

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**SCHEDULE RANDOMIZATION BASED
COUNTERMEASURES AGAINST TIMING
ATTACKS IN REAL-TIME WIRELESS
NETWORKS**

**ANKITA SAMADDAR
SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING
2021**

Schedule Randomization Based Countermeasures Against Timing Attacks In Real-Time Wireless Networks

Ankita Samaddar

School of Computer Science & Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

18/08/2021

.....

Date

ITU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
ITU NTU NTU NTU NTU NTU NTU NTU
.....

.....
Ankita Samaddar
.....

Ankita Samaddar

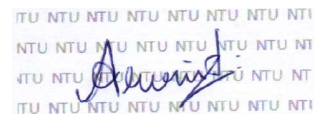
Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

18/08/2021

.....

Date



.....

Arvind Easwaran

Authorship Attribution Statement

This thesis contains material from two papers published in the following peer-reviewed journals and papers accepted at journals and conferences in which I am listed as an author.

Part of Chapter 3, Chapter 4 and Chapter 5 are published as [Ankita Samaddar, Arvind Easwaran and Rui Tan, “A Schedule Randomization Policy to Mitigate Timing Attacks in WirelessHART Networks”, Springer Real-Time Systems, Volume 56, Pages 452-489, October 2020](#) and [Ankita Samaddar, Arvind Easwaran and Rui Tan, “SlotSwapper: A Schedule Randomization protocol for Real-Time WirelessHART Networks”, Real-Time Networks \(RTN\), 2019.](#)

The contributions of the co-authors are as follows:

- Prof. Arvind Easwaran and Prof. Rui Tan guided me and provided me feedback on the problem solution and the paper draft.

18/08/2021

.....

Date

NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU
NTU NTU NTU NTU NTU NTU NTU NTU

.....

Ankita Samaddar

Acknowledgements

I have received constant support and help from several people throughout my Ph.D. I wish to express my sincere gratitude to each of them.

Foremost, I would like to thank my advisor Prof. Arvind Easwaran for unstintingly supporting me during the last four years of my Ph.D. Arvind introduced me to the world of research. His persistence, insightful and intensive discussions helped me to identify and tackle several challenging problems during my Ph.D. He gave me the academic freedom necessary to carry forward my research. His keen eye for details has helped a lot in the overall quality of this thesis. I would also like to thank my co-supervisor Prof. Rui Tan for his support during the initial years of my Ph.D. Rui's novel ideas and various discussions have contributed immensely to this thesis.

Besides my advisor, I would like to thank my thesis advisory committee members, Prof. Rui Tan, Prof. Anupam Chattopadhyay and Prof. Arindam Basu, for their valuable suggestions and encouragement that I received throughout the process. I would also like to thank my Ph.D. panel members Prof. Thambipillai Srikanthan, Prof. Lin Shang-Wei and Prof. Mohamed M. Sabry Ali for their valuable feedback.

It has been a great experience to spend four years in the School of Computer Science and Engineering, NTU. I want to thank the lab administrator Jeremiah Chua for setting up my lab environment. I want to thank all the past and present members of the Cyber-Physical Systems (CPS) research group. Especially, I want to thank Saravanan Ramanathan, Nitin Shivaraman, Chiam Zhonglin, Sundar Vijay Kumar, Sriram Vasudevan, Mohammed Shihabul Haque and others from whom I have learnt a lot. Besides, I also want to thank all other friends in Hardware and Embedded Systems Lab.

I would also like to thank all my friends in the university, Lavie Rekhi, Subhasiny Sankar, Nandish Chattopadhyay, Arko Dutt, Utsa Chattopadhyay and many others. A special thanks to Diotima Chatteraj, Joyjit Chatteraj, Pritijita Chatteraj and Partha Pratim Kundu for making me feel at home and constantly supporting me. I would also like to thank

my undergraduate friends Shreyasi Mukherjee and Mrinmoy Sadhukhan for constantly supporting and encouraging me over phone calls in the weekends.

I would also like to thank my mentor at Indian Statistical Institute, Prof. Ansuman Banerjee, for inspiring me to pursue Ph.D. abroad. I would like to thank Prof. Sourav Sengupta and Sananda Mitra for their constant support during the last four years at NTU.

Finally, I would like to thank my parents and my brother for their unconditional love and care. I am indebted to my family for giving me the freedom to choose my ambitions. Thank you for supporting me always and allowing me to pursue higher studies. Your inspiration helped me to carry forward and reach towards the completion of my Ph.D.

To my dear parents

Contents

Acknowledgements	ix
Summary	xvii
List of Figures	xxiii
List of Tables	xxiii
Abbreviations	xxv
List of Notations	xxvii
1 Introduction	1
1.1 Real-Time Cyber-Physical Systems	1
1.2 Real-Time Wireless Networks in Cyber-Physical System	3
1.3 Timing Attacks in Real-Time Wireless Networks	5
1.4 Thesis Motivation	6
1.5 Contributions	8
2 Background	11
2.1 Network Model	11
2.2 WirelessHART Network Descriptions	12
2.2.1 Key Features of a WirelessHART Network	12
2.2.2 WirelessHART Network Model	16
2.3 The 5G Cellular Network Descriptions	18
2.3.1 Communication Services in 5G Networks	18
2.3.2 5G New Radio (NR) Frame Structure	19
2.3.3 Scheduling URLLC flows in 5G Networks	20
2.3.4 5G Network Model	21
2.4 Metrics to Measure Schedule Randomization	23
3 Related Works	25
3.1 Timing Attacks in Wireless Networks	25
3.1.1 Traffic Analysis Attacks	26
3.1.2 Selective Jamming Attacks	27

3.1.3	Other Types of Timing Attacks in Wireless Networks	28
3.1.4	Timing Attacks in Real-Time Wireless Networks	28
3.2	Timing Attacks on Cache, GPU and Microarchitectural Platforms	29
3.2.1	Timing Attacks on Cache	29
3.2.2	Timing Attacks on GPU or Other Microarchitectural Resources	30
3.2.3	Timing Attacks on Real-Time Processors	31
3.3	Different Metrics on Schedule Randomization	32
4	Threat Models in WirelessHART and 5G Networks	33
4.1	Assumptions of the Threat Model for Real-Time Wireless Networks	33
4.2	Threat Model in WirelessHART Networks	34
4.3	Threat Model in 5G Networks	38
4.4	Consequences of Selective Jamming Attacks in WirelessHART and 5G Networks	41
4.5	Advantages of Selective Jamming over other types of Jamming in Real-Time Networks	42
5	Centralized Schedule Randomization in WirelessHART Networks	44
5.1	Introduction	45
5.2	System Model	46
5.3	Threat Model	46
5.4	Motivation of our work	47
5.5	Proposed MTD technique	49
5.5.1	Offline Schedule Generation Phase	51
5.5.2	Online Schedule Selection Phase	56
5.5.3	Time Complexity Analysis	58
5.6	Optimality of <i>SlotSwapper</i> for harmonic flows in single-channel WirelessHART network	58
5.6.1	Discussion	62
5.7	Measure of Uncertainty	64
5.8	Experiments and Evaluation	68
5.8.1	Experiments	69
5.8.2	Evaluation using K-L divergence	70
5.8.3	Security Analysis	74
5.8.4	Memory and Power Consumption	75
5.9	Conclusion	76
6	Online Distributed Schedule Randomization with Period Adaptation in Industrial Control Systems	78
6.1	Introduction	79
6.2	Related works on wireless networked-control systems	81
6.3	System Model	82
6.4	Threat Model	86
6.5	Proposed Distributed Schedule Randomization Technique	86

6.5.1	Offline Cluster Generation Phase	87
6.5.2	Online Distributed Schedule Generation Phase	88
6.5.3	Time Complexity Analysis	92
6.5.4	Period Adaptation	93
6.5.5	Extension of <i>DistSlotShuffler</i> to support period adaptation	95
6.6	Experiments and Evaluation	96
6.6.1	Experiments	101
6.6.2	Control Performance	102
6.6.3	WirelessHART Network Performance	112
6.6.4	Performance after Period Adaptation	118
6.6.5	Power Analysis	121
6.7	Conclusion	123
7	Online Schedule Randomization in 5G URLLC Communication	124
7.1	Introduction	125
7.2	Related Works on other types of attacks in 5G Networks	126
7.3	System Model	126
7.4	Threat Model	128
7.5	Motivation of our work in 5G	128
7.6	Proposed Countermeasure for Attack	128
7.6.1	Partitioning Scheduling Windows and Admission Control	129
7.6.2	Schedule Randomization Strategy: <i>PerRand</i>	131
7.6.3	Time Complexity of <i>PerRand</i>	136
7.6.4	Online Schedule Distribution	137
7.6.5	Scheduling of flows with 10ms period	138
7.7	Experiments and Evaluation	139
7.7.1	Experiments	140
7.7.1.1	Evaluation by K-L divergence :	144
7.7.1.2	Evaluation by Prediction Probability of slots :	148
7.7.1.3	Computation time	148
7.8	Conclusion	150
8	Conclusion and Future Works	151
8.1	Conclusions	151
8.2	Future Research Directions	154
	List of Author's Publications and Awards	156
	Bibliography	157

Summary

Cyber-physical systems are realized as those systems where the computation, communication and physical processes interact with each other (e.g., robotic systems, automobiles, etc). A large class of cyber-physical systems are associated with strict timeliness requirements. They are known as ‘real-time systems’ (e.g., smart grids, avionics, etc).

Industrial control systems (ICS) are real-time systems that consist of several closed-loop controls. The main components of a control loop are — a physical plant, sensors, actuators and a controller. The sensors transmit the measurement data to the controller that generates appropriate control signals to actuate the physical plant. Large-scale wireless sensor actuator networks form the main communication framework among the network devices in ICS. Most of the communications between each of these devices are periodic real-time flows with hard deadlines. To ensure reliability, the communication in these systems are time-division multiple access (TDMA) based. Dedicated network resources (time-slots and frequencies) are allocated to these devices in advance and the communication schedule is pre-computed to satisfy the hard deadlines of the real-time flows. The same schedule is repeated over time which makes the schedule predictable in nature. However, a malicious attacker can exploit this predictable time-slots in the schedules to launch timing attacks. Since these applications are time-critical, timing attacks can completely undermine the system performance leading the system to an unsafe state.

Schedule randomization is a popular defense mechanism to mitigate timing attacks. Although a few works exist in the literature that use schedule randomization as a countermeasure against timing attacks in real-time uniprocessor systems, none of them are applicable for real-time wireless networks. Schedule randomization as a countermeasure against timing attacks in real-time wireless networks remained largely unexplored. With this objective in focus, this thesis addresses some schedule randomization techniques to mitigate timing attacks in real-time wireless networks.

Among the existing wireless network protocols that are in use, the WirelessHART is the most suitable and widely adopted protocol in ICS. A WirelessHART protocol supports centralized architecture, TDMA based communication, multiple channels, etc. All

of these characteristics of a WirelessHART network facilitate reliable and predictable communication with real-time flow guarantees in ICS. However, the predictable communication exposes the system to timing attacks, such as selective jamming attacks. Selective jamming attacks are stealthy attacks, however, it can lead the system to an unstable state and can disrupt the safety of the system. As a countermeasure against such attack in WirelessHART network, we propose a centralized schedule randomization technique that randomizes the time-slots and channels in the schedule over every hyperperiod (least common multiple of the periods of the real-time flows) without violating the hard deadlines of the real-time flows, while still satisfying the feasibility constraints of a schedule in a WirelessHART network. The centralized technique generates the schedules offline and distributes the schedules online, hence, cannot support any change in topology in the network. Further, this technique has energy overheads in distributing the schedules to all the network devices at runtime. Hence, we propose a distributed online schedule randomization technique that can generate random feasible schedules at runtime in each network device without affecting the closed-loop control stability. To increase the extent of randomization of time-slots in the schedules, this online distributed technique adopts a period adaptation strategy that can adjust the transmission periods of the real-time flows at runtime depending on the stability of the closed-loop controls.

To support an ever-growing number of devices and dense connectivity, the fifth generation (5G) cellular networks is expected to serve as the main communication standard in the future wireless sensor networks. Among the different service categories supported by 5G, the ultra-reliable low-latency communication (URLLC) is mostly suitable for time-critical applications such as the ICS. The communication in 5G is organized into slots over multiple frequencies. To satisfy the hard deadlines of the URLLC flows in ICS, a part of the resources (slots and frequencies) in 5G are reserved for URLLC traffic and the same schedule is repeated over time. The repetition of the same schedule over time makes the slots in the schedules predictable. This, in turn, makes the 5G networks vulnerable to timing attacks. However, the proposed schedule randomization techniques for WirelessHART networks are not applicable for dynamic networks like 5G where the number of flows and the amount of available network resources for URLLC flows vary at runtime. Moreover, the feasibility of the real-time flows need to be guaranteed at runtime. Hence, we propose an online schedule randomization technique that randomizes the slots and the frequencies of the periodic URLLC flows while guaranteeing the feasibility of the flows at runtime.

List of Figures

1.1	An overview of an Industrial Control Systems	2
2.1	Overview of a WirelessHART network	13
2.2	A network graph with five vertices and six edges.	15
2.3	Applications under different services in 5G	19
2.4	Different slot durations in 5G corresponding to each sub-carrier spacing	19
4.1	A network graph with seven nodes	38
4.2	A schedule \mathcal{S} over 6 slots with two flows, F_1 (marked in red) and F_2 (marked in blue), where $src_1 = 1$, $src_2 = 4$, $des_1 = des_2 = D$	38
4.3	A schedule \mathcal{S} over 6 frames and 2 frequencies with three uplink flows F_1 (red), F_2 (blue), F_3 (brown); source devices $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; destination devices $des_1 = des_2 = des_3 = B$; periods: $p_1 = 20ms$, $p_2 = 30ms$, $p_3 = 60ms$; The slot duration in the schedule is $1ms$	40
5.1	A network graph with seven nodes	47
5.2	Two schedules \mathcal{S}_1 and \mathcal{S}_2 over 6 slots with two flows, F_1 (marked in red) and F_2 (marked in blue), where $src_1 = 1$, $src_2 = 4$, $des_1 = des_2 = D$	47
5.3	A network graph with 5 nodes	49
5.4	Three schedules \mathcal{S} , \mathcal{S}_1 and \mathcal{S}_2 over the network graph in Figure. 5.3 with two flows F_1 (red) and F_2 (blue).	49
5.5	Three graphs with $ V =100$ and (a) θ varying between 2 to 4, (b) θ varying between 3 to 6, (c) θ varying between 3 to 8	68
5.6	Mean upper-bound K-L divergence over Graph A, Graph B and Graph C with slot utilization varying between 0.0 to 0.9 and number of channels between 1 to 4	71
5.7	Upper-bound K-L divergence over (a) Graph A, (b) Graph B and (c) Graph C, with number of flows varying between 10 to 40 and number of channels between 1 to 4 over a hyperperiod of 2^{10} slots	73
5.8	Maximum Prediction Probability of slots over 10 hyperperiod schedules each with 1024 slots over Graph C with 40 flows and m varying between 1 to 4.	75
6.1	A wireless control system architecture consisting of a physical plant, a linear feedback controller and a single-channel WirelessHART network	82

6.2	(a) a single-channel WirelessHART network graph G (b) Three feasible schedules over of 10 slots for the graph G in Figure. 6.2(a) with two flows F_1 (marked in red) with $route_1 = \{1 \rightarrow 3 \rightarrow AP\}$, period $p_1 = 50ms$ (5 slots) and F_2 (marked in blue) with $route_2 = \{3 \rightarrow AP\}$, period $p_2 = 100ms$ (10 slots).	91
6.3	A wireless control loop in the GISOO simulator	99
6.4	(a) States $x(t)$ (b) Control Input $u(t)$ (c) Lyapunov function V_{xt} of $PLANT_1$ over 250s executing base schedule \mathcal{S}_B without schedule randomization	103
6.5	(a) States $x(t)$ (b) Control Inputs $u(t)$ (c) Lyapunov function V_{xt} of $PLANT_1$ over 250s executing <i>DistSlotShuffler</i> at the WirelessHART network	104
6.6	(a) States $x(t)$ (b) Control Inputs $u(t)$ (c) Lyapunov function V_{xt} (d) Period Adaptation of $PLANT_1$ under normal condition (with no physical disturbance) over a time period of 250s executing (1) base schedule \mathcal{S}_B (plots to the left) and (2) <i>DistSlotShuffler</i> (plots to the right) over every hyperperiod	108
6.7	A physical disturbance is applied at the output of the physical plant from 131s to 150s (after first period increase at 123s). Corresponding States $x(t)$; Control Input $u(t)$; Lyapunov function V_{xt} ; Period Adaptation of control loop L_1 ; over a time period of 300s upon executing \mathcal{S}_B every hyperperiod (plots shown on the left side) and on executing <i>DistSlotShuffler</i> (plots shown on the right side)	109
6.8	A physical disturbance is applied at the output of the physical plant from 176s to 195s (after second period increase at 173s). Corresponding States $x(t)$; Control Input $u(t)$; Lyapunov function V_{xt} ; Period Adaptation of control loop L_1 ; over a time period of 300s on executing \mathcal{S}_B every hyperperiod (plots shown on the left side) and on executing <i>DistSlotShuffler</i> (plots shown on the right side)	111
6.9	Upper-bound K-L divergence of (a) <i>SlotSwapper</i> (red) (b) <i>DistSlotShuffler</i> (blue) with slot utilization (i) 33.33% (ii) 50% (iii) 66.6% (iv) 75% over $hp = 60slots$. The X-axis shows the maximum memory consumed by the <i>SlotSwapper</i> (number of schedules) and <i>DistSlotShuffler</i> (number of keys: number of base schedules).	114
6.10	Prediction Probability of slots of (a) <i>SlotSwapper</i> (red) (b) <i>DistSlotShuffler</i> (blue) with slot utilization (i) 33.33% (ii) 50% (iii) 66.6% (iv) 75% over $hp = 60slots$. The X-axis shows the maximum memory consumed by the <i>SlotSwapper</i> (number of schedules) and <i>DistSlotShuffler</i> (number of keys: number of base schedules).	117
6.11	The plots (blue) shows the upper-bound K-L divergence of the <i>DistSlotShuffler</i> under normal condition at slot utilization (a) 25% (b) 50% (c) 75%. The plots (red and black) show the same after first and second period increases with no disturbance. The X-axis shows maximum memory consumption represented as (number of keys: number of base schedules).	121

7.1	Upper-bound K-L divergence. SCS: 15KHz (graphs on the left in each sub-figure), 30KHz (graphs on the right). Divergence values of <i>EDF_Rand</i> and <i>PerRand</i> are marked in red and blue, respectively.	143
7.2	PP of slots. SCS: 15KHz (graphs on the left in each sub-figure), 30KHz (graphs on the right). Divergence values of <i>EDF_Rand</i> and <i>PerRand</i> are marked in red and blue, respectively.	147
7.3	Average computation time (<i>ms</i>) of <i>EDF_Rand</i> (plots in pyramids) and <i>PerRand</i> (plots in circles).	149

List of Tables

2.1	A schedule \mathcal{S} over six slots on the network graph in Figure. 2.2 with two channels and two flows, F_1 (marked in red) and F_2 (marked in blue).	17
2.2	A schedule \mathcal{S} over 6 frames and 2 frequencies with two uplink flows F_1 (red), F_2 (blue); The slot duration in the schedule is 1ms.	23
5.1	List of notations used by <i>SlotSwapper</i>	50
5.2	A schedule over the network graph in Figure. 5.1 for two flows F_1 (marked in red) and F_2 (marked in blue) with $src_1 = 3$, $src_2 = 6$, $des_1 = D$, $des_2 = D$, $p_1 = 2$ slots and $p_2 = 3$ slots.	63
5.3	Calculation of K-L divergence on running the <i>Offline_Sched_Gen</i> for 10,000 hyperperiods over the set of flows in Example 5.1 considering \mathcal{S}_1 of Figure. 5.2 as the initial base schedule with reference to \mathbb{A}' that generates all possible feasible schedules	65
5.4	True <i>K-L divergence</i> for the graph in Figure. 5.1 with number of flows between 3 to 5	69
5.5	Number of slots with Maximum Prediction Probability = 0 over 10 hyperperiods each with 1024 slots over Graph <i>C</i> with 40 flows, and m varying between 1 to 4	75
6.1	List of notations used by <i>DistSlotShuffler</i>	87
6.2	Period Adaptation of the four control loops under normal condition (with no physical disturbance)	106
6.3	Mean Absolute Error (MAE) of the four control loops under different conditions	111
6.4	Total estimated power consumption of all the nodes over a hyperperiod of 240 slots executing the (1) <i>DistSlotShuffler</i> and (2) <i>SlotSwapper</i> at different slot utilization under (a) normal condition (b) after 1 st period adaptation (c) after 2 nd period adaptation	122
7.1	Two schedules \mathcal{S}_1 and \mathcal{S}_2 over 6 frames and 2 frequencies with three uplink flows F_1 (red), F_2 (blue), F_3 (brown); source devices $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; destination devices $des_1 = des_2 = des_3 = B$; periods: $p_1 = 20ms$, $p_2 = 30ms$, $p_3 = 60ms$; The slot duration in the schedule is 1ms.	135

Abbreviations

TDMA	time division multiple access
FDD	frequency division duplex
5G	fifth generation
NR	New Radio
URLLC	ultra-reliable low-latency communication
eMBB	enhanced Mobile Broadband
mMTC	massive Machine-Type Communication
OFDM	Orthogonal Frequency Division Multiplexing
V2X	Vehicle to Everything
SCS	Sub-Carrier Spacing
SPS	Semi-Persistent Scheduling
BS	base station
UE	user equipment
LTE	Long Term Evolution

3GPP Third Generation Partnership Project

K-L divergence Kullback-Leibler divergence

List of Notations

G	a network graph
V	set of vertices
E	set of edges
n	number of flows
m	number of channels/frequencies
M	number of control loops
\mathcal{F}	set of flows
F_i	i^{th} flow
F_{ij}	j^{th} instance of the i^{th} flow
f_{ij}	first sub-flow of the j^{th} instance of the i^{th} flow
\hat{f}_{ij}	second sub-flow of the j^{th} instance of the i^{th} flow
\mathcal{F}^k	set of flows associated with the k^{th} control loop
src_i	source of the i^{th} flow
des_i	destination of the i^{th} flow
p_i	period of the i^{th} flow
δ_i	relative deadline of the i^{th} flow
$route_i$	route of the i^{th} flow
r_{ij}	release time of the j^{th} instance of the i^{th} flow
$F_{ij.n_hops}$	total number of hops in the j^{th} instance of the i^{th} flow
F_0	idle flow
W_{ij}	scheduling window of the j^{th} instance of the i^{th} flow
\mathcal{C}	a cluster of flows with the same period and same scheduling window for all of their instances

\mathcal{N}	total number of clusters in a system
κ	a set of \mathcal{N} clusters
hp	hyperperiod of a schedule
\mathcal{U}	a group of n UEs in a 5G network
\mathcal{S}	a schedule over a hyperperiod
B	base station in a 5G network
U_i	i^{th} UE in a 5G network

Chapter 1

Introduction

1.1 Real-Time Cyber-Physical Systems

Cyber-physical systems are systems where the cyber world of communication and computation integrate, and control and interact with the physical world of sensors and actuators [1]. Cyber-physical systems have occupied an important position in today's technology spectrum. In recent years, cyber-physical systems are adopted across a wide range of modern day applications in factories, robotics, avionics, healthcare, etc. A large class of cyber-physical systems are associated with strict timeliness requirements, *i.e.*, these systems need to deliver their output within a fixed duration of time, pre-specified by the system. This implies that the correctness of these systems are verified both in terms of functionality and timeliness. A potential failure in these systems result in catastrophic consequences to human lives and environment. These systems are more specifically known as *real-time systems* and the pre-specified timing constraints are referred to as *deadlines* [2]. The correctness of a real-time system depends not only on the logical correctness of computation but also on the temporal correctness.

Depending on the criticality of the real-time applications and the consequence of deadline miss, real-time systems can be categorized as *hard* or *soft*. For example, a flight control application is a *hard real-time system* where a deadline miss can result in catastrophic consequences such as system loss, environment damage or even death [3]. For this reason, service guarantees are to be provided to these systems within strict deadlines. Such deadlines are considered as *hard deadlines*. On the other hand, an infotainment system is a *soft real-time system* where occasional deadline misses may lead to a degraded

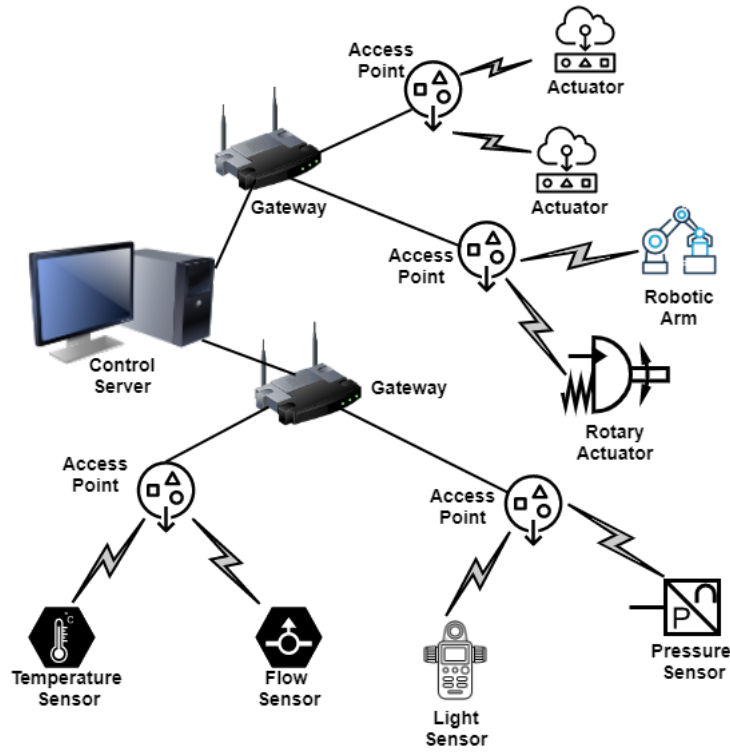


FIGURE 1.1: An overview of an Industrial Control Systems

quality of service, but do not have catastrophic consequences. Security in hard real-time system is a major concern. A malicious attacker may alter the timing behavior of a hard real-time system, resulting in deadline misses [4]. Since these systems are time-critical or safety-critical in nature, such attacks leading to deadline misses, may result in safety hazards.

Real-time cyber-physical systems span across various fields of technology, such as industries, medical devices, automotive, smart power grids, etc. We consider one such real-time cyber-physical system, the industrial control systems (ICSs), as a representative example in this thesis.

A large class of applications in modern day industries are real-time systems, where a network of devices, machinery, sensors, actuators, robots and other automation devices, interact with each other using industrial protocols to take decisions in real-time. Figure. 1.1 provides an overview of an ICS. The communication between these devices are either wired or wireless in nature. With rapid advancement in technology and application of automation in modern day industries, the number of devices in industries keep increasing day by day. However, wired networks have limited flexibility in deployment

and high installation and maintenance costs. Hence, the communication network in most ICSs are switching from wired to wireless communication [5].

Apart from ICSs, this is also true for other types of real-time cyber-physical systems, such as automotives. In the automotive industry, a modern car that integrates hundreds of sensors and actuators requires cables of approximately $4Km$; integrating a new sensor to the system may require the system to re-route the cables which is costly and not scalable [6]. Thus, wireless sensor actuator networks are gaining relevance as the main communication framework in many real-time cyber-physical systems.

1.2 Real-Time Wireless Networks in Cyber-Physical System

A large number of wireless protocols are already being deployed across a multitude of cyber-physical system applications. Examples of such wireless protocols are Bluetooth [7], Zigbee [8], WirelessHART [9], etc. However, only a few of them can satisfy the strict timeliness requirements of real-time systems. Moreover, the wireless networks are deployed in open areas where the network is highly exposed to interference. For example, a remote diagnostics and maintenance unit in an ICS is responsible for identifying, preventing, and recovering the system from abnormal operations or failures [10]. Such devices need to communicate with the remote devices in real-time to maintain safety of the system. Hence, only those wireless protocols that are highly reliable and can guarantee the strict timeliness requirements are suitable to serve as the communication network protocol in real-time cyber-physical systems.

In this thesis, we consider two such real-time wireless protocols:

1. The WirelessHART protocol [9], that has been extensively adopted in real-time cyber-physical systems, particularly in ICSs, because of its reliable guaranteed service within strict deadlines. A survey states that until 2012, WirelessHART had been adopted in over 8,000 manufacturing systems worldwide [11].
2. The ultra-reliable low-latency communication (URLLC) of the fifth generation (5G) cellular networks, that has the capability to provide service to real-time applications in the future wireless systems with very high reliability (99.99%) and very low latency ($1ms$) [12].

Because of the existing popularity of WirelessHART protocol in ICSs and the anticipated popularity of the 5G cellular network protocol in the future wireless networks, these two protocols are chosen in this thesis.

A majority of the applications in real-time systems are associated with periodic data communication, *i.e.*, the communication repeats after a specific interval of time. For example, in automotive, a safety application in modern Vehicle to Everything (V2X) communication is associated with basic safety messages (information about vehicular position, vehicular velocity, etc.) that are broadcasted every $100ms$ with high reliability to avoid collision [13]. Any interruption in the packet transmission results in catastrophic consequence such as collision of the vehicle. Similarly, in an ICS, a conveyor belt application consists of simple processing elements, a programmable logic controller and some input sensors. The communication between the processing elements are periodic and the same sequence of data is transmitted over the network repeatedly [14]. Another example is a Distributed Control System (DCS), mainly used in oil refinery, manufacturing, etc., that consists of a centralized supervisory control loop to manage multiple local controllers or devices in a production process [10]. Each of these communications is a periodic data flow and is associated with a hard deadline. Any communication failure in these systems can cause instability of the control loops, resulting in safety hazards to the environment.

To ensure reliable collision-free communication within hard deadlines, the wireless communications associated with these applications are generally based on Time Division Multiple Access (TDMA) strategy [15]. In order to satisfy the hard deadlines of the real-time flows, a centralized network manager reserves dedicated time-slots and frequencies, and computes the schedule for these communications. The wireless devices associated with these communications are informed of their allocated time-slots and frequencies in a schedule. Each device wakes up in their allocated time-slots to transmit/receive messages to/from the neighboring devices. The same schedule repeats over every hyperperiod, *i.e.*, the least common multiple of the periods of the real-time flows. Due to repetition of the same schedule, the time-slots in which a specific device communicates with its neighboring devices over a hyperperiod become predictable in nature. This predictable radio resource allocation ensures deadline satisfaction of all the real-time flows in the system. However, this predictable behavior also serves as a key vulnerability in the system that can be exploited by an attacker.

