, the total number of slots in $[s+1, d_i]$ conflicting with transmissions in $T_{post,t_{ij}}$ is $\sum_{t \in T_{post,t_{ij}}} q^t_{s+1,d_i}$. The minimum number of slots required to schedule $T_{post,t_{ij}}$ is $|T_{post,t_{ij}}|$. Therefore, the laxity of F_i when scheduled t_{ij} at s is:

$$[s - d_i] - \sum_{t \in T_{post,t_{ij}}} q^t_{s+1,d_i} - |T_{post,t_{ij}}| \tag{5.1}$$

where $s - d_i$ is the number of slots in $[s+1, d_i]$, and $\sum_{t \in T_{post,t_{ij}}} q^t_{s+1,d_i}$ is the estimated number of conflicting slots that cannot be allocated to transmissions in $T_{post,t_{ij}}$. Our channel reuse policy attempts to select slot s for t_{ij} such that F_i's laxity is no less than 0. This means that after scheduling t_{ij} at slot s, there are enough slots to accommodate the remaining transmissions of F_i such that F_i can deliver a packet to the destination within its deadline.

5.4.3 Reuse Conservatively Algorithm (RC)

Algorithm 2: Reuse Conservatively (RC)

Input : A flow set F ordered by priority, a conflict graph $G_R(V, E)$, the minimum channel reuse hop distance ρ_t

Output : A transmission schedule S

```
1  for each flow F_i from F_1 to F_N do
2      ρ ← ∞ ;
3      for each transmission t_ij of F_i do
4          while ρ ≥ ρ_t do
5              < s, c >= findSlot(t_ij, ρ);
6              laxity = calculateLaxity(t_ij, s);
7              if laxity ≥ 0 then
8                  break;
9              else
10                 if ρ = ∞ then
11                     ρ ← λ_R;
12                 else
13                     ρ ← ρ − 1;
14     if s ≤ d_i then
15         Schedule t_ij at < s, c >;
16     else
17         return ∅ ;
18 return S ;
```

In this section, we provide a detailed description of our *Reuse Conservatively Algorithm (RC)*, which integrates our channel reuse policy with a real-time fixed priority scheduling

as presented in Algorithm2. The input of the algorithm is a set of flows F, a channel reuse graph $G_R(V, E)$, and the minimum channel reuse hop count threshold ρ_t. A larger ρ_t will lead to more reliable communication, but may reduce the capacity of the network since channel reuse will be adopted in a more restrictive fashion. In practice, to maintain reliability, a network operator may select the largest channel reuse hop count under which the workload is schedulable. When the workload is schedulable without channel reuse, RC will not introduce channel reuse (i.e., the hop count is effectively infinity).

The output is a transmission schedule S. We schedule flows based on their priorities from the highest to the lowest priority. The ρ variable keeps the current minimum channel reuse hop distance. For each transmission t_{ij}, ρ is first initialized to ∞, which means no channel reuse is allowed.

For every transmission t_{ij}, the algorithm first executes $findSlot()$. This function finds the earliest slot s and channel offset c that comply with the *channel reuse constraints* (see Section 5.4.1) given the channel reuse hop count ρ. It then calculates flow laxity following Equation 5.1 given t_{ij} is scheduled at s. If the computed $laxity \geq 0$, then t_{ij} is scheduled at slot s and assigned channel offset c, and the algorithm proceeds to the next transmission. Otherwise, the algorithm introduces channel reuse for t_{ij} . We define the network diameter as the maximum shortest distance between two nodes in the network. When channel reuse is adopted, ρ is set to λ_R (i.e., the network diameter of $G_R(V, E)$), which gives the maximum channel reuse distance for this network. The algorithm decreases the channel reuse hop count ρ and repeats the $findSlot()$ and $calculateLaxity()$ steps until the calculated $laxity \geq 0$ or until ρ is less than the minimum threshold ρ_t. The algorithm terminates when all transmissions of a flow set F are scheduled, or when any scheduled transmission fails to meet its deadline, which deems F unschedulable.

The algorithm adopts channel reuse conservatively, i.e., when $laxity < 0$. It introduces channel reuse with a larger hop count first to mitigate interference. Note that when there exist multiple channel offsets in slot s that meet the channel reuse constraints, the algorithm chooses a channel with the fewest number of scheduled transmissions to reduce channel contention.

Complexity: There can be at most $N * L$ transmissions scheduled where N is the size of F, and L is the maximum path length among all flows. For each transmission, $findSlot()$ is executed at most λ_R+1 times. For each slot, the algorithm checks the channel reuse constraints, which takes $N*(|E|+|V|log|V|)$ because there can be no more than N transmissions scheduled in each slot, and channel reuse hop count computation on a graph $G_R(V, E)$ takes $|E| + |V|log|V|$. The upper-bound complexity of $findSlot()$ is $O(\lambda_R * P_{max} * N * (|E| + |V|log|V|))$ where P_{max} is the maximum possible number of slots in a schedule, which associates with the hyper-period of F. The total complexity of our channel reuse policy integrating with a fixed priority scheduling algorithm is $O(N^2 * L * \lambda_R * P_{max} * (|E| + |V|log|V|))$.

5.5 Detecting Reliability Degradation Caused by Channel Reuse

While conservative channel reuse can effectively mitigate the impacts of channel reuse on reliability, it may lead to reliability degradation in some cases. In such cases, we need to detect links that suffer poor reliability caused by channel reuse, so that these links can be reassigned to different channels or time slots.

A naive approach is to use a PRR threshold to identify links affected by channel reuse, i.e., if the PRRs of the links associated with channel reuse are lower than the PRR threshold, then

channel reuse may have caused packet loss over these links. However, it is important to note that channel reuse is not the only possible cause of transmission failures. Dynamic changes in channel or environmental conditions and external wireless interference may lead to changes in link reliability as well. It is therefore important to distinguish whether reliability degradation is caused by channel reuse or other factors so that the network manager can resolve the link quality degradation according to the root causes. For example, if a link's PRR drops because of external interference instead of channel reuse, removing channel reuse involving that link would not necessarily improve its quality. Henceforth, we propose a method to distinguish whether a link's reliability degradation is attributed to channel reuse.

We address this challenge by comparing a link's PRR in the time slots when it shares a channel with others and its PRR in the other time slots when there is no channel reuse. Intuitively, if a link is highly reliable in a contention-free channel, but performs poorly when channel reuse is introduced, then the link's reliability is degraded due to channel reuse. Moreover, we minimize detection overhead by leveraging statistics already collected by the network manager of a WirelessHART network.

Each wireless node maintains a neighbor table, which contains statistics pertaining to connectivity between a node and its neighbors, e.g., PRRs, in all channels used. A node learns these statistics based on regular data transmissions and periodic neighbor-discovery packets, which are used to update the topology of the network. The network manager must also reserve enough slots for each node to broadcast neighbor-discovery packets in all channels used. Since the network topology does not change so frequently, these periodic neighbor-discovery packets do not need to be scheduled very often. Based on the schedule computed by the network manager, each node has knowledge of time slots, channel offsets, and its links, which are associated with channel reuse. For each link involved in channel reuse, a node gathers link statistics in two cases: (1) when a transmission on the link occurs without channel reuse and

(2) when a transmission over the link shares a channel with other transmissions. A node periodically reports these statistics to the network manager, which then detects links affected by channel reuse based on the following statistical test.

To detect links affected by channel reuse, our detection policy adopts the *Kolmogorov-Smirnov test (K-S test)* [35] to quantify the difference of link qualities under these two cases. *K-S* test offers the benefits of making no assumption about the distribution of data and making no restriction on the data set size. A two-sample *K-S* test evaluates the difference between two Empirical Cumulative Distribution Functions (ECDF) of two data sets. In our case, we compare the PRR distributions of a link l_i when a transmission over l_i is scheduled with channel reuse (denoted $PRR\_DIST_r(l_i)$), and when a transmission takes place without channel reuse (denoted $PRR\_DIST_{cf}(l_i)$). In a statistical test, under the *null hypothesis*, there is no significant difference between two sample sets. *K-S* test computes the maximum distance between ECDF of $PRR\_DIST_r(l_i)$ and $PRR\_DIST_{cf}(l_i)$. It calculates the *p-value* $\in (0, 1)$, which determines the result of the hypothesis test. The output of the *K-S* test is either *accepting* or *rejecting* the null hypothesis at α significance level. The null hypothesis is rejected when the p-value$< \alpha$. Therefore, for our detection policy, if the outcome of the *K-S* test is *reject*, then we conclude that channel reuse degrades the reliability of link l_i since the $PRR\_DIST_r(l_i)$ differs significantly from $PRR\_DIST_{cf}(l_i)$. Otherwise, channel reuse only has subtle or no impact on l_i.

Our detection policy considers only links associated with channel reuse. Its goal is to identify links whose reliability is degraded by channel reuse. Let $PRR_r(l_i)$ be the PRR of $PRR\_DIST_r(l_i)$. The policy first uses a PRR threshold PRR_t to identify links that fail to meet the reliability requirement, i.e., links with $PRR_r(l_i) < PRR_t$. It then performs *K-S* test on such links to determine if reliability degradation is caused by channel reuse or not. The following is the detection policy:

- If $PRR_r(l_i) < PRR_t$, then perform *K-S* test on $PRR\_DIST_r(l_i)$ and $PRR\_DIST_{cf}(l_i)$.
  - If *K-S* test returns *reject*, then channel reuse degrades link reliability.
  - If *K-S* test returns *accept*, then l_i fails to meet the reliability requirement due to other reasons.
- Otherwise, l_i meets the reliability requirement.

5.6 Evaluation

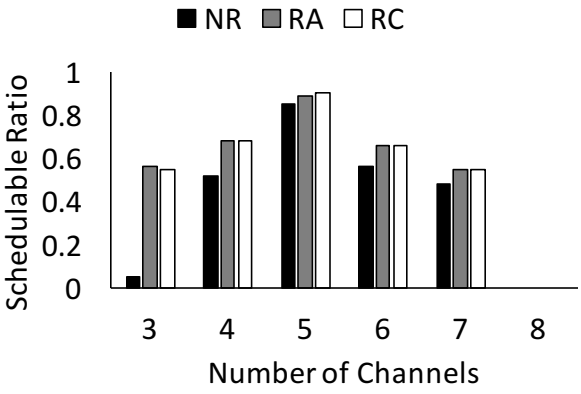
We evaluate our algorithm in terms of the following aspects: (1) real-time performance (2) algorithm efficiency, i.e., the amount of channel reuse and channel reuse hop count (3) execution time (4) network reliability (5) applicability of our detection policy in identifying links whose reliability is degraded by channel reuse. To demonstrate the generality of our approach, (1)-(3) are evaluated based on the collected topologies from: (a) the 80-node Indriya testbed, deployed at the National University of Singapore [18] (b) the 60-node WUSTL testbed deploying across 3 floors. The topology information includes the PRRs of all links in the network in all 16 channels. Based on the network topology and the traffic requirements, the network manager constructs a communication graph (with $PRR_t = 0.9$) and a channel reuse graph. Then, it generates routes and a schedule, which all field devices must follow during run time. To measure reliability performance, evaluation (4) and (5) are conducted on the WUSTL testbed, which runs the WirelessHART protocol on TinyOS 2.1.2 and TelosB motes (See Section 3.3.3).

We randomly generate a set of flows F by varying source and destination nodes. Each flow set contains two access points, which are nodes with a high number of neighbors. Following common characteristics of process monitoring and control applications in process industries, sources of flows generate packets periodically, and the periods of flows are harmonic. The

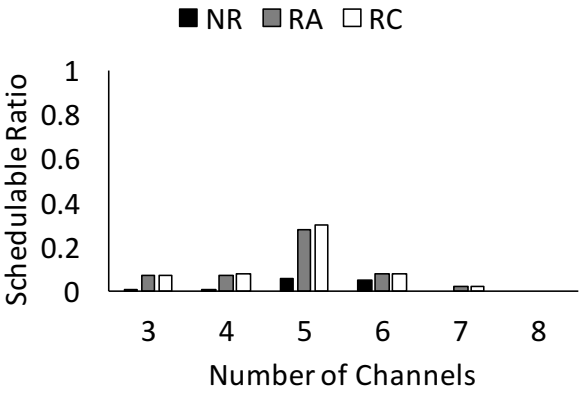
periods are uniformly selected from the range $P = [2^x, 2^y]$ where $x < y$. Given the period range $P = \{2^x, 2^{x+1}, ...s^{x+k}\}$, if a flow F_i is assigned with $P_i = 2^j$, then its deadline D_i is randomly picked from a range $[2^{j-1}, 2^j]$.

For each flow $F_i \in F$, the network manager generates a single route from a source to a destination node based on the shortest path algorithm and the types of traffic. We consider two types of traffic for control applications: (1) *centralized* traffic: a packet is generated by a source (sensor), and routed to a controller through an access point wired to a gateway; then a control message issued by a controller, running on a computer behind the gateway, is delivered to a destination (actuator) and (2) *peer-to-peer* traffic: to enhance scalability, controllers are implemented on field devices in the network, and a data packet can be delivered directly from a source to a destination node without going through the gateway. We run experiments under source routing, which enables every link along the route to retransmit a packet if the first transmission attempt fails. Hence, a scheduler must reserve one more time slot for every transmission over a link.

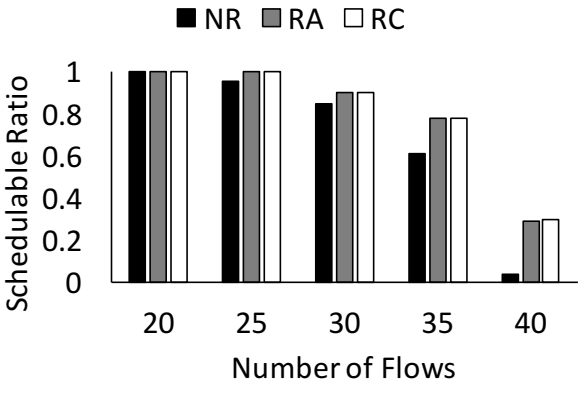
We adopt the *Deadline Monotonic (DM)* algorithm, which is a commonly used fixed priority scheduling policy for real-time systems. Under DM, a flow with the shortest deadline has the highest priority. Our algorithm RC is compared against: (1) DM with no channel reuse (NR): only one transmission can be scheduled on each channel, and (2) DM with aggressive channel reuse (RA): a channel is reused whenever possible, i.e., a scheduler schedules transmissions at the earliest slot and on a channel with at least ρ channel reuse hops. RA adopts similar channel reuse approaches as traditional channel reuse strategies, which always introduce channel reuse when an interference model estimates that concurrent transmissions will not interfere with each other. In addition, RA is also similar to TASA [53], a centralized TSCH scheduling approach that *always* allows transmissions to share a channel when $\rho \geq 2$. For a fair comparison, we set the minimum channel reuse distance ρ_t for RC to 2.



(a) #flows=30, $P = 2^{0\sim2}$



(b) #flows=30, $P = 2^{-1\sim3}$



(c) #ch=5, $P = 2^{0\sim2}$

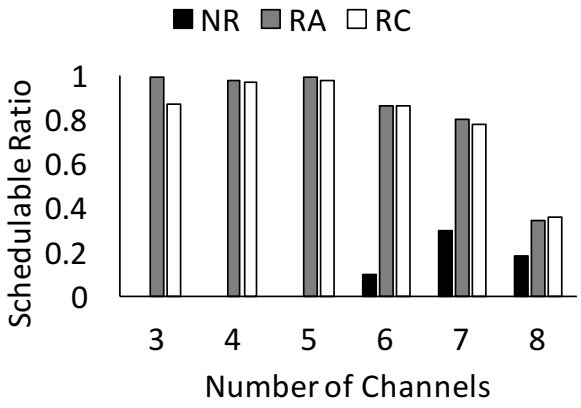
Figure 5.1: Schedulable ratios under varying number of channels and flows based on the centralized traffic (Indriya).

5.6.1 Real-Time Performance

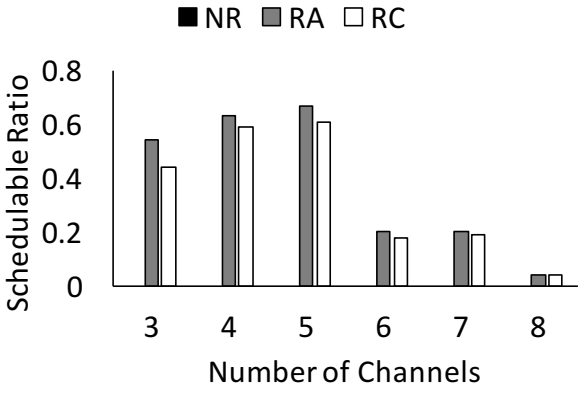
We evaluate the effectiveness of our channel reuse policy in enhancing real-time performance by using a schedulable ratio as a metric. A schedulable ratio is a fraction of flow sets that is schedulable. For each experiment, we generate 100 different flow sets.

Figures 5.1 and 5.2 present schedulable ratios based on the Indriya testbed topology. We compare RC against NR and RA under a varying number of channels, flows, and periods. Figures 5.1a, 5.1b, 5.2a, and 5.2b show RA and RC consistently outperform NR, especially when there are a limited number of channels, i.e., when the number of channels is from 3 to 5. Under the peer-to-peer traffic and the period range $P = [2^{-1}, 2^{3}]$, NR cannot successfully generate any schedule for any number of channels as shown in Figure 5.2b. Note that increasing the number of channels does not always monotonically improve schedulability. Although more channels enhance network capacity, at the same time, they can degrade route diversity in the network and introduce more transmission conflicts as observed in our previous study (See Section 4.3.3).

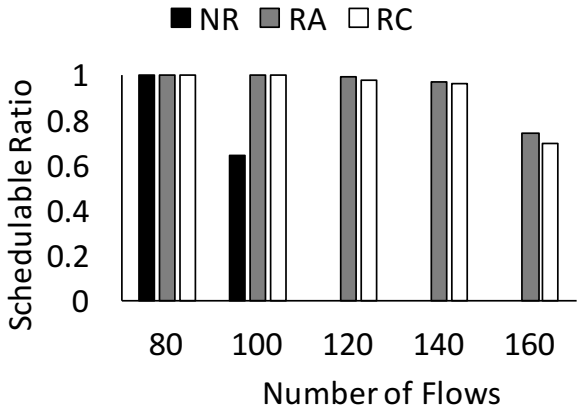
Both RC and RA also yield higher schedulable ratios compared to NR as the number of flows increases. Figure 5.1c shows, under the centralized traffic, RA and RC achieve the maximum of a 7.5X higher schedulable ratio than NR when the number of flows is 40. Figure 5.2c also illustrates that with the peer-to-peer traffic, as the number of flows reaches 120, NR can no longer successfully generate schedules, while RA and RC still obtain almost a 100% schedulable ratio. We also note that a flow path length based on the peer-to-peer traffic is approximately two times shorter than the path length of a flow under the centralized traffic. The results demonstrate the benefits of channel reuse in enhancing scalability of the network under heavy workloads and few available channels. Notably, with lower traffic load, indeed channel reuse is not needed since flows can be scheduled easily to meet their deadlines. We



(a) #flows=120, $P = 2^{0\sim2}$

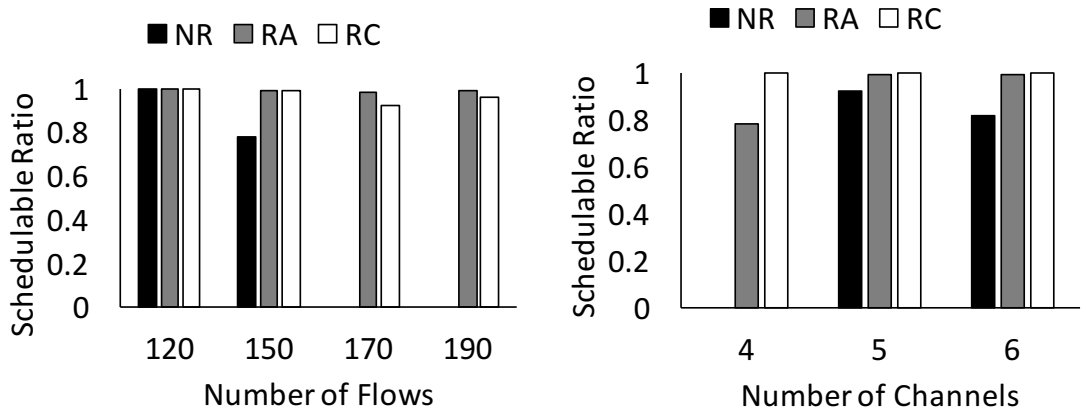


(b) #flows=120, $P = 2^{-1\sim3}$



(c) #ch=5, $P = 2^{0\sim2}$

Figure 5.2: Schedulable ratios under varying number of channels and flows based on the peer-to-peer traffic (Indriya).



(a) #flows=150, $P = 2^{0\sim2}$ (b) #ch=5, $P = 2^{0\sim2}$

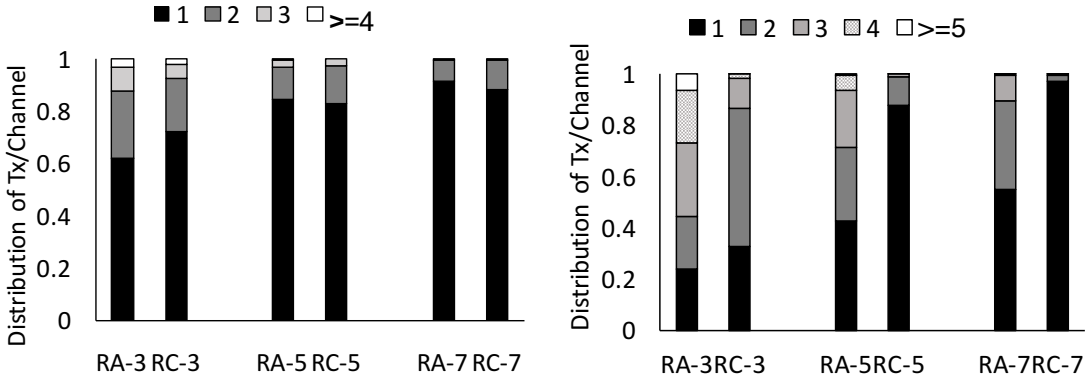
Figure 5.3: Schedulable ratios under varying number of channels and varying number of flows based on the peer-to-peer traffic (WUSTL).

repeat the experiments based on the WUSTL testbed topology, and observe similar results as presented in Figure 5.3.

It can also be noticed that channel reuse is more effective under the peer-to-peer traffic since it enables both RA and RC to outperform NR with a significantly larger margin compared to the centralized approach. The centralized traffic enforces all packets to be routed through access points. As a result, it potentially introduces more transmission conflicts near the access points, which make channel reuse more difficult. Moreover, in most cases, RC can achieve a schedulable ratio comparable to RA. However, in the worst case, RC obtains a 22% lower schedulable ratio than RA as shown in Figure 5.3a.

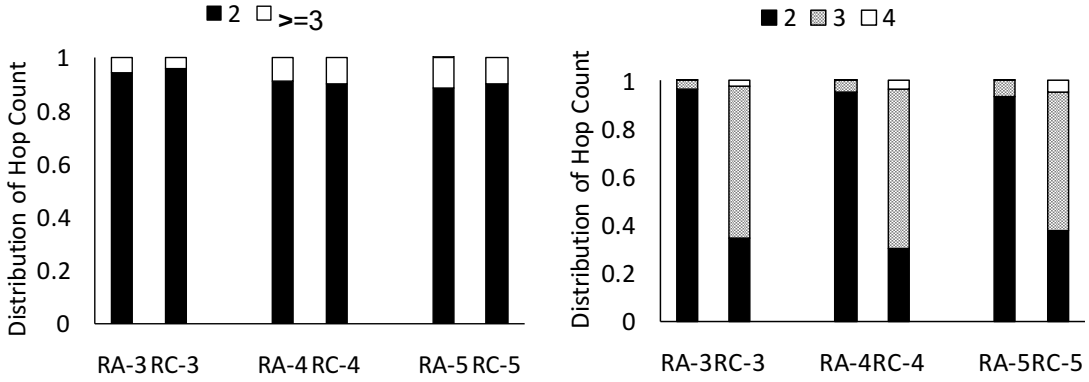
5.6.2 Algorithm Efficiency

We validate if RC indeed meets its goals in: (1) reducing channel reuse and (2) increasing channel reuse hop distance when channel reuse is adopted. We quantify the number of transmissions per channel (Tx/channel), and the minimum channel reuse hop count among



(a) Centralized traffic (#flows = 60, $P = 2^{0\sim3}$) (b) Peer-to-peer traffic (#flows = 120, $P = 2^{0\sim2}$)

Figure 5.4: Number of transmissions per channel under a varying number of channels (Indriya).



(a) Centralized traffic (#flows = 60, $P = 2^{0\sim3}$) (b) Peer-to-peer traffic (#flows = 120, $P = 2^{0\sim2}$)

Figure 5.5: Channel reuse hop count under a varying number of channels (Indriya).

senders and receivers of concurrent transmissions on each channel. The results are obtained from the experiments in Section 5.6.1. Figure 5.4 shows the distribution of different numbers of transmissions per channel of RA and RC. With the peer-to-peer traffic (Figure 5.4b), compared to RA, RC attains a higher proportion of 1 Tx/channel (no channel reuse), especially when there are more available channels. In addition, the results demonstrate that if a channel is reused, RC schedules fewer transmissions on a channel than RA. While, under the centralized traffic (Figure 5.4a), both RA and RC achieve comparable results, except when the number of channels is 3, in which RC achieves a $\sim$10% higher proportion of 1 Tx/channel than RA.

Figures 5.5a and 5.5b plot the distribution of channel reuse hop count under the peer-to-peer and centralized traffic, respectively. Again, the peer-to-peer approach allows RC to increase channel reuse hop count. For every number of channels, RC has the highest proportion for 3-hop count, while RA has the highest proportion for 2-hop count. With the centralized traffic, both RA and RC are dominated by 2-hop channel reuse. RC demonstrates that it meets its goals in mitigating interferences by reducing Tx/channel and increasing channel reuse hop count under the peer-to-peer traffic. However, it is less efficient with the centralized traffic because the centralized approach introduces more transmission conflicts, which limit the number of slots that can accommodate channel reuse.

5.6.3 Execution Time

We measure the execution time of the algorithms on a Macbook Pro laptop with 2.7 GHz Intel core i7. For NR, RA, and RC, we set the number of channels to 5 and the periods $P = [2^0, 2^2]$, and vary the traffic loads. The results are based on the peer-to-peer traffic. Figure 5.6 compares the execution time of NR, RA, and RC. NR obtains the smallest execution time of 0.2 to 0.4 ms, when the number of flows increases from 40 to 100. As the number of flows reaches 120, NR can no longer successfully generate schedules. The execution time of RA

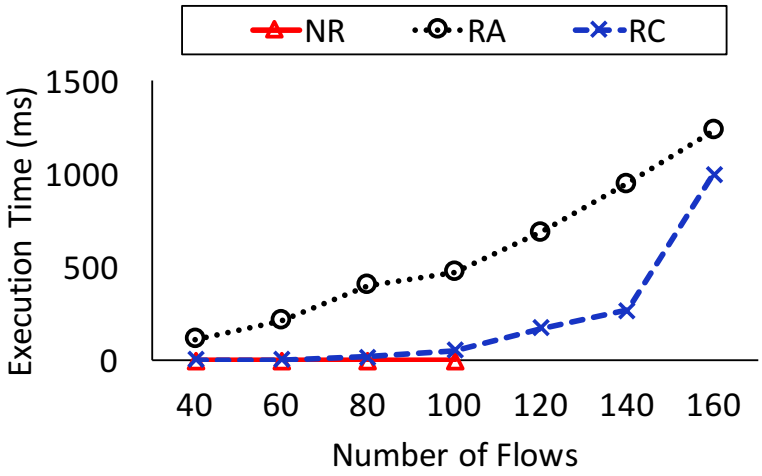


Figure 5.6: Algorithm execution time (in milliseconds).

increases linearly from 1.2 ms to 1200 ms as the number of flows increases from 40 to 160, respectively, while RC incurs 115.1 ms to 290 ms, and sharply increases to 990 ms when the traffic loads grow from 40, 140, to 160. RC requires less execution time than RA since RA attempts to adopt channel reuse for every transmission, while RC is more conservative. Although RC sparingly introduces channel reuse, it obtains longer execution time than NR because it is required to compute laxity for every transmission. A sudden increase in RC's execution time when the number of flows reaches 160 is the result of more workloads, and heavy computation for channel reuse.

RC's execution time is acceptable in the WirelessHART network because a schedule does not need to be reconfigured often. A schedule is recomputed only when the topology or the traffic demands change significantly. Accordingly, WirelessHART network maintenance is in frequent by design, e.g., it can take several minutes for a node to detect and report a link failure to the network manager.

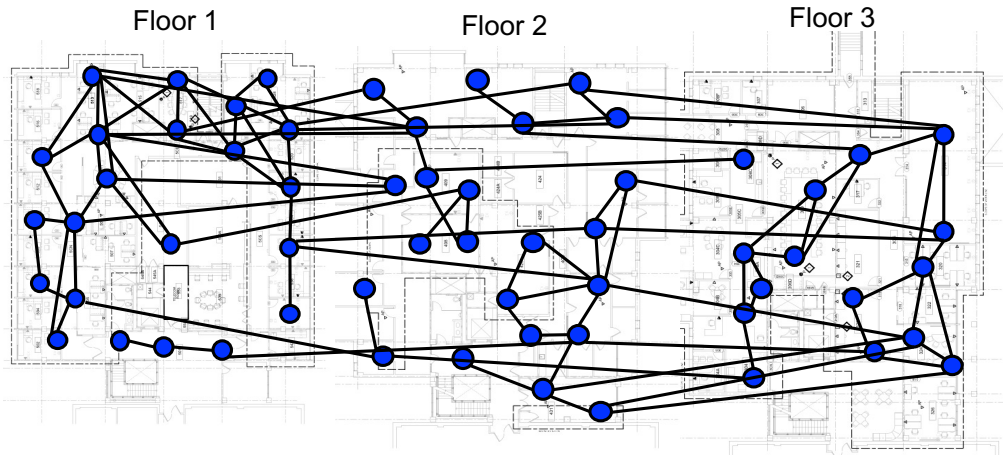


Figure 5.7: WUSTL testbed topology when channels 11-14 are used.

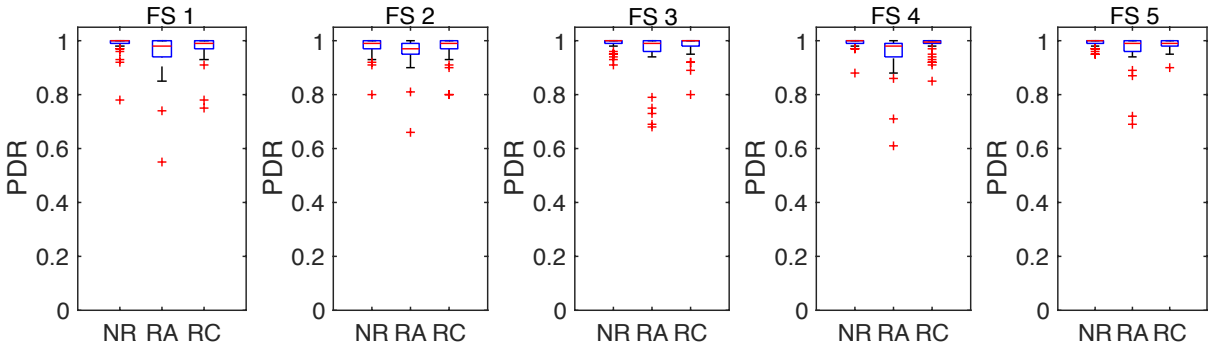


Figure 5.8: Box plots of Packet Delivery Ratio (PDR) of 5 distinct flow sets where #flows=50, #ch=4, $P = 2^{-1\sim0}$

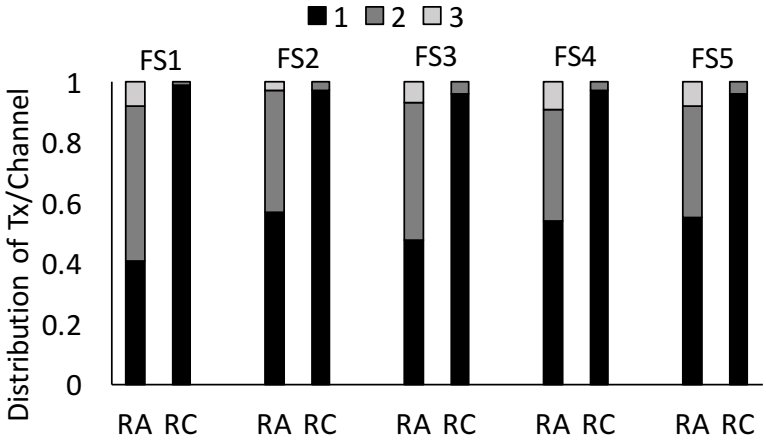


Figure 5.9: Number of transmissions per channel under RA and RC of 5 different flow sets.

5.6.4 Network Reliability

Despite enhancing real-time performance, channel reuse may degrade network reliability. We conduct experiments on the 60-node WUSTL testbed (see Figure 5.7) to obtain the reliability performance. The testbed runs TSCH MAC protocol with 10 ms time slot and source routing at the network layer. Source routing allows a sender to retransmit a packet if its first transmission fails. We generate five different flow sets consisting of 50 flows where 50% of flows release their packets every 2^{-1} s, and the rest release their packets every 2^{0} s. We run experiments on 4 channels with observed good performance (channel 11 to 14) and set the transmission power to 0 dBm. For each flow set, we enable a network to execute the schedule for 100 times.