A *stealthy* attacker can take full advantage of the predictable behavior of the schedule to infer the time-slots in which any two specific devices communicate. The repetitive patterns in the schedules greatly help the attacker to eavesdrop the communication between any two specific devices over a certain period of time, and infer the time-slots in the schedule. With the inferred time-slots, the attacker can further launch various strategic attack steps. For example, the attacker can selectively jam the communications between the two targeted devices in specific time-slots of the schedule. Such selective jamming attacks are very hard to detect as the attacker jams only in specific time-slots of the schedule [16]. However, selectively jamming a critical flow in a real-time system over every hyperperiod may result in safety breaches, which in turn, can cause catastrophic consequences. In a recent work, Cheng *et al.* have illustrated selective jamming attack in a WirelessHART network [17]. In ICSs, a large number of control applications are safety critical in nature. Selective jamming attack on a critical sensor or a critical actuator over every hyperperiod in such applications may result in complete disruption of the flow associated with the jammed sensor or the jammed actuator device. This, in turn, can make the system unstable or can even result in catastrophic damages, thereby, violating the safety of the system.

Selective jamming attack is one of the most common type of timing attacks. However, the predictable behavior in the time-slots of the schedule makes the real-time networks vulnerable to a large class of other timing attacks as well.

1.3 Timing Attacks in Real-Time Wireless Networks

Although the wireless protocols used in real-time wireless networks guarantee reliability, the network devices are generally deployed in remote areas and in open environment, where they are exposed to attacks due to subtle security loopholes in the system. A timing attack is one such exploit that takes advantage of some flaws in the system and compromises the security of the system, resulting in catastrophic damages.

A secure real-time system aims to satisfy the three fundamental criteria of security, more specifically known as the *CIA* triads of security [18] —

1. *Confidentiality* : “Only authorized users and processes should be able to access or modify data”.

2. *Integrity* : “Data should be maintained in a correct state and nobody should be able to improperly modify it, either accidentally or maliciously”.
3. *Availability* : “Authorized users should be able to access data whenever they need to do so”.

An attacker tries to breach at-least one of these three triads of security to launch a successful attack. A timing attack is a side-channel attack that enables the attacker to spot vulnerabilities in a system to extract confidential information by exploiting the timing behavior of the system. Thus, timing attacks mainly aim to breach the “confidentiality” or the “privacy” of the system by analyzing and predicting the timing behavior of the system. The timing behavior of the system can be exploited in various ways, such as by statistically analyzing the power consumption measurements to extract the timing information of an encryption operation [19], or by analyzing the task execution patterns on a processor to predict the time of execution of a specific task [20]. Timing attacks are also possible on cache memories [21, 22] where an attacker analyzes the data flow execution pattern of a specific encryption operation to launch timing attack. Timing attacks can even occur in wireless networks where an attacker eavesdrops the communication to predict the packet transmission patterns in the network [23, 24].

1.4 Thesis Motivation

To launch timing attacks in real-time networks, an attacker uses the schedule as a side-channel and exploits the predictable behavior of the schedule to analyze the time-slots specific to any two targeted devices. Different techniques to mitigate timing attacks exist in the literature depending on the platform where the attack is launched. For example, countermeasures against cache timing attacks involve rescheduling the execution patterns of the instruction [25]. *The main objective of this thesis is to propose countermeasures against timing attacks in real-time wireless networks.* Since, the attacker in real-time networks exploits the predictable behavior of the time-slots in the schedule to launch timing attacks, our proposed countermeasures aim at reducing the predictability in the schedules by obfuscating the communication pattern in the schedules. To achieve this, our countermeasures randomize the time-slots in the schedules over every hyperperiod so that the schedule changes before the attacker can analyze and infer the time-slots

in the schedules. That is, our proposed countermeasures target to obfuscate the time-slots in the schedule at a pace that is faster when compared to the learning pace of the attacker.

Schedule randomization in real-time wireless networks is not straightforward. A few works exist in the literature that perform schedule randomization for processors with single cores [20, 19]. However, these solutions are not applicable to our system. Randomizing the schedule of a collection of n real-time flows over a set of m frequencies in a wireless network is in fact similar to randomizing the schedule of a collection of real-time tasks on a processor with multiple cores. This problem has been shown to be NP-hard [26], where the n real-time flows are similar to n real-time tasks and the m frequencies map to the m processing cores. Tiloca *et al.* proposed schedule randomization as a countermeasure against timing attacks in wireless sensor networks [27, 28]. These works provide countermeasures against timing attacks by permuting the slot utilization pattern at the node level to randomize the schedule in wireless sensor networks. However, these flows are not associated with deadlines, and hence randomization of slot utilization pattern at the node level is feasible. This is not the case for our problem because flows have hard deadlines, and therefore such non real-time solutions are also not applicable.

The flows in real-time wireless networks are associated with hard deadlines. Hence, schedule randomization must be done without violating the hard deadlines of the flows. Apart from the hard deadlines, a feasible schedule has to satisfy three other conditions; (1) there should be no conflicts in the transmissions of the real-time flows, implying two real-time flows cannot have the same sender device or the same receiver device in a time-slot, (2) there should not be any collisions in the transmissions of a schedule implying that if two flows are within their region of interference, then these two flows cannot use the same frequency in the same time-slot, (3) if a flow has multiple hops between the source device, from where it is transmitted, and the destination device, where it needs to be transmitted, then these hop sequences have to be preserved in the schedules. These three conditions, together with the hard deadline requirement, add constraints on the schedule randomization process and hence make it a hard problem to solve.

Apart from these constraints, there are other challenges as well in schedule randomization. The schedule in a real-time wireless network is generated by a centralized network manager and the scheduling information is passed on to the wireless devices. To reduce predictability, a different schedule needs to be executed over every hyperperiod.

However, the wireless devices deployed in real-time networks are generally low power devices with very limited memory. As a result, these devices can only store scheduling information corresponding to a very small number of schedules. Hence, storage and distribution of schedules become a challenge in real-time wireless networks. Although frequent broadcast of scheduling information from the centralized network manager serves as a solution to this problem, it increases the energy consumption of the system. One solution to this problem is to decentralize the schedule randomization process among the wireless devices. This is also a challenging problem, given the limited computational resources in such devices and the scale of real-time wireless networks.

There are additional challenges in schedule randomization in case of a 5G network. A 5G network supports different types of communication service categories, in addition to URLLC. Unlike other real-time wireless networks, the traffic in a 5G network is also very dynamic. Since, multiple types of applications from different communication service categories share the same 5G network, only a fraction of the resources are available to URLLC service. Hence, guaranteeing feasibility with limited available resources while still randomizing the schedule, becomes a major challenge in case of a 5G network.

1.5 Contributions

The main contributions of this thesis is to propose schedule randomization techniques as a countermeasure against timing attacks in real-time wireless networks. The schedule randomization techniques randomize the time-slots over each hyperperiod schedule to reduce the predictability in the time-slots of the schedules. Depending on the characteristic features of the two real-time wireless protocols considered in this thesis, and the type of traffic supported by each of these two protocols, different schedule randomization techniques are proposed for each of them.

Thus, the main contributions of this thesis are summarized below.

- Chapter 2 introduces a generic network model used in real-time wireless networks. It presents a detailed description of the two wireless network protocols, (1) the WirelessHART network protocol and (2) the 5G network protocol, and extends the generic network model for each of these protocols. It also introduces two metrics to evaluate the performance of our proposed countermeasures.

- Chapter 3 presents literature review related to the focus of this thesis: timing attacks in wireless networks as well as on cache, GPU and other microarchitectural platforms. It also presents some related research on different metrics used for schedule randomization.
- Chapter 4 presents the two threat models: threat model for timing attack in WirelessHART networks and threat model for timing attack in 5G, and describes the two models with illustrative examples. It then presents the consequences of these attacks in ICSs. It also presents the advantages of selective jamming attacks compared to other types of jamming attacks in wireless networks.
- Chapter 5 presents a countermeasure against timing attacks, specifically, selective jamming attacks, in a multi-channel WirelessHART network. Although a few works on schedule randomization based countermeasures against timing attacks in wireless sensor networks exist in the literature [27, 28], these countermeasures do not consider hard deadlines for the flows. In this chapter, we present a moving target defense technique that randomizes the time-slots in the schedules over every hyperperiod without violating the hard deadlines of the real-time flows, while still satisfying all the feasibility constraints of a real-time schedule. We show that our solution is optimal for harmonic flows with a single frequency (periods of the flows are multiples of each other). We use *Kullback-Leibler divergence* or *K-L divergence* [29] to measure the randomness in the time-slots of the generated schedules with reference to a truly random solution. We conduct security analysis of our proposed countermeasure by measuring the Prediction Probability of time-slots in the generated schedules based on observations over the previous hyperperiod schedules. We evaluate our results on simulated networks of 100 nodes in the Contiki Cooja simulator [30] as well as on a real testbed of 74 TelosB motes [31]. This work has been published in [32, 33].
- Chapter 6 addresses the drawbacks of the schedule randomization strategy proposed in Chapter 5. The schedule randomization strategy proposed in Chapter 5 is a centralized strategy. The random schedules are distributed periodically to all the devices in the network which increases the energy consumption of the system. Moreover, any addition or removal of devices results in recomputation of all the schedules and also involves their redistribution. Also, the schedule randomization strategy proposed in Chapter 5 does not consider the interaction between the network flows and the closed-loop control tasks in the system. To overcome these

drawbacks, in this chapter, we propose a distributed dynamic online schedule randomization strategy for a single-channel WirelessHART network. This strategy randomizes the time-slots in the schedules at each device in a distributed manner without violating the deadlines of the real-time flows, while still maintaining the stability of the closed-loop control tasks. Additionally, we integrate a period adaptation strategy that can change the period of transmissions of the real-time flows in the network at runtime depending on the control performance. This increases the extent of randomization in the generated schedules as well as reduces the energy consumption of the system, while still maintaining the same control performance. We use K-L divergence to measure the extent of randomization in the time-slots of a schedule compared to a truly random solution. We also use Prediction Probability of time-slots to perform security analysis of our proposed countermeasure. We also compare the distributed online schedule randomization strategy with the centralized strategy presented in Chapter 5. We conduct our experiments in the Gisoo simulator [34] and measure the transmission power using the Tossim [35] simulator.

- Chapter 7 presents a novel online schedule randomization strategy for periodic URLLC flows in a 5G network. The solutions proposed in Chapter 5 and Chapter 6 for WirelessHART networks are not applicable for a dynamic system like 5G. The network in a WirelessHART is static where the schedules are determined at network initialization for a fixed number of flows. However, in a 5G network, the traffic is highly dynamic in nature and the number of flows vary at runtime. Besides, only a fraction of the available resources are allocated to periodic URLLC flows. Hence, schedule randomization needs to be performed online, guaranteeing the feasibility of the flows with the available resources. Hence, the strategy proposed in this chapter randomizes the transmission slots and the frequencies in the schedules for periodic URLLC flows, while still guaranteeing the deadlines of the flows and satisfying schedule feasibility at runtime. We prove that the randomization strategy always generates feasible schedules. We also present comparisons based on K-L divergence and Prediction Probability of time-slots with a baseline strategy that uses an earliest deadline first based heuristic.
- Chapter 8 summarizes this thesis and discusses some future directions for research.

Chapter 2

Background

In this chapter, we introduce a generic network model for real-time wireless networks. Then, we describe two well-known wireless network protocols, (a) the WirelessHART network and (b) the fifth generation (5G) cellular network, and extend the generic network model for each of these protocols. We also present two metrics, the K-L divergence and the Prediction Probability of time-slots, that we will use in this thesis to evaluate the performance of the proposed countermeasure techniques.

2.1 Network Model

We consider a directed network graph $G = (V, E)$, where V denotes the set of network devices and E denotes the set of links or edges between the devices. Our network consists of m channels or frequencies and n real-time flows $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$. Each flow $F_i \in \mathcal{F}$ periodically generates a packet at the source node $src_i \in V$ with period p_i . The packet needs to reach the destination node $des_i \in V \setminus \{src_i\}$ with relative deadline δ_i , *i.e.*, within deadline δ_i from the time when the packet is ready to be released at the source node, src_i . Most control applications in real-time cyber-physical systems such as the ICS are associated with implicit deadline-based tasks, *i.e.*, the relative deadline δ_i of the task is always equal to the period p_i ($\delta_i = p_i$) [36, 37]. Hence, we assume that the real-time flows in our network that are associated with the control applications, also have implicit deadlines. Each flow $F_i \in \mathcal{F}$ is associated with a fixed route $route_i$ which denotes a list of intermediate nodes (hops) for the flow between source src_i and destination des_i .

Definition 2.1 (Release time). The release time of the j^{th} instance of flow F_i ($j \geq 1$) is the time at which the j^{th} instance of F_i is ready to be released at the source node src_i . The release time (r_{ij}) is defined as

$$r_{ij} = (j - 1) \cdot p_i. \quad (2.1)$$

Definition 2.2 (Number of hops). The number of hops ($F_i.n_hops$) in the route $route_i$ of a flow F_i is the number of intermediate nodes between the source (src_i) and the destination (des_i) in $route_i$.

Definition 2.3 (Scheduling Window). Scheduling Window (W_{ij}) of the j^{th} instance of the i^{th} flow F_i is the time-slots (slots in short) between its release time r_{ij} and absolute deadline $r_{ij} + p_i$. Slot $s \in W_{ij}$, if $r_{ij} \leq s < r_{ij} + p_i$.

Depending on the specifications of a WirelessHART network and a 5G network, this network model is extended further in Section 2.2.2 and Section 2.3.4 of this chapter, respectively.

2.2 WirelessHART Network Descriptions

In this section, we first describe the key features of a WirelessHART network followed by the WirelessHART network model that we use in Chapter 4, Chapter 5 and Chapter 6 of this thesis.

2.2.1 Key Features of a WirelessHART Network

The WirelessHART protocol is the first open wireless communication standard for measurement and control in industrial control systems (ICSs) [26]. It is compliant with IEEE 802.15.4 standards. A WirelessHART network uses wireless mesh networking among the network devices and provides reliable digital communications to satisfy stringent deadline requirements of real-time applications. A WirelessHART network consists of a gateway, multiple field devices, access points (APs), a security manager and a centralized network manager. The network manager is installed on or connected to the gateway and is responsible for managing the devices, creating the routes, optimizing the network and generating the schedule for transmissions among the devices. The security manager

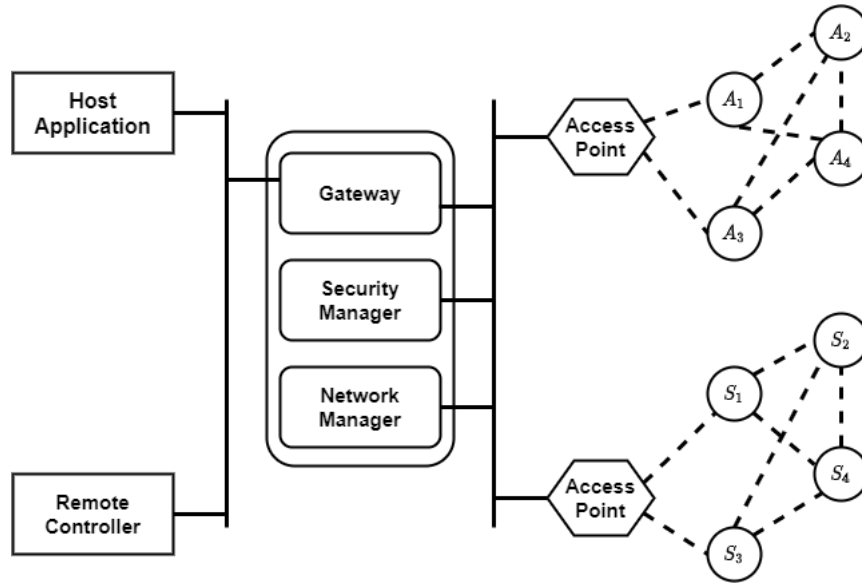


FIGURE 2.1: Overview of a WirelessHART network

is responsible for generation, storage and management of the network keys that are used for encrypting/decrypting the packets flowing in the network [38]. The remote controller either resides on or is connected to the gateway using wired connection. The field devices are mainly wireless sensors, actuators and APs. Multiple APs are connected to the gateway using wired connections to provide redundant paths between the gateway and the field devices. The host applications are also connected to the gateway using wired connections. Thus, the gateway serves as the main communication link between the host applications and the field devices. Figure. 2.1 shows an overview of the different components of a WirelessHART network. The sensors and the actuators are denoted by S_i and A_i respectively in the figure.

The WirelessHART is the most suitable wireless protocol that serves as the medium of communication in large-scale wireless sensor actuator networks. The most common and mature use case of WirelessHART is the ICSs where it provides guaranteed reliable service within hard deadlines and ensures stability of the closed-loop controls. The key features of a WirelessHART network that makes it suitable for ICSs are as follows:

1. **Time Division Multiple Access (TDMA) based communication:** Time is globally synchronized and divided into *slots* of $10ms$ duration. During this $10ms$ duration, a network device sends a packet and receives its corresponding acknowledgement. The TDMA based communications in WirelessHART networks enable reliable communication with predictable latencies.

2. **Size of a network:** A typical WirelessHART network can support up to 80 devices under the control of a single gateway [39, 40]. This facilitates management of the system from a centralized network manager. Large-scale wireless sensor actuator networks are then formed through communication between the gateways. To guarantee that the networking delays do not harm the control performance (stability) of the control applications, the maximum number of hops between the gateway and the source or destination device should not exceed 4 hops [39, 41].
3. **Channel diversity:** WirelessHART supports operation in the 2.4GHz ISM band using IEEE 802.15.4 standard radios [9]. The entire frequency band of WirelessHART is divided into 16 channels as defined in IEEE 802.15.4 physical layer [42]. To avoid interference from neighboring wireless systems, WirelessHART adopts channel hopping *i.e.*, it changes the operating frequency of the communication channel in every slot. The physical channel assigned to a link in a particular slot using the channel hopping mechanism is given by

$$Ch_p = (ASN + Ch_l) \bmod m \quad (2.2)$$

where ASN represents the Absolute Slot Number of the transmission. The ASN increases by one at every slot. Ch_p and Ch_l are the physical channel and the logical channel assigned to a node. m denotes the number of channels in the network.

4. **Channel Blocklisting:** In a WirelessHART network, a channel is *blocklisted* from the network and not used for communication if that channel suffers from persistent external interference [43].
5. **Route diversity:** To mitigate physical obstacles, broken links and interference, WirelessHART allows transmission of a packet via multiple paths over multiple channels.
6. **Spatial re-use of channels:** A WirelessHART network schedules transmissions based on multi-channel TDMA protocol. Generally, each node in the network is surrounded by a circular region which is defined as its interference region. However, due to variability in the interference patterns between the nodes, the interference in WirelessHART network is very hard to detect. Hence, in large networks deployed over a wide area, two distant nodes can use the same channel in parallel if they do not interfere with each other, *i.e.*, a WirelessHART network allows *spatial re-use of channels* [44].

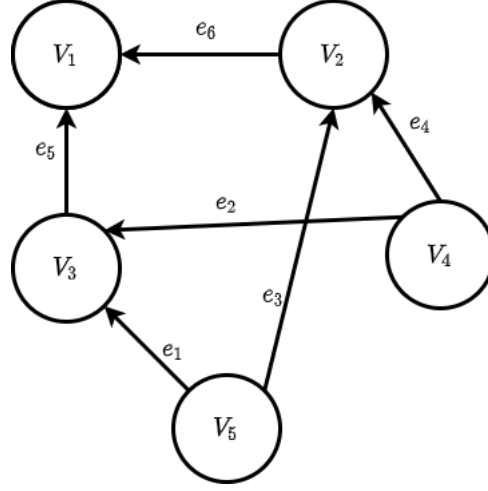


FIGURE 2.2: A network graph with five vertices and six edges.

Based on the above features, a WirelessHART network can be modeled as a network graph $G = (V, E)$, where V is the set of nodes representing the field devices and the APs; E is the set of edges or communication links between the devices. A node $v \in V$, can be either a sensor, an actuator or an AP. An edge $e = u \rightarrow v$ is part of G if and only if device u can reliably communicate with device v . In a transmission along an edge $u \rightarrow v$, the transmitting device u is the *sender* and the receiving device v is the *receiver* of the transmission. In a slot, a device cannot transmit or receive simultaneously. In addition to that, a receiving device can receive from exactly one sender device.

Definition 2.4. Two transmissions along edges $u \rightarrow v$ and $w \rightarrow x$, where $u, v, w, x \in V$, are said to be **conflicting transmissions** if both of them have the same sender or the same receiver, i.e., if $(u = w) \vee (v = w) \vee (u = x) \vee (v = x)$. For each edge, $u \rightarrow v \in E$, there exists a set of conflicting transmissions in G . To keep track of the conflicting transmissions in G , an adjacency list known as the **Conflict List** is stored. Each index i in the list corresponds to an edge in E and the list corresponding to index i stores the list of edges which generate conflicting transmissions with i .

We present an example to explain the *conflicting transmission* and *Conflict List*.

Example 2.1. Consider the network graph in Figure. 2.2. The transmissions along $V_5 \rightarrow V_3$ and $V_4 \rightarrow V_3$ are conflicting transmissions as they have the same destination node V_3 . Similarly, the transmissions along $V_5 \rightarrow V_3$ and $V_5 \rightarrow V_2$ are conflicting transmissions as they have the same source V_5 . Hence, the entry corresponding to edge e_1 in the Conflict List is $\{e_2, e_3\}$. Similarly, the entry corresponding to edge e_5 in the Conflict List is $\{e_1, e_2, e_6\}$.

A WirelessHART network graph $G = (V, E)$ can be sub-divided into an uplink graph (U) which connects the sensors to the APs and a downlink graph (D) which connects the APs to the actuators. An end-to-end communication between a sensor and an actuator occurs in three phases:

1. **Sensing Phase:** during which sensor measurement data is sent to the remote controller residing on the gateway.
2. **Control Phase:** during which the control algorithms run on the remote controller to generate appropriate control signals.
3. **Actuation Phase:** during which the generated control signals are sent to the actuators to update the state of the physical plant.

2.2.2 WirelessHART Network Model

Based on the key features of a WirelessHART network, the network model described in Section 2.1 of this chapter can be extended to represent a WirelessHART network model with some additional definitions and notations.

A WirelessHART network graph G is sub-divided into an uplink graph U and a downlink graph D , such that $G = U \cup D$. The source and the destination nodes are the sensors and the APs respectively, if a flow F_i transmits sensor data from the sensors to the APs via the uplink graph (U). Similarly, the source and the destination nodes are the APs and the actuators respectively, if F_i transmits control signals from the APs to the actuators via the downlink graph (D). Multiple flows can share an edge in the network. Hence, the flows in the network can have overlapping routes.

A *feasible schedule* in a WirelessHART network is defined as:

Definition 2.5. Given an graph $G = (V, E)$ with m channels and a set of flows \mathcal{F} defined over G , a **feasible schedule** \mathcal{S} is a sequence of transmissions over the slots in \mathcal{S} . Each transmission is a mapping of a flow to a channel in a slot satisfying the following conditions:

1. **No transmission conflict** - Two transmissions along $u \rightarrow v$ and $w \rightarrow x$ can be scheduled in the same slot s , if $u \rightarrow v$ and $w \rightarrow x$ are non-conflicting transmissions.

Schedule	Channel	Slots					
		1	2	3	4	5	6
\mathcal{S}	Ch_1	$\mathbf{V}_5 \rightarrow \mathbf{V}_3$			$\mathbf{V}_2 \rightarrow \mathbf{V}_1$	$\mathbf{V}_3 \rightarrow \mathbf{V}_1$	
	Ch_2	$\mathbf{V}_2 \rightarrow \mathbf{V}_1$	$\mathbf{V}_3 \rightarrow \mathbf{V}_1$		$\mathbf{V}_5 \rightarrow \mathbf{V}_3$		

TABLE 2.1: A schedule \mathcal{S} over six slots on the network graph in Figure. 2.2 with two channels and two flows, F_1 (marked in red) and F_2 (marked in blue).

2. **No collision** - If $u \rightarrow v$ uses channel y and $w \rightarrow x$ uses channel z in the same slot s , where $u, v, w, x \in V$, and u, v, w, x lie within the same interference region [45] of G , then $y \neq z, \forall y, z \in [1, m]$.
3. **No deadline violation** - If a flow F_i has n_hops number of hops, ($F_i \in \mathcal{F}$), then for each instance F_{ij} of F_i , all the hops of F_i are to be scheduled within its scheduling window W_{ij} .
4. **Flow sequence preservation** - If a flow F_i has n_hops number of hops ($F_i \in \mathcal{F}$), then the k^{th} hop ($1 < k \leq n_hops$) cannot be scheduled until all the previous $k - 1$ hops are scheduled.

The length of a feasible schedule, hp , is given by the number of slots in the feasible schedule \mathcal{S} and is equal to the length of the hyperperiod, *i.e.*, the lowest common multiple of the periods of the flows in \mathcal{F} .

We explain a feasible schedule in a WirelessHART network with an example.

Example 2.2. Table 2.1 shows a feasible schedule \mathcal{S} over six slots on the WirelessHART network graph in Figure. 2.2 with two channels and two flows F_1 and F_2 . The flows, F_1 and F_2 , are marked in red and blue respectively. Flows F_1 and F_2 have sources at V_5 and V_2 respectively and a common destination at V_1 . Their periods or the relative deadlines are given by 3 slots (30ms) and 6 slots (60ms) respectively. Transmissions $V_5 \rightarrow V_3$ and $V_2 \rightarrow V_1$ are non-conflicting, hence, they can be scheduled in the same slot along different channels.

We assume that the network manager *blocklists* those channels from the network for which the probability of successful transmission is less than a certain threshold [43]. Hence, the packet loss in the m available channels in the network is negligibly small. At the time of network initialization, the network manager decides the *schedule* depending on the number of available channels, the topology of the network and available routes for

each flow [46, 42]. Given a network graph G with m channels and a set of n flows \mathcal{F} , the network manager runs a scheduling algorithm \mathbb{A} that generates a schedule \mathcal{S} satisfying all the conditions of Definition 2.5. We assume that there is no priority constraint among the flows in \mathcal{F} . The network manager then informs all the network devices about the allocated slots in which they can transmit (receive) packets from specific neighbors. The network devices become active only in those slots at which they are required to transmit (receive) packets.

2.3 The 5G Cellular Network Descriptions

In this section, we describe the different communication services and the frame structure supported by the 5G networks. Then, we present the scheduling policy to be followed by the 5G networks to support time-critical applications in future wireless networks. Furthermore, we extend the network model introduced in Section 2.1 of this chapter to support time-critical applications in 5G networks. We use this network model in Chapter 4 and Chapter 7 of this thesis.

2.3.1 Communication Services in 5G Networks

The 5G New radio (NR) supports three broad communication service categories:

- *Enhanced Mobile Broadband (eMBB)*: supports very high bandwidth, high data rate communication across a wide coverage area. eMBB is suitable for a large number of use cases such as UltraHD or 360-degree streaming videos, the emerging Augmented Reality/Virtual Reality media applications, etc.
- *Massive Machine-type Communication (mMTC)*: supports extremely high density online connectivity. mMTC is suitable for Internet of Things applications with densely connected devices.
- *Ultra-Reliable Low-Latency Communication (URLLC)*: supports communication that require very high reliability (99.99%) and very low latency (1ms) requirements [12]. URLLC is designed for time-critical applications such as autonomous driving, industrial automation, smart energy, remote surgery, etc.

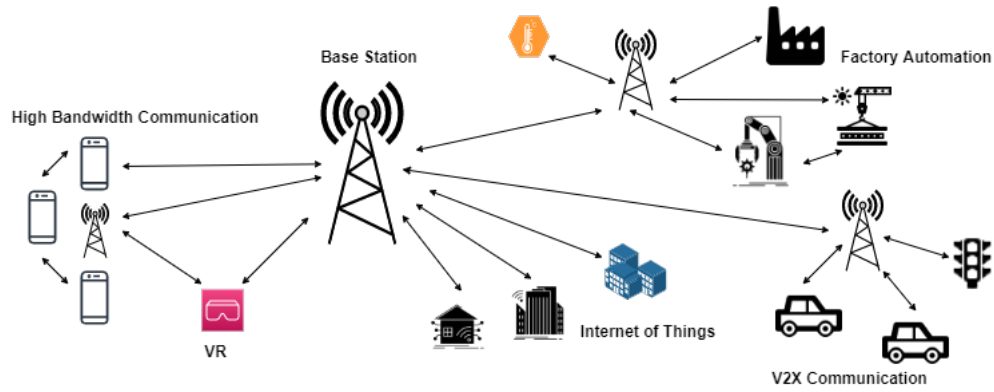


FIGURE 2.3: Applications under different services in 5G

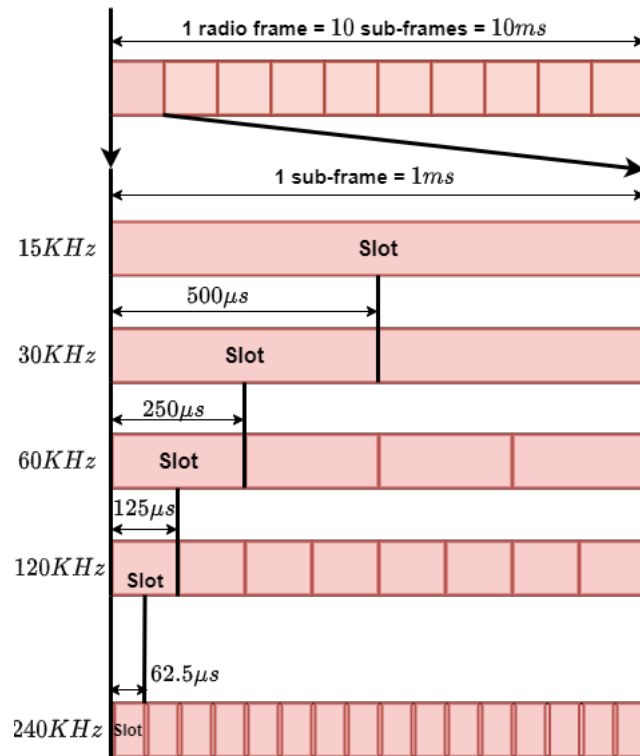


FIGURE 2.4: Different slot durations in 5G corresponding to each sub-carrier spacing

Figure. 2.3 shows different applications belonging to different service categories in 5G. Among the three broad communication service categories, the URLLC is mostly applicable for real-time applications with strict timeliness requirements.

2.3.2 5G New Radio (NR) Frame Structure

5G NR access technology is based on flexible orthogonal frequency division multiplexing (OFDM) that enables 5G to operate in a wide range of frequency bands to support

different use cases across multiple spectrum [47]. The uplink/downlink transmissions in 5G are organized into *frames*, each of $10ms$ duration. Each frame is divided into 10 *sub-frames* of $1ms$ duration each, which is further sub-divided into *transmission slots* or *slots*. We denote the number of slots in each sub-frame by N_{sf} . 5G NR frame structure allows multiple sub-carrier spacing (SCS) to support different latencies specific to each URLLC application [48]. Figure. 2.4 shows a $10ms$ frame in 5G and the different slot durations corresponding to each SCS. With increase in the SCS, the number of slots per sub-frame increases with a corresponding decrease in the duration of a slot. The SCS in 5G varies from $15KHz$ upto $240KHz$, the corresponding slot duration varies from $1ms$ down to $62.5\mu s$.

2.3.3 Scheduling URLLC flows in 5G Networks

In a typical 5G network, a base station (BS) serves a set of user equipment (UEs) under its coverage area. Existing Long Term Evolution (LTE) systems use a grant-based scheduling policy in which each UE sends an access request to the BS to transmit data [49]. The BS grants the request and issues an access grant through a four step random access procedure. The communication setup phase takes at-least $11.5ms$, considering a Transmission Time Interval of $1ms$ and a scheduling request periodicity of $10ms$ [50]. Hence, URLLC services cannot be served with existing LTE systems. To satisfy the low latency requirements of URLLC traffic, grant-free access, more specifically known as configured grant scheduling, has been proposed by the Third Generation Partnership Project (3GPP) in which a UE can transmit a packet without any request for grant access [51].

A broad category of URLLC applications in real-time cyber-physical systems are periodic in nature with hard deadlines, *i.e.*, the packet transmissions must be done periodically within pre-specified time windows. We call such network flows *periodic real-time URLLC flows*. These type of applications are mainly found in use cases such as autonomous vehicles or ICSs. For example, a vehicular safety application in Vehicle to Everything (V2X) broadcasts basic safety messages every $100ms$ to avoid collisions [13]. Such messages are needed to be highly reliable and to be transmitted within strict deadlines. To guarantee service for such flows, the BS allocates dedicated resources for the corresponding UEs in advance. Preallocation of dedicated radio resources is known as Semi-Persistent-Scheduling (SPS) [51]. The BS then computes the resource schedule to

serve such UEs and informs them about the generated schedule. This ensures that the flow deadlines are guaranteed and the same schedule is repeated over time.

2.3.4 5G Network Model

We extend the network model described in Section 2.1 to account for additional features of a 5G network with some new notations.

We consider a set of *single-antenna UEs* under the coverage area of one BS B . The UEs in our system are either static devices (e.g., sensors and actuators) or mobile within a certain bounded range (e.g., robots on a manufacturing shopfloor). Hence, we assume that a UE is served under a single BS for the entire duration of time for which it is active. However, a UE can join (leave) the network at any point in time due to addition (removal) of devices to (from) the network. In most real-time cyber-physical systems, such as the ICSs, the URLLC flows of different applications operate at different SCS depending on their latency requirements. For example, a discrete automation application operates at a latency of $10ms$, whereas, a process monitoring automation operates at a latency of $60ms$ [12]. We further assume that all UEs operate in Frequency Division Duplex (FDD) mode, *i.e.*, the frequency bands involved are partitioned into two separate frequency bands, one for transmitting data to the BS (uplink communication) and the other for receiving data from the BS (downlink communication). Hence, each UE is associated with a set of operating frequencies. We *strictly partition* the UEs into groups such that all UEs belonging to the same group operate with the same SCS and use the same set of frequencies. *Therefore, each such UE group has a fixed value for N_{sf} (refer to Section 2.3.2 of this chapter) and operates without interference from the other groups. Hence, each such UE group can be scheduled independently.* Thus, from here onwards, we focus on such a single UE group. We also assume that each UE in the group is within the radio range of B , *i.e.*, at one hop distance from B and requires periodic real-time URLLC service with hard deadline, and is associated with either an uplink or a downlink flow of data.

Based on the above assumptions we consider a single UE group with n UEs, $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$. The flows and the operating frequencies associated with these UEs can be represented by the flow set \mathcal{F} and the m frequencies introduced in the network model in Section 2.1 of this chapter. All of these flows use any frequency from these m set of frequencies and operate under the same SCS. We assume that there is no spatial

reuse of frequencies, *i.e.*, no two flows in \mathcal{F} can be scheduled in the same slot using the same frequency. The source and the destination devices are UE U_i (BS B) and BS B (UE U_i) depending on whether the flow is an uplink (downlink) transmission. In addition to the flow parameters defined in the network model in Section 2.1 of this chapter, we have an additional flow parameter in 5G, *i.e.* the execution time e_i or the number of slots required to transmit (receive) data packets in each period. 5G URLLC is expected to reach a maximum transmission time of $1ms$ for a packet size of 32 bytes with a reliability of 99.99% [52]. Most periodic applications in real-time cyber-physical systems such as the ICSs are associated with short periods and small amounts of data (approximately 20 bytes) [53]. Usually, the data size is sufficiently small to get transmitted in one slot. Further, to satisfy the low latency requirements, the UEs do not wait for acknowledgement [54]. Instead, one re-transmission slot for each data is preallocated in the schedule [55, 56, 57]. This assumption is reasonable because there is extremely low chance of collision due to preallocation of resources and short flow periods [54]. Hence, we assume that $e_i = 2$ for all the flows $F_i \in \mathcal{F}$ (one slot for the original transmission and another for the re-transmission).

We also assume that this UE group only uses α fraction of the sub-frames per frame in each of the m frequencies for periodic real-time URLLC flows where $\alpha \geq 0.1$ and $\alpha \in [0.1, 1]$. This is to ensure that at-least one sub-frame per frame is allocated for periodic URLLC flows. The remaining $(1 - \alpha)$ fraction of the sub-frames are used for traffic that do not have hard deadline requirements (sporadic URLLC, eMBB, etc.). *Since these allocated sub-frames may not be uniformly distributed within each frame, we further assume that the flow periods are aligned with the frame boundaries.* That is, for each flow F_i , p_i is a multiple of frame duration $10ms$. This gives flexibility in the distribution of slots to different traffic within a frame, without affecting the hard deadlines of URLLC flows. Generally, the flow periods in most motion control or factory automation applications vary between few hundreds of milliseconds to a few seconds [58, 52, 59]. *Hence, we consider the minimum period of any flow in our system to be $10ms$.*

We define a *feasible schedule* for a set of periodic URLLC flows in a 5G network as follows.

Definition 2.6. Given a set of n periodic URLLC flows, \mathcal{F} , transmitting over a set of m frequencies with α fraction of the sub-frames per frame in each frequency, a **feasible schedule** \mathcal{S} is a sequence of transmissions in the allocated slots of each frame along m frequencies, satisfying the following conditions.

Schedule	Freq.	Slots											
\mathcal{S}		2	5	14	18	27	29	31	32	48	49	54	58
	fr_1	F_1		F_2		F_1				F_1		F_2	
		1	3	12	15	25	28	31	32	43	49	52	55
	fr_2		F_1		F_2		F_1				F_1		F_2

TABLE 2.2: A schedule \mathcal{S} over 6 frames and 2 frequencies with two uplink flows F_1 (red), F_2 (blue); The slot duration in the schedule is $1ms$.

1. A transmission of a flow instance of a UE can only be scheduled along a single frequency in a specific slot. It may use different frequencies in different slots.
2. Since, the UEs are equipped with single antenna, two UEs cannot transmit data in the same slot using the same frequency.
3. The two slots required to transmit (receive) data for the j^{th} instance of flow F_i must be scheduled within its scheduling window W_{ij} .

The duration of a schedule \mathcal{S} is equal to the hyperperiod which is the lowest common multiple of the flow periods, and is denoted by hp .

We explain a feasible schedule in a 5G network with an example.

Example 2.3. Table 2.2 shows a feasible schedule over six frames and two frequencies with two flows, F_1 (marked in red) and F_2 (marked in blue). The slot duration $N_{sf} = 1$ and $\alpha = 0.2$. The source devices of the flows are UE U_1 and UE U_2 respectively. The destination device is the BS B . The periods of the flows are $20ms$ and $30ms$ respectively. Since $\alpha = 0.2$ and $N_{sf} = 1$, therefore, only two slots are allocated per frame along each frequency for periodic URLLC flows in the schedule \mathcal{S} .

2.4 Metrics to Measure Schedule Randomization

To mitigate timing attacks in real-time networks, our proposed countermeasures randomize the communication slots and the operating frequencies in the schedules over every hyperperiod such that the hard deadlines of the real-time flows as well as all the feasibility constraints of a schedule (Definition 2.5 in Section 2.2.2 for a WirelessHART network and Definition 2.6 in Section 2.3.4 for a 5G network) are not violated. In this

section, we introduce two metrics that we use in Chapter 5, Chapter 6 and Chapter 7 to evaluate the performance of our proposed countermeasures.

Given two probability distributions p and q , *Kullback-Leibler divergence* or *K-L divergence* [29] is a measure in statistics that quantifies in bits how close a probability distribution $p = \{p_i\}$ is to a candidate probability distribution $q = \{q_i\}$ [60]. From information theoretic point of view, *K-L divergence* measures the amount of information we lose if we choose probability distribution $p = \{p_i\}$ instead of $q = \{q_i\}$ [61]. It is represented as

$$\mathcal{KL}(p||q) = \sum \Pr(p_i) \log \frac{\Pr(p_i)}{\Pr(q_i)} \quad (2.3)$$

We use this metric to compare the extent of randomization in the schedules generated by our proposed countermeasure with reference to a truly random algorithm that can generate all possible feasible schedules with equal probability. We re-define this metric for each of our countermeasures depending on the network model in Chapter 5, Chapter 6 and Chapter 7. Lower the value of the K-L divergence, lesser is the divergence of the solution space of our proposed countermeasure with reference to a truly random algorithm, hence better is the solution.

We perform *security analysis* of the generated schedules by using *Prediction Probability (PP)* of slots in the schedules. Given the observation over $t - 1$ hyperperiod schedules, PP of slots in the t^{th} hyperperiod schedule is the probability of correctly predicting the transmission in the slots of the t^{th} hyperperiod schedule based on the observations in the previous $t - 1$ hyperperiod schedules.

We use this metric to measure the success probability of the attacker in correctly predicting the slots in the schedules depending on the previous hyperperiod schedules. We re-define this metric for each of our countermeasures depending on the network model in Chapter 5, Chapter 6 and Chapter 7. Lower the PP of slots in the schedules, less predictable are the slots to the attacker and therefore, better is the security provided by the generated schedules.

Chapter 3

Related Works

The idea of timing attack was first introduced by *Kocher* in 1996 [62]. Since then, several works have been done on timing attacks on caches [63] where an attacker exploits the predictable behavior of the instruction execution pattern when an encryption operation is being executed. Similarly, timing attacks are also possible on other platforms such as GPU [64], computing hardware [65] or in the communication network [17]. In this chapter, we present past research studies on timing attacks on different platforms, with a particular focus on wireless networks. We also present some past research studies on the K-L divergence metric we use in this thesis.

3.1 Timing Attacks in Wireless Networks

Timing attacks in wireless networks and their countermeasures are well studied in the literature. Timing attacks can be passive such as traffic analysis attacks, or active such as selective jamming attacks [16]. Among the different types of timing attacks, selective jamming attack is considered as one of the most popular type of timing attacks in wireless networks. However, selective jamming attack can be launched only after analyzing the traffic for a sufficiently long period of time.

3.1.1 Traffic Analysis Attacks

Traffic analysis attacks are passive attacks where the attacker analyzes the traffic to predict the communication patterns in the network without being detected by the system. After analyzing the traffic for a considerably long period of time, the attacker gains sufficient knowledge about the timing behavior of the system. Upon successfully predicting the timing behavior of the system, the attacker can launch active attacks such as selective jamming attacks, compromising the normal functionalities of the system.

Goethem *et al.* proposed a novel type of remote timing attack, where the attacker exploits a series of network timing measurements by performing statistical analysis to reveal the differences in the execution time by different applications. This work describes the attack in HTTP/2 web servers, Tor onion services, and EAP-pwd (a popular Wi-Fi authentication method) [66]. Ramesh *et al.* presented a web traffic analysis based timing attack where the attacker analyzes the web traffic to use the packet timing information on the uplink communication [67]. As a countermeasure against this attack, this work uses spatial correlation of the received signal strength inherited from the wireless nodes. It then uses clustering based mechanism to detect the number of attackers and finally uses Support Vector Machines to further improve the accuracy of the countermeasure. However, this type of countermeasure cannot be implemented on low power wireless devices, and may not guarantee the timeliness requirements of the system. Some notable works provided countermeasures against traffic analysis attacks by hiding the location information of the victim nodes [24, 68]. The countermeasures proposed in these works either introduce randomness in routing, involve dummy packet injection or follow anonymous communication scheme in order to confuse the attacker. However, this countermeasure cannot guarantee the timeliness requirements of real-time network, hence not applicable to our system. Feghhi *et al.* presented a defense mechanism against timing attack by introducing a class of lower overhead tunnel resistant to traffic analysis [23]. The traffic analysis resistant tunnel is made by opportunistically reducing the number of dummy packets during busy times while increasing the number of dummy packets when the traffic load is less in the network. However, this countermeasure consumes a large amount of energy. Hence, this countermeasure cannot be adopted in our network with low power devices. Meng *et al.* proposed a secure scheme against timing attack that combines hybrid continuous-time Markov Chain with a queueing model for wireless sensor networks [69]. This countermeasure provides protection against attacks in wireless sensor networks with very low energy consumption. However, the flows considered in this work

are not associated with deadlines, hence the system model in this work is different from our work.

3.1.2 Selective Jamming Attacks

Selective jamming attack is a common timing attack in wireless networks [16]. To launch selective jamming attack, the attacker first eavesdrops or hears the communication patterns in the network by traffic analysis. After gaining sufficient knowledge on the communication patterns in the traffic, the attacker launches selective jamming attack by jamming specific transmissions in the wireless network. Proano *et al.* illustrated a selective jamming attack by compromising the node and modifying different information at the transport layer [16]. On the other hand, Daidone *et al.* proposed a GTS based selective jamming attack [70]. The countermeasure against the attack involves encryption of the messages followed by adoption of a central coordinator. The main objective of the coordinator is to randomize the slot allocations in each superframe such that the slots in the superframe become harder to predict. However, this countermeasure cannot be applied to our system as the flows in our system are associated with deadlines. Randomization over superframe does not guarantee flow deadlines. Pirayesh *et al.* provided a survey of different types of selective jamming attacks [71]. Countermeasures against selective jamming attacks in wireless sensor networks are well studied in the literature [70, 16, 72, 73]. Proano *et al.* proposed countermeasures against selective jamming attacks by using cryptographic primitives [73]. Wood *et al.* proposed DEEJAM, a countermeasure against selective jamming attack by adding a MAC-layer protocol to detect and defeat the stealthy jammers in IEEE 802.15.4 based platforms [72]. However, this type of countermeasure cannot protect our system against selective jamming attack as the traffic in our system is predictable in nature.

Like other wireless sensor actuator networks, the fifth generation cellular network (5G) is also vulnerable to jamming attacks [74, 75]. Several studies focusing on physical layer countermeasures to jamming attacks by adopting either Frequency Hopping Spread Spectrum (FHSS) or Direct Sequence Spread Spectrum (DSSS), exist in the literature. However, DSSS has high complexity in its implementation in 5G networks [74] and the frequency hopping in FHSS is predictable in real-time [74] in a 5G network. Hence, the above countermeasures are not fully effective against selective jamming attacks in 5G networks.

3.1.3 Other Types of Timing Attacks in Wireless Networks

Apart from traffic analysis attacks or selective jamming attacks, there are other side-channel attacks that exploit the timing behavior of the system to launch attacks in wireless networks. Power consumption attack is one such attack where the attacker analyzes the patterns in the power consumption of a specific node to launch an attack. Pongaliur *et al.* presented a survey on different types of side-channel attacks in wireless sensor networks such as power consumption attacks, selective jamming attacks, traffic analysis attacks [76]. This survey also provides a broad discussion on the possible countermeasures to each of these attacks. Brumley *et al.* provided a mechanism to attack weak computing devices such as smartcards [77]. The timing attack demonstrated in this work extracts private keys from an OpenSSL-based web server running on a machine in a local network [77]. Goethem *et al.* presented a web-based cross-site timing attack in which an adversary can obtain information of a user from a cross-origin website [78]. The attacker can exploit new side-channels in modern browser and can acquire accurate timing measurements from the new side-channels regardless of the network conditions.

Apart from the different defense mechanisms discussed in the above works, schedule randomization is a very popular defense mechanism to mitigate timing attacks in different platforms. Tiloca *et al.* proposed two countermeasures against timing attacks using schedule randomization in single and multi-channel wireless sensor networks respectively [27, 28]. These countermeasures permute the slot utilization patterns at the node level over a superframe to randomize the schedules in wireless sensor networks. Algin *et al.* proposed a similar defense mechanism in the context of smart meters [79]. However, all these proposed defense mechanisms consider non real-time communications, *i.e.*, the communications in these networks are not associated with hard deadlines. The deadline satisfaction of the flows in real-time systems impose additional constraints on the system which restricts permuting the slots over superframe. Hence, they cannot be adopted to provide a valid countermeasure to our system.

3.1.4 Timing Attacks in Real-Time Wireless Networks

Cheng *et al.* illustrated smart selective jamming attacks in WirelessHART network [17]. However, this work does not address any real-time solutions to the attack. Although timing attacks in wireless networks have been well-explored in the literature, most of these

works mainly focus on non real-time based wireless network, *i.e.*, networks where the flows are not associated with any deadlines. As a result, the countermeasures proposed in these works do not follow the strict timeliness requirements of real-time systems and hence, they are not applicable to our system. Countermeasures against timing attacks in real-time networks remain widely unexplored.

With this objective into our focus, in this thesis, we propose countermeasures against timing attacks in real-time wireless networks. Since, the attacker in real-time wireless networks exploits the predictability in the schedules to launch timing attacks, we aim at reducing the predictability in the schedules. Hence, we focus on schedule randomization based countermeasures in this thesis.

3.2 Timing Attacks on Cache, GPU and Microarchitectural Platforms

Timing attacks are also popular on cache memory [63], GPU [64], or other microarchitectural platforms [80], etc. A large number of applications use cryptographic algorithms to make their systems secure. However, cryptographic algorithms are vulnerable to timing attacks when they are implemented on real systems. By predicting the execution behavior and exploiting the measurements of physical quantities such as timing or power consumption, an attacker can launch timing attacks on cache or GPU systems.

3.2.1 Timing Attacks on Cache

Several works exist in the literature in the context of cache timing attacks across multiple applications. A few of them are mention-worthy. *Bernstein* was the first to demonstrate the complete AES key recovery process from known plaintext timings of a network server on another computer [21]. Since then, a lot of works have been proposed that exploit the vulnerabilities in the execution of AES encryption algorithm by launching cache timing attacks. *Bonneau et al.* presented the most powerful attack that is capable of recovering a full 128-bit AES over 213 timing samples [22]. This work shows an improvement of almost four orders of magnitude over *Bernstein's* work [21]. This work describes a general attack strategy and uses a simplified model of the cache to predict the timing variation due to cache-collisions in a sequence of lookups performed

by the encryption operations in AES [22]. This work proposes to use a small patch to reduce the vulnerability specific to cache-collision attacks [22]. Alawatugoda *et al.* proposed to hide the natural cache-timing pattern during encryption while still preserving its semantics [63]. Takarabt *et al.* proposed a recent work in IoT platforms that uses a methodology to access the robustness of MbedTLS library against timing attacks on cache [81]. The methodology presented in this work checks the whole source code to find out leakage of sensitive information in the system [81]. Settana *et al.* proposed a weighted average masking time algorithm to reduce the variances in the encryption time in cache [82]. By repeating the weighted average masking time, the average encryption time changes due to changes in the implementation environment. However, all these types of countermeasures are platform specific, hence they are not applicable to our system.

Jayasinghe *et al.* proposed a countermeasure against cache timing attacks that performs rescheduling of instructions such that the encryption rounds in the encryption process takes constant amount of time independent of cache hits or misses [25]. However, this type of rescheduling can only be applicable to our system if the flows in our system preserve deadlines even after rescheduling. Osvik *et al.* presented several software side-channel attacks based inter-process leakage that can reveal all the memory access patterns [83]. The revealed memory access patterns uses data-dependent table lookups that can be used for cryptanalysis of cryptographic primitives. These are very strong type of attacks that are possible even if the attacker has no knowledge of the specific plaintexts or ciphertexts. This attack can be launched only by monitoring the effect of the cryptographic process on the cache. However, the countermeasure to this attack is platform specific, hence, not applicable to our system. For further literature on cache timing attacks, we suggest interested readers to refer to these surveys [84, 85].

3.2.2 Timing Attacks on GPU or Other Microarchitectural Resources

Different other types of timing attacks involve exploiting microarchitectural or GPU resources to launch potential attacks by leaking sensitive information over time [65]. Andreou *et al.* demonstrated cache timing attacks on recent microarchitectures [86]. Ge *et al.* presented an extensive survey of timing attacks on contemporary hardware [87]. Jiang *et al.* presented the first work that demonstrated the vulnerability of a commercial GPU architecture to timing attacks [64]. This work was able to recover all key bytes for

AES-128 in less than 30 minutes by launching timing attacks in GPU system. Timing attacks are also possible by exploiting the variations in the power consumption when an encryption operation is performed. Lipp *et al.* presented PLATYPUS attacks which are a novel software-based power side-channel attacks on Intel server, desktop, or laptop CPUs [88]. This work showed how an unprivileged access to the RAPL interface exposes values that are directly correlated with the power consumption, thereby, forming a low resolution side-channel. Randolph *et al.* provided an exhaustive survey of power side-channel attacks in the literature [89]. All of these attacks and their countermeasures are mainly applicable for non real-time applications. Hence, they are not applicable to our systems.

3.2.3 Timing Attacks on Real-Time Processors

A few notable works on timing attacks in the context of real-time systems are presented [20, 19, 90, 91, 92, 93]. Yoon *et al.* proposed countermeasures against timing inference attacks such as those based on side-channels and covert-channels [20]. This work proposes a schedule randomization protocol, called *TaskShuffler*, that shuffles a set of fixed priority real-time tasks on a uniprocessor system to make the schedule unpredictable to the attacker [20]. Jiang *et al.* proposed a scheduler that randomizes the leakage points in a schedule to protect the system from Differential Power Analysis attacks [19]. Kruger *et al.* also proposed a schedule randomization policy that uses fine-grained slot-level slacks in a schedule and provides choices for the scheduler to randomly select a task at runtime [90]. Chen *et al.* presented a novel side channel attack in real-time task scheduler that can be exploited by an attacker to launch side-channel attack in a real-time scheduler [91]. Liu *et al.* proposed a side-channel attack in automotives where the attacker can infer the speed of the engine after observing and analyzing the real-time scheduling sequences on the Engine Control Unit [92]. This work provides two engine-triggered task period tracing methods to infer the periods of the engine-triggered tasks [92]. Liu *et al.* proposed a schedule-based timing side channel attack that can infer the number of tasks in the system and the period and execution time of the tasks directly from the execution sequence of the tasks without any prior knowledge of the task parameter information [93]. However, all of these works are applicable on uniprocessor platforms. The countermeasures proposed in the thesis in Chapter 5 and Chapter 7 are as hard as multiprocessor scheduling. m frequencies/channels and n real-time flows in a WirelessHART/5G network can be mapped to m processors and n real-time tasks in a

multiprocessor system respectively. Hence, the schedule randomization techniques on uniprocessor platforms are not applicable to our systems.

Several studies on different other types of attacks in wireless sensor networks such as other types of jamming attacks, denial of service attacks, spoofing attacks, packet modification attacks, etc exist in the literature. However, all these studies are orthogonal to the focus of this thesis, and hence, we do not present them here.

3.3 Different Metrics on Schedule Randomization

In this section, we present some related research on different metrics to measure the randomness in the schedules generated by our proposed countermeasures.

Schedule randomization is a popular defense mechanism against timing attacks. The main objective of schedule randomization is to generate random schedules such that the predictable behavior in the schedules are obfuscated to an attacker. To measure the randomness or uncertainty in the randomly generated schedules, Yoon *et al.* introduced the *upper-approximate schedule entropy* [20]. Upper-approximate schedule entropy measures the entropy of the slots over a hyperperiod across a set of randomly generated schedules. However, the upper-approximate schedule entropy fails to preserve the relative ordering of joint probability distribution of slots, when approximated using their empirical probability distribution. Another metric, commonly used in the literature is the *cross entropy* that measures the randomness in the schedules [94]. Cross-entropy measures the average number of total bits needed to estimate a probability distribution rather than a true distribution. However, cross-entropy cannot measure the divergence between two probability distributions. Hence, in this thesis, we use *Kullback-Leibler divergence (K-L divergence)* [29] as a metric to quantify the performance of our algorithm with reference to a truly random algorithm by measuring the divergence in their probability distribution of slots in the schedules. This metric is also used in the fields of machine learning [95] to compare two classification algorithms and in information retrieval [29] to compare between two query processing models. This metric has been used in Chapters 5, Chapter 6 and Chapter 7 of this thesis.

Chapter 4

Threat Models in WirelessHART and 5G Networks

In this chapter, we present the assumptions of our threat model to launch timing attacks in real-time networks. Based on these assumptions, we present the threat in the context of (1) the WirelessHART networks and (2) the fifth generation (5G) cellular networks, and illustrate the threats with examples. Then, we provide a discussion on the consequences of selective jamming attacks in WirelessHART and 5G networks, followed by its advantage over other types of jamming attacks. We propose countermeasures to the threats in Chapter 5, Chapter 6 and Chapter 7 of this thesis.

4.1 Assumptions of the Threat Model for Real-Time Wireless Networks

We present a *stealthy attacker* whose main objective is to select a specific node with periodic hard real-time flow as the *victim node* and the flow associated with the victim node as the targeted flow, and disrupt all the communications to/from that victim node. Since the schedules in real-time wireless networks are pre-computed to meet hard deadlines of the real-time flows and the same schedule is repeated over time, the communication slots in which the victim node communicates with a specific neighboring node become predictable over time. By selectively and repeatedly jamming specific communication slots associated with the victim node over every hyperperiod, the attacker can degrade

the performance of the system. Since, these systems are associated with hard deadlines, selectively jamming specific communications may even disrupt the safety of the system leading to catastrophic consequences.

Our threat model is based on the following assumptions.

- **Assumption 1:** The attacker needs to be within the transmission range of the victim node [96].
- **Assumption 2:** The attacker is equipped with directional antenna/antennae using which it can listen to the channels/frequencies used by the victim node [96, 17].
- **Assumption 3:** The attacker is aware of the network graph, *i.e.*, the nodes and the communication links between any two nodes in the network [96].
- **Assumption 4:** The attacker has no knowledge about the periods, deadlines and routes of the flows in the network. The attacker only knows the source and destination of the targeted flow [17].
- **Assumption 5:** All the nodes in the network are assumed to be secure and authenticated using join keys at the time of joining the network. Besides, all the packets flowing in the network are end-to-end encrypted using network keys. We assume that the attacker is not an authenticated user inside the system, *i.e.*, the attacker is not aware of the join keys, network keys and session keys in the system [96, 17]. Hence, it cannot capture the nodes or tamper with the packets flowing in the network.

We present the threat in the context of WirelessHART network and 5G cellular network based on these assumptions.

4.2 Threat Model in WirelessHART Networks

In WirelessHART networks, the communication schedule is decided at the time of network initialization as discussed in the last paragraph of Section 2.2.2 of Chapter 2 and the same hyperperiod schedule is repeated over time. As a result, the communication slots in which a specific node communicates with its neighboring nodes become predictable over time. This predictable behavior of the schedule serves as a vulnerability in

the communication network. An attacker can exploit this vulnerability to launch timing attacks in a WirelessHART network.

A WirelessHART network is generally deployed in industries to monitor and control applications remotely. The attacker can be a battery powered device deployed in such setups. Since the WirelessHART networks operate in the 2.4GHz ISM band, the attacker can listen to all the 16 channels in the network [96]. Hence, *the attacker can easily guess the number of operating channels (the m channels introduced in the System Model in Section 2.1 of Chapter 2) in the network*. Additionally, the Network Protocol Data Unit (NPDU) header which contains the sender and the receiver addresses of a transmission is unencrypted [97]. Hence, the attacker can analyze the traffic in the network. To launch an attack in such setups, the attacker selects a critical sensor or a critical actuator as the victim node and the critical flow associated with the victim node as the targeted flow. The victim node can either be the source or the destination of the targeted flow. It can even be a relay node of some other flows [98]. To remain stealthy, the attacker does not disrupt those flows which use the victim node as the relay node.

Based on the assumptions in Section 4.1, the attacker has the following capabilities.

- **Capability 1:** *The attacker can target a critical flow as the targeted flow and can select a critical sensor or a critical actuator (the source or destination of the targeted critical flow) as the victim node. It can use its directional antenna to detect the communication between any two nodes [99].*
- **Capability 2:** *The NPDU header field in a WirelessHART packet contains the source and the destination address of a transmission. Since the NPDU header is unencrypted in a WirelessHART network, the attacker can identify the transmissions associated with the victim node [97].*
- **Capability 3:** *Due to the repetitive nature of the communication schedule, the same communication repeats every hyperperiod. The attacker can estimate the hyperperiod of the schedule from the observed traces over sufficiently long period of time. Then, it can use this estimate in the subsequent hyperperiods to infer the communication slots of the victim node [17].*
- **Capability 4:** *The attacker can reverse engineer the channel hopping sequences by silently observing the channel activities in the network [96]. Since it knows*

the number of channels in the network, it can easily crack the channel hopping sequences following the steps as discussed in [96] with probability 1.0.