Figure 5.8 presents box plots comparing Packet Delivery Ratio (PDR) of NR, RA, and RC under 5 different flow sets. A PDR is a fraction of packets successfully delivered to its destination. RC achieves the median PDR at most 1% lower than NR's median PDR in flow sets 1, 4, and 5, whereas NR's median PDR surpasses that PDR of RA by a maximum of 2% under flow sets 1, 2, and 4. This shows that most of the flows of both RC and RA can achieve good reliability performance. In terms of worst-case reliability, RC clearly outperforms RA. RC achieves lower worst-case PDRs than NR by a margin of 3%, 0%, 8%, 3%, and 5% for flow sets 1 to 5, respectively, while RA decreases the worst-case PDRs by 29%, 18%, 22%, 31%, and 27% under flow sets 1 to 5, respectively, compared to NR. This is because RC adopts channel reuse more conservatively compared to RA as shown in Figure 5.9. It is important to note that *worst-case* reliability is important in industrial applications because severe data loss can degrade stability and control performance and may lead to system failure. Although RC may still suffer from interferences due to channel reuse, in exchange for a much higher schedulable ratio, advances in networked control have contributed control designs that can

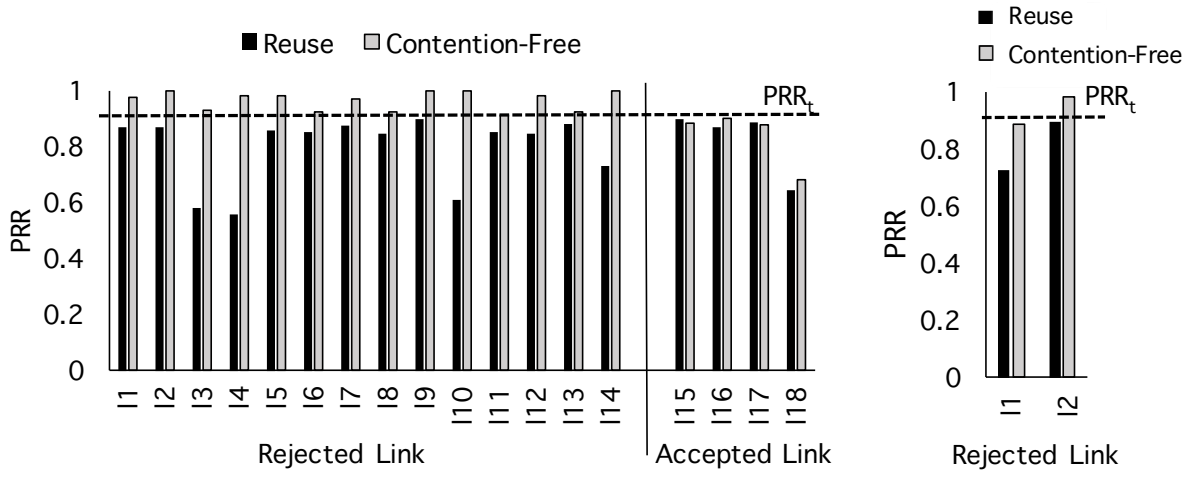
tolerate packet losses systems (e.g., [32] and [69]). Our approach is suitable for such resilient control systems.

5.6.5 Detecting Links Affected by Channel Reuse

We evaluate our detection policy in terms of its applicability to identify links whose reliability is degraded by channel reuse under both RA and RC approaches. In this set of experiments, we generate 50 peer-to-peer flows, which release a packet every 1 s. By default, a WirelessHART node must deliver a health report to the network manager every 15 minutes (=1 epoch). In each epoch, we obtain the PRR distribution consisting of 18 samples. We also calculate the overall PRR of each link in each epoch. Note that we consider the PRR distribution of each link over all channels used.

We run experiments under a clean environment on channel 11 to 14. We then generate external interference using WiFi in order to demonstrate that our detection policy can differentiate unreliable links caused by channel reuse from those caused by external interference. Because every node in the WUSTL testbed is connected to a Raspberry PI through a USB port for testbed management and instrumentation, we set up three pairs of Raspberry PIs, one pair on each floor, to generate interference on WiFi channel 1, which overlaps with 802.15.4 channels 11 to 14. For each pair of raspberry PIs, one is set up as a server (receiver), and another is a client sending 1 Mbps UDP traffic. We set the significance level α to 0.05 and PRR_t to 0.9. We run the experiments for 6 epochs. There are 95 and 20 links associated with channel reuse when the flow set is scheduled by RA and RC, respectively.

With RA under clean environment, there are 10 links whose reliability is below PRR_t under channel reuse. Because only links with $PRR \geq PRR_t$ are used for communication, this link quality degradation is therefore caused by channel reuse because there exists no external

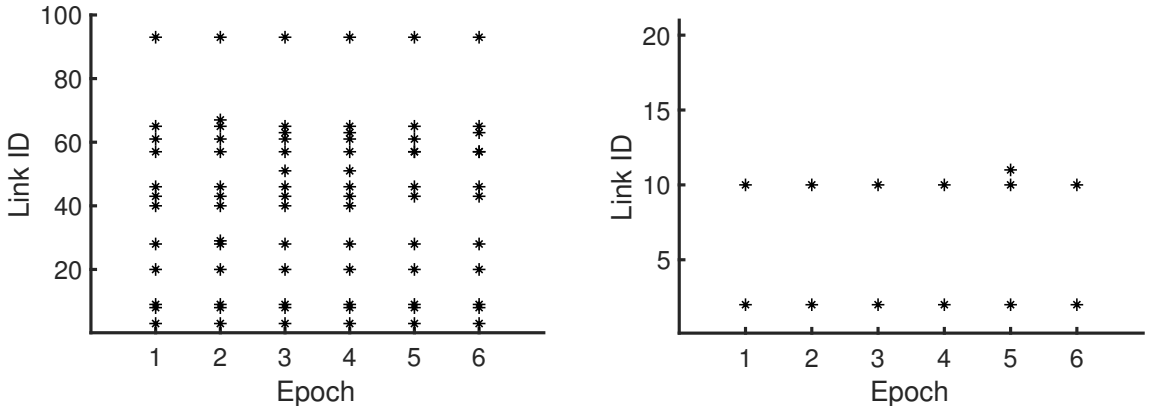


(a) RA

(b) RC

Figure 5.10: PRRs of rejected and accepted links failing to meet the reliability requirement when scheduled by RA and RC.

interference. After WiFi traffic is injected to cause external interference on the network, we observe 18 links with $PRR < PRR_t$. Out of the 18 links with low PRRs, our detection policy determines that only 14 cases (including all the 10 links with low PRRs in the clean environment) are attributed to channel reuse (reject), while transmission failures over the other 4 links are caused by external interference (accept). Our detection policy correctly detects all 10 links that suffered from channel reuse in the clean environment. Note that those same links are likely to remain vulnerable to channel reuse under external interference. Moreover, the additional 4 rejected links were also rejected by the K-S test in the clean environment. This indicates that channel reuse had a negative impact on these links. However, as their PRRs still met the reliability requirement in the clean environment, these links do not need to be rescheduled. When under external interference, however, the external interference degrades the PRRs of the links while the effect of channel reuse remains. Our policy hence includes these links among the links that require channel reuse to be avoided.



(a) RA (b) RC

Figure 5.11: Rejected links failing to meet the reliability requirement in each epoch under external interference.

We run similar experiments using RC to generate a schedule. In the clean environment, all links satisfy the reliability requirement, while under external interference, our detection policy identifies two rejected links and no accepted link with $PRR < PRR_t$. Again, the two rejected links were also rejected by the K-S test in the clean environment demonstrating that they were affected by channel reuse. But because their PRRs were above PRR_t in the clean environment, we do not need to readjust the schedule for these links. Due to RC's more conservative channel reuse policy, fewer number of links suffer from link quality degradation due to channel reuse compared to RA.

Figure 5.10 compares the PRRs of links that fail to meet the reliability requirements for both rejected and accepted cases when transmissions are scheduled by RA and RC. It can be observed that rejected links consistently achieve better performance on a contention-free channel but obtain low PRR when a channel is reused. While accepted links attain poor reliability performance for both cases. This result shows that our detection policy can effectively distinguish if link quality degradation is a result of channel reuse or external interference. Figure 5.11 presents rejected links in each epoch when transmissions are

scheduled by RA and RC. The result demonstrates that our detection policy consistently obtains almost the same set of rejected links over the course of the experiments under external interference.

5.7 Summary

WSANs have become an enabling technology for many IIoT applications that impose strict demands for real-time and reliable performance. In contrast to traditional approaches designed to maximize channel reuse, we propose a *conservative* channel reuse to enhance the real-time performance of industrial WSANs while mitigating the impact of channel reuse on network reliability. We further design a classifier to detect reliability degradation caused by channel reuse. Experimental results from two testbeds demonstrate that conservative channel reuse significantly outperforms the traditional industrial WSANs without channel reuse in real-time performance, while achieving higher reliability than aggressive channel reuse.

Chapter 6

REACT: an Agile Control Plane for a Centralized Industrial Wireless Network Architecture

To enhance predictability, industrial WSAN standards such as WirelessHART employ a centralized management architecture. While the centralized scheme offers deterministic and reliable communication, it has a significant impact on the network's ability to adapt to changing wireless conditions. Most prior efforts in the context of centralized industrial WSANs focus on improving the performance of the data plane. However, these works often overlook the control plane, which is crucial for sustaining the data plane performance in dynamic environments. To address the limitation of the centralized architecture in dynamic environments, we present **REACT**, a Reliable, Efficient, and Adaptive Control Plane for WirelessHART. Specifically designed to support network adaptation, REACT significantly reduces the latency and energy cost of network reconfiguration, thereby improving the agility

of WirelessHART under network dynamics. REACT comprises a *Reconfiguration Planner* featuring flexible scheduling and routing algorithms and reactive update policies to reduce rescheduling cost, and an *Update Engine* providing efficient and reliable mechanisms to report link failures and disseminate updated schedules. Experimental results on a 50-node wireless testbed demonstrate that REACT can reduce network reconfiguration latency by 60% at 50% of energy cost when compared to standard approaches.

6.1 Introduction

Industrial WSAN standards such as WirelessHART [83] and ISA100 [33] adopt a centralized management architecture that guarantees timeliness and highly reliable communication, i.e., a centralized manager that can generate a collision-free schedule and enable all packets to be delivered to meet their deadlines. However, such a solution impairs the network's ability to adapt quickly in response to network dynamics. This is due to the fact that the centralized manager must gather network connectivity information from network devices to generate routes and a global schedule. The newly computed schedule is then disseminated into the network through a multi-hop wireless mesh network. This schedule update process can incur significant communication overhead, especially in a large network.

Despite extensive research on WirelessHART, most works have largely emphasized WirelessHART data plane performance, such as route and schedule optimization, rate selection, and channel selection. Comprehensive reviews of these works can be found in [42, 49]. However, few works consider the control side of the network, which is critical for monitoring and maintaining the performance of the data plane. Due to the unpredictable nature of wireless environment, a network may suffer link quality degradation. While different techniques are used to enhance network reliability, e.g., multi-path forwarding [25] and per-link

retransmissions [10, 46], a network must have the capability to promptly reconfigure routes and schedules when link failure occurs.

Additionally, although the WirelessHART standard provides some high-level guidelines and specifications for its control plane, it leaves open the details regarding the actual design and implementation. Therefore, our work explores different design spaces for the WirelessHART control plane to support reliable and efficient network adaptation. Our contributions are four-fold.

1. We propose ***REACT***, a reliable, efficient, and adaptive control plane optimized for adaptation in industrial WSANs based on centralized architecture.

2. We develop a ***Reconfiguration Planner*** featuring strategies to compute and update routes and transmission schedules. Designed to minimize the changes to transmission schedules, the Planner effectively reduce the latency and energy cost in disseminating a new schedule.

3. We build an ***Update Engine*** that provides the mechanisms necessary for network adaptation, including health reporting, failure notification, and an efficient and reliable schedule dissemination mechanism. Our design also allows critical flows impacted by link failures to recover faster than other ones.

4. We implement REACT as the control plane of a WirelessHART protocol stack. Experiments on a 50-node wireless testbed demonstrate that REACT can significantly reduce the latency and energy cost of network reconfiguration, thereby supporting agile adaptation in industrial WSANs.

The rest of the chapter is organized as follows. Section 6.2 reviews related works. Section 6.3 introduces TSCH scheduling. Section 6.4 described REACT architecture. Section 6.5 presents

the policies used in the Reconfiguration Planner, and Section 6.6 details the design of the Update Engine. Section 6.7 presents evaluation results, and Section 6.8 summarizes the chapter.

6.2 Related Work

Earlier works on real-time scheduling for centralized TDMA-based networks addressed different objectives, such as optimizing real-time performance [13, 63] or enhancing reliable communication [25, 46]. However, these works schedule transmissions without taking into account the need for a schedule to be updated due to network dynamics. Consequently, a network may incur considerable adaptation cost. Our system includes a scheduling policy that helps mitigate schedule reconfiguration cost by reducing schedule-related information that must be disseminated by a network manager.

There are other scheduling approaches to facilitate network adaptation. Dezfouli et al. introduced Rewimo [17], a wireless solution for real-time scheduling in mobile networks. Rewimo incorporates a scheduling technique for a TSCH network that schedules nodes additively, so a network manager disseminates only a schedule of newly admitted nodes in response to workload changes. Nevertheless, the scheduling policy does not consider the need for schedule updates due to link failure. To tackle the unreliable nature of wireless links, Yang et al. [87] proposed slot assignment algorithms supporting efficient slot rescheduling, and Shen et al. [66] developed a source-aware scheduling algorithm that trades off schedule length and schedule reconfiguration overhead. While our scheduling and reconfiguration policies are built upon a TSCH network and allow multiple flow classes, their scheduling polices are based on a single channel TDMA network, and consider only one class of flow, i.e., all flows have the same period and deadline.

There are also previous efforts to support network maintenance and adaptation mechanisms for industrial WSANs, based on a centralized architecture in which a network manager collects network statistics and centrally computes a global schedule. Pottner et al. [55] developed a schedule construction algorithm and proposed mechanisms to monitor link quality and verify the validity of a schedule throughout the network lifetime. O'Donovan et al. [50] proposed GINSENG, a complete industrial WSAN solution that enables topology maintenance and adaptation through BurstProbe [9] and dynamic tree construction. Both of these solutions are designed for a single channel TDMA network, whereas REACT is geared towards a TSCH network operating on multiple channels.

Additionally, Livolant et al. [41] compared the cost of installing and updating schedules among existing protocols such as CoAP [15] and CoMI [16] that run on top of a centralized 6TiSCH [31] network, and a custom industrial WSAN protocol OCARI [4]. This work investigated only the impact of these protocols on the number of messages required to install a global schedule, while REACT offers a more complete solution for schedule reconfiguration in a WirelessHART network.

Conversely, a decentralized architecture allows nodes to construct their own schedules, which enables them to adapt quickly when network connectivity changes. The following works are built on a network running TSCH and RPL [61]. Duquennoy et. al [20] proposed Orchestra, where nodes autonomously build their own schedules without requiring additional signaling among neighbors. In the context of the OpenWSN (an open-source implementation of 6TiSCH) [51], uRes [79] introduced a schedule negotiation mechanism between neighbor nodes to compute local schedules. Accettura et al. [1] presented DeTAS, a decentralized scheduling protocol, implemented on OpenWSN. DeTas ensures the smallest end-to-end latency, and reduces neighbor-to-neighbor signaling to generate local schedules. Although

these protocols enhance network adaptability, they provide only best-effort service, and cannot guarantee real-time performance.

Glossy [23] provides fast and efficient network flooding by exploiting constructive interference and the capture effect. Blink [90] combines a Glossy-based Low-Power Wireless Bus protocol [22] and real-time scheduling, and offers a promising alternative to TSCH-based networks. In contrast to the above-mentioned efforts, Blink is topology independent. Any change in network connectivity does not impact its global schedule. Taking another approach, REACT adopts the WirelessHART network model, which is the state-of-the-art industrial solution.

6.3 TSCH Scheduling

A centralized scheduler residing on the network manager constructs a global schedule. Building a schedule involves allocating a *time slot* and a *channel offset* to each transmission. To generate a feasible schedule, a scheduler must follow these ***scheduling constraints***:

1. Slot constraints:
   (a) Due to a half-duplex nature of the IEEE 802.15.4 radio, a node can only send or receive at a given time, so no two transmissions in a time slot can share a common node.
   (b) To prevent intra-network interference, only one transmission can be scheduled on each channel in a time slot. With c available channels, there can be at most c concurrent transmissions in a slot, and the channel offset is within the range $[0, c]$.
2. Precedence constraint: a sender must receive a packet before forwarding it, i.e., transmission t_{ij} must be scheduled in a time slot before transmission t_{ij+1}.
3. Real-Time constraint: all flows must be scheduled to meet their deadlines.

Our work adopts ***fixed priority scheduling***, which offers a efficient and computationally inexpensive heuristic. Hence, it is commonly used for real-time systems. Each flow F_i is associated with a priority. A flow F_i has higher priority than a flow F_l if $i < l$. Priorities are commonly assigned based on a flow's period or deadline, and a scheduler schedules flows in decreasing order of priority. For each flow F_i, with respect to the scheduling constraints, a scheduler allocates a time slot and a channel offset to each transmission t_{ij}.

6.4 REACT Architecture

Figure 6.1 depicts REACT architecture, consisting of (1) a gateway enabling communication between network devices (sensors or actuators) and the centralized controller (2) a host running network manager software (3) a WSAN comprising multiple network devices forming a wireless mesh network, and (4) two access points (AP) wired to the network manager and the gateway. Providing two access points offers redundant paths between the WSAN and the back plane. To support network reconfiguration, REACT includes two major components: a *reconfiguration planner* and an *update engine*.

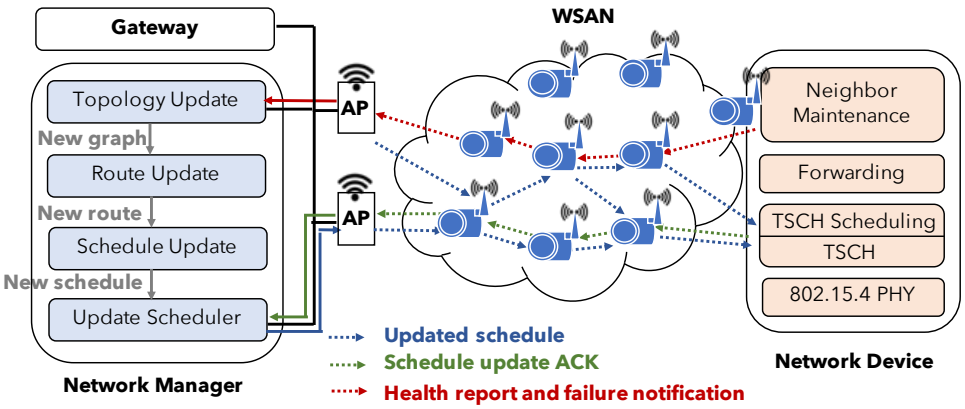


Figure 6.1: REACT architecture and adaptation process.

6.4.1 Reconfiguration Planner

The Reconfiguration planner resides on the network manager, and consists of the following components:

- Topology update: A manager collects link statistics from all network devices, and generates a network topology represented as a graph $G(V, E)$, where V is a set of network devices and E is a set of links reliably connecting different pairs of network devices. When a link fails to meet the reliability requirement, the link is removed from the topology. Conversely, if a node detects a new neighbor with reliable connectivity, a link between the node and its neighbor is added into the topology.

- Route update: The module is responsible for creating flow routes and constructing a *broadcast graph* for disseminating schedule information from the manager downward to all network devices. With an updated topology, it computes new routes for flows associated with a failed link and adjusts the broadcast graph if necessary. Our work incorporates a *partial reroute policy* enabling the flow schedule to be modified partially, and consequently reducing the amount of flow's schedule that must be updated.

- Schedule update: Based on the new routes, the scheduler recalculates a new global schedule. It utilizes *gap-induced scheduling* scheme, which complements our *rescheduling policy* in granting only schedules of flows affected by link failure to be updated, thereby lowering the schedule dissemination cost.

6.4.2 Update Engine

The update engine is composed of two different mechanisms to enable schedule reconfiguration.

- Health report and failure notification: Each network device maintains statistics pertaining to connectivity between a node and its neighbors. It learns these statistics through regular transmissions of data packets and periodic neighbor-discovery packets. These statistics are then reported to the network manager through *upstream control flows*, which provide routes from nodes to access points. When a node identifies a failed link, it notifies the network manager through the same upstream flow.

- Schedule dissemination: An *update scheduler* installed on the network manager constructs packets. It schedules updates based on flow priority. Therefore, more critical flows receive updates and recover faster than less critical flows. Schedule-related information is propagated through a broadcast graph to destinations. Nodes receiving update commands modify their TSCH schedules, and send acknowledgement (ACK) back to the network manager, using the *upstream control flows*. The ACK allows the network manager to handle packet losses and ensure all affected nodes receive its new schedule.

6.5 Configuration Planner

In this section, we present policies that are incorporated into our reconfiguration planner to reduce schedule reconfiguration cost. These policies include a gap-induced scheduling policy, which the scheduler follows while creating or updating a schedule, the partial reroute scheme for a route update, and the rescheduling policy for a schedule update.

6.5.1 Gap-Induced Scheduling Policy

We propose a scheduling policy based on fixed priority scheduling to support efficient transmission rescheduling in response to link failure. Our policy is designed to (1) decrease

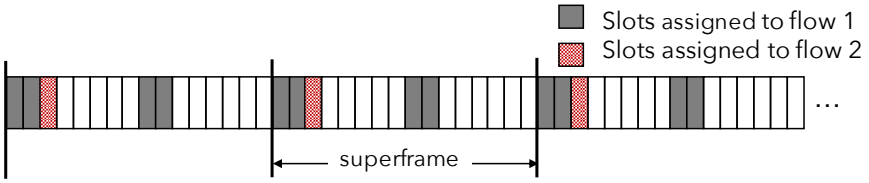


Figure 6.2: A superframe consists of 16 slots containing two flows (flow 1 with $P_1 = 8$, and flow 2 with $P_2 = 16$.)

the likelihood that an update to the schedule of a higher priority flow will impact the schedules of its lower priority flows. (2) enable a flow's schedule to be reused partially when possible. Therefore, only limited portions of a schedule are modified, which lowers the schedule dissemination overhead.

Key Ideas

With fixed priority scheduling, flows are scheduled in descending order of priority. For each flow, (1) transmissions are assigned to the *earliest* feasible slot in a sequential order [62] or (2) a scheduler schedules transmissions in reverse order by selecting the *latest* possible slot meeting the scheduling constraint [17]. In contrast to these two traditional approaches, our scheduling algorithm adopts the following polices to reduce the rescheduling cost.

1. It ensures that for each flow F_i, the same schedule is repeated for every released packet $1, 2, ...T/P_i$ within a superframe. As shown in Figure 6.2, flow 1 (with $P_1 = 8$) repeats its schedule twice in the superframe of size 16. With a regular scheduling pattern, the network manager can disseminate less schedule-related information to wireless devices. In practice, flow periods are usually harmonic, so this policy is then easy to satisfy when we schedule or reschedule transmissions.

2. It adds gaps between transmissions belonging to the same flow. This policy provides two benefits.

Schedule 1: Earliest Possible

ch/slot	1	2	3	4	5	6	7	8	9	10
0	a→b	b→c	c→d			a→b	b→c	c→d		

Schedule 2: Introducing Gap

ch/slot	1	2	3	4	5	6	7	8	9	10
0	a→b		b→c		c→d	a→b		b→c		c→d

Figure 6.3: Examples of schedules based on two slot scheduling heuristics. Flow 1 ($P_1 = D_1 = 5$) sends a packet through route $a \to b \to c \to d$.

Schedule 1: Earliest Possible

ch/slot	1	2	3	4	5	6	7	8	9	10
0	a→b	b→c	c→d			a→b	b→c	c→d		
1			b→f	f→g	g→h					

Schedule 2: Introducing Gap

ch/slot	1	2	3	4	5	6	7	8	9	10
0	a→b		b→c		c→d	a→b		b→c		c→d
1		b→f				f→g				g→h

Schedule 3: Fewer Transmissions per Slot

ch/slot	1	2	3	4	5	6	7	8	9	10
0	a→b	b→c	c→d			a→b	b→c	c→d		
1				b→f	f→g				g→h	

Figure 6.4: Examples of schedules based on different heuristics. Each schedule consists of two flows: flow 1 ($P_1 = D_1 = 5$, $\phi_1 = a \to b \to c \to d$) is labeled in black, and flow 2 ($P_1 = D_2 = 10$, $\phi_2 = b \to f \to g \to h$) is labeled in red.

First, it allows a flow to be rescheduled partially when the route of a flow does not entirely change. For instance, Figure 6.3 presents two schedules: (1) where a scheduler picks the earliest possible slot for each transmission, (2) when transmissions are spread out from each other. Initially, a flow sends a packet through route $a \to b \to c \to d$. Suppose link $b \to c$ fails, and a newly computed route is $a \to b \to f \to c \to d$. To obtain a new schedule, with the first schedule, a scheduler has to replace $b \to c$ with $b \to f$, replace $c \to d$ with $f \to c$, and move $c \to d$ to slot 4. However, for the second schedule, a scheduler needs only to add $b \to f$ to slot 2 and replace $b \to c$ with $f \to c$. Hence, the second schedule incurs fewer operations to update.

Second, it reduces the chance of lower priority flow schedules hindering the modification of a higher priority flow's schedule, so a change made to higher priority flows has less chance of impacting lower priority flows. For example, in Figure 6.4, there are two flows $F1$ and $F2$. Similar to the previous example, suppose $b \rightarrow c$ fails and a new route of F_1 is $a \rightarrow b \rightarrow f \rightarrow c \rightarrow d$. With the first schedule, in order to schedule F_1 to meet its deadline at slot 5, the scheduler has to reschedule F_2 as well, since $b \rightarrow f$ and $f \rightarrow g$ block time slots 3 and 4. Otherwise, these two slots can be allocated to $f \rightarrow c$ of F_1. On the other hand, with the second schedule, a scheduler can reschedule F_1 without affecting F_2 by removing $b \rightarrow c$, then adding $b \rightarrow f$ and $f \rightarrow c$ to slots 3 and 4, respectively.

3. It reduces the number of concurrent transmissions in each time slot, which spreads out transmissions belonging to different flows. This heuristic also helps prevent lower priority flow from blocking updates on higher priority flow schedules. As shown in Figure 6.4, for schedule 3, the scheduler avoids scheduling multiple transmissions in the same time slot. Again, supposing link $b \rightarrow c$ fails, then the scheduler needs only to modify the schedule of F_1 by adding $b \rightarrow f$ to slot 2, $f \rightarrow c$ to slot 3, and moving $c \rightarrow d$ to slot 4, and does not need to update F_2's schedule.

Gap-Induced Scheduling Algorithm

A detailed description of our gap-induced scheduling policy is presented in Algorithm 3. Here, a time slot means a slot offset in the superframe. With a superframe length of T, the slot offset is within the range $[1, T]$. Both the periods and deadlines of flows are measured in slots.

The input of the algorithm is a set of flows F and c available channels, and the output is a global schedule S. The algorithm schedules flows in descending order of flow priority. Each flow F_i consists of a sequence of transmissions $\Gamma_i = \{t_{i1}, t_{i2}, ...t_{in}\}$, and $|\Gamma_i|$ is the size of

Algorithm 3: Gap-Induced Scheduling

Input : A flow set F ordered by priority, c = the number of available channels
Output : A global schedule S

foreach *flow* F_i, *where* i=*1 to* k **do**
  foreach *transmission* t_{ij}, *where* $j = n$ *to 1* **do**
    $\chi_{ij} = ObtainFeasibleSlots(t_{ij})$;
    if j==*1* **then**
      x_{i1} = earliest slot in χ_{ij};
      $S.assign(t_{i1}, x_{i1}, unusedChannel(x_{i1}))$;
    else if j==n **then**
      x_{in} = latest slot in χ_{ij};
      $\lambda_{in} = x_{in}$;
      $S.assign(t_{in}, x_{in}, unusedChannel(x_{in}))$;
    else
      λ_{ij} = latest slot in χ_{ij}, where $\lambda_{ij} < \lambda_{ij+1}$;
  foreach *transmission* t_{ij}, *where* $j = 2$ *to* $n-1$ **do**
    $y = x_{ij-1} + (x_{in} - x_{ij-1}+1)/(|\Gamma_i| - j + 1)$;
    $x_{ij} = SelectSlot(t_{ij}, \chi_{ij}, x_{ij-1}, \lambda_{ij+1}, y)$;
    if x_{ij}==*-1* **then**
      return $\emptyset$;
    $S.assign(t_{ij}, x_{ij}, unusedChannel(x_{ij}))$;
return S;
Function $ObtainFeasibleSlots(t_{ij})$:
  for q = *1 to* T/P_i **do**
    Find X_q = set of slots in $[r_{iq}, d_{iq}]$ that meets the slot constraint
    foreach *slot* x *in* X_q **do**
      $x = (x-1) \mod P_i + 1$;
    $\chi_{ij} = \bigcap_{q=1}^{T/P_i} X_q$;
  return χ_{ij};
Function $SelectSlot(t_{ij},\ \chi_{ij},\ x_{ij-1},\ \lambda_{ij+1},\ y)$:
  if $\forall x \in \chi_{ij}, x \notin (x_{ij-1}, \lambda_{ij+1})$ **then**
    return -1;
  foreach *slot* $x \in \chi_{ij}$ *and* $x \in (x_{ij-1}, \lambda_{ij+1})$ **do**
    $dist(x, y) = |x - y| + 1$;
    $w(x) = (\#trans(x) + 1)/c$;
    $cost(x) = dist(x, y) * w(x)$;
  return $\arg\min_x cost(x)$;

Γ_i. During a superframe, a flow generates T/P_i packets. Each packet q is released at time r_{iq} and has a deadline d_{iq}. The first step is to obtain a set of feasible slots χ_{ij} for each transmission t_{ij} (function $ObtainFeasibleSlot()$) starting from the last transmission t_{in} to the first transmission t_{i1}. A time slot x is added to χ_{ij} if $\forall q$, a slot $(q-1)*P_i+x$ satisfies the slot constraint for t_{ij} and is within $[r_{iq}, d_{iq}]$. Because the scheduler will assign only a slot in χ_{ij} to each t_{ij}, it is guaranteed that a flow can repeat the same schedule for every packet released in the superframe. The scheduler also determines the latest feasible slot $\lambda_{ij} \in \chi_{ij}$ for each transmission t_{ij}, where $\lambda_{ij} < \lambda_{ij+1}$ for transmission 2 to n-1. Note that a function $S.assign()$ assigns a time slot x_{ij} and an unused channel offset in x_{ij} to t_{ij}.

To spread out transmissions and introduce gaps between transmissions of a flow, the algorithm schedules the first transmission t_{i1} of flow F_i at the earliest possible slot $x_{i1} \in \chi_{i1}$, and assigns the latest feasible slot $x_{in} \in \chi_{in}$ to the last transmission t_{in}. For each remaining transmission t_{ij}, from $j=2$ to $n-1$, the scheduler computes an ideal slot y (line 14). Suppose x_{ij-1} is a slot assigned to the previous transmission t_{ij-1}, and $|\Gamma_i|-j$ is the number of remaining unscheduled transmissions. Here, if the scheduler divides slots in the range $[x_{ij-1}+1, x_{in}-1]$ into $|\Gamma_i|-j+1$ equal segments, and assigns a time slot at the end of segment 1 to segment $|\Gamma_i|-j$ to each remaining transmission, then t_{ij} and remaining transmissions after t_{ij} can be spaced evenly between $[x_{ij-1}+1, x_{in}-1]$. Hence, in other words, y is the last slot in the first segment.

A function $SelectSlot()$ determines the best slot for t_{ij}. It computes the cost of choosing slot $x \in \chi_{ij}$ (line 33), considering the distance between x and y and the number of transmissions already scheduled in x ($\#trans(x)$). The scheduler selects the lowest-cost slot x_{ij} for t_{ij} (i.e., the slot closest to y and already allocated to the fewest transmissions), where $x_{ij} \in [x_{i,j-1}+1, \lambda_{ij+1}-1]$. The algorithm enforces t_{ij} to be scheduled after a slot $x_{i,j-1}$ (to preserve the precedence constraint) and before the latest feasible slot λ_{ij+1} of the next

transmission $t_{i,j+1}$ (to ensure the flow's schedulability). The algorithm terminates when (1) transmissions of all flows are scheduled within their deadline, or (2) a scheduler cannot find a slot for t_{ij}, i.e., no slot in χ_{ij} is within the range $(x_{i,j-1}, \lambda_{ij+1})$.

6.5.2 Partial Reroute

Selecting a new route for a flow associated with link failure influences the flow's new schedule. To allow affected flows to reuse parts of their old schedules, the routing algorithm should choose a new route that includes links in the old route. A network topology is represented as a graph $G(V, E)$, where V is a set of network devices and E is a set of links. A link is added between nodes u and v if u and v can reliably communicate with each other. In particular, the packet reception ratios (PRR), the number of packets successfully received over the total number of packets sent, of both links $u \rightarrow v$ and $v \rightarrow u$ must be higher than the reliability requirement in all c channels used. Due to channel hopping, nodes must be able to reliably communicate in all channels used.

After a graph is updated by removing a failed link, we calculate a new route for each affected flow in decreasing order of flow priority. Let ϕ_i be an old route of F_i. A flow F_i can obtain a new route ϕ_i' in two steps.

1. We use the Dijkstra's algorithm to calculate F_i's route. For each link in $G(V, E)$ in the old route ϕ_i, we sets its weight to $w/2$, whereas we set the weight of the remaining links to w. By doing so, the algorithm will output the least-cost route, i.e., a shorter route containing links used in ϕ_i. Due to the centralized control architecture, a packet needs to be propagated upstream from a source to an access point and a gateway, and downstream from a gateway and an access point to a destination. Because our work

considers two access points in the network, Dijkstra's algorithm must compute four routes per flow (one upstream and one downstream route for each access point).

2. Because a new route ϕ_i' is a combination of an upstream and a downstream route, the routing algorithm must select the combination with the lowest cost. Here, the cost is computed based on the number of operations required to update a route from ϕ_i to ϕ_i', which reflects the minimum cost of modifying a flow schedule. There are two possible update operations, DELETE and ADD. If a link in ϕ_i no longer exists in ϕ_i', we must perform a DELETE operation. On the other hand, if a link is in ϕ_i', but not in ϕ_i, then an ADD operation is required. The cost is defined as $(nDEL * a) + (nADD * b)$, where $nDEL$ and $nADD$ are the numbers of deleting and adding operations, respectively. Here, a and b are the numbers of bytes required to DELETE or ADD a schedule of a transmission (see Section 6.6.2).

6.5.3 Schedule Reconfiguration Policy

Based on fixed priority scheduling, if a schedule of a higher priority flow is updated due to a failed link, all of its lower priority flows must be rescheduled as well, although these lower priority flows are not pertinent to the failed link. Such an approach introduces more modification to a global schedule, which implies a corresponding higher schedule dissemination cost. The goal of this policy is to reduce the schedule update cost by (1) rescheduling only flows affected by the failed link and (2) reusing a flow's schedule as much as possible. Consequently, fewer packets are required to update a global schedule. A schedule constructed based on the gap-induced scheduling policy has a higher chance of being successfully reconfigured to meet these two goals. For the same reason, the reconfiguration policy also adopts gap-induced scheduling scheme when rescheduling transmissions.