By using the above capabilities, our attacker works in two phases—

1. ***Schedule Observation Phase***, during which the attacker uses the above three capabilities to gather knowledge about the communication schedules. For instance, using Capability 1 and Capability 2, the attacker has a knowledge of the slots in which a victim node communicates with its neighbors in a hyperperiod schedule. Using Capability 3, the attacker can guess the channel through which the victim node communicates with a specific neighbor in a specific slot. Three possible cases can occur based on whether the victim node is only a source or only a destination of a targeted critical flow or a relay node of other flows.
 - (a) If the victim node acts only as the source (destination) of one or more targeted flows, then all the transmissions from (to) the victim node is associated with the targeted flow (flows). In this case, the attacker can easily predict the slots specific to the targeted flow (flows) from the repetitive patterns in the hyperperiod schedules.
 - (b) If the victim node acts as the source node of the targeted flow as well as relay node of other flows, then there are outgoing transmissions from the victim node specific to other flows. To determine the slots specific to those transmissions which belong to flows other than the targeted flow, the attacker considers the incoming transmissions (transmissions that are incident on the victim node) individually in each hyperperiod and selectively jams the transmission over that hyperperiod. This restricts further propagation of the selectively jammed flow from the victim node in that particular hyperperiod. Thus, the attacker can easily distinguish between the slots associated with the targeted flow from those associated with other flows having the victim node as the relay node.
 - (c) If the victim node acts as the destination node of the targeted flow as well as relay node of other flows, then there are outgoing transmissions from the victim node specific to other flows. In a similar manner, by selectively jamming the incoming transmissions (transmissions that are incident on the victim node) individually in each hyperperiod, the attacker can figure out the

slots associated with the targeted flow. If any flow other than the targeted flow gets jammed while being incident on the victim node in a hyperperiod, then there will be no subsequent transmissions of the jammed flow from the victim node over that hyperperiod. This helps the attacker to distinguish the slots associated with the targeted flow from those which consider the victim node as the relay node.

Note that, selectively jamming an intermediate hop of a flow over one hyperperiod appears to be a mere packet loss due to transmission failure to the network manager. However, the attacker uses this trick to determine the transmissions associated with the targeted flow. Once the slot(s) associated with the targeted flow is(are) known to the attacker, it switches to the *Attack Phase*.

2. **Attack Phase**, during which the attacker launches the attack by targeting specific transmissions from (to) certain critical sensors (actuators) and selectively jamming those transmissions in specific slots. Due to repetitive nature of the hyperperiod schedules, same flow gets transmitted in the same slot over every hyperperiod. Hence, *selectively jamming the predicted channel in specific slots over every hyperperiod results in jamming the targeted flow with probability 1.0*.

We illustrate the attack in WirelessHART network with an example.

Example 4.1. Consider the network graph as shown in Figure. 4.1 with two channels. Consider two flows, F_1 (marked in red) and F_2 (marked in blue) in Figure. 4.2, over the network graph with the following settings: the sources are $src_1 = 1$, $src_2 = 4$; the destinations are $des_1 = des_2 = D$; the periods/relative deadlines are $p_1 = \delta_1 = 6$ slots (60ms), $p_2 = \delta_2 = 3$ slots (30ms); the routes are $route_1 = [1 \rightarrow 2 \rightarrow 3 \rightarrow D]$ and $route_2 = [4 \rightarrow 5 \rightarrow D]$. Consider \mathcal{S} in Figure. 4.2 to be the hyperperiod schedule over the flows in the traditional time division multiple access based WirelessHART network. Now, consider node 1 to be the victim node. The network starts with schedule \mathcal{S} which repeats every 6 slots. An attacker listening to the channels in the network will find nodes 1 and 2 communicating every 6 slots. In particular, to identify this repetitive pattern, the attacker needs to listen to the network for at-least two hyperperiods, i.e., 12 slots. The attacker can launch selective jamming attack earliest in the 13th slot. By selectively jamming the transmission $[1 \rightarrow 2]$ over every hyperperiod, the attacker can prevent further transmission of F_1 in the subsequent slots of the schedule, thereby, degrading the performance of the system.

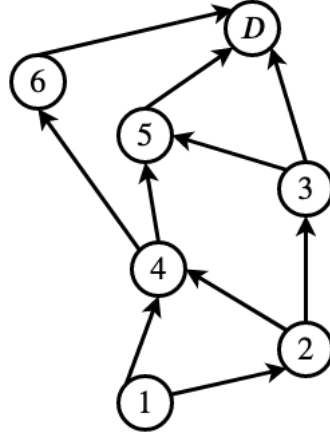


FIGURE 4.1: A network graph with seven nodes

Schedule	Channel	Slots					
		1	2	3	4	5	6
\mathcal{S}	ch ₁	1 → 2 (F ₁)	4 → 5 (F ₂)				5 → D (F ₂)
	ch ₂		2 → 3 (F ₁)	5 → D (F ₂)	3 → D (F ₁)	4 → 5 (F ₂)	

FIGURE 4.2: A schedule \mathcal{S} over 6 slots with two flows, F_1 (marked in red) and F_2 (marked in blue), where $src_1 = 1$, $src_2 = 4$, $des_1 = des_2 = D$.

4.3 Threat Model in 5G Networks

In 5G networks, a broad category of the URLLC flows that are associated with real-time cyber-physical systems are periodic in nature with hard deadlines. To ensure that all the flow deadlines are met, the base station (BS) reserves radio resources for such flows and pre-computes the schedule over a period of time. However, the periodic nature of the real-time flows makes the transmissions predictable in nature. The predictable behavior of the schedule can be exploited by an attacker in a stealthy manner to launch timing attacks in 5G networks.

We present an attacker whose main objective is to select a user equipment (UE) as the *victim node* and the uplink (downlink) flow associated with the victim node as the *critical flow*, and degrade the performance of the system by selectively and repeatedly (across flow instances) jamming a part of both the transmitted and re-transmitted data packets. Selectively jamming a part of the transmission results in wrong sensor data or actuation signal, leading to generation of wrong control signals and eventually degrading the system performance and safety.

We have a few assumptions specific to the 5G networks in addition to the assumptions in Section 4.1.

Additional Assumptions: The attacker is a wireless device equipped with 5G directional antenna. Due to high densification of devices in 5G networks, the average inter-UE distance is less than 100 meters [100]. Hence, the attacker needs to be in close proximity to the victim node. The attacker knows the operating frequencies of the victim node, (the set m introduced in the System Model in Section 2.1 of Chapter 2). Additionally, we make two standard assumptions regarding the flow schedules. (1) The original and the re-transmitted data packets are scheduled in successive slots in a schedule for periodic URLLC flows [101]. (2) The UEs in a group follow a pre-determined frequency hopping pattern to avoid interference and channel fading [101, 102]. *Note that, although these two assumptions are commonly used in the literature to simplify flow scheduling, they are not application requirements (refer to Definition 2.6 in Section 2.3.4 of Chapter 2). Hence, our proposed countermeasure on schedule randomization in Chapter 7 precisely eliminates these two assumptions.*

Based on the assumptions in Section 4.1 and the above assumptions, our attacker has the following capabilities.

- **Capability 1:** *The attacker can use its directional antenna to listen to the transmissions between the victim UE and the base station B. The attacker can guess the duration of a slot from the duration of each transmission.*
- **Capability 2:** *The attacker can use a time based marker to note the timestamp at which the victim UE communicates with B in each operating frequency.*
- **Capability 3:** *Since the same schedule repeats in each hyperperiod, the attacker can predict the slots in which the victim UE communicates with B by tuning its directional antenna to each operating frequency one at a time and analyzing the traffic for a sufficiently long period of time.*

Given these capabilities, the attacker uses the following steps to launch an attack. Steps 1 and 2 are the *Schedule Observation Phase* in which the attacker gathers knowledge to predict the flow associated with the victim UE. Step 3 is the *Attack Phase* in which the attacker actually launches the attack.

Schedule	Freq.	Slots											
\mathcal{S}		2	5	14	18	27	29	31	32	48	49	54	58
	fr_1	F_1		F_2		F_1			F_3	F_1		F_2	
		1	3	12	15	25	28	31	32	43	49	52	55
	fr_2		F_1		F_2		F_1	F_3			F_1		F_2

FIGURE 4.3: A schedule \mathcal{S} over 6 frames and 2 frequencies with three uplink flows F_1 (red), F_2 (blue), F_3 (brown); source devices $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; destination devices $des_1 = des_2 = des_3 = B$; periods: $p_1 = 20ms$, $p_2 = 30ms$, $p_3 = 60ms$; The slot duration in the schedule is $1ms$.

- **Step 1:** The attacker uses Capability 1 to guess the duration of a slot. It uses Capability 2 and Capability 3 to gather knowledge about the slots in which the victim UE communicates with B in each operating frequency. The attacker listens to a specific frequency till it observes a repetition in the communication pattern in that frequency. By repeating the above process in each operating frequency, the attacker can predict the slots used in each operating frequency.
- **Step 2:** To predict the frequency hopping pattern of the victim UE, the attacker marks a specific transmission, say tr_1 , in one operating frequency, say fr_1 , and uses Capability 2 to note the timestamp of the observed transmission. Then, it tunes the antenna immediately to another operating frequency, say fr_2 , and notes the timestamp of the first observed transmission in fr_2 . The difference between the two timestamps gives the number of slots between the two transmissions. Upon listening to subsequent transmissions in fr_2 , the attacker can easily guess the relative position of the first observed transmission in fr_2 . Upon repeating the same procedure for the remaining operating frequencies, the attacker can estimate the entire channel hopping pattern of the victim UE.
- **Step 3:** The attacker launches the attack by selectively jamming a portion of the data packets of both the transmissions associated with the flow of the victim UE in the predicted slots and frequencies.

We present an example to illustrate the steps of the attack.

Example 4.2. Consider three uplink flows F_1 , F_2 , and F_3 (marked in red, blue and brown respectively in Figure. 4.3), corresponding to three UEs. The source devices are $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; All the flows have a common destination device, $des_1 = des_2 = des_3 = B$. Consider schedule \mathcal{S} in Figure. 4.3 over two frequencies and with slot duration = $1ms$. All the re-transmissions in \mathcal{S} are scheduled in the slot immediately

after the original transmission. Consider U_1 to be the victim node and F_1 to be the critical flow. Assume that the attacker starts observing from the third frame in frequency fr_1 . It marks the timestamp at which U_1 transmits in slot 27 in fr_1 . It listens to the next transmissions of U_1 in fr_1 . Due to repetition of the schedule every $60ms$, the next transmissions of U_1 occur in slot 48 in the same hyperperiod, and in slots 2 and 27 in the next hyperperiod. The time difference between each of these transmissions in fr_1 are $21ms$, $14ms$ and $25ms$, respectively, after which the attacker observes repetition in the transmissions. Likewise, on switching to fr_2 , it observes a similar repetitive pattern in the transmissions associated with U_1 . To detect the frequency hopping pattern, the attacker tunes to fr_2 (say) and marks the timestamp of any transmission in fr_2 , say slot 3. Thereafter, it immediately switches to fr_1 and listens to the transmissions of U_1 along fr_1 . It notes the timestamp of the first observed transmission along fr_1 after $24ms$ in slot 27 of schedule \mathcal{S} . The subsequent transmissions of U_1 along fr_1 are already known to the attacker. Hence, the attacker can link the two observed sequence of transmissions along fr_1 and fr_2 to get the frequency hopping pattern. Now, the attacker can selectively jam specific transmissions associated with U_1 .

Note that the URLLC devices can support a maximum of 16 frequencies [52]. Since our system uses Frequency Division Duplexing mode (refer to Section 2.3.4 of Chapter 2), a part of these frequencies are used for uplink communication, while the remaining are used for downlink. In most real-time cyber-physical system applications, the periods of the real-time flows are not too long (usually range from a few milliseconds to a few seconds). For example, a safety application in V2X generally has a periodicity of $100ms$ [13]. Similarly, in industrial control systems the periods of these flows range from a few milliseconds to a few seconds [58, 59], and are chosen to be harmonic (*i.e.*, multiples of each other) [103]. As a result, the hyperperiod of a set of flows in these systems is typically not very long, which makes the schedule observation phase, and consequently, the attack, practical.

4.4 Consequences of Selective Jamming Attacks in WirelessHART and 5G Networks

In case of a WirelessHART network, selectively jamming a specific transmission from the victim node over every hyperperiod is a stealthy attack as the attack is perceived

as a mere transmission failure to the network manager and hence, remains undetected. However, selectively jamming the transmissions from a critical sensor node results in blocking the sensor data from reaching the access points or gateway. As a result, appropriate control commands cannot be generated at the gateway. Similarly, jamming the communication between the access points and the actuators prevents timely delivery of the control signals. Since these systems are time-critical in nature these attacks can undermine the system performance severely and can even cause catastrophic safety incidents [104]. Similarly, in case of a 5G network, selective jamming attack has a higher success probability as the attacker knows the exact communication slots to jam [74]. Since both the original and re-transmitted data packets of the periodic URLLC flows are partially jammed, it becomes difficult for the destination device to retrieve the actual data. Although 5G proposes some detection strategies at the physical layer to detect inconsistent transmissions [75], they are not fully effective in protecting the system from selective jamming attacks [74, 75].

Therefore, a secure and efficient flow scheduling strategy with deadline guarantees is needed to ensure that the attacker is not able to detect the communication patterns, while still ensuring that all the flow deadlines are met.

4.5 Advantages of Selective Jamming over other types of Jamming in Real-Time Networks

Unlike constant jamming attack that jams all the transmissions, selective jamming attack is more stealthy and hard to detect as it allows the attacker to strategically target certain critical devices within their proximity with much lower radio transmission power. This reduces the overhead and cost for the attacker to implement the jamming attack [105]. In contrast, random jamming attack does not infer the slots in the schedule. Instead, it jams in randomly selected slots. Randomly jamming a slot does not guarantee jamming the targeted flow with probability 1.0. Hence, it is much less effective [106]. Reactive jamming attack, on the other hand, constantly monitors the network and jams a channel/frequency whenever a specific transmission is observed without inferring the schedule [105]. Hence, it is far less effective. In reactive jamming, since the flows in the network have overlapping routes and the attacker does not infer the schedule before launching the attack, it may jam flows other than the targeted flow which makes reactive

jamming attacks far less stealthy than selective jamming attacks. Moreover, in reactive jamming, the amount of energy required to constantly monitor all the channels/frequencies in every slot in the network over every hyperperiod seems considerably high for a battery powered device. Thus, reactive jamming attack seems to be non-scalable from an attacker's point of view.

Chapter 5

Centralized Schedule Randomization in WirelessHART Networks

In this chapter, we present a moving target defense (MTD) technique to mitigate timing attacks in industrial control systems (ICSs). ICSs are associated with periodic real-time flows with hard deadlines. To ensure deadline satisfaction of the flows, the communication schedules are computed offline and the same schedule repeats over every hyperperiod which makes the communication predictable in nature. As a result, ICSs are vulnerable to timing attacks. Our main objective is to propose a MTD technique that reduces the predictability in the slots of the communication schedules so that the schedule changes before the attacker can predict it. ¹

We present the vulnerability of the scheduling policy in a WirelessHART network in ICSs and present the motivation of this work in Section 5.1. In Section 5.2, we briefly present the assumed system model. In Section 5.3, we briefly present the threat model. We illustrate a motivating example for this work in Section 5.4. Section 5.5 presents the proposed MTD technique and the time complexity analysis of the MTD technique. Section 5.6 shows the optimality of our proposed technique. Section 5.7 presents the metrics we used to measure the performance of our proposed technique. In Section 5.8, we evaluate the proposed MTD technique against timing attacks in WirelessHART networks. We conclude this chapter in Section 5.9.

¹The work in this chapter has been published in [33, 32]

5.1 Introduction

In Chapter 2, Section 2.2, we presented the characteristic features of a WirelessHART network that makes it the most suitable wireless protocol for real-time communications in ICSs. Most of the applications in ICSs are periodic in nature. The communication between each of the devices are periodic real-time flows with a hard deadline. To satisfy the hard deadline requirements in ICSs, the communication schedule is computed offline by the centralized network manager. Due to the periodic nature of the real-time flows, the communication schedule is repeated over every hyperperiod. Such repetition greatly helps the attacker to analyze the eavesdropped traces and infer the schedule. With the inferred schedule, the attacker can further launch various strategic attack steps. In Chapter 4, Section 4.2, we presented a description with an illustrative example of how a stealthy attacker can exploit the predictable behavior of the communication schedule to launch timing attacks in a WirelessHART network. For instance, the attacker can selectively jam the transmissions from (to) a certain critical sensor (actuator) which can eventually breach the safety of the system. In Chapter 4, Section 4.4, we discussed about the consequences of selective jamming attacks in a WirelessHART network.

In Chapter 1, Section 1.4, we discussed schedule randomization based countermeasures in wireless sensor networks [27, 28]. However, such countermeasures are not applicable for real-time networks, as discussed in Section 1.4. In Chapter 1, Section 1.4, we also discussed schedule randomization based countermeasures on real-time uniprocessor platforms [20, 90, 19]. However, such countermeasures cannot hold for multi-channel networks as discussed in Section 1.4.

To reduce the predictability of slots in the communication schedule of a WirelessHART network, in this chapter, we propose a *moving target defense (MTD) technique*, the *SlotSwapper*, that randomizes the slots in the communication schedule over every hyperperiod, while still satisfying the strict deadlines of all the real-time flows. We found that the attacker, capable of monitoring the wireless transmissions in the network, requires at-least two hyperperiods to infer the communication schedule. Thus, randomizing the schedule over every hyperperiod renders the attacker's inference ineffectual and greatly improves the confidentiality of the WirelessHART network's operations. However, randomizing the communication slots of a WirelessHART network is not straightforward. A feasible schedule over real-time flows in a WirelessHART network should satisfy the four constraints — (1) there should not be any conflicting transmissions

among the flows in the network, (2) there should not be any channel collisions in the network, (3) deadlines of all the real-time flows should be met to ensure safety of the system and (4) the communication slots should preserve the hop sequences of every flow instances. We defined a feasible schedule in Definition 2.5 in Section 2.2.2 of Chapter 2. These four constraints impose restrictions on the selection of slots which makes the schedule randomization problem harder.

5.2 System Model

In this chapter, we assume that our network model is represented by the generic Network Model described in Chapter 2, Section 2.1. As described in Section 2.1, our network model consists of a set of n flows, $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$, and m channels over a network graph $G = (V, E)$. The notations for the flow parameters, src_i , des_i , p_i , δ_i and $route_i$, have the same semantics as discussed in Section 2.2.2. Hence, Definition 2.1, Definition 2.2 and Definition 2.3 are also applicable to the flows in our system model. We consider a multi-channel WirelessHART network as described in Section 2.2.1 to be the main communication network in our system. Hence, our network model is extended based on the key features of a WirelessHART network as described in Section 2.2.2. A feasible schedule, \mathcal{S} , for our WirelessHART network is represented as Definition 2.5 of Section 2.2.2.

5.3 Threat Model

In this chapter, we consider a *stealthy* attacker with the assumptions as in Chapter 4, Section 4.1. Based on these assumptions, the attacker has three capabilities as described in Section 4.2. Using the capabilities as in Section 4.2, the attacker launches the attack in two phases — (1) the Schedule Observation Phase during which the attacker gathers knowledge about the communication slots in the schedules to predict the hyperperiod and the communication schedule. (2) the Attack Phase during which the attacker actually launches the attack. The attacker launches selective jamming attack in a similar manner as in Section 4.2. Example 4.1 in Chapter 4 illustrates the selective jamming attack that we consider in this chapter.

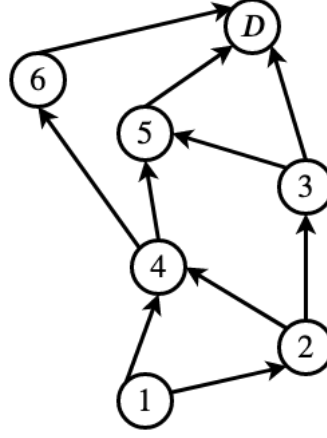


FIGURE 5.1: A network graph with seven nodes

Schedule	Channel	Slots					
		1	2	3	4	5	6
\mathcal{S}_1	ch ₁	$1 \rightarrow 2$ (F_1)	$4 \rightarrow 5$ (F_2)				$5 \rightarrow D$ (F_2)
	ch ₂		$2 \rightarrow 3$ (F_1)	$5 \rightarrow D$ (F_2)	$3 \rightarrow D$ (F_1)	$4 \rightarrow 5$ (F_2)	
\mathcal{S}_2		1	2	3	4	5	6
	ch ₁		$5 \rightarrow D$ (F_2)	$2 \rightarrow 3$ (F_1)	$4 \rightarrow 5$ (F_2)	$3 \rightarrow D$ (F_1)	$5 \rightarrow D$ (F_2)
	ch ₂	$4 \rightarrow 5$ (F_2)	$1 \rightarrow 2$ (F_1)				

FIGURE 5.2: Two schedules \mathcal{S}_1 and \mathcal{S}_2 over 6 slots with two flows, F_1 (marked in red) and F_2 (marked in blue), where $src_1 = 1$, $src_2 = 4$, $des_1 = des_2 = D$.

5.4 Motivation of our work

In this chapter, our main objective is to develop a moving target defense (MTD) technique, the *SlotSwapper*, that randomizes the communication slots over every hyperperiod schedule such that the schedule changes before the attacker can estimate it. In Chapter 4, Section 4.2, we present the three key capabilities of the attacker. Our MTD technique, the *SlotSwapper*, removes these three key capabilities of the attacker without which the attacker cannot launch further strategic attack steps. We present a motivating example to illustrate how the threat can be addressed by randomizing the slots over every hyperperiod schedule.

Example 5.1. Consider the same network graph as in Figure. 4.1 and flow settings as in Figure. 4.2 of Section 4.2 in Chapter 4. The two flows F_1 (marked in red) and F_2 (marked in blue), over the network graph have the following flow settings: the sources are $src_1 =$

1, $src_2 = 4$; the destinations are $des_1 = des_2 = D$; the periods/relative deadlines are $p_1 = \delta_1 = 6$ slots (60ms), $p_2 = \delta_2 = 3$ slots (30ms); the routes are $route_1 = [1 \rightarrow 2 \rightarrow 3 \rightarrow D]$ and $route_2 = [4 \rightarrow 5 \rightarrow D]$. Due to repetition of the same schedule \mathcal{S}_1 as explained in Example 4.1 in Section 4.5 of Chapter 4, the communication slots associated with a specific target node become predictable in nature. For instance, on executing schedule \mathcal{S}_1 over every hyperperiod, nodes 1 and 2 communicate every 6 slots. Taking the full advantage of this opportunity, an attacker can selectively jam specific transmissions, such as the transmission between node 1 and 2, and can degrade the performance of the system.

However, with our proposed moving target defense technique, a new schedule is executed in each hyperperiod, i.e., if \mathcal{S}_1 is followed in the first six slots, then another schedule, say \mathcal{S}_2 is followed in the next six slots and so on. For instance, from Figure 5.2, we can see that node 1 and node 2 communicate in the first slot in \mathcal{S}_1 , however, there is no communication between them in the first slot in \mathcal{S}_2 . Therefore, by changing the schedule in every hyperperiod, the system will change at a faster pace compared to the learning pace of the attacker, rendering further strategic destructive attack steps (e.g., selective jamming) infeasible.

Key differences of our attacker with reference to the attacker presented in [107]

Nasri *et al.* suggested the limitations and vulnerabilities of schedule randomization based countermeasures for real-time uniprocessor systems [107]. However, there are certain differences between our attacker and the one presented in [107]. We present an example to highlight the difference between our attacker and the one presented in [107].

Example 5.2. Consider the simple network graph, in Figure 5.3, with two flows, F_1 and F_2 , and with the following parameters; routes of the flows, $route_1 = [A \rightarrow C \rightarrow E]$ and $route_2 = [B \rightarrow C \rightarrow E]$; periods $p_1 = 4$ slots and $p_2 = 8$ slots. Without randomization, the schedule \mathcal{S} is repeated every hyperperiod. Hence, selectively jamming the transmission of at any slot, slot 4 (say), jams the transmission $[B \rightarrow C]$ of F_2 over every hyperperiod. According to the work presented in [107], the attacker knows the periods and the Worst Case Execution Time of the real-time tasks. This implies that the attacker knows the periods and the routes of the real-time flows in our network. Therefore, if $[B \rightarrow C]$ communication is scheduled at the first slot in \mathcal{S}_1 , then according to the work in [107], the attacker can easily infer the transmission $[C \rightarrow E]$ within the next 7 slots even after randomization with probability 1.0. However, schedule randomization is still

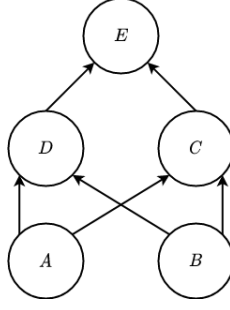


FIGURE 5.3: A network graph with 5 nodes

Schedule	Slots							
	1	2	3	4	5	6	7	8
\mathcal{S}	$A \rightarrow C$		$C \rightarrow E$	$B \rightarrow C$		$A \rightarrow C$	$C \rightarrow E$	$C \rightarrow E$
\mathcal{S}_1	$B \rightarrow C$	$A \rightarrow C$		$C \rightarrow E$	$A \rightarrow C$	$C \rightarrow E$		$C \rightarrow E$
\mathcal{S}_2	$A \rightarrow C$	$B \rightarrow C$	$C \rightarrow E$		$C \rightarrow E$	$A \rightarrow C$	$C \rightarrow E$	

FIGURE 5.4: Three schedules \mathcal{S} , \mathcal{S}_1 and \mathcal{S}_2 over the network graph in Figure. 5.3 with two flows F_1 (red) and F_2 (blue).

a valid countermeasure in our case because of two main reasons. Firstly, our attacker has no information about the periods and routes of the real-time flows in our network. Secondly, the flows in our network have multiple hops and overlapping routes. If a different schedule executes in each hyperperiod, then it becomes very difficult for an attacker to predict the slot in which a hyperperiod schedule begins and another schedule ends. If we assume that our attacker has somehow managed to guess the start and end point of a hyperperiod schedule even after randomization, then on observing transmission $[B \rightarrow C]$ in the first slot in \mathcal{S}_1 , our attacker is unaware of the scheduling window of F_2 . Hence, it cannot predict when the transmission $[C \rightarrow E]$ corresponding to F_2 will occur in the subsequent slots of the schedule. Also, it becomes impossible for the attacker to predict and jam the transmission $[C \rightarrow E]$ corresponding to F_2 due to overlapping routes between F_1 and F_2 .

5.5 Proposed MTD technique

Our MTD technique, the *SlotSwapper*, starts with an initial feasible schedule \mathcal{S}_B over a WirelessHART network graph G with m channels and a set of n flows $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$ and generates a list of *random feasible schedules* σ . We denote the initial feasible schedule \mathcal{S}_B as the base schedule. Table 5.1 presents the list of notations used by our algorithms.

SlotSwapper consists of two phases—

1. An offline schedule generation phase, the *Offline_Sched_Gen*, which runs at the network manager $K - 1$ times to generate a list of feasible schedules, σ . If the cardinality of σ is greater than 1, then a feasible schedule is randomly selected from σ as the base schedule in every iteration. To ensure variations in the slots of the generated schedules, K should be a large number. We provide a discussion on an estimate of K in Section 5.6.1.
2. The online schedule selection phase which runs at every hyperperiod in each node.

Algorithm 1 presents an overview of the *SlotSwapper*.

Algorithm 1: *SlotSwapper*(\mathcal{S}_B)

```

1  $\sigma = \{\mathcal{S}_B\};$  // initial base schedule
2 for  $i=1,2$  upto  $K-1$  // offline schedule generator
3 do
4   if  $|\sigma| > 1$  then
5      $\mathcal{S}_B =$  select a random schedule from  $\sigma$  as the base schedule;
6   Generate hop_list, edge_list over  $\mathcal{S}_B$ ,  $\mathcal{F}$  and  $G$ ;
7    $\mathcal{S} = \text{Offline\_Sched\_Gen}(\mathcal{S}_B);$ 
8    $\sigma = \sigma \cup \mathcal{S};$ 
9  $\mathcal{S}_R =$  select a random schedule from  $\sigma$  every hp; // online schedule
   selector

```

TABLE 5.1: List of notations used by *SlotSwapper*.

G	a network graph over V nodes and E edges
\mathcal{F}	a set of n flows defined over G
m	number of channels in the network
hp	hyperperiod of n flows
\mathcal{S}_B	an initial feasible schedule or the base schedule over hp with n flows and m channels
C_List	conflict list for the set of edges in E
<i>edge_list</i>	a list to store channel to edge mapping for each slot in \mathcal{S}_B over hp
<i>hop_list</i>	a list to store the mapping of hop number of every flow to the slot where it is scheduled in \mathcal{S}_B over hp
<i>elig</i>	eligible list of candidate flows for a particular slot
σ	list of random feasible schedules over hp

5.5.1 Offline Schedule Generation Phase

The *Offline_Sched_Gen* considers a feasible schedule \mathcal{S}_B uniformly at random from σ for a set of n flows, \mathcal{F} , over a network graph G with m channels, and generates a new feasible schedule \mathcal{S} by randomizing the slots in \mathcal{S}_B . Randomization of slots in \mathcal{S}_B are to be done in such a way that all the conditions of a feasible schedule (Definition 2.5 in Section 2.2.2 of Chapter 2) are satisfied. Two lists, *hop_list* and *edge_list*, are also generated from \mathcal{S}_B , \mathcal{F} and G , that are used by the *Offline_Sched_Gen*.

Algorithm 2 presents an overview of the *Offline_Sched_Gen*. The base schedule \mathcal{S}_B is copied in \mathcal{S} so that all updates are reflected in \mathcal{S} (line 1 of Algorithm 2). The slots in \mathcal{S}_B are scanned till deadline of any flow is encountered in the t^{th} slot, ($1 \leq t \leq hp$). Since the flows have implicit deadlines (period is equal to relative deadline) we use p_i to check this condition in line 4 of Algorithm 2. The instance number of the flow is calculated (line 5 of Algorithm 2). For each hop h of the flow instance F_{ij} , ($1 \leq h \leq F_{ij}.n_hops$), *hop_list*(F_{ij}, h) returns the slot at which the h^{th} hop of F_{ij} is scheduled in \mathcal{S}_B (line 7 of Algorithm 2). Note that, there are *no conflicting transmissions* in case of a *single-channel WirelessHART network*. As a result, we can ignore the edge associated with each hop of a flow instance for a single channel WirelessHART network. If the h^{th} hop of a flow F_i appears at any slot s_t , the edge associated with that hop gets automatically mapped to it. For each hop of F_{ij} , a list of slots, *elig*, that are eligible to be swapped with the current hop of F_{ij} are selected (lines 12-13 of Algorithm 2). Note that these slots are selected in such a way that the deadlines of the flow instances scheduled in these slots in \mathcal{S}_B are not violated after swapping with the current flow instance, F_{ij} . This condition is ensured by Algorithm 3. A slot is randomly selected from *elig* (line 14 of Algorithm 2) and swapped with the current flow instance (line 15 of Algorithm 2). The hop sequences of the current flow instance and the flows that are swapped with the current flow instance are updated in *hop_list* (line 16 of Algorithm 2).

Algorithm 3 takes two slots as inputs and checks whether the deadlines of the flow instances scheduled in these two slots in \mathcal{S}_B are violated if these two slots are swapped. The condition in line 4 of Algorithm 3 checks this condition and returns *true* if this condition is satisfied.

For a *multi-channel WirelessHART network*, additional constraints are imposed on the flows due to transmission conflicts and collisions (Condition 1 and 2 of Definition 2.5 in Section 2.2.2 in Chapter 2). As a result, unlike single-channel network, the edges

Algorithm 2: *Offline_Sched_Gen*(\mathcal{S}_B)

```

1 Copy  $\mathcal{S}_B$  to  $\mathcal{S}$  over  $hp$ ;
2 for  $t = 1, 2, \dots, hp$  do
3   for  $i = 1, 2, \dots, n$  do
4     if  $t \% p_i == 0$  then
5        $j = \frac{t}{p_i}$ ; // current instance of  $F_i$ 
6       for  $h = 1, 2, \dots, F_{ij}.n\_hops$  do
7          $s_t = \text{hop\_list}(F_{ij}, h)$ ;
8          $\text{elig} = \{\}$ ; // empty list
9         if  $m == 1$  // single-channel
10          then
11            for  $s'_t \in W_{ij}$  do
12              if  $\text{deadCheck}(s_t, s'_t) == \text{true}$  then
13                Add  $s'_t$  to  $\text{elig}$ ;
14             $s_r = \text{random}(\text{elig})$ ; // random slot
15             $\text{swap}(s_t, s_r)$ ;
16            update  $\text{hop\_list}, \mathcal{S}$ ;
17          else
18             $ch_1 = \text{channel of } h^{\text{th}} \text{ hop of } F_{ij}$ ;
19            if  $h == 1$  // first hop
20              then
21                 $lb = \text{first slot in } W_{ij}$ ;
22              else
23                 $lb = \text{hop\_list}(F_{ij}, h-1) + 1$ ; // slot next to the
24                   $(h-1)^{\text{th}}$  hop of  $F_{ij}$ 
25            if  $h == F_{ij}.n\_hops$  // last hop
26              then
27                 $ub = \text{last slot in } W_{ij}$ ;
28              else
29                 $ub = \text{hop\_list}(F_{ij}, h+1) - 1$ ; // slot previous to
30                  the  $(h+1)^{\text{th}}$  hop of  $F_{ij}$ 
31            for  $s'_t \in [lb, ub]$  do
32              for  $ch = 1, 2, \dots, m$  do
33                if  $(\text{transConf}(s_t, ch_1, s'_t, ch) \ \&\& \ \text{deadCheck}(s_t, s'_t) \ \&\& \ \text{flowSeq}(s_t, s'_t) == \text{true})$  then
34                  Add  $(s'_t, ch)$  to  $\text{elig}$ ;
35             $(s_r, c_r) = \text{random}(\text{elig})$ ; // random slot channel pair
36             $\text{swap}(s_t, ch_1, s_r, c_r)$ ;
37            update  $\text{hop\_list}, \text{edge\_list}$  and  $\mathcal{S}$ ;
38 return  $\mathcal{S}$ ;

```

Algorithm 3: *deadCheck*(s_t, s'_t)

```

1  $F_{i'} = \mathcal{S}_B[s'_t]$ ;
2  $j' = \lceil \frac{s'_t}{p_{i'}} \rceil$ ; // get the instance number of  $F_{i'}$ 
3 Assign the scheduling window of  $F_{i'j'}$  to  $W'$ ;
4 if ( $s'_t \in W_{ij}$ ) && ( $s_t \in W'$ ) then
5   | return true;
6 return false;

```

Algorithm 4: *transConf*(s_t, ch_1, s'_t, ch)

```

1  $e_1 = edge\_list[s_t].get(ch_1)$ ;
2  $e_2 = edge\_list[s'_t].get(ch)$ ;
3 for ( $i = 1, 2, \dots, m$ ) && ( $i \neq ch$ ) do
4   |  $e_3 = edge\_list[s'_t].get(i)$ ;
5   | if  $e_1 \in C\_List[e_3]$  //  $e_1$  is in Conflict List of  $e_3$ 
6   |   then
7   |     | return false;
8 for ( $i = 1, 2, \dots, m$ ) && ( $i \neq ch_1$ ) do
9   |  $e_3 = edge\_list[s_t].get(i)$ ;
10  | if  $e_2 \in C\_List[e_3]$  //  $e_2$  is in Conflict List of  $e_3$ 
11  |   then
12  |     | return false;
13 return true;

```

associated with each hop of a flow instance needs to be considered. The window for h^{th} hop of a flow instance, $1 < h < n_hops$, is represented by the slots between the $(h-1)^{st}$ hop (lines 22-23 of Algorithm 2) and $(h+1)^{st}$ hop (lines 27-28 of Algorithm 2). If h is the first hop, then the window of the h^{th} hop is between the release time and $(h+1)^{st}$ hop of the flow instance (lines 19-21 of Algorithm 2). Similarly, if h is the last hop, then the window is between the $(h-1)^{st}$ hop and deadline of the flow instance (lines 24-26 of Algorithm 2). The main reason for restricting the window within this small range is to avoid additional computations involved in reverting back some of the swaps to preserve the hop sequences of the flows. However, this may restrict us from selecting all possible (slot,channel) pairs in *elig* and hence prevent us from exploring all possible feasible schedules in σ . A list of eligible (slot,channel) pairs, *elig*, is generated by selecting (slot,channel) pairs for each hop of the current flow instance (lines 29-33 of Algorithm 2).

Algorithm 5: $flowSeq(s_t, s'_t)$

```

1  $F_{i'j'} = \mathcal{S}_B[s'_t]$ ;
2  $j' = \lceil \frac{s'_t}{p_{i'}} \rceil$ ; // instance number of  $F_{i'j'}$ 
3  $h$  = the hop number of  $F_{i'j'}$  at slot  $s'_t$  in  $\mathcal{S}_B$ ;
4 if  $h$  is the first hop then
5   if  $s_t \notin W_{i'j'}$  then
6     return false;
7 else if  $h$  is the last hop then
8   if  $s_t \notin W_{i'j'}$  then
9     return false;
10 else
11   if  $s_t < s'_t$  &&  $hop\_list(F_{i'j'}, h-1) < s_t$  then
12     return true;
13   else if  $s_t > s'_t$  &&  $hop\_list(F_{i'j'}, h+1) > s_t$  then
14     return true;
15 return false;

```

Given any two (slot,channel) pairs, (s_t, ch_1) and (s'_t, ch) , Algorithm 4 checks for conflicting transmissions between the two pairs and returns *true* if no such conflicts exist. Lines 1-2 finds the edge corresponding to a specific slot and channel in the *edge_list*. Lines 3-7 and lines 8-12 check if there is any transmission conflict on swapping (s_t, ch_1) and (s'_t, ch) by searching the entries corresponding to s_t and s'_t in the conflict list, *C_List*, respectively, and returns *true* if no such conflicts exist.

Algorithm 5 checks whether the flow instances to be swapped preserve the hop sequences of the flows even after swapping. The hop number of $F_{i'j'}$ scheduled in s'_t in \mathcal{S}_B is assigned to h (line 3 of Algorithm 5). If h is the first hop of $F_{i'j'}$, then lines 4-6 check whether $F_{i'j'}$ is still scheduled within its scheduling window $W_{i'j'}$ even after swapping. Similarly, lines 7-9 check the same condition for the last hop of $F_{i'j'}$. If h is an intermediate hop, then depending on the position of h with respect to s_t in \mathcal{S}_B , lines 11-12 check whether the hop sequences of the previous hops of $F_{i'j'}$ are preserved even after swapping and returns *true* if the condition is satisfied. Similarly, lines 13-14 check the same for the next hops of $F_{i'j'}$.

Finally, a random (slot,channel) pair is selected from *elig* and swapped with the current flow instance (lines 34-35 of Algorithm 2). *hop_list*, *edge_list* and \mathcal{S} are updated corresponding to each flow instance (line 36 of Algorithm 2). The process is repeated for

each hop of every flow instance in the hyperperiod schedule to get a completely new schedule \mathcal{S} .

We illustrate the steps of the *Offline_Sched_Gen* with an example.

Example 5.3. Consider the flow setting as in Figure. 5.2 and Example 5.1. Let \mathcal{S}_1 in Figure. 5.2 be the base schedule. Let us consider the 3rd hop of F_1 in \mathcal{S}_1 which occurs at the 4th slot of 2nd channel. The window corresponding to 3rd hop of F_1 is [3,6]. For every slot in [3,6] and for each channel, we check for conflicting transmissions. [3 \rightarrow D] has conflicting transmission with [5 \rightarrow D] in \mathcal{S}_1 . Therefore, (slot,channel) pairs such as, (3,1) and (6,2) are rejected due to transmission conflict with [3 \rightarrow D]. Similarly, (slot,channel) pair such as (3,2) is also rejected by function *deadCheck()* due to violation of deadline of the first instance of F_2 . The (slot,channel) pair (3,1) corresponds to the first instance of F_2 with scheduling window between 1st slot and 3rd slot. Hence, the first instance of F_2 cannot be swapped with any slots after slot 3 in \mathcal{S}_1 . Similarly, *flowSeq()* does not allow (slot,channel) pair (6,1) in the eligible list in order to preserve the hop sequences of flows. If the transmission corresponding to 2nd hop of 2nd instance of F_2 (via edge 5 \rightarrow D) of (slot,channel) pair (6,1) is allowed to swap with (4,2), then the second hop of that instance of F_2 would have been scheduled before the first hop, violating the hop sequences of the flow instances. Finally, the list of eligible (slot,channel) pairs are — [(4,1), (5,1), (5,2)]. Let (5,1) be the randomly selected element. Swapping the transmissions and the flow instances between (4,2) and (5,1) and iterating the same procedure over all the flow instances generates a completely new feasible schedule.

Note that, in this work we have assumed that there is no spatial re-use of channels in the network. However, the *Offline_Sched_Gen* can be easily extended to support spatial re-use of channels. We need to consider a list of entries corresponding to each (slot,channel) pair in the representation of a schedule instead of a single entry in the (slot,channel) pair in the current algorithm. In addition, we need to check the conflicts for the list of (slot,channel) pairs instead of a single (slot,channel) pair in the existing algorithm. To maintain reliable communication, WirelessHART re-transmits a packet once via the same route if the initial transmission fails. Note that, we have not reserved slots in our schedules that correspond to re-transmission of packets along the same path in the network. However, to facilitate re-transmission, we need to duplicate the flows in our schedules with the same periods, routes in the Offline Schedule Generation Phase. In the Online Schedule Selection Phase, we need to implement some additional checks at

the node level to determine whether a packet corresponding to a specific flow needs to be re-transmitted or not.

Theorem 5.1. *The `Offline_Sched_Gen` always generates feasible schedules.*

Proof. Consider a WirelessHART network G with a set of n flows, $\mathcal{F} = \{F_1, \dots, F_n\}$, and a feasible base-schedule \mathcal{S}_B . Each modification in the schedule \mathcal{S}_B is accompanied by swapping a pair of slots in \mathcal{S}_B . Let s_1 and s_2 be two such slots. Each of these slots is associated with an instance of (say) F_i and F'_i respectively, $F_i, F'_i \in \mathcal{F}$. For a single-channel network, an instance of F_i is eligible to be swapped with an instance of F'_i , if and only if both of these instances lie within their respective release time and deadline even after swapping, satisfying Condition 3 of feasible schedule (refer to Definition 2.5 in Section 2.2.2 of Chapter 2). The hop sequence of each flow instance is updated after swapping to satisfy Condition 4 of a feasible schedule (Definition 2.5 in Section 2.2.2 of Chapter 2). The remaining two conditions are not applicable for single-channel networks.

In case of a multi-channel network with m channels, the `Offline_Sched_Gen` associates a window corresponding to each hop of a flow instance to satisfy Condition 4 of Definition 2.5 in Section 2.2.2 of Chapter 2 for the current flow instance. Two slots s_1 and s_2 are only eligible to be swapped within this window. Also, `flowSeq()` in line 31 checks whether the hop sequences of other flow instances are preserved on swapping with the current flow instance. In addition, s_1 and s_2 are also checked for conflicting transmissions (`transConf()` in line 31) to satisfy Condition 1 and Condition 2 of Definition 2.5 in Section 2.2.2 of Chapter 2. The deadline satisfaction of flow instances, Condition 3 of Definition 2.5 in Section 2.2.2 of Chapter 2, is checked in `deadCheck()` in line 31. Thus, for both single and multi-channel WirelessHART networks, each candidate slot in the eligible list satisfies all the conditions of feasible schedules. Hence, the schedules generated by randomly selecting any one of these eligible candidate slots in each iteration always generates feasible schedules. \square

5.5.2 Online Schedule Selection Phase

On executing the offline `Offline_Sched_Gen` $K - 1$ times in the network manager, we get a list of feasible schedules, σ . At the time of network initialization, each node is informed about the slots in which it can transmit (receive) messages in each of these

K hyperperiod schedules ($K - 1$ schedules generated by *Offline_Sched_Gen* and one initial base schedule \mathcal{S}_B). Each node wakes up only at its allocated slots during which it can transmit (receive) messages to (from) other nodes. At every hyperperiod, each node in the network selects a schedule \mathcal{S}_R from σ uniformly at random and executes \mathcal{S}_R over that hyperperiod.

To ensure that all the nodes in the network select the *same schedule in a distributed manner* without any additional communication, we suggest to use a pseudo-random number generator (PRNG) [108], initialized with the same value and same seed at each node in the network. Since, we assume that the attacker cannot capture the nodes in the network (refer Assumption 5 of the Threat Model in Section 4.1 of Chapter 4), the initial value and the seed of the PRNG are unknown to the attacker. Hence, the PRNG is assumed to be secure. If a node loses its synchronization or if a node resets, then the network manager should send the intermediate value of the secure PRNG to the node along with the synchronization messages [109] so that the node can generate the same random number like all other active nodes after joining the network. To ensure large variations in the slots of a schedule, K should be a large number. However, the amount of memory available in each sensor node is limited. The number of schedules, K_{node} , that can be stored in each sensor node depends highly on the maximum number of slots in which a node becomes active in a hyperperiod schedule. If Mem be the amount of memory available in each node and t_i be the number of slots in which the i^{th} node becomes active (transmit or receive packets) in each hyperperiod schedule \mathcal{S} , then the number of hyperperiod schedules K_{node} that can be stored in each node is

$$K_{node} = \left\lfloor \frac{Mem}{\max(t_i) \mid \forall i \in |V|} \right\rfloor. \quad (5.1)$$

All the nodes in the network are assumed to be secure and authenticated. As a result, the number of hyperperiod schedules that each node can store in the network, K_{node} , is assumed to be secure. Each of these K_{node} hyperperiod schedules needs to be replenished periodically by new schedules so that there is enough variation in the slots of the executed schedules. The gateway broadcasts new schedules to all the active nodes in the network after a few hyperperiods. In Section 5.8.4 of this chapter, we provide a discussion from experiments on the number of schedules K_{node} that can be stored in each sensor node and the time duration after which the schedules are replenished.

5.5.3 Time Complexity Analysis

Generating the lists, *hop_list* and *edge_list* take $\mathcal{O}(n \times hp)$ time and $\mathcal{O}(hp \times m)$ time respectively. Consider the *Offline_Sched_Gen* and an initial base schedule of length *hp* with *n* flows and *m* channels. Copying \mathcal{S}_B to \mathcal{S} takes $\mathcal{O}(hp \times m)$ time. For a single-channel network, creating a list of eligible candidates for swapping, takes $\mathcal{O}(hp)$ time in the worst-case (lines 11-13). For a multi-channel network, selecting the window for h^{th} hop of a flow instance takes $\mathcal{O}(1)$ time (lines 19-28). Creating a list of eligible (slot,channel) pairs involves scanning all the slots in the scheduling window of the flow instance and checking for all the four conditions of a feasible schedule. Each call to *transConf()* function takes $\mathcal{O}(m)$ time (line 32). The functions *flowSeq()* and *deadCheck()* take $\mathcal{O}(1)$ time each. If $p_i = hp$, the scheduling window corresponding to a flow instance can be of length *hp* in the worst-case. In that case, there will be $hp \times m$ calls to these functions. Considering the number of hops between the sensor and gateway or between the gateway and the actuator in a WirelessHART network to be at-most 4 [110], we have $4 \times hp \times n$ iterations. On executing *Offline_Sched_Gen* *K* times, the overall time complexity is $\mathcal{O}((hp)^2 nmK)$. The online schedule selection phase generates a random number between 1 to *K* which takes $\mathcal{O}(1)$ time. Thus, the overall time complexity of *SlotSwapper* is $\mathcal{O}((hp)^2 nmK)$.

5.6 Optimality of *SlotSwapper* for harmonic flows in single-channel WirelessHART network

An algorithm is said to be *optimal* if it generates any feasible schedule with equal probability. An *optimal* algorithm makes it very difficult for an attacker to deduce the schedule because it is able to generate and deploy any feasible schedule. To prove the *optimality* of *SlotSwapper*, presented in Algorithm 1 of this chapter, we use the following definitions.

Definition 5.1. A schedule generation and deployment algorithm in a WirelessHART network is said to be optimal if it can generate any feasible schedule with equal probability.

Definition 5.2. A set of *n* flows $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$ with periods p_1, p_2, \dots, p_n respectively is said to be harmonic if any pairwise periods divide each other. More specifically,

the set of periods p_1, p_2, \dots, p_n is harmonic if for every p_i and p_j , either $p_i/p_j \in \mathbb{N}$, or $p_j/p_i \in \mathbb{N}$.

Consider a set of n flows $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$ with periods p_1, p_2, \dots, p_n (represented in slots) and the number of hops h_1, h_2, \dots, h_n respectively. We assume that the flows are sorted in ascending order of their periods, i.e., $p_1 \leq p_2 \leq \dots \leq p_n$ and the flows are harmonic i.e., $p_n = q_{n-1} \cdot p_{n-1} = q_{n-1} \cdot q_{n-2} \cdot p_{n-2} = \dots = q_{n-1} \cdot q_{n-2} \cdot q_{n-3} \dots q_1 \cdot p_1$, where $q_1, q_2, \dots, q_{n-1} \in \mathbb{N}$. The k^{th} instance of flow F_i with period p_i has a scheduling window W_{ik} . The slots in W_{ik} is given by $[(k-1) \cdot p_i + 1, k \cdot p_i]$. Since each flow has an implicit deadline, the length of its scheduling window is equal to its period.

The total number of feasible schedules N_{sched} for a set of n harmonic flows over a single-channel WirelessHART network is given by

$$N_{sched} = \prod_{i=1}^n \beta_i \quad (5.2)$$

where β_i is given by

$$\beta_i = \begin{cases} \left(C_{h_i}^{p_i} \right)^{\frac{h_i p_i}{p_i}}, & i = 1 \\ \left(C_{h_i}^{(p_i - \sum_{1 \leq j \leq i-1} \frac{p_i}{p_j} \cdot h_j)} \right)^{\frac{h_i p_i}{p_i}}, & 2 \leq i \leq n \end{cases}$$

$C_{h_i}^{p_i}$ is the number of ways of selecting h_i slots out of p_i slots for the flow with minimum period.

For all other flows, F_i , with period $p_i > p_1$, $C_{h_i}^{(p_i - \sum_{1 \leq j \leq i-1} \frac{p_i}{p_j} \cdot h_j)}$ denotes the number of ways of selecting h_i slots from the available slots after allocating the slots for the flows with periods less than p_i . If an algorithm generates any feasible schedule with equal probability, then the probability of occurrence of each schedule is given by N_{sched}^{-1} . Thus, if an algorithm generates any feasible schedule with probability N_{sched}^{-1} , then that algorithm is said to be optimal.

Theorem 5.2. *If all the flows are harmonic, the `Offline_Sched_Gen`, defined in Algorithm 2, generates each feasible schedule with probability N_{sched}^{-1} for a single-channel WirelessHART network, where N_{sched} is given by Equation 5.2.*

Proof. To show that the *Offline_Sched_Gen* generates each feasible schedule with probability N_{sched}^{-1} , we calculate the probability of generating each feasible schedule for a single-channel WirelessHART network with a single flow and two flows as our base case. Then, we induct on the number of flows and show that for any number of flows, $n \in \mathbb{N}$, the *Offline_Sched_Gen* generates each feasible schedule with probability N_{sched}^{-1} .

Base Case: Consider flow F_1 with period p_1 , i.e., $n = 1$. In this case, the hyperperiod is equal to p_1 and consists of only one instance of F_1 . Each instance of F_1 consists of h_1 hops. Therefore, the number of feasible schedules with a single flow, F_1 , is given by $N_{sched} = C_{h_1}^{p_1}$. Consider the scheduling window of F_1 , i.e., $[1, p_1]$. Since F_1 is the only flow, F_1 is equally likely to be scheduled at any slot in $[1, p_1]$. Assuming that the random slot selector (line 14 of Algorithm 2 of this chapter) selects slots from the *elig_list* uniformly at random, F_1 can occur at any slot in $[1, p_1]$ with probability $\frac{1}{p_1}$. There are h_1 hops in F_1 . Thus, the probability of generating any feasible schedule for a single flow by the *Offline_Sched_Gen* is given by $\left(C_{h_1}^{p_1}\right)^{-1}$.

Now, consider we have two flows, $\mathcal{F} = \{F_1, F_2\}$ with periods p_1 and p_2 respectively, such that $p_2 = q_1 \cdot p_1$, i.e., $n = 2$. This implies, there are q_1 instances of F_1 within each scheduling window of F_2 . Since, $p_2 = q_1 \cdot p_1$, the hyperperiod of any schedule with two flows, F_1 and F_2 , is equal to p_2 and the schedule consists of q_1 instances of F_1 and one instance of F_2 in it. Each instance of F_1 has h_1 hops within its scheduling window. Since each instance of F_1 can occur at any slot within its scheduling window and Algorithm 2 of this chapter randomizes the hops of F_1 followed by F_2 based on their deadlines, the probability of generating each feasible schedule of length p_2 with q_1 instances of F_1 is given by $\frac{1}{\left(C_{h_1}^{p_1}\right)^{\frac{p_2}{p_1}}}$. Now, consider the only instance of F_2 within scheduling window

$[1, p_2]$. Any two slots, say s_1 and s_2 containing a hop of the x^{th} instance of F_1 and the first instance of F_2 , with scheduling windows $[(x-1) \cdot p_1 + 1, x \cdot p_1]$ and $[1, p_2]$ respectively are eligible to be swapped with each other if $(x-1) \cdot p_1 + 1 \leq s_1, s_2 \leq x \cdot p_1$ (line 14 of Algorithm 2). Hence, for each hop of F_2 , $(q_1 - 1) \cdot h_1$ hops of F_1 belonging to remaining $(q_1 - 1)$ scheduling windows of F_1 within interval $[1, p_2]$ cannot be considered in the *elig_list* of F_2 . Since our slot selector selects each slot from the *elig_list* uniformly at random (line 14 of Algorithm 2), for each feasible schedule of length p_2 with q_1 instances of F_1 , the probability with which the only instance of F_2 occurs at any slot within $[1, p_2]$ is given by $\left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1}\right)^{-1}$. Since each flow is independent, the probability

of generating any feasible schedule with two flows by the *Offline_Sched_Gen* is given by $\left(\left(C_{h_1}^{p_1} \right)^{\frac{p_2}{p_1}} \times \left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1} \right) \right)^{-1}$.

Inductive Hypothesis: There are m flows in \mathcal{F} , i.e., $n = m$ (say). Consider first instance of m^{th} flow F_m with period p_m such that $p_m = q_{m-1} \cdot p_{m-1} = q_{m-1} \cdot q_{m-2} \cdot p_{m-2} = \dots = q_{m-1} \cdot q_{m-2} \cdot \dots \cdot q_2 \cdot q_1 \cdot p_1$, where q_1, q_2, \dots, q_{m-1} are positive integers greater than or equal to 1. This implies that there are one instance of F_m , q_{m-1} instances of F_{m-1} , $q_{m-1} \cdot q_{m-2}$ instances of F_{m-2} , \dots , $q_{m-1} \cdot q_{m-2} \cdot \dots \cdot q_2 \cdot q_1$ instances of F_1 within $[1, p_m]$. Assuming that the random slot selector in line 14 of Algorithm 2 of this chapter selects slots uniformly at random, let each of the feasible schedules over m flows be generated by the *Offline_Sched_Gen* with probability

$$\frac{1}{\left(C_{h_1}^{p_1} \right)^{\frac{p_m}{p_1}} \times \left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1} \right)^{\frac{p_m}{p_2}} \times \dots \times \left(C_{h_m}^{p_m - \sum_{1 \leq j \leq m-1} \frac{p_m}{p_j} \cdot h_j} \right)}.$$

We need to show that the above hypothesis is true for any number of flows in \mathcal{F} .

Consider we have $n = m + 1$ flows in \mathcal{F} . Let us consider the first instance of F_{m+1} with period p_{m+1} , such that $p_{m+1} = q' \cdot p_m$. Since, the flows are harmonic, we have $p_{m+1} = q' \cdot p_m = q' \cdot q_{m-1} \cdot p_{m-1} = q' \cdot q_{m-1} \cdot q_{m-2} \cdot p_{m-2} = \dots = q' \cdot q_{m-1} \cdot q_{m-2} \cdot \dots \cdot q_1 \cdot p_1$. The hyperperiod of any schedule with F_{m+1} flows is equal to p_{m+1} . Consider the scheduling window $[1, p_{m+1}]$ corresponding to the first instance of F_{m+1} . By induction hypothesis, each feasible schedule in the window $[1, p_m]$ shows up with probability

$$\frac{1}{\left(C_{h_1}^{p_1} \right)^{\frac{p_m}{p_1}} \times \left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1} \right)^{\frac{p_m}{p_2}} \times \dots \times \left(C_{h_m}^{p_m - \sum_{1 \leq j \leq m-1} \frac{p_m}{p_j} \cdot h_j} \right)}.$$

There are q' scheduling windows corresponding to the q' instances of F_m in $[1, p_{m+1}]$. The number of hops of all the flow instances in each of these q' scheduling windows is fixed. Since the *Offline_Sched_Gen* selects slots from the *elig_list* uniformly at random, the probability of each feasible schedule of length p_{m+1} with m flows is

$$\frac{1}{\left(C_{h_1}^{p_1} \right)^{\frac{p_{m+1}}{p_1}} \times \left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1} \right)^{\frac{p_{m+1}}{p_2}} \times \dots \times \left(C_{h_m}^{p_m - \sum_{1 \leq j \leq m-1} \frac{p_m}{p_j} \cdot h_j} \right)^{\frac{p_{m+1}}{p_m}}}.$$

Now consider the only instance of $(m + 1)^{th}$ flow, F_{m+1} , having a scheduling window of $[1, p_{m+1}]$. Each hop of the flow instance of F_{m+1} can occur at any slot belonging to any of the q' scheduling windows of F_m . Since our slot selector selects each slot from the *elig_list* uniformly at random (line 14 of Algorithm 2 of this chapter), for each

feasible schedule of length p_{m+1} with q' instances of F_m , $q' \cdot q_{m-1}$ instances of F_{m-1} , \dots , $q' \cdot q_{m-1} \cdot q_{m-2} \cdot \dots \cdot q_1$ instances of F_1 , the probability with which the only instance of F_{m+1} occurs at any slot within $[1, p_{m+1}]$ is given by $\frac{1}{C_{h_{m+1}}^{p_{m+1} - \sum_{1 \leq j \leq m} \frac{p_{m+1}}{p_j} \cdot h_j}}$. Since each of these $m+1$ flows in \mathcal{F} are independent, the probability of generating each feasible schedule with $m+1$ flows in \mathcal{F} is given by

$$\frac{1}{\left(C_{h_1}^{p_1}\right)^{\frac{p_{m+1}}{p_1}} \times \left(C_{h_2}^{p_2 - \frac{p_2}{p_1} \cdot h_1}\right)^{\frac{p_{m+1}}{p_2}} \times \dots \times \left(C_{h_{m+1}}^{p_{m+1} - \sum_{1 \leq j \leq m} \frac{p_{m+1}}{p_j} \cdot h_j}\right)}.$$

The above hypothesis holds for any number of flows $n \in \mathbb{N}$. Thus, it can be concluded that the *Offline_Sched_Gen* generates each feasible schedule with probability N_{sched}^{-1} for n harmonic flows over a single-channel WirelessHART network. \square

5.6.1 Discussion

From Theorem 5.2, it can be concluded that the *Offline_Sched_Gen* generates any feasible schedule with probability N_{sched}^{-1} for a set of n harmonic flows over a single-channel WirelessHART network. Since multiple runs of the *Offline_Sched_Gen* in the offline phase may result in generating the same schedule, the exact value of K or $|\sigma|$ for which the *Offline_Sched_Gen* needs to be run to encounter all the N_{sched} distinct feasible schedules with probability N_{sched}^{-1} is not known. To get an estimate of the value of $|\sigma|$, we consider hp random variables, X_1, X_2, \dots, X_{hp} corresponding to each of the hp slots in the hyperperiod schedules. Each X_i is defined as follows.

$$X_i = \begin{cases} i, j, & \text{if } F_j \text{ occurs in slot } i \text{ in any schedule} \\ 0, & \text{otherwise.} \end{cases}$$

Each distinct feasible schedule corresponds to a sample in our sample space. Therefore, the sample size in our case is $|N_{sched}|$. For a single-channel WirelessHART network, there are no conflicting transmissions among the flows in \mathcal{F} . As a result, a flow instance can occur at any slot inside its scheduling window. Since each flow F_i has period p_i and number of hops h_i , each flow F_i shows up at each slot s_t in the $|N_{sched}|$ distinct schedules with probability $\frac{h_i}{p_i}$. Hence, the sample mean is calculated as follows.

$\bar{X} = \frac{\sum_{i=1}^{hp} X_i}{|N_{sched}|} = \sum_{i=1}^{hp} i \times \sum_{j=1}^n \frac{h_j}{p_j} \times j$. Each run of the *Offline_Sched_Gen* generates a feasible schedule which corresponds to samples in our population space σ . At every

hyperperiod, the population mean is calculated over the existing population using the expression $\bar{\sigma} = \frac{\sum_{i=1}^{hp} X_i}{|\sigma|}$, where $|\sigma|$ denotes the size of our population space. From Theorem 5.2, it can be concluded that the *Offline_Sched_Gen* generates each feasible schedule with probability N_{sched}^{-1} . The online phase of *SlotSwapper* selects each schedule from σ uniformly at random. According to strong law of large numbers, the sample mean \bar{X} converges almost surely to the population mean $\bar{\sigma}$ when the population size tends to infinity [111], i.e., $\lim_{|\sigma| \rightarrow \infty} \bar{X} = \bar{\sigma}$. Thus, assuming a sufficiently large population size, σ , *SlotSwapper* can generate all possible feasible schedules and can execute each of them with equal probability for a single-channel WirelessHART network with harmonic flows.