Algorithm 4: Flow Rescheduling

Input : A set of affected flows F', ordered by priority, and the current global schedule S
Output : A new global schedule S'

```
1  S' = S;
2  foreach affected flow F_i ∈ F' do
3      Remove from S' schedules of all transmissions in Γ_i that no longer exist in Γ'_i;
4      while not all transmissions in Γ' have been scheduled do
5          Γ'_s = latest set of unscheduled transmissions in Γ' that cannot be reused,
             {t_ip, t_ip+1, ...t_iq};
6          if p == 1 then
7              lower_bound = 1;
8          else
9              lower_bound = x_ip−1+1;
10         if q == m then
11             upper_bound = D_i;
12         else
13             upper_bound = x_iq+1-1;
14         success = false;
15         while success == false do
16             success = reschedule(Γ_s, lower_bound, upper_bound);
17             if not success then
18                 if p == 1 and q == m then
19                     return ∅;
20                 else if p == 1 and q! = m then
21                     < Z, upper_bound > = ExpandRight();
22                 else if p! = 1 and q == m then
23                     < Z, lower_bound > = ExpandLeft();
24                 else
25                     < ZR, upperR > = ExpandRight();
26                     < ZL, lowerL > = ExpandLeft();
27                     if (upper_bound − lowerL + 1)/|Γ'_s ∪ ZL| ≥
                         (upperR − lower_bound + 1)/Γ'_s ∪ ZR| then
28                         < Z, lower_bound >=< ZL, lowerL >;
29                     else
30                         < Z, upper_bound >=< ZR, upperR >;
31                 Γ'_s = Γ'_s ∪ Z;
32 return S';
```

```
33 Function ExpandLeft():
34     Z.add(t_{ip-1});
35     foreach transmission t_{ij} ∈ Γ' where j = p − 2 to 1 do
36         if t_{ij} is already assigned a time slot then
37             return < Z, x_{ij} + 1 >
38         Z.add(t_{ij});
39     return < Z, 1 >
40 Function ExpandRight():
41     Z.add(t_{iq+1});
42     foreach transmission t_{ij} ∈ Γ' where j = q + 2 to n do
43         if t_{ij} is already assigned a time slot then
44             return < Z, x_{ij} − 1 >;
45         Z.add(t_{ij});
46     return < Z, D_i >;
```

Schedule Reconfiguration Algorithm

Algorithm 4 describes the procedure for rescheduling flows associated with a failed link. The input of the algorithm is a set of affected flows F', ordered from highest to lowest priority, and a current global schedule S. The output is a new global schedule S'. The algorithm first assigns S to S'. Let Γ_i be the old set of transmissions, and let $\Gamma'_i = \{t_{i1}, t_{i2}, ...t_{im}\}$ be the new set of transmissions of flow F_i. For each affected flow F_i, the algorithm removes the schedule of transmissions that no longer exist in Γ'_i, i.e., transmissions that cannot be reused. Let $\Gamma_s \subset \Gamma'_i$ be a set of sequential transmissions $\{t_{ip}, t_{ip+1}, ...t_{iq}\}$, where all transmissions in Γ_s do not exist in Γ_i. If Γ_s contains only one transmission, then $p = q$. There can be one or more discrete sets of Γ_s in Γ'_i. Our policy aims to reschedule only transmissions in Γ_s, without modifying the schedules of other transmissions. For instance, suppose an old set of transmissions is $\{a \to b, b \to c, c \to d, d \to e\}$ and a new set of transmissions comprises $\{a \to b, b \to c, c \to g, g \to e\}$. Here, there is only one set of transmissions Γ'_s to be scheduled,

namely, $\{c \rightarrow g, g \rightarrow e\}$. Transmissions $a \rightarrow b$ and $b \rightarrow c$ should be allocated to the same slots and channel offsets as in schedule S.

The $reschedule()$ function schedules transmissions in Γ'_s following the gap-induced scheduling policy, and time slots allocated for these transmissions must be within the range $[upper\_bound, lower\_bound]$ to avoid interfering with the schedule of those reusable transmissions. If $p = 1$ (first transmission), then the $lower\_bound$ is the first slot of the superframe. And if $q = m$ (last transmission), the $upper\_bound$ is a flow deadline D_i. Otherwise, the $upper\_bound$ and $lower\_bound$ slots are x_{ip-1}+1 and x_{iq+1}-1, respectively, and x_{ip-1} and x_{iq+1} are time slots already assigned to the transmissions t_{ip-1} and t_{iq+1} (in the old schedule S).

If the scheduler fails to schedule transmissions in Γ'_s, i.e., $reschedule()$ returns $false$, the algorithm will add more transmissions into Γ'_s and adjust either the $lower\_bound$ or the $upper\_bound$ through the functions $ExpandLeft()$ or $ExpandRight()$. In this case, the algorithm needs to reschedule some of the transmissions that can be reused as well. Then, it attempts to reschedule Γ'_s again. This step repeats until all transmissions in Γ'_s can be scheduled successfully. $ExpandLeft()$ adds one or more transmissions prior to t_{ip} to Γ'_s and adjusts a $lower\_bound$, whereas $ExpandRight()$ adds at least one transmission after t_{iq} to Γ'_s and computes a new $upper\_bound$. In both functions, the algorithm keeps adding transmissions to Γ'_s until it reaches the next transmission that is already assigned a time slot, which sets a new $upper\_bound$ or a new $lower\_bound$. By doing so, the algorithm gradually updates the schedule of those reusable transmissions only when necessary, which enables a flow to preserve its old schedule as much as possible. The algorithm chooses to $ExpandLeft()$ if its $upper\_bound$ has reached D_i, and $ExpandRight()$ when the $lower\_bound$ is already set to 1. Otherwise, the algorithm will compute which option is better. The better option is

the one providing the higher ratio of the number of slots in $[lower\_bound, upper\_bound]$ to the number of transmissions in Γ_s after the expansion.

The algorithm terminates if it cannot successfully reschedule a flow, i.e., when Γ'_s contains all the transmissions in Γ'_i and the $upper\_bound$ and $lower\_bound$ reach their limits ($upper\_bound = D_i$ and $lower\_bound = 1$). In this case, the algorithm cannot modify a schedule of F_i without rescheduling other flows. Otherwise, the algorithm ends when all the affected flows are scheduled to meet their deadlines

6.6 Update Engine

In this section, we present the designs of our update engine to handle network adaptation. We first discuss health reporting and failure notification mechanisms, and then a schedule dissemination mechanism.

6.6.1 Health Report and Failure Notification Mechanisms

Before a WSAN network is made operational, to determine a reliable set of links and channels for calculating routes and schedules, the network manager must collect complete topology information from every node and on all 16 channels. Because environmental conditions in a plant change over time, the manager must update the network topology throughout the network's lifetime, and must ensure the current flow routes and schedule stay valid. Therefore, while the network is operating, nodes must have the capability to report link condition to the network manager.

To maintain neighbor statistics, each node deploys a neighbor table of link statistics for each of its neighbors, e.g., PRR. A node obtains the PRR from periodic data packets and

neighbor-discovery packets. Each node is required to broadcast a neighbor discovery packet periodically on all channels used, so other nodes hearing the packet can maintain their connectivity with the sender node. In particular, for each link, a node records if a packet is successfully received, and uses a sliding window to compute the current PRR.

Instead of periodically reporting to the network manager the link statistics of every neighbor, our system reduces the amount of data uploading to the network manager by letting nodes send a health report only when they discover new neighbors or when the link quality between a node and any of its neighbor falls below the reliability requirement, thereby saving the network resources. A node activates a link failure notification whenever an active link, i.e., a link used by any data flow, fails to meet the reliability requirement. To ensure that a notification will eventually arrive at the gateway, both the sender and receiver associated with the failed link will repeatedly send out notifications until they receive a new schedule.

Reserving periodic upstream control flows for health reports and failure notifications from every network device to the gateway can be expensive, because these flows are left unused when there is no change in the network condition. To conserve network resources, these control data can be piggybacked onto periodic data packets. We allocate a separate upstream control flow for a node only when it has no upstream data flow passing through. The amount of control data required to indicate a newly added link or a failed link does not add much overhead to the data packet. Additionally, data flows tend to run at higher rates than control flows, thus control information can be delivered to the network manager faster.

6.6.2 Schedule Dissemination Mechanism

In this section, we describe our schedule dissemination mechanism that supports reliable and efficient schedule updates. Our design provides the following features: (1) it updates the

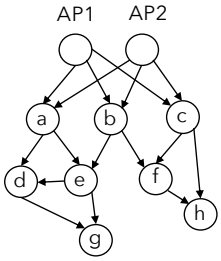


Figure 6.5: A broadcast graph with two are access points AP1 and AP2.

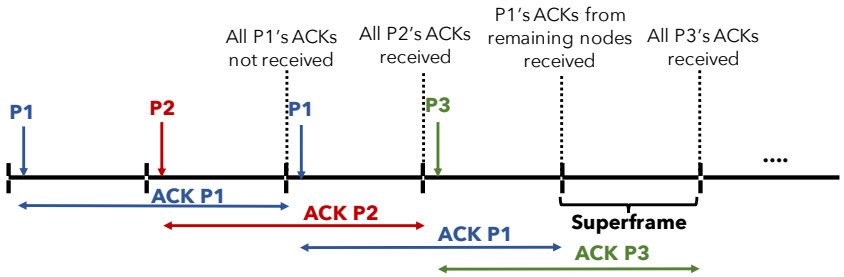


Figure 6.6: A timeline of schedule update with three packets P1, P2, and P3.

schedules of critical flows first, and enables these flows to recover quickly from link failure, (2) it disseminates schedule information through a broadcast graph that offers route diversity, and forwards only selected packets to enhance energy efficiency, (3) it guarantees that new schedules reach all affected nodes, and (4) it can handle multiple link failures.

Update Scheduler

It is important that critical flows (flows with higher priority) promptly recover from link failure. Since these critical flows usually have shorter periods, a longer schedule update latency results in more packet losses. Therefore, we design our system to ensure that schedules of more critical flows are updated first. To facilitate such a design, the update scheduler disseminates packets containing schedules of higher priority flows before packets with lower priority flow schedules. The update scheduler must first form packets containing schedule update commands and then determine the sequence in which these packets will be disseminated.

To reduce schedule dissemination overhead, the network manager identifies the differences between the old and the newly computed schedules, and distributes only the modified portions of the global schedule to the affected network devices. Our system supports two schedule modification commands ADD and DELETE. A DELETE command removes the transmission

of a flow from a time slot, while an ADD command schedules a new transmission in a time slot. An ADD command contains a schedule entry associated with a new transmission. A schedule entry is represented by the following attributes: slot offset (2 bytes), channel offset (4 bits for the maximum of 16 channels), sender (1 byte), receiver (1 byte), flow ID (1 byte), and slot type (1 bit). The flow ID indicates which flow a transmission belongs to, and the slot type specifies whether a slot is dedicated or shared. For a DELETE command, it is sufficient to specify only the sender (1 byte), the receiver (1 byte), and the time slot (2 bytes) of the transmission to be removed. The WirelessHART standard suggests that the number of nodes in a network should not exceed 80, therefore, 1 byte is enough to address all nodes in the network. Hence, it requires 6 bytes to ADD a new schedule, and 4 bytes to remove a schedule entry. Note that the maximum packet length of the IEEE 802.15.4 PHY layer is 127 bytes [29], and the MAC payload can contain up to 98 bytes [30].

To constructs packets, the update scheduler follows these steps:

1. It sorts the schedule update commands in decreasing order of flow priority, so that higher priority flow schedules are included in earlier packets. For each flow, DELETE commands precede ADD commands, since a schedule entry needs to be removed before a new entry is added.

2. It adds schedule update commands into a packet until the packet is full, then creates a new packet.

3. It assigns each packet a sequence number. Packets receiving smaller sequence numbers contain schedules of higher priority flows.

The update scheduler distributes packets with lower sequence numbers first, and ensures that all intended recipients have received the packets before disseminating packets with higher

sequence numbers. Once a node receives each packet, it immediately updates its TSCH schedule, and executes a new schedule in the next superframe. Therefore, higher priority flows can recuperate and use a new schedule, while the update scheduler is still disseminating and updating schedules of the lower priority flows.

Broadcast Graph

We use a broadcast graph to disseminate update commands. Compared to installing multiple downstream control flows from access points to every network device, a broadcast graph requires fewer time slots to deliver a packet to every network device. In addition, with fewer allocated time slots, nodes can save the energy cost when no packet is being distributed, because every node scheduled to receive in a time slot must always listen on a channel for at least 2.2 ms [83] to detect if there is an incoming packet or not. If there is not, the node will turn off its radio. Otherwise, it continues to receive the packet.

A broadcast graph consists of two root nodes (access points), multiple intermediate nodes that receive and forward packets, and multiple leaf nodes, which only receive packets. To provide route diversity and ensure reliable dissemination of packets containing update commands, we require that an intermediate or a leaf node must have two parents, and an intermediate node must receive a packet from both parents before broadcasting a packet to its children.

Since each packet contains schedules intended for different sets of nodes, it is energy-inefficient to disseminate every packet to every node. Therefore, we allow intermediate nodes in a broadcast graph to forward a packet only when they are on a path to packet's destinations. To realize such a forwarding mechanism, we install on intermediate nodes information about each node's children and its descendants in a broadcast graph. Hence, when a node checks the destinations of a packet, it can decide whether to forward a packet. Figure6.5 presents a

broadcast graph in which each node has two parents. Here, if a packet is destined for nodes f and h, then only nodes b, c, and f can forward the packet.

When a broadcast graph is also affected by link failure, its schedule will be altered as well. A broadcast graph's new schedule will be disseminated after the schedules of all affected data flows have been updated. To ensure that all nodes use the same broadcast graph during the current schedule update, a new broadcast graph schedule is executed at a later time, when the network again requires schedule reconfiguration.

Handling Packet Losses

We use a simple acknowledgement scheme to confirm that nodes have received all their schedule update packets. For every new packet a node receives, the node sends an ACK back to the network manager through an upstream flow. Similar to health reports and failure notifications, we allow an ACK to be piggybacked onto a data packet. Since there can be more than one packet intended for a node, an ACK must contain a packet number identifying the specific packet a node received. The update scheduler ensures that it receives ACKs from all recipients of the packet. Otherwise, it needs to resend that packet until the packet reaches all destinations. If the manager sends a packet at the beginning of superframe x, then it expects to receive ACKs from all destination nodes at the end of superframe x+1, because a packet may arrive at the node after an upstream flow has already been scheduled in that superframe. In this case, an ACK will be sent in the next superframe.

Figure6.6 shows the timeline for a schedule update consisting of three packets, P1, P2, and P3. In the first superframe, the update scheduler distributes the first packet P1. It expects all ACKs for P1 to be delivered by the end of superframe 2. While waiting for P1's ACKs, the update scheduler distributes P2 in the second superframe, and checks if it has received all

ACKs for P1 at the end of that superframe. If some ACKs for P1 are absent, the scheduler will resend P1. If at least one node belonging to a flow does not receive a schedule, then the entire flow is not operational. Hence, to ensure that higher priority flows can complete the update first, the scheduler must resend P1 before sending P3.

Handling multiple link failures

Our system can handle multiple incoming link failure notifications. If a new failure notification arrives, while the network manager is computing a new schedule or when a new schedule has not been disseminated yet, then the network manager can halt those operations and recalculate a new schedule. Contrarily, if the dissemination process is on going as a new notification arrives, the manager will have to stop disseminating packets and start recomputing a new schedule again. In the meantime, nodes that already have received a schedule update will execute those partial new schedules. Since our system updates schedules based on flow priority, flows that already received an update will run a new schedule, while the other flows execute on the old schedule, as they wait for the manager to compute and distribute the most recent schedule.

6.7 Evaluation

To evaluate the performance of REACT, we develop a reconfiguration planner and an update engine to support schedule reconfiguration. We implement network manager software running on our server, and a protocol stack running on TinyOS 2.1.2 [78] and TelosB motes. The network manager can update the network topology, generate and update schedules and routes, and schedule update commands. In addition, we incorporate features necessary for schedule reconfiguration including health reporting, failure notification, and TSCH schedule updating into our previous WirelessHART stack implementation (see Section 3.3.3), which provides

Fixed-Priority Scheduling Policy
GAP: schedule transmissions based on a gap-induced scheduling policy EARLY: schedule transmissions of a flow in sequential order and select the earliest feasible slot for a transmission LATE: schedule transmissions of a flow in a reverse order and select the latest feasible slot for a transmission
Route Update Policy
PR: apply a partial reroute policy, and compute a new route using Dijkstra's shortest path algorithm RR: reroute using Dijkstra's shortest path algorithm
Schedule Update Policy
AFO: reschedule affected flows only ALL: reschedule affected flows and all of their lower priority flows

Table 6.1: Scheduling, route, and schedule update policies

basic network features such as TSCH MAC and source routing. We designate two nearby nodes in our testbed as access points. The server communicates with the access points through serial interfaces.

Following common practices for process monitoring and control applications, flows release packets periodically, and the periods of flows are harmonic. The periods are uniformly selected from the range $P = \{2^x, 2^{x+1}, ...2^y\}$. The manager constructs a collision-free TSCH schedule using fixed-priority scheduling, where only one transmission is allowed per channel in a time slot. We adopt WirelessHART source routing, which provides a single route from a source to a destination. For each transmission belonging to a data flow, the manager reserves an additional time slot for a sender to retransmit a packet if the sender does not receive an ACK from the receiver.