Counterexample: *The Offline_Sched_Gen is not optimal for non-harmonic flows.*

Consider the network graph in Figure 5.1 with one channel and two flows F_1 and F_2 with $src_1 = 3$, $src_2 = 6$, $des_1 = des_2 = D$, $p_1 = 2$ slots and $p_2 = 3$ slots. Let us consider a schedule \mathcal{S}_1 as follows.

TABLE 5.2: A schedule over the network graph in Figure 5.1 for two flows F_1 (marked in red) and F_2 (marked in blue) with $src_1 = 3$, $src_2 = 6$, $des_1 = D$, $des_2 = D$, $p_1 = 2$ slots and $p_2 = 3$ slots.

Schedule	Slots					
	1	2	3	4	5	6
\mathcal{S}_1	$\mathbf{3 \rightarrow D}$ $\mathbf{F_1}$		$\mathbf{6 \rightarrow D}$ $\mathbf{F_2}$	$\mathbf{3 \rightarrow D}$ $\mathbf{F_1}$	$\mathbf{6 \rightarrow D}$ $\mathbf{F_2}$	$\mathbf{3 \rightarrow D}$ $\mathbf{F_1}$

Consider the scheduling window $[1, 3]$ corresponding to the first instance of F_2 . The first instance of F_2 is equally likely to occur at any slot $s_i \in [1, 3]$. An optimal algorithm would have scheduled F_2 at any slot s_i with probability $\frac{1}{3}$. According to the *Offline_Sched_Gen*, the eligible list corresponding to the first instance of F_2 is $[2, 3]$. This is because the first instance of F_2 at slot 3 cannot be swapped with slot 1 to preserve the deadline of F_1 . Hence, the probability of F_2 occurring at slot 1 is 0. Thus, some schedules are less probable to be generated by the *Offline_Sched_Gen* if the flows are non-harmonic.

5.7 Measure of Uncertainty

SlotSwapper, discussed in Section 5.5, generates a list of new feasible schedules, σ , to reduce the predictability of slots in the base schedules \mathcal{S}_B . The main objective is to make the schedules dynamic so that the predictability in the slots is reduced at every hyperperiod. More distinct the slots are in a schedule, more difficult it is for an attacker to predict them. The uncertainty of a flow occurring in a particular slot of a schedule comes from the random selection of flow instances from the available choices. Yoon *et al.* used *schedule entropy* as a measure of uncertainty of a given schedule for a uniprocessor system [20]. Samadder *et al.* re-defined *schedule entropy* as a function of the slot and channel entropy to measure the uncertainty in the generated schedules [32]. However, the calculation of schedule entropy involves joint probability distribution of the slots and channels in σ , which is exponential in nature if the length of the hyperperiod is long. Further, a multi-channel WirelessHART network consists of a maximum of 16 channels. If there are $n + 1$ flows where $n + 1 > 16$, there are P_{16}^{n+1} possible permutations in the calculation of the joint probability for each slot, which is also exponential in nature. Hence, *upper-approximate slot entropy* and *upper-approximate schedule entropy* are used by considering empirical probability distribution of flows across the slots and channels in the generated schedules, σ [32].

Although upper-approximate schedule entropy effectively captures the variation of slots in the generated schedules, σ , it cannot be used as a valid security metric. Consider two lists of schedules, σ_1 and σ_2 , with true schedule entropies, $\mathcal{H}(\sigma_1)$ and $\mathcal{H}(\sigma_2)$, and upper-approximate schedule entropies, $\widetilde{\mathcal{H}}(\sigma_1)$ and $\widetilde{\mathcal{H}}(\sigma_2)$ respectively. Calculation of true schedule entropy, $\mathcal{H}(\sigma_1)$ and $\mathcal{H}(\sigma_2)$, involves joint probability distribution of the slots and the channels in σ_1 and σ_2 respectively. On the other hand, calculation of upper-approximate schedule entropy, $\widetilde{\mathcal{H}}(\sigma_1)$ and $\widetilde{\mathcal{H}}(\sigma_2)$, involves empirical probability distribution of the slots and the channels in σ_1 and σ_2 respectively. The relative ordering of the joint probability distribution of the slots and the channels in the two lists of schedules, σ_1 and σ_2 , is not always preserved when their empirical probability distributions are considered. As a result, $\widetilde{\mathcal{H}}(\sigma_1) > \widetilde{\mathcal{H}}(\sigma_2)$, does not always imply $\mathcal{H}(\sigma_1) > \mathcal{H}(\sigma_2)$. Thus, *upper-approximate schedule entropy* cannot be used as a valid metric to measure the performance of our algorithm.

Schedule entropy is an absolute measure. Consider an algorithm \mathbb{A} that generates a list of feasible schedules σ_1 . Even if \mathbb{A} 's schedule entropy is high, \mathbb{A} may not be a truly random

Probability Mass Function	(slot,channel) pair					
	(1,1)	(1,2)	(2,1)	(2,2)	(3,1)	(3,2)
$\Pr(g_{ik} = 1)$	0.0215	0.0212	0.0225	0.0226	0.0222	0.0217
$\Pr(g'_{ik} = 1)$	0.0219	0.0219	0.0219	0.0219	0.0219	0.0219
$\Pr(g_{ik} = 2)$	0.0280	0.0285	0.0276	0.0275	0.0270	0.0279
$\Pr(g'_{ik} = 2)$	0.0281	0.0281	0.0281	0.0281	0.0270	0.0270
$\Pr(g_{ik} = 0)$	0.0337	0.0335	0.0330	0.0331	0.0340	0.0336
$\Pr(g'_{ik} = 0)$	0.0332	0.0332	0.0332	0.0332	0.0342	0.0342
$\Pr(g_{ik}) \log_2 \frac{\Pr(g_{ik})}{\Pr(g'_{ik})}$	0.000009	0.00002	0.00001	0.00002	0.000004	0.00002

Probability Mass Function	(slot,channel) pair					
	(4,1)	(4,2)	(5,1)	(5,2)	(6,1)	(6,2)
$\Pr(g_{ik} = 1)$	0.0228	0.0244	0.0211	0.0205	0.0146	0.0144
$\Pr(g'_{ik} = 1)$	0.236	0.0236	0.0210	0.0210	0.0144	0.0144
$\Pr(g_{ik} = 2)$	0.0292	0.0267	0.0312	0.0313	0.0244	0.0236
$\Pr(g'_{ik} = 2)$	0.0294	0.0294	0.0294	0.0294	0.0243	0.0243
$\Pr(g_{ik} = 0)$	0.0312	0.0321	0.0310	0.0314	0.0442	0.0452
$\Pr(g'_{ik} = 0)$	0.0301	0.0301	0.0327	0.0327	0.0445	0.0445
$\Pr(g_{ik}) \log_2 \frac{\Pr(g_{ik})}{\Pr(g'_{ik})}$	0.00005	0.00032	0.00013	0.00013	0.000004	0.00002
$\mathcal{KL}(\mathbb{A} \mathbb{A}') = \sum \Pr(g_{ik}) \log_2 \frac{\Pr(g_{ik})}{\Pr(g'_{ik})} = \mathbf{0.00077}$						

TABLE 5.3: Calculation of K-L divergence on running the *Offline_Sched_Gen* for 10,000 hyperperiods over the set of flows in Example 5.1 considering \mathcal{S}_1 of Figure. 5.2 as the initial base schedule with reference to \mathbb{A}' that generates all possible feasible schedules

algorithm. A truly random algorithm generates and executes all possible feasible schedules with equal probability. To compare \mathbb{A} to the truly random algorithm, one can use the probability distribution of the schedules generated by both. Given two probability distributions, p and q , *cross entropy* [94] is a well-known metric used in information theory that measures the average number of bits per message needed to encode events drawn from the true distribution p , if using an optimal code for the distribution q . However, cross entropy does not quantify how much one probability distribution diverges from another. Thus, we propose to use *Kullback-Leibler (K-L divergence)* or *relative entropy* [29] as a security metric to measure the performance of our algorithm in terms of its ability to randomize the slots of the generated schedules with reference to a truly random algorithm.

Consider our algorithm \mathbb{A} and a truly random algorithm \mathbb{A}' that generates and selects all

possible feasible schedules with equal probability. *K-L divergence* measures the divergence in the solution space of \mathbb{A} with reference to the solution space generated by \mathbb{A}' . Lower the value of K-L divergence, lesser is the divergence of the solution space of \mathbb{A} with reference to the solution space generated by \mathbb{A}' . Thus, lower the divergence values of \mathbb{A} with reference to \mathbb{A}' , more diverse are the slots in the generated schedules and hence less predictable are the schedules generated by our algorithm.

Definition 5.3. Given an algorithm \mathbb{A} , a truly random algorithm \mathbb{A}' and a set of $n + 1$ flows $\mathcal{F} = \{F_0, F_1, \dots, F_n\}$, (F_0 being the idle flow corresponding to the idle slots in the hyperperiod schedules with release time 0 and deadline hp) over a WirelessHART network with m channels, the **relative entropy** of \mathbb{A} with reference to \mathbb{A}' measures the divergence of the probability distribution of flows in \mathcal{F} across all the channels over all the slots in the hyperperiod schedules generated by \mathbb{A} with reference to \mathbb{A}' . It is represented as

$$\mathcal{KL}(\mathbb{A}||\mathbb{A}') = \sum_{i=1}^{hp} \sum_{k=1}^m \sum_{j=0}^n \Pr(g_{ik} = j) \log_2 \frac{\Pr(g_{ik} = j)}{\Pr(g'_{ik} = j)} \quad (5.3)$$

where g_{ik} and g'_{ik} are discrete random variables and $\Pr(g_{ik} = j)$ and $\Pr(g'_{ik} = j)$ are the probability mass functions of the j^{th} flow occurring in the k^{th} channel of the i^{th} slot in the hyperperiod schedules generated by \mathbb{A} and \mathbb{A}' , respectively.

The minimum value of $\mathcal{KL}(\mathbb{A}||\mathbb{A}')$ is 0, however there is no upper-bound of K-L divergence. A divergence value of 0 indicates the solution space is identical to the truly random solution space. Note that \mathbb{A}' considers all possible feasible schedules, *i.e.*, $\mathbb{A} \subseteq \mathbb{A}'$. However, computation of all possible feasible schedules for a very large hyperperiod is exponential in nature. Hence, we calculated an *upper-bound value of K-L divergence* for large hyperperiods in our experiments. We compare our algorithm \mathbb{A} with a hypothetical truly random algorithm \mathbb{A}'' . Note that, \mathbb{A}'' , cannot meet the various constraints that our solution satisfies, however. If there are h_j hops in a flow F_j with period p_j , then the number of flow instances n_{inst} in a hyperperiod schedule with m channels and hp slots is $n_{inst} = \frac{hp}{p_j}$. The total number of slots (n_j) in which F_j occurs in a hyperperiod schedule is given by : $n_j = n_{inst} \times h_j$. For a hypothetical truly random algorithm \mathbb{A}'' , the probability mass function of the j^{th} flow in the k^{th} channel of the i^{th} slot, ($\Pr(g''_{ik} = j)$), is given by

$$\Pr(g''_{ik} = j) = \frac{n_j}{hp} \times \frac{1}{hp \times m}. \quad (5.4)$$

where $\frac{n_j}{hp}$ is the probability mass function of a single slot which when multiplied by $\frac{1}{hp \times m}$ gives the probability mass function for all possible schedules with m channels and hp slots. Table 5.3 shows the calculation of K-L divergence on running the *Offline_Sched_Gen* (A) upto 10,000 hyperperiods. The K-L divergence calculates divergence of the *Offline_Sched_Gen* (A) with reference to a truly random algorithm (A') generating all possible feasible schedules over 6 slots and 2 channels for the set of flows in Example 5.1. To calculate K-L divergence of A with reference to A', we generate all possible permutation of flows and prune the solution space by removing those permutations which result in infeasible schedules.

K-L divergence measures the strength of our algorithm in terms of the divergence in the solution space of our algorithm with reference to a truly random algorithm. However, it does not give a measure of how strong the generated schedules are taking the attacker's success probability into consideration. Note that, the attacker's success probability depends on the amount of randomization that is feasible in a given network. If the amount of randomization feasible in a given network is low due to high slot utilization and conflicts in the network, the K-L divergence can still be small because less number of feasible schedules are possible to be generated by the truly random algorithm. Thus, small values of K-L divergence does not always imply low success probability of the attacker. However, in case of a WirelessHART network, several slots are reserved in the schedule for re-transmission to make the system reliable and robust. As a result, the slot utilization in a WirelessHART network is comparatively low [44]. Hence, it is reasonable to expect that the amount of schedule randomization feasible in such a network is high. According to Nasri *et al.*, a security metric must quantify the success of a defense mechanism against the attack being considered [107]. In this work, an attacker's success probability depends on how correctly it can predict a transmission over a specific slot depending on its observation over previous hyperperiod schedules. To measure this, we use *Prediction Probability (PP)* of slots of a schedule which measures the security provided by our algorithm in terms of the probability of correctly predicting the communication in each slot of a schedule based on the observations over previous hyperperiod schedules. To define Prediction Probability (PP) of a slot, we consider a random variable O_{jiet} to denote the observation of the j^{th} flow in the i^{th} slot via the e^{th} link in the t^{th} hyperperiod. $O_{jiet} = 1$, if j^{th} flow occurs in the i^{th} slot via the e^{th} link in the t^{th} hyperperiod. Otherwise, $O_{jiet} = 0$.

Definition 5.4. Consider observations over t hyperperiod schedules. Prediction Probability (PP) of the i^{th} slot in the t^{th} hyperperiod, denoted by $PP(s_{it})$, is the probability of

predicting the occurrence of the j^{th} flow in the i^{th} slot via link e in the t^{th} hyperperiod, from the observations of i^{th} slot over previous $t - 1$ hyperperiods. It is represented as

$$PP(s_{it}) = \frac{\sum_{t'=1}^{t-1} \{O_{jiet'} \mid O_{jiet} = 1\}}{t - 1} \quad (5.5)$$

5.8 Experiments and Evaluation

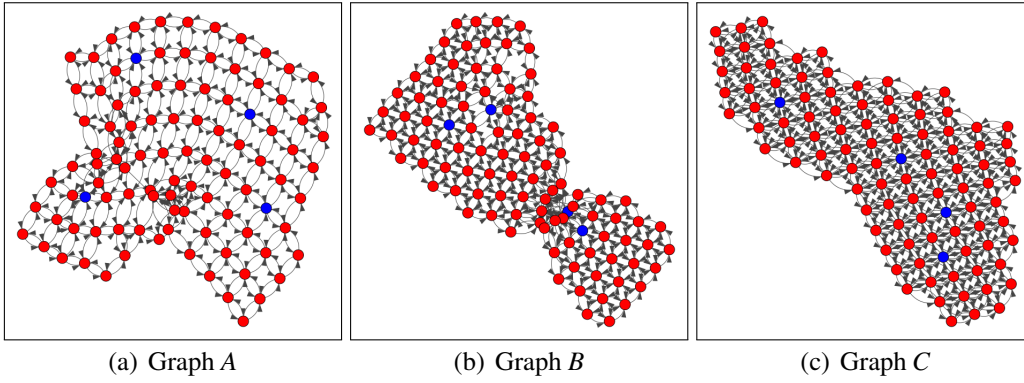


FIGURE 5.5: Three graphs with $|V|=100$ and (a) θ varying between 2 to 4, (b) θ varying between 3 to 6, (c) θ varying between 3 to 8

Metrics: The performance of our algorithm, the *SlotSwapper*, is measured by *upper-bound K-L divergence* and *Prediction Probability* of the slots in the generated schedules. Further, we provide some analysis on additional power and memory consumption on running our algorithm.

Simulation setup: We carried out extensive evaluation of our algorithm on Cooja simulator [30] of Contiki 3.0. We found that a typical WirelessHART network consists of approximately 80 devices [40, 39] under the control of one centralized network manager. However, as we increase the number of edges, the number of collisions and conflicts increase which affect the performance of our algorithm. We generated three random topologies on 100 simulated Tmote sky motes by varying the degree of nodes (θ). The degree of a node (θ) is the number of incoming and outgoing edges from a node in a graph. We considered a unit disk of radius 50m as the transmission range of each mote. Figure. 5.5 shows three random topologies — (a) Sparsely connected (Graph A with θ varying between 2 to 4), (b) Moderately connected (Graph B with θ varying between 3 to 6) and (c) Densely connected (Graph C with θ varying between 3 to 8). The nodes

TABLE 5.4: True K -L divergence for the graph in Figure. 5.1 with number of flows between 3 to 5

Number of flows	hyperperiod	True K-L divergence
3	10 slots	0.0404
4	10 slots	0.0739
5	10 slots	0.1175

with highest number of neighbors are considered to be the access points. The access points are marked in blue in each of the graphs in Figure. 5.5.

Flow Generation: A fraction (γ) percent of the nodes are randomly selected as the source and destination nodes. We varied (γ) upto 80% which implies 40% of the nodes are source and 40% are destination nodes which are disjoint. We selected the shortest path (number of hops) between the source and the destination node as the route of a flow. We selected the number of hops of each flow to be between 2 to 4 as the number of hops between the access point and the source or destination node is no more than 4 hops [39]. The flows have implicit-deadlines. The periods of the flows are harmonic and generated randomly in the range of 2^7 to 2^{10} slots [112]. We found that most of the flow sets are not schedulable for periods less than 2^7 slots.

5.8.1 Experiments

The *Offline_Sched_Gen* algorithm is written in Python and the experiments are run on a Linux machine with 3.4GHz Intel Xeon 12 processors and 16GB RAM. For the evaluation of our algorithm, we generated 4800 flow sets over the three synthetically generated topologies in Figure. 5.5 by varying the number of flows between 10 to 40 and the number of channels between 1 to 4. Each of these cases in our experiments is represented by 100 randomly generated flow sets with random periods, (range 2^7 to 2^{10}) over a hyperperiod of 2^{10} slots. We also evaluated our algorithm based on the slot utilization in the schedules. We generated three buckets with utilization varying between $[0.0, 0.3]$, $[0.3, 0.6]$ and $[0.6, 0.9]$. We generated 2100 flow sets over the random topologies in Figure. 5.5, by varying the utilization between $[0.0, 0.9]$ and the number of channels between 1 to 4 with 100 flow sets in each bucket. For each such flow sets, we generated the initial base schedule following the conditions of WirelessHART network [110]. We ran the *Offline_Sched_Gen* on each of these flow sets to generate a list of feasible schedules, σ , in offline mode. We continued the simulation upto 10,000

hyperperiods, to get enough confidence in the probabilities of each flow in the generated schedules. We calculated the *K-L divergence* to measure the divergence of our algorithm (\mathbb{A}) w.r.t. a truly random algorithm (\mathbb{A}'). Table 5.4 shows *true K-L divergence* of \mathbb{A} with reference to \mathbb{A}' over a hyperperiod of 10 slots for a single-channel WirelessHART network. We observed that as the length of the hyperperiod schedule is more than 12 slots, generation of all possible feasible schedules by \mathbb{A}' becomes non-scalable. So, we considered *upper-bound K-L divergence* to measure the maximum divergence of our algorithm (\mathbb{A}) with reference to a hypothetical truly random algorithm (\mathbb{A}'') in our experiments. We checked the feasibility of the generated schedules on the Cooja simulator [30] in Contiki 3.0 and also on the Indriya2 testbed [31]. We found that the memory and power consumption to run the pseudo random number generator in the online phase is negligible. In Section 5.8.4, we provide a discussion on the extra memory and power consumption in storing or broadcasting the K schedules in the network.

5.8.2 Evaluation using K-L divergence

Figure. 5.6 shows the mean value of *upper-bound K-L divergence* on 2100 flow sets for the three different topologies (shown in Figure. 5.5) by varying the slot utilization in a schedule and the number of channels. Each point in the plot is the average of 100 runs of the *Offline_Sched_Gen* on different flow sets. In addition, Figures. 5.7(a), 5.7(b), 5.7(c) show the distribution of upper-bound K-L divergence on 4800 randomly generated flow sets, by varying the number of flows and number of channels (m) on the three topologies.

From Figure. 5.6, it can be concluded that for a fixed slot utilization or for a fixed number of flows, the mean upper-bound K-L divergence is minimum for a single-channel WirelessHART network ($m = 1$). This is because, for a single-channel network, there are no conflicts among the flows. Thus, a flow instance can occur at any slot within its release time and deadline which results in more variations in the slots of a schedule. More variations in the slots imply more number of feasible schedules, which imply lesser divergence of our solution space from a truly random solution. From Figure. 5.6, it can be observed that the mean upper-bound K-L divergence increases with increase in m from 1 to 2 for a fixed utilization. With increase in m , the hypothetical truly random algorithm generates more number of schedules due to increase in the number of available locations in the schedule. To keep utilization fixed, the number of flows increase, which in turn, increase the number of conflicts in the network and hence reduce the number of

feasible schedules. Thus, mean divergence increases. However, as we increase m further from 2 to 4 keeping utilization constant, the mean divergence decreases. This is because, the number of schedules generated by the hypothetical truly random algorithm increase. However, to maintain fixed utilization, the number of flows also increase which result in more variations in the slots of a schedule. Note that this is true at lower utilization (typically for utilization below 0.6) when θ is typically between 2 to 6. For a dense graph, the number of transmission conflicts increase as slot utilization increases which in turn provides less scope to randomize. Note that in Figure. 5.6, we did not find any flow set for Graph C that can generate any feasible schedule for utilization greater than 0.6. Due to very high number of edges, the number of conflicts in Graph C is very high. As a result, no feasible schedule with utilization greater than 0.6 exists for Graph C. The number of edges in Graph B is higher than that of Graph A, resulting in more conflicts in Graph B. Hence, the mean divergence for Graph B is slightly higher than Graph A at higher utilization (> 0.6) and for $m = 3$ and $m = 4$ (refer Figure. 5.6).

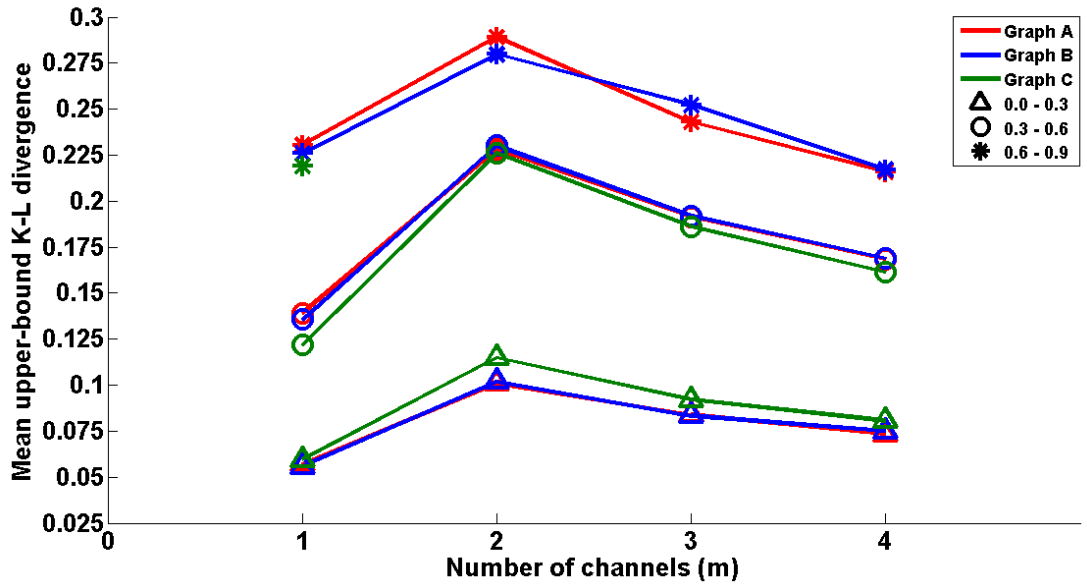
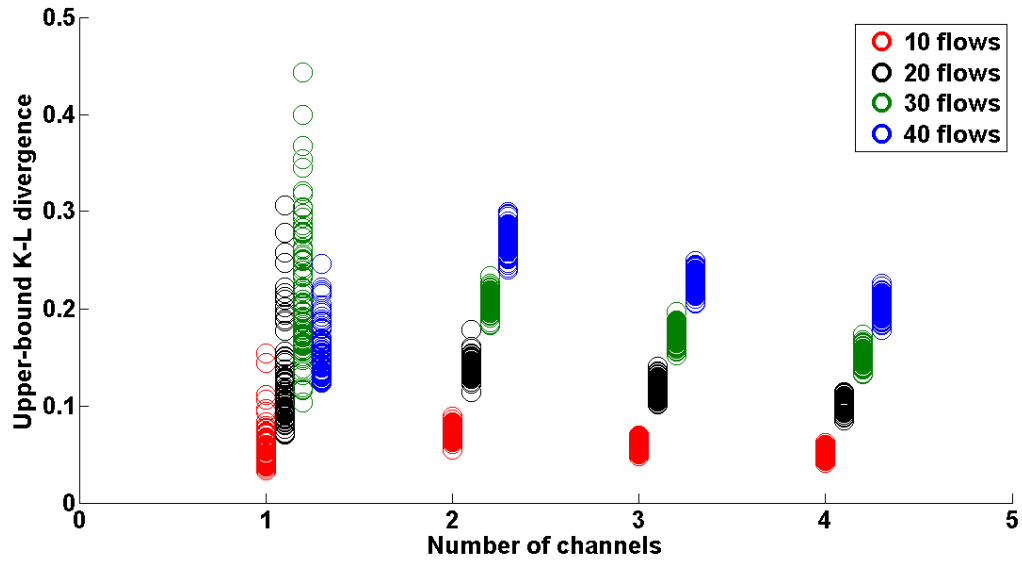
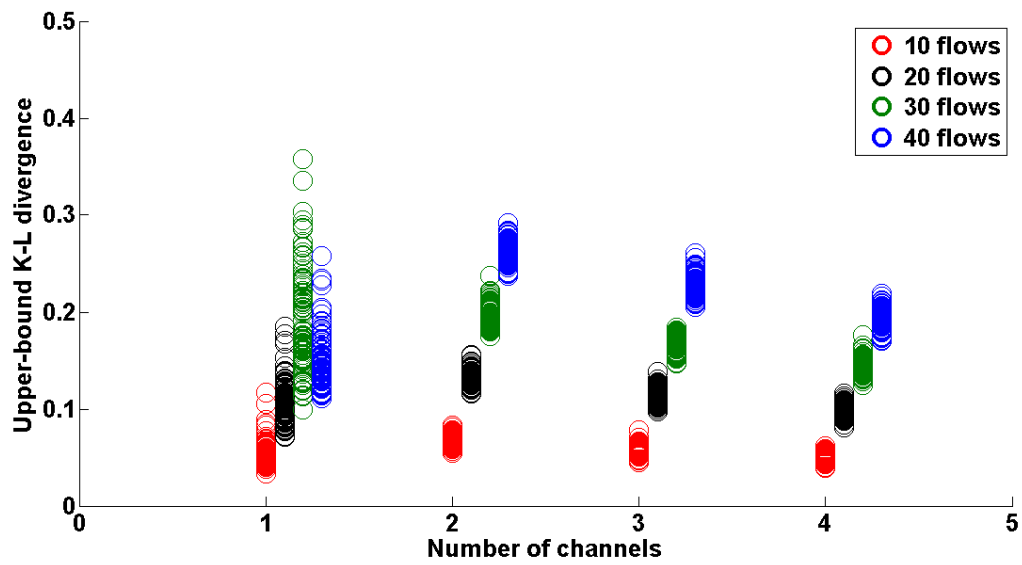


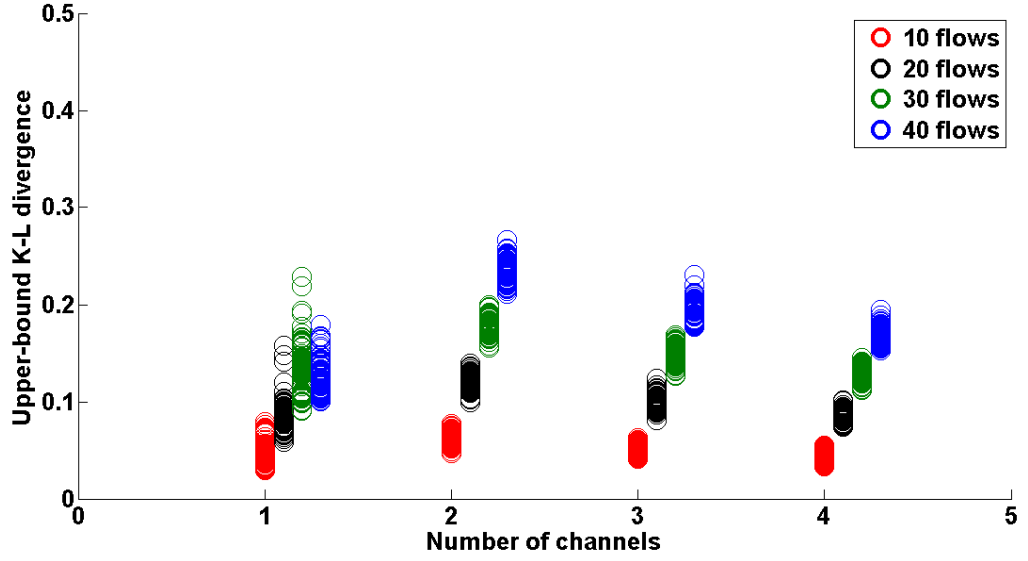
FIGURE 5.6: Mean upper-bound K-L divergence over Graph A, Graph B and Graph C with slot utilization varying between 0.0 to 0.9 and number of channels between 1 to 4



(a) Upper-bound K-L divergence over Graph A



(b) Upper-bound K-L divergence over Graph B



(c) Upper-bound K-L divergence over Graph C

FIGURE 5.7: Upper-bound K-L divergence over (a) Graph A, (b) Graph B and (c) Graph C, with number of flows varying between 10 to 40 and number of channels between 1 to 4 over a hyperperiod of 2^{10} slots

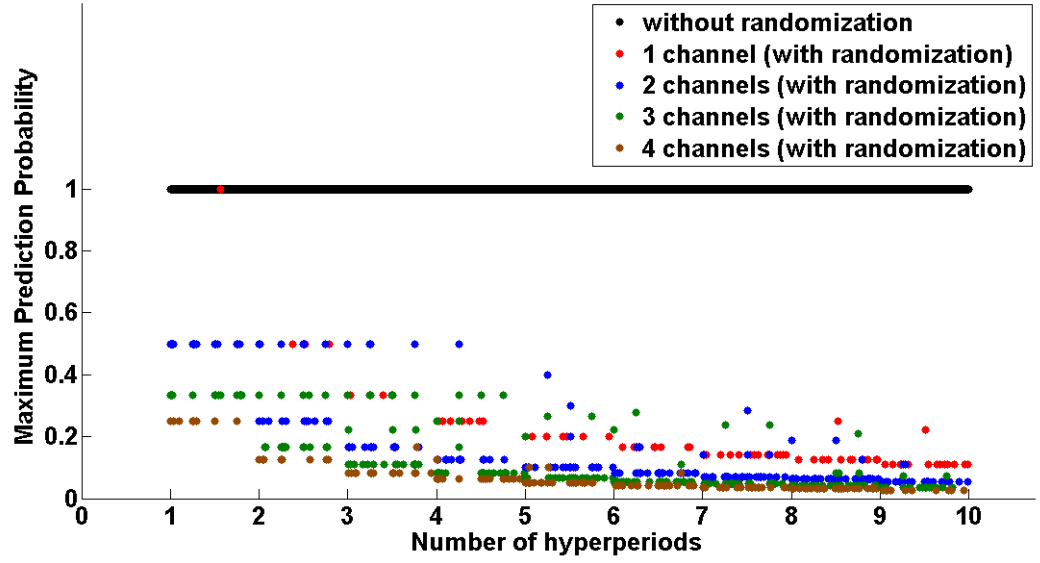
From Figures. 5.7(a), 5.7(b), 5.7(c), it can be observed that for a fixed number of flows, the divergence increases with increase in the number of channels (m) from 1 to 2. This is because, with increase in m , the number of available locations in a schedule increase. However, the variation in the slots of the schedules does not increase parallelly due to conflicting transmissions. Hence, K-L divergence increases. However, from Figures. 5.7(a), 5.7(b), 5.7(c), it can be observed that if we keep the number of flows fixed, the divergence decreases as we increase the number of channels between 2 to 4. This is because, with increase in the number of channels, the number of conflicts remain the same, however, the parallelism in the schedules increases at a higher rate compared to number of conflicts in the network. Hence, number of feasible schedules increase, which result in decrease in divergence. From Figures. 5.7(a), 5.7(b), 5.7(c), as the number of flows increase keeping m fixed, the number of schedules generated by a hypothetical truly random algorithm increase. However, due to conflicting transmissions, the number of feasible schedules generated is much lower compared to that of the hypothetical truly random algorithm. Hence, upper-bound K-L divergence increases.