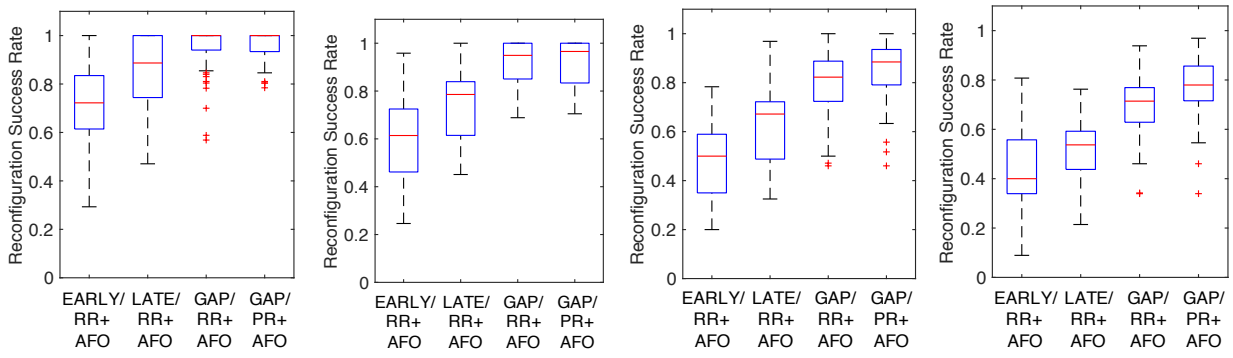
We quantify the performance of the reconfiguration planner through simulations based on two metrics: (1) the success rate in rescheduling a flow without modifying schedules of other flows, and (2) the number of packets required to update flow schedules. For the update engine, we run experiments on the testbed, present the resulting schedule reconfiguration

timelines, and measure the schedule dissemination latency and the energy cost. Table 6.1 summarizes different scheduling, route update, and schedule update policies that we compare against our work. By choosing combinations of the three policy types, we construct different approaches, each of the form *scheduling policy/route update policy + schedule update policy*.

6.7.1 Reconfiguration Planner Evaluation

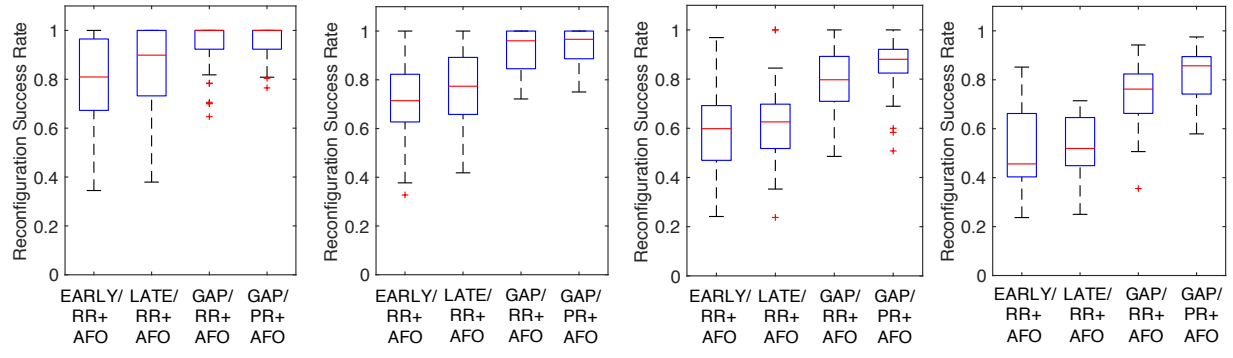
To evaluate the reconfiguration planner, we conduct simulation studies based on our testbed topology containing 60 nodes. The topology information includes the PRRs of all links in the network in all 16 channels. We use the topology to construct a communication graph in which links added to the graph have PRRs of no less than 90% in all channels and in both directions. We use channels 11 through 14. We randomly generate 50 flow sets under different traffic loads (i.e., when the numbers of flows are 20, 24, 28, and 32) by varying the locations of sources and destinations of flows and access points. For each flow set, we obtain a set of links, where each link in the set is randomly picked. In each experiment, one link in this set is selected as a failed link.

We consider two fixed-priority scheduling policies commonly adopted for real-time systems: deadline monotonic and rate monotonic policies. Following a deadline monotonic policy, flow priorities are determined according to deadlines (i.e., flows with shorter deadlines have higher priority), while a rate monotonic policy assigns priorities based on rate (i.e., flows with higher rates have higher priority). We set $P = \{2^{-1}, 2^0, 2^1\}$. Periods are uniformly assigned to flows in a flow set. With the deadline monotonic policy, if a flow F_i has a period $P_i = 2^x$, then its deadline D_i is randomly selected from the range $\{2^{x-1} + |\Gamma_i| * 2, 2^x\}$, where $|\Gamma_i|$ is the number of transmissions of F_i. For the rate monotonic policy, D_i is configured to be equal to P_i.



(a) 20 flows (b) 24 flows (c) 28 flows (d) 32 flows

Figure 6.7: Schedule reconfiguration success rates of AFO under the deadline monotonic policy.

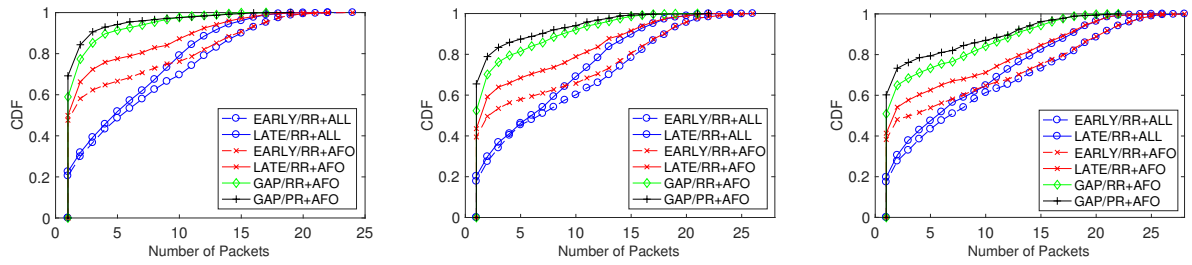


(a) 20 flows (b) 24 flows (c) 28 flows (d) 32 flows

Figure 6.8: Schedule reconfiguration success rates of AFO under the rate monotonic policy.

Schedule Reconfiguration Success Rate

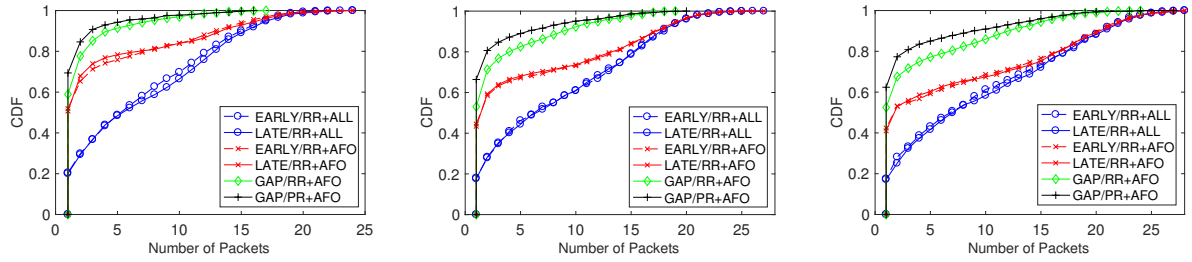
To evaluate the effectiveness of our gap-induced scheduling policy in enabling the AFO reconfiguration policy to update only the schedules of those flows affected by link failure, we quantify the schedule reconfiguration success rate of AFO. The success rate is defined as the fraction of cases of a flow set in which AFO successfully reschedules only flows using the failed link. We compare our two approaches GAP/RR+AFO and GAP/PR+AFO, against EARLY/RR+AFO and LATE/RR+AFO.



(a) 24 flows (b) 28 flows (c) 32 flows

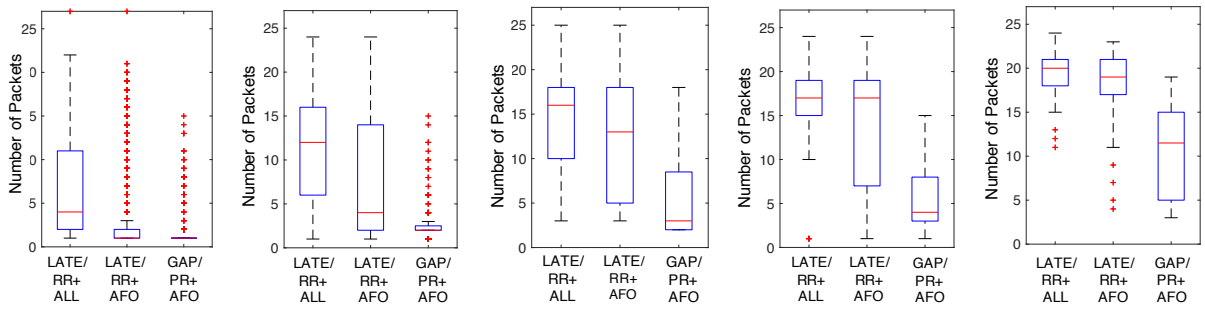
Figure 6.9: Total number of packets to be disseminated under the deadline monotonic policy.

Figure 6.7 shows box plots of the reconfiguration success rates of AFO under the deadline monotonic policy. GAP/PR+AFO demonstrates better improvement as the traffic load increases because it is more difficult for AFO to avoid rescheduling flows unaffected by link failure as more flows occupy the schedule, especially under the EARLY and LATE scheduling policies. With 32 flows, GAP/PR+AFO increases the median success rate by 95% and 44% compared to EARLY/RR+AFO and LATE/RR+AFO, respectively. Similar results can be observed under the rate monotonic policy, as shown in Figure 6.8. GAP/PR+AFO attains a higher median success rate than EARLY/RR+AFO and LATE/RR+AFO by 87.9% and 65%, respectively. Scheduling transmissions with the LATE policy provides better results than the earliest slot policy (EARLY). With LATE, a lower priority flow with a longer deadline can be allocated to time slots that are closer to its deadline and are not usable by a higher priority flow (i.e., slots between the deadline of the current packet and the release time of the next packet of a higher priority flow), which helps spread out transmissions of different flows, and makes it easier for AFO to reconfigure the schedule. The results manifest the benefit of GAP in improving the reconfiguration success rate of AFO and in preventing the modification of higher priority flow schedules from impacting those of lower priority flows. In addition, it also shows that PR policy can further enhance the reconfiguration success rate of AFO by at most 12.5% (Figure 6.8d) because PR allows flows' schedules to be partially reused.



(a) 24 flows (b) 28 flows (c) 32 flows

Figure 6.10: Total number of packets to be disseminated under the rate monotonic policy.



(a) 1-4 flows (b) 5-8 flows (c) 9-12 flows (d) 13-16 flows (e) 17+ flows

Figure 6.11: Impact of the number of flows associated with link failure on the number of packets required to update a schedule (under the rate monotonic policy).

Number of Packets to Disseminate

We examine the ability of GAP/PR+AFO to reduce the cost of adapting to a failed link by computing the number of packets required to update the global schedule, which is a direct quantification of the schedule reconfiguration overhead. We compare our approach with two additional baselines, EARLY/RR+ALL and LATE/RR+ALL, which utilize ALL as their reconfiguration policy. Moreover, if AFO cannot successfully reconfigure a flow's schedule, the manager will reschedule the flow, along with all of its lower priority flows, starting from the highest priority flow that AFO fails to reconfigure.

Figures 6.9 and 6.10 present the CDFs of the total number of packet to be disseminated. Again, with higher workloads, GAP/PR+AFO proves to be most effective. The results also demonstrate that GAP/PR+AFO outperforms other approaches in all settings, especially with 32 flows (Figure 6.10c), where only 15% of the cases need more than 5 packets to be disseminated. Conversely, under EARLY/RR+ALL, LATE/RR+ALL, EARLY/RR+AFO, and LATE/RR+AFO, for 57%, 58%, 41%, and 40% of the times, respectively, a network manger must distribute more than 5 packets. Compared to GAP/RR+AFO, GAP/PR+AFO reduces the number of cases requiring more than 5 packets by 32%.

Furthermore, we investigate how the number of flows associated with link failure impacts the number of packets needed to update a schedule. Figure 6.11 shows box plots of the number of packets to be disseminated when the number of the affected flows is withing the range $[a, b]$. Here, we consider flow sets consisting of 56 flows and schedules generated with the rate monotonic policy. We show only the results of LATE/RR+AFO and LATE/RR+ALL, since the LATE policy performs better than or similar to the EARLY scheme. GAP/PR+AFO outperforms the baselines, with drastically larger margins as the number of flows associated with link failure increases from [1,4] to [13, 16]. For example, LATE/RR+AFO has a 4.25X higher median number of packets than GAP/PR+AFO when the number of flows is within [13, 16]. However, as the number of flows exceeds 16, more flows need to be reconfigured, preventing AFO from successfully rescheduling flows, which in turn increases the number of packets to be disseminated. GAP/PR+AFO offers a notable reduction over the baselines in the number of packets required to reconfigure a schedule, which translates into shorter reconfiguration latency and lower energy consumption.

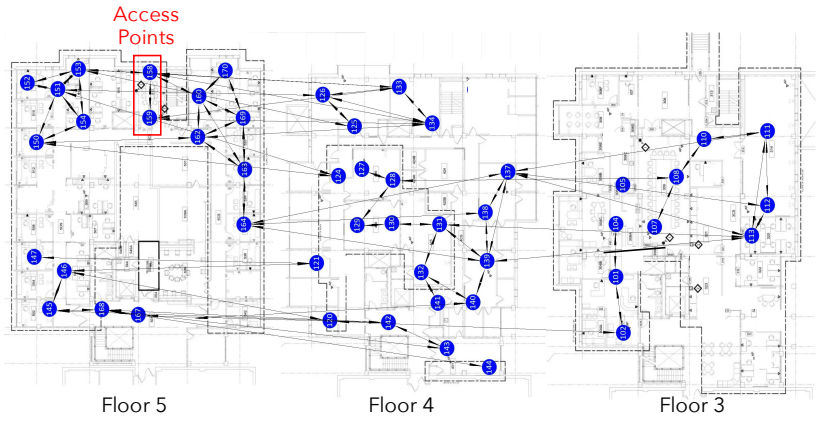


Figure 6.12: WUSTL testbed topology.

6.7.2 Update Engine Evaluation

In testbed experiments, We assess REACT's ability to improve schedule reconfiguration latency and reduce energy cost. Figure 6.12 shows the WUSTL testbed topology, consisting of 50 TelosB motes, which is equipped with an MSP430 MCU and a CC2420 radio compatible with the IEEE 802.15.4 PHY layer. The testbed spans three floors of an office building. Two nodes on the 5th floor are designated as access points, and are connected to the server through wired links. We run the network manager software on the server, and our TinyOS implementation of the WirelessHART protocol with schedule reconfiguration functionality on the motes.

We explore three different configurations: 16, 24, and 32 flows. Transmission schedules are generated based on the deadline monotonic policy. We opt to compare our GAP/PR+AFO approach against only LATE/RR+ALL and LATE/RR+AFO, since they outperform the baselines employing the EARLY policy under the deadline monotonic policy. We schedule a broadcast graph every 1 second. All flows generate packets with similar periods of 1 second. The deadline of each flow is chosen randomly from the range $[|\Gamma| * 2, 2^0]$, where $|\Gamma|$ is the number of transmissions of a flow. We allow control data (e.g., failure notification and

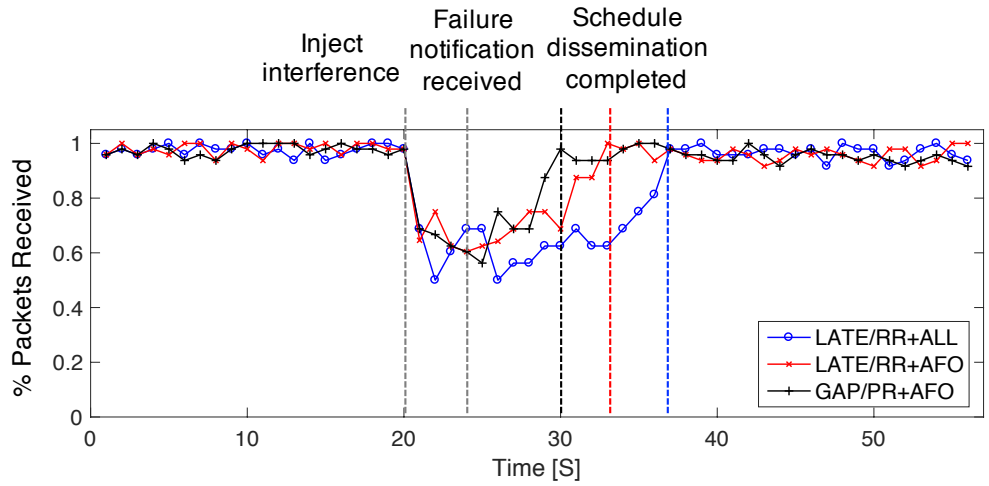
ACK) to be piggybacked onto data packets, so no additional control flow is installed. All configurations follow the same settings, except where stated otherwise.

To demonstrate the benefit of REACT in reducing rescheduling overhead, we pick one critical link used by 85-90% of flows as a failed link. To simulate network dynamics, WiFi interference is introduced close to the selected link. We use two Raspberry PIs, which are parts of our testbed infrastructure, to generate 5MB traffic on WiFi channel 1 overlapping with IEEE 802.15.4 channels 11 to 14 while the nodes communicate on channels 13 to 15. A node reports a link failure to the network manager once a link's PRR falls below 90%, and employs a sliding window of size 100 to calculate the link's PRR. To obtain average measurements, we repeat the experiment five times for each configuration.

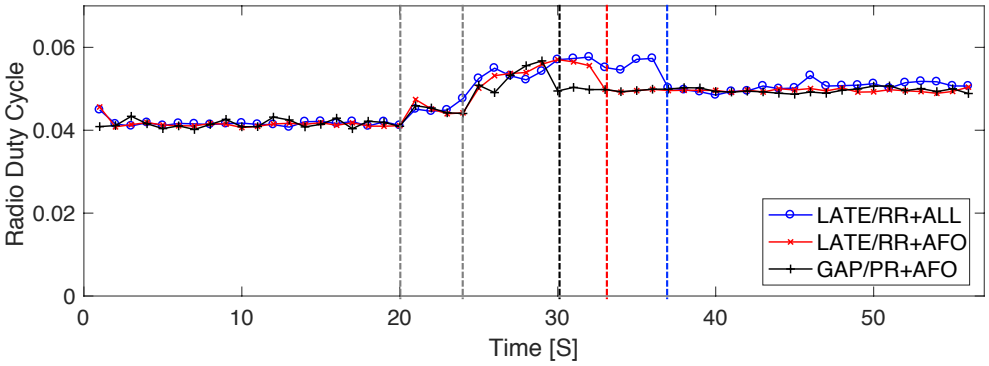
Note that with centralized scheduling, the schedule reconfiguration overhead depends on several factors (e.g., the periods of a broadcast graph and control flows, the reliability requirement, and the sliding window size), which introduce different tradeoffs. For instance, scheduling a broadcast graph less frequently preserves network resources, but incurs more reconfiguration latency. Selecting the best values for these parameters is not within the scope of this work, which focuses on reducing the portions of the global schedule being modified so fewer packets are required for the schedule update.

Schedule Reconfiguration Timeline

Figure 6.13 presents the schedule reconfiguration timeline to validate the correctness of our implementation. We quantify two metrics: (1) the radio duty cycle, the fraction of time a node has its radio on, and (2) the percentage of packets successfully received at their destinations. The network consists of 16 flows. When the network is stable, all approaches achieve a high percentage of packets received and incur low duty cycles. At time=20, interference is injected



(a) Percentage of packets received.



(b) Radio duty cycle.

Figure 6.13: Schedule reconfiguration timeline.

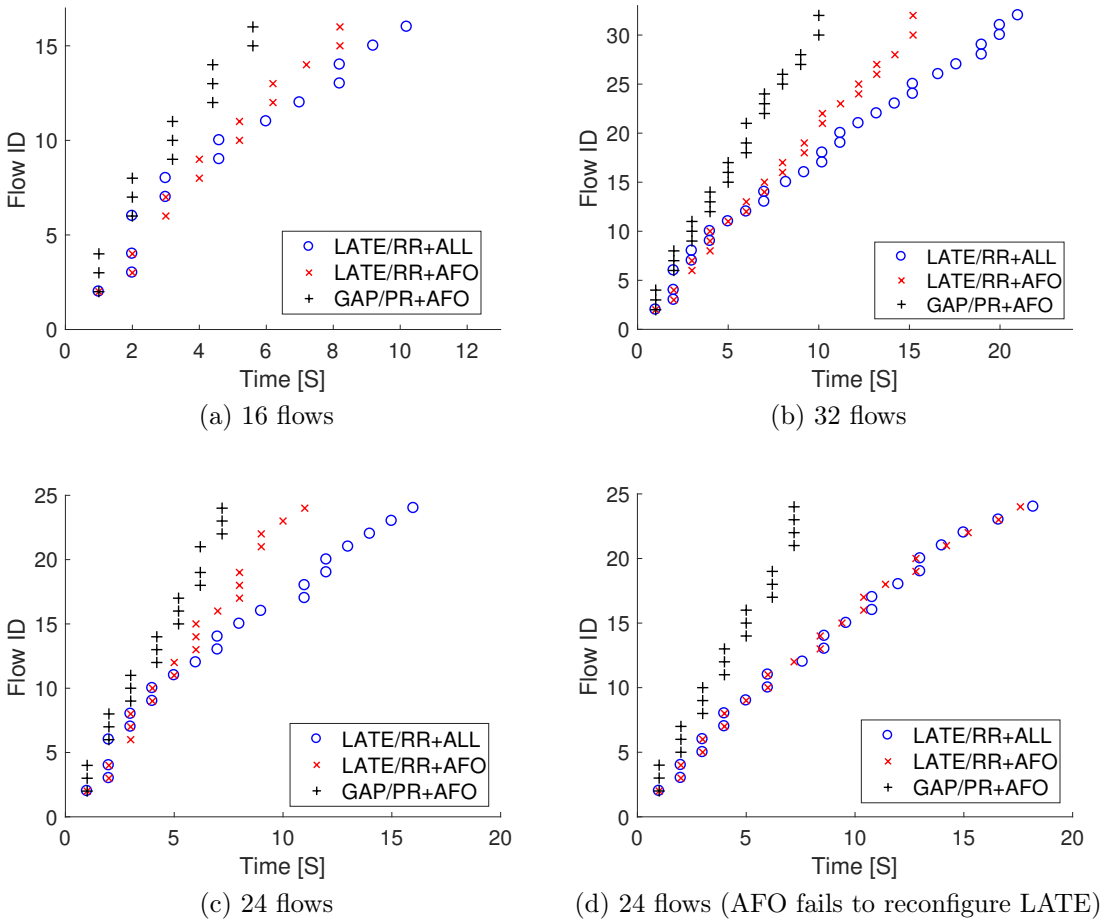
to degrade the link quality, and the percentage of packets received starts to decrease. Because the failed link is used by multiple flows, nodes can detect link quality degradation and notify the network manager quickly. In this setting, the network manager takes less than 10 ms to recompute a new schedule. So after receiving the notification at time=24, the manager can promptly begin disseminating a new schedule at time=25. Note that based on our simulation studies, with 32 flows, we observe a maximum execution time of 13.5 ms for PR+AFO. The execution time is measured on a Macbook Pro laptop with a 2.7 GHz Intel Core i7.

During the schedule reconfiguration phase, we notice the percentage of packets received drops, and the radio duty cycle increases because nodes need to retransmit a packet more often and they also participate in schedule distribution. Schedule reconfiguration finishes at times 30, 33, and 37 for GAP/PR+AFO, LATE/RR+AFO, and LATE/RR+ALL, respectively. GAP/PR+AFO provides 25% and 43.8% improvements in schedule reconfiguration latency over LATE/RR+AFO and LATE/RR+ALL, since GAP/PR+AFO requires fewer packets to alter the schedule. After schedule reconfiguration, the network performance returns to normal, and the slightly higher radio duty cycle is due to longer flow routes, which requires more transmissions.

Note that process monitoring applications may be able to tolerate some packet losses during the reconfiguration process. However, for more time-sensitive control applications, it is crucial to ensure that flows can still operate to meet the real-time requirement when the network suffers from link failure. Therefore, graph routing (a multi-path routing strategy supported by the WirlessHART protocol) should be adopted for this class of application to ensure reliable communication.

Schedule Dissemination Latency

We first validate that our policies indeed meet their goal of reducing schedule dissemination time. Latency is measured from when the first packet is disseminated until the schedules of all impacted flows are modified or all the affected nodes receive their new schedules. Figure 6.14 presents the latency (in seconds) when a schedule of each flow is updated under different workloads, and Figure 6.15 plots the time required for each related node to receive a complete schedule. Thus, the dissemination latency here is the time when the last flow is updated or the last node has received the full schedule. In Figures 6.14a, 6.14b, and 6.14c, GAP/PR+AFO reduces the schedule dissemination latency by approximately 45.1-55% and 31.7-34.5% over



(a) 16 flows

(b) 32 flows

(c) 24 flows

(d) 24 flows (AFO fails to reconfigure LATE)

Figure 6.14: Flow schedule update latency under different traffic loads.

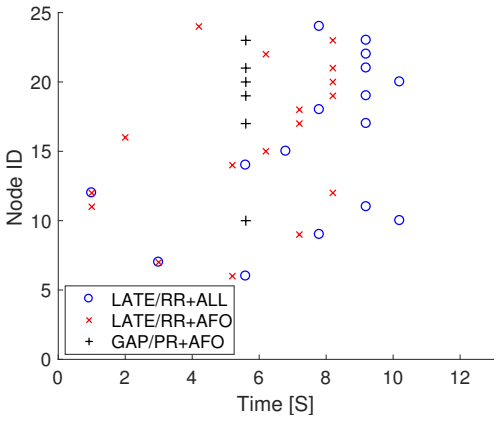
LATE/RR+ALL and LATE/RR+AFO, respectively. In these cases, the network manager successfully reschedules only flows affected by the failed link for LATE/RR+AFO. With more flows, it takes longer to disseminate a new schedule, since more flows are affected by the change.

We introduce an extra scenario where AFO fails to update only flows associated with the failed link for LATE/RR+AFO. Here, the manager reschedules the remaining flows that could not be reconfigured by AFO, and also reschedules all of their lower priority flows. To demonstrate such a case, we adjust the flows' periods by letting 50% of flows release their packets every 0.5 second, and the rest release their packets every 1 second. Figure 6.14d shows the results under such a scenario with 24 flows. Compared to LATE/RR+ALL and LATE/RR+AFO, GAP/PR+AFO further lowers the schedule update latency by 60.4% and 60%, respectively.

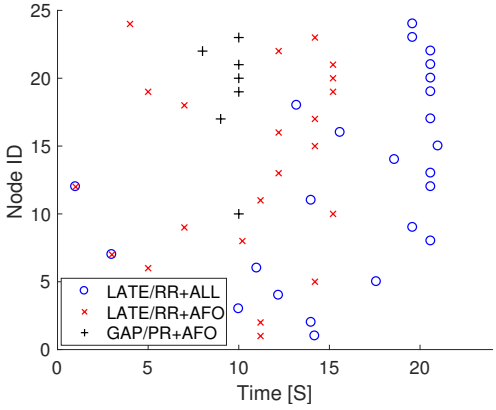
These results indicate the schedule dissemination latency achieved with GAP/PR+AFO is considerably lower than that of other approaches. This reduction shows the complementary benefit of our scheduling and reconfiguration policies in enabling partial schedule reuse, which reduces the amount of schedule-related information to be broadcast once link failure is detected. Furthermore, the results in Figure 6.14 also verify that our update scheduler indeed modifies schedules of higher priority flows (i.e., flows with smaller ID) first. Therefore, these more critical flows running at higher rates suffer less from packet losses.

Energy Consumption

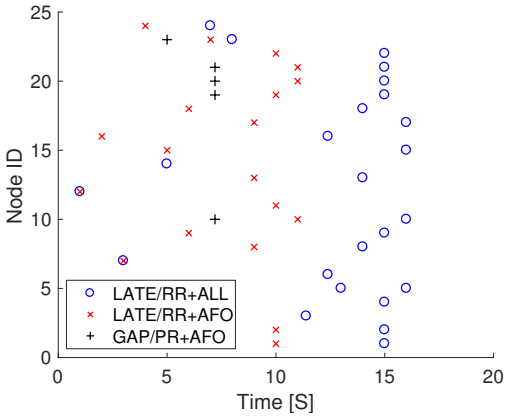
We next examine the performance of REACT in terms of energy efficiency. We measure the radio on time on each node and compute the energy consumption in mJ. According to the CC2420 radio specification [12], the power requirements for a transmission and a reception



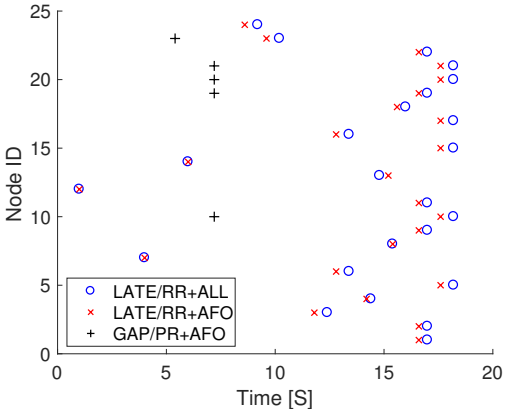
(a) 16 flows



(b) 32 flows

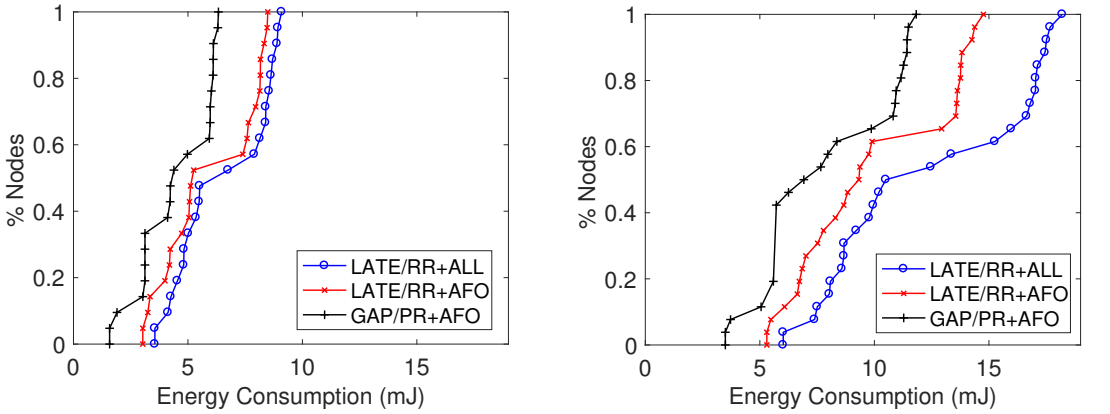


(c) 24 flows



(d) 24 flows (AFO fails to reconfigure LATE)

Figure 6.15: Latency in which each affected node has received a completed schedule.



(a) 16 flows

(b) 32 flows

(c) 24 flows

(d) 24 flows (AFO fails to reconfigure LATE)

Figure 6.16: CDF of node's energy consumption in mJ.

are 52.2 mW and 59.2 mW, respectively. The results presented in this section are obtained from the experiments in Section 7.2.2. Figure 6.16 shows the CDFs of a node's average energy consumption. GAP/PR+AFO significantly improves the energy cost over the two baselines. For example, with 32 flows (Figure 6.16b), under GAP/PR+AFO, 42% of the nodes consume less than 6 mJ, while for LATE/RR+AFO, only 8% of the nodes have energy costs lower than 6 mJ. In contrast, with LATE/RR+ALL, all nodes require more than 6 mJ.

Our approach proves to be energy efficient in all configurations for two reasons: It disseminates fewer packets, and it reduces the number of nodes affected by the schedule update. As shown in Figure 6.15, many fewer nodes are affected by the schedule modification under GAP/PR+AFO than under either LATE/RR+AFO or LATE/RR+ALL. For instance, when the number of flows is 24, compared to LATE/RR+AFO and LATE/RR+ALL, GAP/PR+AFO decreases the number of affected nodes by 78.2% and 73.7%, respectively. Reducing the number of nodes impacted by the schedule update results in fewer nodes participating in schedule dissemination, since we allow nodes in the broadcast graph to forward a packet only when they are on a path to the packet's destinations, thereby improving the node's energy efficiency.

6.8 Summary

To meet the stringent demands of industrial applications for real-time and reliable performance, WirelessHART adopts a centralized management architecture to provide deterministic and reliable communication. Despite these benefits, the centralized architecture introduces high adaptation overhead for updating a transmission schedule when a network experiences link failures. In this work, we design and implement REACT, a novel WirelessHART control plane to handle network adaptation. REACT includes a reconfiguration planner and an update engine to support efficient and reliable schedule reconfiguration. We implement and

evaluate REACT on a WSAN testbed. The results show that our system reduces the schedule dissemination latency by over 60%. At the same time, it significantly decreases the number of nodes affected by schedule updates, which in turn improves the node energy efficiency.

Chapter 7

Conclusion

With the emergence of industrial IoT, process industries such as oil refinery and chemical plants have increasingly adopted WSANs. WSANs offer a promising solution to reduce deployment and maintenance cost compared to wired infrastructure. To meet the stringent real-time and highly reliability requirements of industrial applications, WirelessHART adopts a set of specific features such as TSCH MAC and centralized network architecture. In this dissertation, we explored different networking designs of industrial WSANs, and developed and experimentally assessed new network protocols and algorithms to enhance performance, efficiency, and agility of industrial WSANs. First, complementary to recent research on theoretical aspects of WSAN design, we implemented a suite of network features of the WirelessHART standard in TinyOS and TelosB motes and then performed a series of empirical studies on WSAN protocol designs. Then, we investigated WSAN's channel selection mechanism, and designed link and channel selection strategies that balance channel and route diversity. Experimental results showed that our algorithms can drastically improve routing and scheduling of industrial WSANs. Next, to enhance real-time performance, we developed a conservative channel reuse algorithm that introduces channel reuse based on the real-time constraints of the

flow set. we further presented a policy to detect reliability degradation caused by channel reuse. Our approach significantly improves real-time performance while maintaining a high degree of reliability. Finally, we designed and implemented REACT a novel WirelessHART control plane to handle network adaptation. REACT includes a reconfiguration planner and an update engine to support efficient and reliable network reconfiguration in response to changing wireless environment. Evaluation results suggested significant reduction in schedule reconfiguration overhead in terms of the schedule dissemination latency and energy efficiency.

References

[1] Nicola Accettura, Elvis Vogli, Maria Rita Palattella, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. "Decentralized Traffic-Aware Scheduling in 6TiSCH Networks: Design and Experimental Evaluation." In: *IEEE Internet of Things Journal* 2.6 (2015), pp. 455–470.