TestBed Setup : We tested the feasibility of the generated schedules on the Indriya2 testbed [31] over 100 random topologies generated over 74 TelosB motes with 4 channels. Based on the connectivity and positions of the nodes, we assigned three access

points in our network. We selected the routes of the flows based on the reliability of a link which is measured in terms of Packet Reception Ratio (PRR) of that link [113]. In the testbed, we were able to generate a maximum of 28 random flows over a hyperperiod of 1000 slots.

5.8.3 Security Analysis

We collected the communication schedules generated by *SlotSwapper* over 10 hyperperiods, *i.e.*, 10240 slots with 40 flows varying the number of channels from 1 to 4. We calculated the *Prediction Probability (PP)* of slots in each of these hyperperiod schedules based on the observations in the previous hyperperiod schedules according to Equation (5.5). Figure 5.8 shows the Maximum Prediction Probability of slots in each of these 10 hyperperiods by varying the number of channels from 1 to 4. Table 5.5 shows the number of slots with $PP = 0$ over different number of channels. We observed that the PP is 0 for the first hyperperiod for both the cases (with and without randomization) as we do not have any observation. Hence, we have not considered the PP values over the first hyperperiod in Figure 5.8 and Table 5.5. However, from the second hyperperiod onwards, the same schedule is repeated over every hyperperiod without randomization. As a result PP is 1 from the second hyperperiod onwards for schedules with no randomization. Upon executing *SlotSwapper* over every hyperperiod, the schedules generated have most of their PP values equal to 0. From Table 5.5, it can be said that with increase in the number of channels, the number of mispredicted (slot,channel) pairs increase due to more variations in the observed schedules. For a fixed number of channel, as we increase the number of hyperperiods, the maximum PP decreases with increase in the length observation period (number of hyperperiods) due to variation in the (slot,channel) pairs caused by execution of different schedules in each hyperperiod (refer to the plots in Figure 5.8). Also, for a fixed observation period, as we increase the number of channels, the maximum PP decreases due to more variations in the slots of the schedules. In Figure 5.8, for a fixed observation period, the maximum PP for four channels is minimum (plots in brown).



(a)

FIGURE 5.8: Maximum Prediction Probability of slots over 10 hyperperiod schedules each with 1024 slots over Graph C with 40 flows and m varying between 1 to 4.

Number of channels (m)	Number of the (slot,channel) pairs with Prediction Probability = 0	Total number of (slot,channel) pairs in the observed hyperperiods
1	9143	10240
2	18170	20480
3	27394	30720
4	36690	40960

TABLE 5.5: Number of slots with Maximum Prediction Probability = 0 over 10 hyperperiods each with 1024 slots over Graph C with 40 flows, and m varying between 1 to 4

5.8.4 Memory and Power Consumption

To maintain high variation in the slots, the number of hyperperiod schedules K should be large. According to the datasheet specification of Tmote sky mote [114], each mote is equipped with 10k RAM and 48k external Flash memory which is not sufficient to store K hyperperiod schedules if K is large. We experimentally found that we were able to store upto 10 hyperperiod schedules, *i.e.*, $K_{node} = 10$, where each schedule consists of 2^{10} slots with a maximum usage of upto 58 slots per node per hyperperiod. We found that to store the remaining 9990 schedules, we need additional 1.88MB of memory. We

suggest to select 10 schedules randomly from σ and store them in RAM and provide an external flash memory [115] of 2MB to each node. All the nodes in the network can replenish these 10 schedules periodically after some fixed number of hyperperiods by dynamically loading new schedules from the external memory during idle slots.

An alternative approach to replenish the schedules is to transmit a schedule using a fraction of the idle slots in every hyperperiod. Corresponding to each slot in a schedule, we need to store the sender node, the receiver node, the channel and the slot in which the communication is scheduled. WirelessHART operates at a data rate of 250kbps per channel and supports a maximum payload size of 125 bytes of data [116] before the High Level Data Link Control (HDLC) encoding. The total number of bits required to encode one transmission for a realistic WirelessHART setup with 100 nodes, 4 channels over a hyperperiod of 2^{10} slots is given by: (sender) + (receiver) + (channel) + (time-slot) = $(7 + 7 + 2 + 10)$ bits = 26 bits. To encode a schedule of 2^{10} slots, we require a maximum of 27 packets. We found that a maximum of 648 broadcasts are required to transmit one hyperperiod schedule across all the nodes in our randomly generated topologies. We suggest to use 1% of the slots which are idle to transmit a schedule (*i.e.*, approximately, 10 slots every hyperperiod). To transmit 648 broadcasts, we require 65 hyperperiod schedules. Hence, a schedule gets replenished every 65 hyperperiods. We measured the additional power consumption on broadcasting the schedules based on the power-trace collected from the Cooja simulator [30]. According to the operating current and voltage collected from the datasheet specifications of Tmote sky mote [114], we found that the extra power consumed in each broadcast is approximately 1.04mW. From the experiments, we found that 10 broadcasts every hyperperiod accounts for approximately 0.07% increase in the total power consumption every hyperperiod.

5.9 Conclusion

In this chapter, we proposed a schedule randomization strategy to mitigate timing attacks in ICSs. Our proposed schedule randomization strategy, the *SlotSwapper*, randomizes the slots and channels in the schedules over every hyperperiod without violating the hard deadlines and the feasibility constraints of the real-time flows in ICSs. We have shown that the *SlotSwapper* is optimal for a single-channel WirelessHART network with harmonic flows. We used Kullback-Leibler divergence as a metric to measure divergence in the schedules generated by our algorithm with reference to a truly random algorithm.

We also used the Prediction Probability of slots to measure the security provided by the *SlotSwapper* in terms of the probability of predicting a particular communication in a schedule based on previous observations. We illustrated the feasibility of the schedules on a real testbed with 74 TelosB motes and on simulated networks in Cooja with 100 Tmote sky motes.

There are some issues with the proposed schedule randomization strategy that are yet to be addressed. The proposed strategy, the *SlotSwapper*, is a centralized schedule randomization strategy where the generated schedules are static in nature. As a result, any change in the network topology due to node failure or due to change in the routes will result in recomputation of the schedules by the network manager. Further, this strategy involves high power consumption in distributing the schedules to the nodes. Hence, a decentralized schedule randomization strategy that can ensure all the constraints of a feasible schedule while still flexible to adapt with the changes in the network topology is needed. We propose a distributed schedule randomization strategy for closed-loop controls in ICSs as an extension of this problem in Chapter 6 of this thesis.

Chapter 6

Online Distributed Schedule Randomization with Period Adaptation in Industrial Control Systems

In this chapter, we propose a distributed online schedule randomization strategy, to mitigate timing attacks in industrial control systems (ICSs). In Chapter 5, we already presented a centralized schedule randomization strategy to mitigate such attacks. However, the centralized schedule randomization strategy consumes a high amount of energy to broadcast the schedules periodically. Since, the centralized schedule randomization strategy generates the schedules offline, any change in the network topology or any change in the route of a flow results in recomputation of the schedules at the network manager. Hence, to address the above problems, we present an online distributed schedule randomization strategy that randomizes the slots over every hyperperiod while still guaranteeing the stability of the closed-loop controls and the hard deadlines of the real-time flows in ICSs. Further, to increase the scope of randomization and to reduce the energy consumption of the system, we incorporate a period adaptation strategy that adjusts the periods of transmission of the flows depending on the stability of closed-loop controls.

We briefly present the vulnerability of the scheduling policy in a WirelessHART network in Section 6.1. We present some related research on wireless networked control systems (WNCSs) in Section 6.2. Section 6.3 describes the system model and Section 6.4 briefly describes the threat. Section 6.5 presents the online distributed schedule randomization

strategy as a countermeasure against timing attacks in ICSs. In Section 6.6.4, we present the period adaptation strategy and Section 6.6 evaluates the performance of our proposed countermeasure. Finally, Section 6.7 concludes the chapter.

6.1 Introduction

In ICS, most of the real-time applications consist of closed-loop controls that operate in a feedback mechanism. Each of these applications is associated with periodic real-time flows with a hard deadline. Since, multiple flows share the same wireless sensor actuator network (WSAN), each flow experiences some communication delay which introduces jitter at the controller. Hence, the period assigned to each real-time flow in ICS should ensure closed-loop stability even with the worst-case delay [117].

In a WirelessHART network, the centralized network manager computes the communication schedule during network initialization and repeats the same schedule over every hyperperiod so that the hard deadlines of the real-time flows are satisfied and the stability of the closed-loop controls are guaranteed. However, the repetitive nature of the communication schedule over every hyperperiod can be exploited by an attacker to launch timing attacks, specifically selective jamming attacks. By eavesdropping and analyzing the communications associated with a specific target device, the attacker can infer the communication schedule. Thereafter, using the inferred schedule in the subsequent hyperperiods, the attacker can selectively jam specific transmissions from a specific sensor or actuator and make the system unstable, eventually disrupting the safety of the system.

In Chapter 5, we presented a centralized schedule randomization strategy, the *SlotSwapper*, for WirelessHART network as a countermeasure against such selective jamming attacks. However, *SlotSwapper* has certain drawbacks. The schedules generated by *SlotSwapper* are static and generated offline at the centralized network manager. The network manager needs to broadcast the generated schedules periodically to all the nodes in the network which consumes a large amount of energy. Since the generated schedules are static in nature, any change in the network topology results in recomputation of the schedules at the network manager. Further, *SlotSwapper* does not consider the interaction between closed-loop controls and network flows. It only considers the real-time flows with fixed deadlines as the flow sets without accounting for the stability of the closed-loop controls. To overcome these drawbacks of *SlotSwapper*, we propose an

online dynamic distributed schedule randomization strategy, the *DistSlotShuffler*, as a countermeasure against timing attacks, such as selective jamming attacks. The flow sets considered by *DistSlotShuffler* model the interactions between closed-loop controls and the corresponding flows. The main objective of *DistSlotShuffler* is to reduce the predictability of slots in the communication schedules by randomizing the slots at runtime in a distributed manner without violating the deadlines of the real-time flows in the network, while still maintaining the stability of each control loop in the system.

Cervin *et al.* has presented that the periods of the flows can be adjusted depending on the stability of the closed-loop controls [117]. A closed-loop wireless control system requires smaller periods to guarantee stability under worst-case conditions. However, the number of slots available to each flow instance is less if the flows have smaller periods. Smaller periods result in smaller number of slots available to each flow which in turn, reduce the scope to randomize the slots in the schedules. On the other hand, larger the periods of the flows are, more is the number of slots available to each flow instance and better is the scope of randomization. Moreover, it has been observed that although smaller periods are required to achieve stability under worst-case scenarios because of more frequent transmissions, the same periods result in unnecessary transmissions when the system is stable under normal conditions, thereby, increasing the power consumption of the system. Hence, we use a period adaptation strategy that adjusts the periods of the flows at runtime depending on the stability of the closed-loop controls, thereby also increasing the scope of randomization of the slots in the hyperperiod schedules.

The period adaptation strategy in the context of wireless networked-control systems (WNCS) has been explored [59, 118]. However, these works do not consider the security aspects of the system. Existing works on schedule randomization apply randomization either on uniprocessor platform [20, 90] or in WSN [27, 28]. However, none of these works consider the interactions between closed-loop controls and the flows. Our work applies schedule randomization technique on closed-loop controls to reduce the predictability of slots in the communication schedules without affecting the stability of the system, and incorporates a period adaptation strategy depending on the stability of the system to achieve better randomization. In addition, it also reduces the power consumption of the system by reducing the number of packet transmissions under normal conditions.

6.2 Related works on wireless networked-control systems

In WNCS, any change in the wireless network has a direct impact on the control performance. Designing a resilient and efficient wireless control system has been well studied in the fields of control theory, wireless networks, and recently in the fields of network control co-designs. In the field of control theory, Kalman filters are used as state observers to handle the packet loss and communication latency in WNCS [119]. Control in WNCS can be aperiodic or periodic in nature. Aperiodic controls are generally adopted to minimize the communication cost. Aperiodic controls can be either event-triggered control systems [120, 121] or self-triggered control systems [122]. However, implementing aperiodic communication triggered by aperiodic controls in time division multiple access (TDMA) based multi-hop networks is challenging. Recent works on wireless network control co-design optimizes the network and control at design time. Sampling the rate of control to optimize the control performance under various network protocols and system settings are well explored in the literature [123, 124]. However, all of these works focus on optimization of the control performance. None of these works consider attacks and the required countermeasures to ensure safety of the system.

Demiral *et al.* suggested a packet forwarding policy in wireless controls under unreliable energy constrained WSN [125]. Kim *et al.* proposed control over IEEE 802.11 networks [126], while Saifullah *et al.* optimized the sampling rates to meet end-to-end deadlines of the data flows in wireless mesh networks [127]. Li *et al.* applied asymmetric routing to ensure enhanced control performance and network efficiency due to application of different routing strategies in the network [128]. A holistic controller design that can interact with the network manager in networked-control system is needed to make the system evolve with the dynamics of the physical system [59, 118]. Dynamic stability can be guaranteed under different network configurations such as delay in the packet delivery or packet losses in the networks [129, 130, 131, 132]. Existing works focus on the trade-offs between latency, reliability, energy consumption of the system and the overall system performance in wireless control systems [133, 125, 134]. Bernardini *et al.* proposed a communication strategy that minimizes the data exchange in wireless networks [135]. A few notable works on guaranteeing stability under specific scheduling and control setups include [133, 136, 137]. However, all of these works develop new network protocols and none of these protocols are resilient to attacks. Hence, they are not suitable for our system. Recent works on network control co-design to lower the energy consumption of the system also exist in the literature [59, 118, 138]. These works

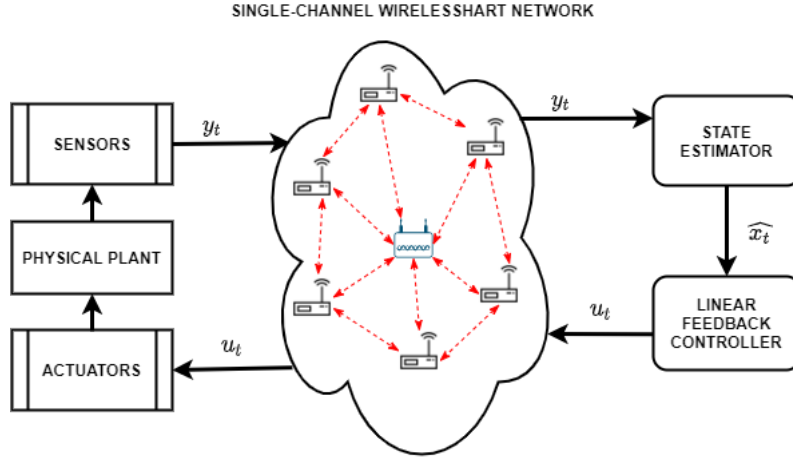


FIGURE 6.1: A wireless control system architecture consisting of a physical plant, a linear feedback controller and a single-channel WirelessHART network

propose two control strategies—rate adaptation and self-triggered control that lowers the energy consumption of the system as well as maintains the stability of the system. In this chapter, we use a period adaptation strategy that is similar to this rate adaptation strategy [59, 118].

6.3 System Model

In this chapter, we consider a WNCS. Our system model consists of the following components—(a) a linear time-invariant (LTI) physical plant model (b) a linear feedback controller (c) a single-channel WirelessHART network. Figure. 6.1 shows an overview of our system.

The LTI physical plant model can be characterized by the following sets of difference equations:

$$x_{t+1} = Ax_t + Bu_t; y_t = Cx_t + Du_t \quad (6.1)$$

where $t \in \mathbb{N}$ is the time instant, $x_t \in \mathbb{R}_a$ denotes the state vector, $u_t \in \mathbb{R}_b$ denotes the control input vector, $y_t \in \mathbb{R}_c$ denotes the output or the measurement vector. $A \in \mathbb{R}_{a \times a}$ and $B \in \mathbb{R}_{a \times b}$ denote the state matrices of the physical plant. $C \in \mathbb{R}_{c \times a}$ and $D \in \mathbb{R}_{c \times b}$ denote the output matrices. The matrix D is typically a null matrix in most physical systems. We assume that the dynamic system is both controllable and observable [139].

State matrices pair (A, B) is assumed to be controllable and (A, C) is assumed to be observable [139].

We consider a linear state feedback controller characterized by the equation, $u_t = Kx_t$, where K is the gain vector. The controller runs periodically and makes the closed-loop system asymptotically stable. At any time instant t , the sensor data y_t is transmitted to the gateway via the multi-hop large-scale WirelessHART network. A Kalman filter residing at the gateway, acts as the state observer [119], estimates the states of the plant and generates the estimated state vector \hat{x}_t from y_t .

Kalman filter [140] consists of two stages: “prediction stage” and “update stage”. The predicted state estimate is given by—

$$\hat{x}_t^{Pr} = S_f \hat{x}_{t-1} + B u_{t-1} \quad (6.2)$$

where S_f is the state transition matrix applied to the previous estimated state vector \hat{x}_{t-1} , B is the control input matrix applied to the control input vector u_{t-1} . The predicted error covariance is given by—

$$E_t^- = S_f E_{t-1}^+ S_f^T + \mathcal{G} \quad (6.3)$$

where E_t^- and E_{t-1}^+ are the prior and posterior state error covariances at time step t and $t - 1$ respectively. \mathcal{G} is the zero-mean Gaussian noise with covariance Q_G .

In the update stage, measurement residual ∇y_t is computed as—

$$\nabla y_t = y_t - H \hat{x}_t^{Pr} \quad (6.4)$$

where H is the measurement matrix. The Kalman gain K_{gain} is given by—

$$K_{gain} = E_t^- H^T (N_R + H E_t^- H^T)^{-1} \quad (6.5)$$

where N_R is the measurement noise covariance matrix. The updated state estimate is given by—

$$\hat{x}_t = \hat{x}_t^{Pr} + K_{gain} \nabla y_t \quad (6.6)$$

The Kalman filter calculates the updated error covariance E_t^+ which will be used in the next time step and is given by—

$$E_t^+ = (I - K_{gain}H)E_t^- \quad (6.7)$$

The remote controller, residing at the gateway, generates the control input $u_t = K\hat{x}_t$ based on the estimated state vector \hat{x}_t [119]. The control input u_t needs to be transmitted to the actuators via the WirelessHART network within a strict deadline to update the state of the physical plant. If any sensor measurements fail to reach the gateway, then the Kalman filter uses the output of the previous time step y_{t-1} in Equation (6.4). Similarly, in case of a transmission failure at the actuators, the actuator reuses the control input of the previous period u_{t-1} .

The stability of our system is determined by the Lyapunov function [141]. Our system, denoted by Eq. (6.1), is said to be stable, if there exists a positive definite Lyapunov function,

$$V(x_t) = x_t^T \mathcal{P} x_t \quad (6.8)$$

such that

$$V(x_{t+1}) - V(x_t) = x_t^T ((A + BK)^T \mathcal{P} (A + BK) - \mathcal{P}) x_t = -x_t^T \mathcal{Q} x_t \quad (6.9)$$

where \mathcal{P} and \mathcal{Q} are positive definite matrices. P satisfies the discrete-time Lyapunov equation

$$(A + BK)^T \mathcal{P} (A + BK) - \mathcal{P} = -\mathcal{Q} \quad (6.10)$$

Our wireless control system consists of a set of n real-time flows as introduced in Section 2.1 in Chapter 2 and M independent control loops $L = \{L_1, L_2, \dots, L_M\}$. The n real-time flows can be represented as $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$ over a single-channel WirelessHART network graph $G = (V, E)$. The notations for the flow parameters, src_i , des_i , p_i , δ_i and $route_i$, have the same semantics as discussed in Section 2.2.2. Each of these n flows is associated with any one of the M control loops in L . We use the notation \mathcal{F}^k to represent the set of flows associated with the k^{th} control loop. Since, a flow F_i can be associated with only one control loop, therefore, $n = \sum_{k=1}^M |\mathcal{F}^k|$ and if $F_i \in \mathcal{F}^k$, then

$F_i \notin \mathcal{F}^j$, where $|\mathcal{F}^k|$ denotes the cardinality of the set of flow in \mathcal{F}^k , \mathcal{F}^k and \mathcal{F}^j are the set of flows associated with L_k and L_j respectively, $\forall k, j \in [1, M] \mid k \neq j$.

In a typical ICS, the physical plant operates at a very high rate. However, the controllers and the real-time flows in the wireless network operate at a lower rate due to computation and communication latencies [59]. Each control loop $L_k \in L$ is associated with a rate R_k which is the rate at which the remote controller runs the control algorithms to generate proper control signals and updates the state of the physical plant. Hence, we assume that all flows in \mathcal{F}^k that are associated with the k^{th} control loop, L_k , have the same period which is equal to $\frac{1}{R_k}$, where R_k is the rate associated with L_k . In an ICS, there can be multiple control loops operating at the same rate. We represent all the flows in \mathcal{F} that have the same period and same scheduling window for each of their instances as a cluster.

We define a cluster as follows.

Definition 6.1. All flows $F_i \in \mathcal{F}$ having the same period p_i (this may apply to flows that belong to different control loops but have the same period as p_i) and the same scheduling window W_{ij} for all the j instances can be represented as a cluster \mathcal{C} . Each cluster \mathcal{C} is associated with a set of flows $\mathcal{C}.\mathcal{F}$, a list of slots, $\mathcal{C}.s$, a sequence of flow transmissions, $\mathcal{C}.FSeq$, a pseudo random number generator (PRNG), $\mathcal{C}.prng$ [108] and a period $\mathcal{C}.p$ which is equal to the periods of all the flows in $\mathcal{C}.\mathcal{F}$.

We assume that there is no spatial re-use of channels in the network. Since, we consider a single-channel WirelessHART network, there is only one location (channel) per slot to schedule a flow. Hence, the first condition of a feasible schedule *i.e.*, there should not be any conflicting transmission in a schedule (refer to Definition 2.5 in Section 2.2.2 of Chapter 2) is not applicable for a single-channel WirelessHART network. Therefore, a *feasible schedule* should satisfy condition 2, condition 3 and condition 4, *i.e.*, no collision, no deadline violation and flow sequence preservation of Definition 2.5 in our system.

6.4 Threat Model

In this chapter, we consider the same attacker as presented in Chapter 4, 4.2. The assumptions of the attacker are discussed in Section 4.1 and the capabilities of the attacker are discussed in Section 4.2. Using the capabilities, as in Section 4.2, the attacker launches the attack in two phases. Example 4.1 in Chapter 4 illustrates the attack.

6.5 Proposed Distributed Schedule Randomization Technique

We propose an *online distributed schedule randomization technique*, the *DistSlotShuffler*, as a countermeasure against timing attack. *DistSlotShuffler* aims at reducing the predictability of the slots in the schedules while still satisfying the flow deadlines in the system. *DistSlotShuffler* considers a base schedule \mathcal{S}_B with M control loops and $n + 1$ flows, F_0 being the idle flow corresponding to the idle slots in \mathcal{S}_B , over a single-channel WirelessHART network graph G and generates randomized partial schedules online at each node in a distributed manner. *DistSlotShuffler* consists of two phases —

1. The **Cluster Generation Phase**, an offline pre-processing phase that runs on the network manager at the time of network initialization to create a set of \mathcal{N} clusters, κ , and a set of keys of varying lengths, *Key*.
2. The **Online Distributed Schedule Generation Phase**, that runs *online* to generate partial schedules at each network device. If a network device is a part of a flow that belongs to the j^{th} cluster, $\kappa[j]$, then the partial schedule generated at that network device corresponding to $\kappa[j]$ is denoted by ω_j and that device runs the online phase once every $\kappa[j].p$ ms. If a device is part of multiple flows belonging to different clusters, then the device needs to run the online phase for each of these clusters. The total number of distinct partial schedules is equal to the total number of clusters in κ . The hyperperiod schedule \mathcal{S} is the union of all the partial schedules generated by each cluster over all scheduling windows, *i.e.*, $\mathcal{S} = \bigcup_{j=1}^{\mathcal{N}} \omega_j^t \cdot \omega_j^{t+1} \forall t \in [1, \frac{hp}{\kappa[j].p})$, where ω_j^t denotes the partial schedule generated by the j^{th} cluster over t^{th} scheduling window.

TABLE 6.1: List of notations used by *DistSlotShuffler*

M	number of control loops
hp	hyperperiod of the schedule
\mathcal{F}^k	set of flows associated with the k^{th} control loop
\mathcal{S}_B	base schedule over a network graph G with n flows and hyperperiod hp
\mathcal{C}	a cluster
κ	set of clusters
$dist_period$	set to store distinct periods of flows in the system
ω_j	partial schedule generated by the nodes associated with the j^{th} cluster
R_k	rate of the controller
$Allocated$	list to store the allocated slots in \mathcal{S}_B
$Free$	list to store the idle slots in \mathcal{S}_B
Key	a set of randomly generated keys of varying lengths
$Member_x$	a set of clusters associated with node V_x , $V_x \in G.V$ along with the index of that cluster in κ
W_t	t^{th} scheduling window of all flow instances in a cluster

Algorithm 6 shows the basic steps of *DistSlotShuffler*. Table 6.1 contains the notations used by our algorithms.

Algorithm 6: *DistSlotShuffler*($n, \mathcal{F}, G, \mathcal{S}_B$)

```

// Offline Cluster Generator running at the network
manager
1 ( $\kappa, Key$ ) = Gen_Cluster( $\mathcal{F} \cup \{F_0\}, n, \mathcal{S}_B$ );
// Online partial schedule generator at each node
 $V_x \in G.V$ 
2 for each  $j : \mathcal{C} \in Member_x$  &&  $\kappa[j] == \mathcal{C}$  do
3    $\omega_j = Gen\_Partial\_Sched(\kappa[j]);$ 

```

6.5.1 Offline Cluster Generation Phase

The *Gen_Cluster* algorithm runs on the network manager during network initialization. It considers an initial base schedule \mathcal{S}_B over a set of $n + 1$ flows, F_0 being the idle flow, and generates a set of clusters, κ . Algorithm 7 presents the Cluster Generation algorithm. Two lists *Allocated* and *Free* are generated to store the utilized slots and idle slots in the base schedule \mathcal{S}_B respectively (lines 2-7). A set known as the *dist_period* is created to store all the distinct periods of flows in \mathcal{F} (lines 8-10). For each entry $p_i \in dist_period$, a new cluster \mathcal{C} is initialized and p_i is assigned to cluster \mathcal{C} (line 13). A PRNG *prng*, initialized with a seed is also assigned to \mathcal{C} (line 14). Note that, the PRNG is assumed

to be secure and not accessible to the attacker. The PRNG ensures that all the nodes belonging to the same cluster generate the same random number in a distributed manner. Other parameters of cluster \mathcal{C} are initialized and \mathcal{C} is added to the list κ (lines 11-16).

Note that all the flows with the same period and same scheduling window for each of its instances are assigned to the same cluster. All slots and transmissions associated with flow F_i in \mathcal{S}_B are assigned to F_i 's allocated cluster, where $1 \leq i \leq n$ (line 19-24). Slot utilization of each cluster is calculated. Slot utilization of the j^{th} cluster, $\kappa[j]$, is denoted as the ratio of the number of slots assigned to the flows in $\kappa[j]$ to the total number of utilized slots in the base schedule \mathcal{S}_B (line 25). The number of idle slots assigned to cluster $\kappa[j]$ is directly proportional to the slot utilization of cluster $\kappa[j]$. These idle slots are selected uniformly at random from the list of idle slots, $Free$ and assigned to each cluster $\kappa[j]$ (lines 26-29). A set of random keys of varying length is generated and stored in Key (line 32).

For each node $V_x \in G.V$, a list $Member_x$ is loaded onto V_x during network initialization. For each cluster $\mathcal{C} \in \kappa$, the cluster information associated with \mathcal{C} is added to $Member_x$ if node V_x belongs to the route of the flows that are associated with cluster \mathcal{C} . If V_x belongs to multiple flows that are associated with more than one cluster, then $Member_x$ needs to be loaded with information related to all the clusters to which each of these flows belongs to. The list of random keys, Key , is also loaded on each node $V_x \in V$.