[2] Adnan Aijaz and Usman Raza. "DeAMON: A Decentralized Adaptive Multi-Hop Scheduling Protocol for 6TiSCH Wireless Networks." In: *IEEE Sensors Journal* 17.20 (2017), pp. 6825–6836.

[3] Johan Åkerberg, Mikael Gidlund, and Mats Björkman. "Future Research Challenges in Wireless Sensor and Actuator Networks Targeting Industrial Automation." In: *INDIN*. 2011, pp. 410–415.

[4] Khaldoun Al Agha et al. "Cross-Layering in an Industrial Wireless Sensor Network: Case Study of OCARI." In: *Journal of Networks* 4.6 (2009), pp. 411–420.

[5] Mansoor Alicherry, Randeep Bhatia, and Li Li. "Joint Channel Assignment and Routing for Throughput Optimization in Multi-Radio Wireless Mesh Networks." In: *MobiCom*. 2005, pp. 58–72.

[6] Lichun Bao and JJ Garcia-Luna-Aceves. "A New Approach to Channel Access Scheduling for Ad Hoc Networks." In: *MobiCom*. 2001, pp. 210–221.

[7] *Bluetooth 4.0 Specification*. URL: `https://www.bluetooth.com/specifications`.

[8] Gurashish Brar, Douglas M Blough, and Paolo Santi. "Computationally Efficient Scheduling with the Physical Interference Model for Throughput Improvement in Wireless Mesh Networks." In: *MobiCom*. 2006, pp. 2–13.

[9] James Brown, Ben McCarthy, Utz Roedig, Thiemo Voigt, and Cormac Sreenan. "BurstProbe: Debugging Time-Critical Data Delivery in Wireless Sensor Networks." In: *Wireless Sensor Networks*. 2011, pp. 195–210.

[10] Ryan Brummet, Dolvara Gunatilaka, Dhruv Vyas, Octav Chipara, and Chenyang Lu. "A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks." In: *ICII*. 2018, pp. 79–88.

[11] Marco Caccamo, Lynn Y Zhang, Lui Sha, and Giorgio Buttazzo. "An Implicit Prioritized Access Protocol for Wireless Sensor Networks." In: *RTSS*. 2002, pp. 39–48.

[12] *CC2420 Documentation*. URL: http://www.ti.com/lit/ds/symlink/cc2420.pdf.

[13] Octav Chipara, Chengjie Wu, Chenyang Lu, and William Griswold. "Interference-Aware Real-time Flow Scheduling for Wireless Sensor Networks." In: *ECRTS*. 2011, pp. 67–77.

[14] Kaushik R Chowdhury and Ian F Akyildiz. "Interferer Classification, Channel Selection and Transmission Adaptation for Wireless Sensor Networks." In: *ICC*. 2009, pp. 1–5.

[15] *CoAP: The Constrained Application Protocol*. URL: https://tools.ietf.org/html/rfc7252.

[16] *CoMI: CoAP Management Interface*. URL: https://tools.ietf.org/html/draft-ietf-core-comi-01.

[17] Behnam Dezfouli, Marjan Radi, and Octav Chipara. "REWIMO: A Real-Time and Reliable Low-Power Wireless Mobile Network." In: *ACM Trans. Sen. Netw.* 13.3 (2017), 17:1–17:42.

[18] Manjunath Doddavenkatappa, Mun Choon Chan, and Akkihebbal L Ananda. "Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed." In: *TRIDENTCOM*. 2011, pp. 302–316.

[19] Manjunath Doddavenkatappa, Mun Choon Chan, and Ben Leong. "Improving Link Quality by Exploiting Channel Diversity in Wireless Sensor Networks." In: *RTSS*. 2011, pp. 159–169.

[20] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. "Orchestra: Robust Mesh Networks through Autonomously Scheduled TSCH." In: *Sensys*. 2015, pp. 337–350.

[21] *Emerson Industrial Wireless Technology*. URL: https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology.

[22] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. "Low-power Wireless Bus." In: *Sensys*. 2012, pp. 1–14.

[23] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. "Efficient Network Flooding and Time Synchronization with Glossy." In: *IPSN*. 2011, pp. 73–84.

[24] Jimmi Grönkvist. "Interference-Based Scheduling in Spatial Reuse TDMA." PhD thesis. KTH, 2005.

[25] Song Han, Xiuming Zhu, Aloysius K. Mok, Deji Chen, and Mark Nixon. "Reliable and Real-time Communication in Industrial Wireless Mesh Networks." In: *RTAS*. 2011, pp. 3–12.

[26] Jan-Hinrich Hauer, Vlado Handziski, and Adam Wolisz. "Experimental Study of the Impact of WLAN Interference on IEEE 802.15.4 Body Area Networks." In: *EWSN*. 2009, pp. 17–32.

[27] Cunqing Hua, Song Wei, and Rong Zheng. "Robust Channel Assignment for Link-Level Resource Provision in Multi-Radio Multi-Channel Wireless Networks." In: *ICNP*. 2008, pp. 157–166.

[28] *IEEE 802.15.4 Physical Layer*. URL: https://standards.ieee.org/standard/802_15_4-2015.html.

[29] *IEEE 802.15.4e Standard*. URL: https://standards.ieee.org/standard/802_15_4e-2012.html.

[30] *IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH)*. URL: https://tools.ietf.org/html/rfc7554.

[31] *IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH)*. URL: https://datatracker.ietf.org/wg/6tisch/about/.

[32] George Irwin, J Chen, Adrian Mckernan, and William Scanlon. "Co-design of Predictive Controllers for Wireless Network Control." In: *IET Control Theory Applications* 4.2 (2010), pp. 186–196.

[33] *ISA 100 Standard*. URL: http://www.isa.org/isa100.

[34] Youngmin Kim, Hyojeong Shin, and Hojung Cha. "Y-MAC: An Energy-Efficient Multi-channel MAC Protocol for Dense Wireless Sensor Networks." In: *IPSN*. 2008, pp. 53–63.

[35] *Kolmogorov–Smirnov Test (K-S test)*. URL: https://en.wikipedia.org/wiki/Kolmogorov%5C%E2%5C%80%5C%93Smirnov_test.

[36] Ravish Kumar, Apala Ray, and Mallikarjun Kande. "Integration of WirelessHART Devices into Mitsubishi PLC for Plant Monitoring." In: *ICCRE*. 2018, pp. 209–212.

[37] Hieu Khac Le, Dan Henriksson, and Tarek Abdelzaher. "A Practical Multi-Channel Media Access Control Protocol for Wireless Sensor Networks." In: *IPSN*. 2008, pp. 70–81.

[38] Xiaojun Lin and S. Rasool. "A Distributed Joint Channel-Assignment, Scheduling and Routing Algorithm for Multi-Channel Ad-hoc Wireless Networks." In: *INFOCOM*. 2007, pp. 1118–1126.

[39] Jane W. S. Liu. "Real-Time Systems." In: 2000.

[40] Shucheng Liu, Guoliang Xing, Hongwei Zhang, Jianping Wang, Jun Huang, Mo Sha, and Liusheng Huang. "Passive Interference Measurement in Wireless Sensor Networks." In: *ICNP*. 2010, pp. 52–61.

[41] Erwan Livolant, Pascale Minet, and Thomas Watteyne. "The Cost of Installing a 6TiSCH Schedule." In: *AdHoc-Now*. 2016, pp. 17–31.

[42] Chenyang Lu et al. "Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems." In: *Proceedings of the IEEE* 104.5 (2016), pp. 1013–1024.

[43] Ritesh Maheshwari, Shweta Jain, and Samir R Das. "A Measurement Study of Interference Modeling and Scheduling in Low-power Wireless Networks." In: *Sensys*. 2008, pp. 141–154.

[44] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. "The Flooding Time Synchronization Protocol." In: *SenSys*. 2004, pp. 39–49.

[45] Mobashir Mohammad, XiangFa Guo, and Mun Choon Chan. "Oppcast: Exploiting Spatial and Channel Diversity for Robust Data Collection in Urban Environments." In: *IPSN*. 2016, pp. 1–12.

[46] Sirajum Munir, Shan Lin, Enamul Hoque, S. M. Shahriar Nirjon, John A. Stankovic, and Kamin Whitehouse. "Addressing Burstiness for Reliable Communication and Latency Bound Generation in Wireless Sensor Networks." In: *IPSN*. 2010, pp. 303–314.

[47] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. "Using Channel Hopping to Increase 802.11 Resilience to Jamming Attacks." In: *INFOCOM Minisymposium*. 2007, pp. 2526–2530.

[48] SM Shahriar Nirjon, John A Stankovic, and Kamin Whitehouse. "IAA: Interference Aware Anticipatory Algorithm for Scheduling and Routing Periodic Real-time Streams in Wireless Sensor Networks." In: *INSS*. 2010, pp. 14–21.

[49] Marcelo Nobre, Ivanovitch Silva, and Luiz Affonso Guedes. "Routing and Scheduling Algorithms for WirelessHART Networks: a Survey." In: *Sensors* 15.5 (2015), pp. 9703–9740.

[50] Tony O'Donovan et al. "The GINSENG System for Wireless Monitoring and Control: Design and Deployment Experiences." In: *ACM Transactions on Sensor Networks* 10.1 (2013), 4:1–4:40.

[51] *OpenWSN*. URL: `https://openwsn.atlassian.net/wiki/spaces/OW/overview`.

[52] Jorge Ortiz and David Culler. "Multichannel Reliability Assessment in Real World WSNs." In: *IPSN*. 2010, pp. 162–173.

[53] Maria Rita Palattella, Nicola Accettura, Mischa Dohler, Luigi Alfredo Grieco, and Gennaro Boggia. "Traffic-Aware Scheduling Algorithm for Reliable Low-power Multi-hop IEEE 802.15.4e Networks." In: *PIMRC*. 2012, pp. 327–332.

[54] Sofia Pediaditaki, Phillip Arrieta, and Mahesh K. Marina. "A Learning-Based Approach for Distributed Multi-Radio Channel Allocation in Wireless Mesh Networks." In: *ICNP*. 2009, pp. 31–41.

[55] Wolf-Bastian Pottner, Hans Seidel, James Brown, Utz Roedig, and Lars Wolf. "Constructing Schedules for Time-Critical Data Delivery in Wireless Sensor Networks." In: *ACM Transactions on Sensor Networks* 10.3 (2014), pp. 1–31.

[56] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung Han, and Ratul Mahajan. "A General Model of Wireless Interference." In: *MobiCom*. 2007, pp. 171–182.

[57] K. N. Ramachandran, E. M. Belding, K. C. Almeroth, and M. M. Buddhikot. "Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks." In: *INFOCOM*. 2006, pp. 1–12.

[58] Bhaskaran Raman. "Channel Allocation in 802.11-Based Mesh Networks." In: *INFOCOM*. 2006, pp. 1–10.

[59] Bhaskaran Raman, Kameswari Chebrolu, Sagar Bijwe, and Vijay Gabale. "PIP: A Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer." In: *SenSys*. 2010, pp. 15–28.

[60] Anthony Rowe, Rahul Mangharam, and Raj Rajkumar. "RT-Link: A Global Time-synchronized Link Protocol for Sensor Networks." In: *Ad Hoc Networks* 6.8 (2008), pp. 1201–1220.

[61] *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. URL: `https://tools.ietf.org/html/rfc6550`.

[62] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. "End-to-End Communication Delay Analysis in Industrial Wireless Networks." In: *IEEE Transactions on Computers* 64.5 (2015), pp. 1361–1374.

[63] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. "Real-time Scheduling for WirelessHART Networks." In: *RTSS*. 2010, pp. 150–159.

[64] Mo Sha, Gregory Hackmann, and Chenyang Lu. "ARCH: Practical Channel Hopping for Reliable Home-Area Sensor Networks." In: *RTAS*. 2011, pp. 305–315.

[65] Mo Sha, Gregory Hackmann, and Chenyang Lu. "Real-world Empirical Studies on Multi-Channel Reliability and Spectrum Usage for Home-Area Sensor Networks." In: *IEEE Transactions on Network and Service Management* 10.1 (2013), pp. 1–9.

[66] Wei Shen, Tingting Zhang, Mikael Gidlund, and Felix Dobslaw. "SAS-TDMA: A Source Aware Scheduling Algorithm for Real-time Communication in Industrial Wireless Sensor Networks." In: *Wirel. Netw.* 19.6 (2013), pp. 1155–1170.

[67] Hossein Shokri-Ghadikolaei, Carlo Fischione, and Eytan Modiano. "On the Accuracy of Interference Models in Wireless Communications." In: *ICC*. 2016, pp. 1–6.

[68] *Siemens-WirelessHART: Innovative for Process Industry*. URL: `https://w3.siemens.com/mcms/sensor-systems/en/process-instrumentation/communication-and-software/wirelesshart/pages/wirelesshart.aspx`.

[69] Bruno Sinopoli, L Schenato, M Franceschetti, Kameshwar Poolla, Michael Jordan, and Shankar Sastry. "Kalman Filtering with Intermittent Observations." In: *IEEE Transactions on Automatic Control* 49.9 (2004), pp. 1453–1464.

[70] Jamila Ben Slimane, Ye-Qiong Songi, and Anis Koubaa. "Control and data channels allocation for Large-Scale UWB-based WSNs." In: *ComNet*. 2009, pp. 1–8.

[71] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. "An Empirical Study of Low Power Wireless." In: *ACM Trans. Sen. Netw.* 6.2 (2010), 16:1–16:49.

[72] Kannan Srinivasan, Mayank Jain, Jung Il Choi, Tahir Azim, Edward S Kim, Philip Levis, and Bhaskar Krishnamachari. "The k-factor: Inferring Protocol Performance Using Inter-Link Reception Correlation." In: *MobiCom*. 2010, pp. 317–328.

[73] Kannan Srinivasan, Maria A. Kazandjieva, Saatvik Agarwal, and Philip Levis. "The Beta-Factor: Measuring Wireless Link Burstiness." In: *SenSys*. 2008, pp. 29–42.

[74] Jian Tang, Guoliang Xue, and Weiyi Zhang. "Maximum Throughput and Fair Bandwidth Allocation in Multi-Channel Wireless Mesh Networks." In: *INFOCOM*. 2006, pp. 1–10.

[75] Lei Tang, Yanjun Sun, Omer Gurewitz, and David B. Johnson. "EM-MAC: A Dynamic Multichannel Energy-Efficient MAC Protocol for Wireless Sensor Networks." In: *MobiHoc*. 2011, p. 23.

[76] *TelosB Datasheet*. URL: `http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf`.

[77] Pascal Thubert, Maria Rita Palattella, and Thomas Engel. "6TiSCH Centralized Scheduling: When SDN Meet IoT." In: *CSCN*. 2015, pp. 42–47.

[78] *TinyOS*. URL: `http://webs.cs.berkeley.edu/tos/`.

[79] *uRES*. URL: `https://openwsn.atlassian.net/wiki/spaces/OW/pages/688147/uRES`.

[80] Ramanuja Vedantham, Sandeep Kakumanu, Sriram Lakshmanan, and Raghupathy Sivakumar. "Component Based Channel Assignment in Single Radio, Multi-Channel Ad Hoc Networks." In: *MobiCom*. 2006, pp. 378–389.

[81] Thomas Watteyne, Vlado Handziski, Xavier Vilajosana, Simon Duquennoy, Oliver Hahm, Emmanuel Baccelli, and Adam Wolisz. "Industrial Wireless IP-Based Cyber-Physical Systems." In: *Proceedings of the IEEE* 104.5 (2016), pp. 1025–1038.

[82] *Wireless Industrial Networking Alliance*. URL: `http://www.wina.org`.

[83] *WirelessHART Standard*. URL: `https://fieldcommgroup.org/technologies/hart`.

[84] Chengjie Wu, Dolvara Gunatilaka, Abusayeed Saifullah, Mo Sha, Paras Babu Tiwari, Chenyang Lu, and Yixin Chen. "Maximizing Network Lifetime of WirelessHART Networks under Graph Routing." In: *IoTDI*. 2016, pp. 176–186.

[85] Yafeng Wu, John A. Stankovic, Tian He, and Shan Lin. "Realistic and Efficient Multi-Channel Communications in Wireless Sensor Networks." In: *INFOCOM*. 2008, pp. 1193–1201.

[86] Kai Xing, Xiuzhen Cheng, Liran Ma, and Qilian Liang. "Superimposed Code Based Channel Assignment in Multi-Radio Multi-Channel Wireless Mesh Networks." In: *MobiCom*. 2007, pp. 15–26.

[87] Dong Yang, Youzhi Xu, Hongchao Wang, Tao Zheng, Hao Zhang, Hongke Zhang, and Mika Gidlund. "Assignment of Segmented Slots Enabling Reliable Real-Time Transmission in Industrial Wireless Sensor Networks." In: *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3966–3977.

[88] Tang Zhong, Cheng Mengjin, Peng Zeng, and Wang Hong. "Real-Time Communication in WIA-PA Industrial Wireless Networks." In: *ICCSIT*. 2010, pp. 600–605.

[89] Gang Zhou, Tian He, John A Stankovic, and Tarek Abdelzaher. "RID: Radio Interference Detection in Wireless Sensor Networks." In: *INFOCOM*. Vol. 2. 2005, pp. 891–901.

[90] Marco Zimmerling, Luca Mottola, Pratyush Kumar, Federico Ferrari, and Lothar Thiele. "Adaptive Real-Time Communication for Wireless Cyber-Physical Systems." In: *ACM Trans. Cyber-Phys. Syst.* 1.2 (2017), 8:1–8:29.