6.5.2 Online Distributed Schedule Generation Phase

Algorithm 8 runs online on each node $V_x \in G.V$ in a distributed manner. If the list $Member_x$ of a node V_x contains the j^{th} cluster, $\kappa[j]$, then V_x runs the online partial schedule generator once every $\kappa[j].p$ ms. At the t^{th} scheduling window, W_t , node V_x computes the partial schedule for the $(t+1)^{th}$ scheduling window, W_{t+1} . The function *randomKeySelect()* uses the PRNG to select a random key from the available keys (line 1). The PRNG of each cluster ensures that each node associated with cluster $\kappa[j]$ selects the same random key in a distributed manner. Otherwise, two nodes may schedule two different transmissions in the same slot, resulting in collision in the network, violating condition 2 of a feasible schedule (refer to Definition 2.5 in Section 2.2.2 of Chapter 2).

If a node is currently in the t^{th} scheduling window of a hyperperiod schedule, then all slots belonging to W_{t+1} and part of $\kappa[j].s$ act as our *plaintext* in the partial schedule for

Algorithm 7: *Gen_Cluster*($\mathcal{F} \cup \{F_0\}, n, \mathcal{S}_B$)

```

1  $\kappa = \{\}$ ; // Empty list of clusters
2 Assign Allocated, Free, dist_period to empty list;
3 for  $slots = 1, 2, \dots, hp$  do
4   if  $\mathcal{S}_B[slots]$  is an idle slot then
5     Add  $\mathcal{S}_B[slots]$  to list Free;
6   else
7     Add  $\mathcal{S}_B[slots]$  to list Allocated;
8 for  $i = 1, 2, \dots, n$  do
9   if  $F_i.p_i \notin dist\_period$  then
10    Add  $p_i$  to the set dist_period; // Add all the distinct periods
    of the flows in  $\mathcal{F}$  to the set dist_period
11 for  $i = 1, 2, \dots, |dist\_period|$  do
12    $\mathcal{C} = \text{Cluster}()$ ; // Initialize an empty cluster
13    $\mathcal{C}.p = dist\_period[i]$ ;
14    $\mathcal{C}.prng = \text{Initialize}()$ ; // initialize PRNG
15   Initialize empty list for  $\mathcal{C}.s$ ,  $\mathcal{C}.FSeq$ ,  $\mathcal{C}.\mathcal{F}$ ;
16   Add  $\mathcal{C}$  to the list  $\kappa$ ;
17 for each  $p_j \in dist\_period$  do
18   Assign a variable util to 0;
19   for each  $F_i \in \mathcal{F}$  do
20     if  $F_i.p_i == dist\_period[j]$  then
21       Add  $F_i$  to the list  $\kappa[j].\mathcal{F}$ ;
22       for  $slots = 1, 2, \dots, hp$  do
23         if  $F_i$  is scheduled at  $\mathcal{S}_B[slots]$  then
24           Allocate  $slots$  to  $\kappa[j].s$  and  $F_i$  to  $\kappa[j].FSeq$ ;
25        $util = util + \frac{hp * F_i.n\_hops}{p_i * |Allocated|}$ ;
26    $idle = util \times |Free|$ ;
27   for  $k = 1, 2, \dots, idle$  do
28      $slots = \text{random}(Free)$ ; // Select a random slot from Free
29     Add  $slots$  to  $\kappa[j].s$  and  $F_0$  to  $\kappa[j].FSeq$ ;
30   Sort the slots in  $\kappa[j].s$ ;
31   Map the transmissions in  $\kappa[j].FSeq$  with the slots in  $\kappa[j].s$ ;
32 Generate a set of random keys of varying length Key;
33 return ( $\kappa, Key$ );

```

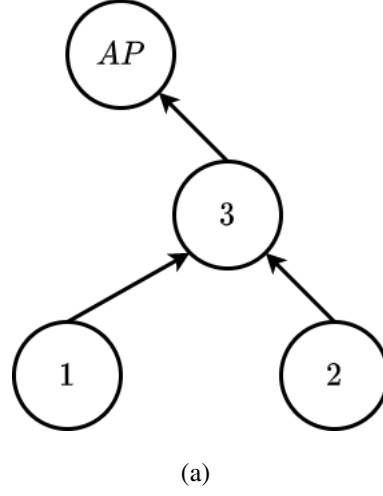
Algorithm 8: *Gen_Partial_Sched*($\kappa[j]$)

```

1 key = randomKeySelect(Key,  $\kappa[j].prng$ );
2 for slots : slots  $\in W_{t+1}$  && slots  $\in \kappa[j].s$  do
3   | Add slots to plaintext;
4 if key.length()  $\geq |plaintext|$  then
5   |  $n_{blocks} = 1$ ;
6   | if key.length()  $> |plaintext|$  then
7   |   | Add dummy slots to plaintext to make it equal to key.length();
8 else if key.length()  $< |plaintext|$  then
9   | Split the plaintext into blocks of length key.length();
10  |  $n_{blocks} = \lceil \frac{|plaintext|}{key.length()} \rceil$ ;
11 Cipher = {};
12 for num = 1, 2, ...,  $n_{blocks}$  do
13   | // Permute slots of plaintext using key
14   | Cipher = Cipher  $\cup$  PermutateCipher(plaintext, key);
15 Remove dummy slots from Cipher;
16 Add the updated slots from Cipher to  $\kappa[j].s$ ;
17 Update the transmissions in  $\kappa[j].FSeq$  corresponding to Cipher;
18 Update the permuted slots in  $\kappa[j].s$  and  $\omega_j$ ;
19 return  $\omega_j$ ;

```

W_{t+1} (line 2-3). If in any scheduling window, the size of *plaintext* is smaller than the key length, then dummy slots are added to the *plaintext* till the length of the *plaintext* is equal to the key length (lines 4-7). On the other hand, if in any scheduling window, the size of the *plaintext* is larger than the key length, then the *plaintext* is sub-divided into blocks of length equal to the selected key (lines 8-10). The variable n_{blocks} keeps track of the number of sub-divided blocks of the *plaintext*. The same *key* is then applied to each of these n_{blocks} number of blocks of the *plaintext* to generate *Cipher*, a permutation of the slots in *plaintext* (line 13). The function *PermutateCipher*() is similar to the permutation cipher used in classical cryptography [142]. Dummy slots if added, are then eliminated from *Cipher* (line 14). The slots in *Cipher* corresponding to the scheduling window considered, are updated in $\kappa[j].s$ (line 15). The corresponding transmissions are updated in $\kappa[j].FSeq$ to generate a new partial schedule ω_j (line 17). The permuted slots in $\kappa[j].s$ form the plaintext in the next hyperperiod during computation of partial schedule for the same scheduling window.



Schedules	Slots									
	1	2	3	4	5	6	7	8	9	10
\mathcal{S}_1	$1 \rightarrow 3$ F_1	$3 \rightarrow AP$ F_1	$3 \rightarrow AP$ F_2			$1 \rightarrow 3$ F_1	$3 \rightarrow AP$ F_1			
\mathcal{S}_2		$1 \rightarrow 3$ F_1		$3 \rightarrow AP$ F_1	$3 \rightarrow AP$ F_2			$1 \rightarrow 3$ F_1	$3 \rightarrow AP$ F_1	
\mathcal{S}_3	$1 \rightarrow 3$ F_1		$3 \rightarrow AP$ F_2	$3 \rightarrow AP$ F_1		$1 \rightarrow 3$ F_1			$3 \rightarrow AP$ F_1	

(b)

FIGURE 6.2: (a) a single-channel WirelessHART network graph G (b) Three feasible schedules over of 10 slots for the graph G in Figure. 6.2(a) with two flows F_1 (marked in red) with $route_1 = \{1 \rightarrow 3 \rightarrow AP\}$, period $p_1 = 50ms$ (5 slots) and F_2 (marked in blue) with $route_2 = \{3 \rightarrow AP\}$, period $p_2 = 100ms$ (10 slots).

We illustrate the steps of our algorithms with an example.

Example 6.1. Consider schedule \mathcal{S}_1 in Figure. 6.2(b) as our base schedule with two flows, F_1 (marked in red) and F_2 (marked in blue). The Offline Cluster Generation Phase generates two lists, $Allocated = \{1, 2, 3, 6, 7\}$ and $Free = \{4, 5, 8, 9, 10\}$ from the allocated and idle slots in \mathcal{S}_1 respectively. There are two instances of F_1 and one instance of F_2 in \mathcal{S}_1 . Since, the two flows have different periods and different scheduling window for each of their instances, they belong to two clusters. Let the two clusters be \mathcal{C}_1 and \mathcal{C}_2 respectively. Slot utilization of \mathcal{C}_1 and \mathcal{C}_2 are given by 0.8 and 0.2 respectively. Let the idle slots allocated to \mathcal{C}_1 and \mathcal{C}_2 be $\{4, 8, 9, 10\}$ and $\{5\}$ respectively.

Now, consider the Online Partial Schedule Generation phase. The scheduling windows corresponding to \mathcal{C}_1 is given by $[1, 5]$ and $[6, 10]$ and that of \mathcal{C}_2 is given by $[1 - 10]$. Let

us consider cluster \mathcal{C}_1 . The plaintext belonging to \mathcal{C}_1 corresponding to the scheduling windows $[1, 5]$ and $[6, 10]$ are given by $\{1, 2, 4\}$ and $\{6, 7, 8, 9, 10\}$ respectively. Similarly, the plaintext belonging to \mathcal{C}_2 corresponding to the scheduling window $[1, 10]$ is given by $\{3, 5\}$. Let us consider any random key, say $K_1 = (3, 2, 1)$. On permuting the slots of cluster \mathcal{C}_1 within the scheduling window $[1, 5]$ in \mathcal{S}_1 using key K_1 , we get a new partial schedule (refer slots of \mathcal{S}_2 in Figure. 6.2(b) within scheduling window $[1, 5]$). Similarly, by permuting the slots of \mathcal{S}_1 in each scheduling window with different random keys, new schedules can be generated such as $\mathcal{S}_2, \mathcal{S}_3$, etc in Figure. 6.2(b).

6.5.3 Time Complexity Analysis

In the offline *Gen_Cluster* algorithm, creating two lists, *Allocated* and *Free*, involves scanning the hyperperiod schedule once which takes $\mathcal{O}(hp)$ time (lines 3-7 of Algorithm 7). Creating a set of distinct periods of flows in the system, *dist_period*, involves accessing the periods of n flows which takes $\mathcal{O}(n)$ time (lines 8-10 of Algorithm 7). In the worst-case, if all the n flows have distinct periods, the time taken to create n empty clusters and generate cluster list κ is given by $\mathcal{O}(n)$ (lines 11-16 of Algorithm 7). For each cluster $\kappa[j]$, assigning slots associated with each flow F_i in \mathcal{S}_B to $\kappa[j].s$ requires $\mathcal{O}(n^2 \times hp)$ time in the worst-case (when all flows have distinct periods). Allocation of idle slots to each cluster requires $\mathcal{O}(hp)$ time in the worst-case (lines 27-29 of Algorithm 7). Finally, sorting the slots in $\kappa[j].s$ takes $\mathcal{O}((hp) \times \log(hp))$ in the worst-case (line 30). Mapping the transmissions of $\kappa[j].FSeq$ with the slots in $\kappa[j].s$ takes $\mathcal{O}((hp) \times \log(hp) + n^2 \times hp)$.

Now, consider the online *Gen_Partial_Sched* algorithm. For each scheduling window of the j^{th} cluster, $\kappa[j]$, creating the *plaintext* requires copying slots from $\kappa[j]$ to *plaintext* corresponding to each scheduling window (lines 2-3 in Algorithm 8) which requires $\mathcal{O}(hp)$ in the worst-case if $\kappa[j].p = hp$. Since, the keys are of constant length, addition (removal) of dummy slots in each scheduling window to (from) the *plaintext* takes $\mathcal{O}(1)$ time each (lines 6-7 and line 14 of Algorithm 8). Dividing the *plaintext* into n_{blocks} takes $\mathcal{O}(hp)$ in the worst-case (for $|plaintext| = hp$). If N be the maximum length of the key, then generating permutation cipher with a key of length N involves N^2 swaps in the worst-case for each block (line 13 of Algorithm 8). In the worst-case, the length of plaintext can be at-most hp . This implies that the maximum value of n_{blocks} can be $\frac{hp}{N}$.

Therefore, the total time taken to run the online partial schedule generation phase is given by $\mathcal{O}(hp + \frac{hp \times N^2}{N})$, which is equal to $\mathcal{O}(hp \times N)$ in the worst-case.

6.5.4 Period Adaptation

The periods of the real-time flows directly impact the control performance in ICS. The smaller the periods of the flows are, the higher the rate at which the sensor data (control signals) are being transmitted to the access points (actuators). However, smaller periods of the flows imply less scope of schedule randomization due to smaller scheduling window of each instance of a flow. In this section, we present a period adaptation strategy at the WirelessHART network that can ensure the control performance as well as increase the scopes of randomization of the slots in the schedules depending on the stability of each control loop. Ma *et al.* proposed a rate adaptation strategy to lower the energy consumption of the system [59]. In this section, we incorporate a similar period adaptation strategy [59].

Control Design: The bounds of the state error is determined from the Lyapunov function (6.8) at the controller side and is denoted by

$$a_1 \|x_t\| \leq V(x_t) \leq a_2 \|x_t\| \quad (6.11)$$

where a_1 and a_2 are the smallest and largest eigenvalues of \mathcal{P} respectively. Considering a certain customized state error bound, $SE = \|x_{se}\|^2$, the threshold for increasing period is denoted by $V_{Inc} = a_1 \|x_{se}\|^2$ and that for decreasing period is denoted by $V_{Dec} = \lambda a_1 \|x_{se}\|^2$, where $\lambda \in (0, 1)$. Our control system is said to be in good condition if $V(x_t)$ remains below V_{Dec} for a customized time interval τ . From (6.9), it can be shown that,

$$V(x_{t+1}) - V(x_t) \leq -\beta \|x_t\|^2 \quad (6.12)$$

From (6.9) and (6.12), the upper bound of ideal Lyapunov function is given by (6.13) which is the upper bound to trigger succeeding decrease in the period.

$$V(x_{t+j}) \leq (1 - \frac{\beta}{a_2})^j V(x_t) \quad (6.13)$$

where β is the smallest eigenvalue of \mathcal{Q} (6.9).

Network reconfiguration: *DistSlotShuffler* can further be extended to handle period adaptation of the flows at runtime. To facilitate period adaptation, all flows in \mathcal{F}^k , associated with L_k , are assigned a set of potential periods $P_k = \{p_k^1, p_k^2, \dots, p_k^{max}\}$, where $p_k^1 = \frac{1}{R_k}$. The base schedule \mathcal{S}_B is generated with the minimum period or base period p_k^1 in P_k . The potential periods of transmission associated with the k^{th} control loop are chosen to be integral multiples of p_k^1 . To ensure stability of the system, multiples of p_k^1 that are less than the maximum stabilizing period are selected in the set P_k . The controller decides the period of the flows based on the stability of the system. If the current period of the transmissions associated with the k^{th} control loop, L_k , changes, then the controller broadcasts the updated period, p_k^{adapt} (say), and the corresponding control loop number “ k ” to all the nodes in the uplink and downlink graph. We reserve a few slots at the beginning of every hyperperiod for this broadcast. The slots are only used if a period adaptation gets triggered, otherwise, the reserved slots remain unused. Only the flows that are associated with L_k will adapt its period of transmission to p_k^{adapt} . All the nodes that belong to the route of the flows associated with L_k , will update their period of transmission to p_k^{adapt} .

Algorithm 9: *Per_Adapt_Controller*($x_t, t, t_0 = t, \tau, \lambda, Per_k, p_k^{current}$)

```

1 Calculate  $V(x_t), V_{Inc}, V_{Dec}$ ;
2 if  $V(x_t) < V_{Inc}$  for a time interval  $\tau$  &&  $p_k^{current} < p_k^{max}$  then
3    $p_k^{current} = P_k[current + 1]$  // period increases
4 else if  $V(x_t) > V_{Dec}$  &&  $p_k^{current} > p_k^1$  then
5   . if last_adapt is an increase then
6      $p_k^{current} = P_k[current - 1]$  // period decreases
7      $t_0 = t$ ;
8   else if last_adapt is a decrease &&  $V(x_t) > (1 - \frac{\beta}{a_2})^{t-t_0} V(x_{t_0})$  then
9      $p_k^{current} = P_k[current - 1]$  // successive period decrease
10 else
11    $p_k^{current}$  remains constant;
12 return  $p_k^{current}$ ;

```

Algorithm 9 presents the period adaptation strategy at the controller side for control loop L_k . If $V(x_t)$ remains below V_{Inc} for a customized time interval, τ , and the current period $p_k^{current}$ is less than the maximum period p_k^{max} in P_k , then the period of transmission is increased. On the other hand, if $V(x_t)$ is greater than V_{Dec} and the current period of transmission is greater than the minimum period p_k^1 in P_k , then the period of transmission is decreased to restore the system stability. If the condition in line 8 is satisfied further,

then the period of transmission is decreased further. Otherwise, the flows belonging to control loop L_k continues to transmit with the current period $p_k^{current}$.

Stability Analysis: Due to the period adaptation strategy, a closed-loop system can be rendered as a switched system. The switching is governed by Algorithm 9 at the controller side. Since, analytically formulating the switching sequence is quite complex, we use the stability result for switched system with arbitrary switching [143]. If there exists a common Lyapunov function for all subsystems, then we can guarantee stability of the switched system under arbitrary switching. We need to construct a common Lyapunov function

$$V(x_t) = x_t^T \mathcal{P} x_t \quad (6.14)$$

among all candidate periods such that the Linear Matrix Inequality (LMI) problem represented by the set of equations

$$(A_{p_k^i} + B_{p_k^i} K)^T \mathcal{P} (A_{p_k^i} + B_{p_k^i} K) - \mathcal{P} < 0, \forall p_k^i \in Per_k \quad (6.15)$$

can be solved for \mathcal{P} . Here, $A_{p_k^i}$ and $B_{p_k^i}$ are the discretized state matrices of control loop L_k corresponding to sampling time p_k^i . If there exists a feasible solution for the LMI problem given by Equation (6.15), $\forall p_k^i \in Per_k$, then $V(x_t) = x_t^T \mathcal{P} x_t$ is the common Lyapunov function for all candidate periods in P_k and stability can be achieved. The existence of a common Lyapunov function is simple but easy to check stability condition. From here onwards, we use the common Lyapunov function, represented by Eq. (6.14), in Eq. (6.11), Eq. (6.12) and Eq. (6.13) to support arbitrary switching during period adaptation.

6.5.5 Extension of *DistSlotShuffler* to support period adaptation

If the period of all the flows in \mathcal{F}^k associated with control loop L_k , changes to any period higher than p_k^1 , then the number of slots in the scheduling window corresponding to each instance of these flows increase. This, in turn, reduces the predictability of the slots in the schedules due to increase in the number of available choices for these flows to appear. For a fixed number of control loops with flows sharing the same WirelessHART network, an increase in the period of the flows results in decrease in the utilization of the slots in the schedule, which in turn, reduces the energy consumption of the system.

Algorithm 10 presents the period adaptation at the WirelessHART network for all flows in \mathcal{F}^k , that are associated with the k^{th} control loop L_k and are part of cluster $\kappa[j]$. Algorithm 10 executes on each node $V_x \in G.V$ that belongs to the route of any flow in \mathcal{F}^k . With period adaptation, the nodes in the route of \mathcal{F}^k starts transmitting the sensor data (control signals) with period, p_k^{adapt} , where $p_k^{adapt} = n_{occur} \cdot p_k^1$, n_{occur} is an integer and $n_{occur} \geq 1$. The variable n_{occur} denotes the number of times the base period p_k^1 can occur in the adapted period p_k^{adapt} (line 2 of Algorithm 10). This implies that there are n_{occur} scheduling windows of size p_k^1 within the current scheduling window corresponding to the adapted period. All nodes associated with any flow in \mathcal{F}^k needs to select any one of these n_{occur} scheduling windows to transmit/receive data. The modulo operation in line 9 of Algorithm 10 ensures that all the nodes select the same scheduling window to transmit/receive the packet. The *key* used in line 9 of Algorithm 10 is the same *key* used in line 1 of Algorithm 8. The nodes continue to transmit at the adapted period until no further period adaptation is triggered. Generally, the *key* used in line 9 of Algorithm 10 is greater than n_{occur} . However, if $key < n_{occur}$, then t will be assigned the same value as *key*. Note that, if the base period p_k^1 is equal to the hyperperiod hp , then the online schedule generator, *Gen_Partial_Sched()*, runs once every hp slots. If the period p_k^1 increases to any larger period, such as p_k^2 (say), the hyperperiod of the schedule is not impacted as the online schedule generator still executes *Gen_Partial_Sched()* at every base period p_k^1 . This is done to ensure that the $\kappa[j].prng$ and the slots in $\kappa[j].s$ are permuted at the same rate in all the nodes which are part of $\kappa[j]$.

Note that, the controller in our system still continues to operate at the same rate R_k . Some well-known control techniques such as self triggered control [59] exists in the literature to change the rate of the controller. However, this is not the focus of this chapter. We refer interested readers to [144, 145, 146].

6.6 Experiments and Evaluation

Uncertainty Measure: The main objective of *DistSlotShuffler* is to randomize the slots in a hyperperiod schedule at runtime in a distributed manner to generate partial schedules at each network device in the WSN, such that the predictability in the slots of the hyperperiod schedule reduces, thereby, making it difficult for the attacker to predict the slots and launch an attack. More random the slots are in a hyperperiod schedule,

Algorithm 10: *Per_Adapt_Network*($p_k^{adapt}, L_k, \kappa[j]$)

```

1 if period adaptation is triggered then
2    $n_{occur} = (p_k^{adapt} / p_k^1);$ 
3    $p_k^{current} = p_k^{adapt};$ 
4    $counter = 0;$ 
5 while no period change is triggered do
6    $\omega_j = Gen\_Partial\_Sched(\kappa[j]);$ 
7   if  $p_k^{current} > p_k^1$  then
8     if  $counter == 0$  then
9        $t = key \% n_{occur};$ 
10    if  $counter == t$  then
11      Transmit/receive messages in the slots corresponding to  $\omega_j$ ;
12     $counter = counter + 1;$ 
13    if  $counter == n_{occur}$  then
14       $counter = 0;$ 

```

more difficult it is for an attacker to predict the slots of a schedule. A truly random algorithm can generate and run all possible feasible schedules with equal probability. However, *DistSlotShuffler* can generate only a subset of the all possible feasible schedules depending on the distribution of slots in the base schedule, number of keys in the set *Key* and the PRNG associated with each cluster.

We use *Kullback-Leibler divergence* or *K-L divergence* as a security metric to measure the uncertainty in the slots of the generated schedules. K-L divergence has been introduced in Section 2.4 of Chapter 2. K-L divergence measures the divergence in the slots of the hyperperiod schedules generated by our algorithm with reference to a truly random algorithm. Lower the value of K-L divergence, more diverse are the slots in the hyperperiod schedules and thus less predictable are the slots in the generated schedules.

We define K-L divergence in the context of a single-channel WirelessHART network with $n + 1$ flows ($\mathcal{F} \cup \{F_0\}$, F_0 being an idle flow).

Definition 6.2. Given an algorithm \mathbb{A} , a truly random algorithm \mathbb{A}' that can generate all possible feasible schedules of length hp with equal probability, over a single-channel WirelessHART network with a set of $n + 1$ flows, (F_0 being an idle flow corresponding to the idle slots in the schedules) the relative entropy or K-L divergence of \mathbb{A} with reference to \mathbb{A}' measures the divergence in the probability distribution of the flows over all the slots

in the hyperperiod schedules generated by \mathbb{A} with reference to \mathbb{A}' . It is denoted by

$$\mathcal{KL}(\mathbb{A}||\mathbb{A}') = \sum_{s=1}^{hp} \sum_{i=0}^n \Pr(g_s = F_i) \log_2 \frac{\Pr(g_s = F_i)}{\Pr(g'_s = F_i)} \quad (6.16)$$

where g_s and g'_s are discrete random variables and $\Pr(g_s = F_i)$ and $\Pr(g'_s = F_i)$ are the probability mass functions of the i^{th} flow occurring in the s^{th} slot in the hyperperiod schedules generated by \mathbb{A} and \mathbb{A}' respectively.

A truly random algorithm, \mathbb{A}' , generates all possible feasible schedules. However, the computation of all possible feasible schedules in \mathbb{A}' is exponential for a very long hyperperiod. Hence, we calculate the *upper-bound K-L divergence* in our experiments by considering a hypothetical truly random algorithm. Note that the hypothetical truly random algorithm \mathbb{A}' may not satisfy all the feasibility conditions of a schedule. The hypothetical truly random algorithm has a probability mass function of $\Pr(g'_s = F_i) = \frac{F_i \cdot n_hops}{p_i \cdot hp}$, where $\frac{F_i \cdot n_hops}{p_i}$ gives the probability mass function of a single slot, which when multiplied by $\frac{1}{hp}$ gives the probability mass function of all possible schedules over hp .

Security Analysis: Upper-bound K-L divergence provides a measure of the randomness in the slots of the schedules generated by our algorithm, the *DistSlotShuffler*, with reference to a truly random algorithm. However, it does not quantify the success of the proposed defense mechanism against the attack. The success probability of our attacker is dependent on how correctly it can predict the transmission in a specific slot depending on its observation in the previous hyperperiod schedules. We use Prediction Probability (PP) of slots in a schedule to measure the effectiveness of the generated schedules in mitigating the attack. PP of slots in a schedule have been defined in Definition 5.4 in Chapter 5.

Metrics: The control performance of our system is measured in terms of mean absolute error (MAE) which is given by

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_t - \hat{y}_t| \quad (6.17)$$

where y_t and \hat{y}_t are the actual outputs and estimated outputs of the LTI plant. The performance of the *DistSlotShuffler* is measured by upper-bound K-L divergence which measures the divergence in the solution space of our algorithm with reference to a truly random algorithm. In addition, we provide some measure on the overall estimated power

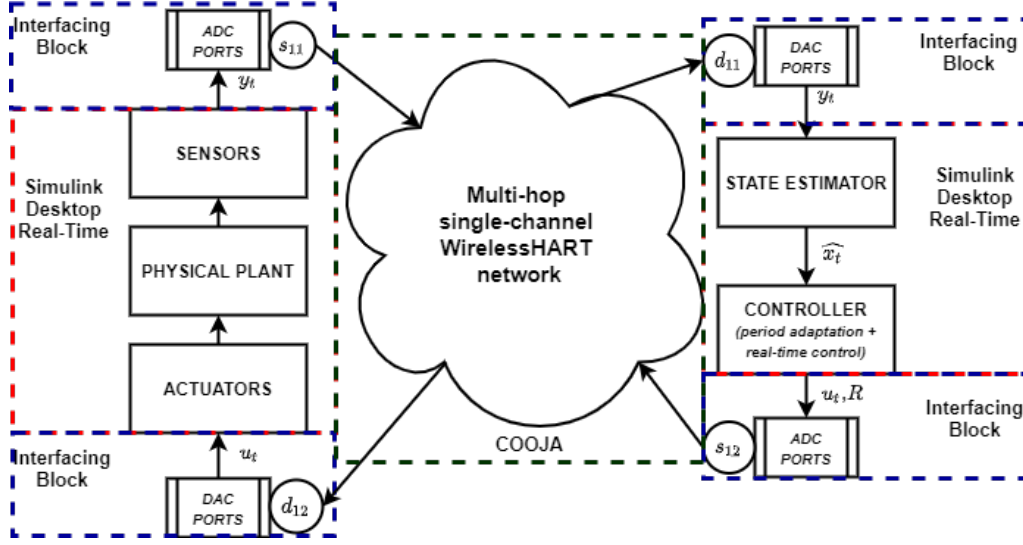


FIGURE 6.3: A wireless control loop in the GISOO simulator

consumption of the telosb motes in the WirelessHART network on running this algorithm.

Simulation Setup: We conducted our experiments on the GISOO simulator [34]. GISOO is a virtual testbed that integrates MATLAB/Simulink Desktop Real-time (SLDRT) and Cooja [30]. GISOO provides a platform to evaluate actual embedded code for the wireless nodes in realistic cyber-physical experiments, observing the effects of both the control and communication components. Figure. 6.3 shows the architecture of one wireless control loop in the GISOO simulator.

Each wireless control loop consists of a LTI physical plant, a state estimator and a linear feedback controller, all of which are modeled in SLDRT (Simulink Desktop Real-Time). The offline Cluster Generation Phase of *DistSlotShuffler* is implemented in *Python* and the online Partial Schedule Generation Phase of *DistSlotShuffler* is implemented in *TinyOS*. The multi-hop single-channel WirelessHART network is modeled in the Cooja simulator. Our experiments were conducted on four wireless control loops, each of which share the same WSAN. The interfacing block between the SLDRT and the Cooja simulator consists of 16 – bit ADC/DAC ports to facilitate communication between the physical plant (WSAN) and the WSAN (controller) in the uplink (downlink) graph. Each telosb mote [147] at the interfacing block serves as the connection between the SLDRT and the Cooja simulator. The ADC/DAC port of each telosb mote consists of 12 bits. The sensor data (control signals) generated at the sensor device (remote controller) in each control loop consists of floating point values in the range of -100.00 to $+100.00$.

However, we required a total of 7 bits to represent the part of the data before the decimal point, 7 bits to represent the part of the data after the decimal point (considering two places after the decimal point), 1 bit to represent sign (positive or negative) and 2 bits to represent the control loop number associated with the reading, *i.e.*, a total of 17 bits. Hence, we used two ADC/DAC ports at each interfacing block for the transmission (reception) of sensor data or control signals.

Control Loop Generation: The physical plant model consists of four LTI control plants, each of which is characterized by Eq. (6.1) and modeled in SLDRT. Generally, the physical plants operate continuously at very high rates. The real-time flows, the state estimator and the remote controller execute at a comparatively lower rate due to communication and computation delays in the closed-loop control. In our experiments, the physical plants are continuous and updated every $1ms$. The physical plant models are simulations of a realistic load positioning system [148, 149]. The load positioning system consists of a load (L) that uses a motor and a ballscrew transmission. The motor is attached very rigidly to a movable base platform (B). The system can be modeled as a 4-state non-linear system [149]. On operating the system at lower rates, the stiffness of the ballscrew and the stored potential energy in it can be neglected in the model and the system can be simplified to a 4-state linear system [149]. The 4-state linear system can be characterized by —

$$\dot{x}_t = A_c x_t + B_c u_t; y_t = C_c x_t; \quad (6.18)$$

where

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -d_L(\frac{1}{m_L} + \frac{1}{m_B}) & \frac{k_B}{m_B} & \frac{d_B}{m_B} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{d_L}{m_B} & -\frac{k_B}{m_B} & -\frac{d_B}{m_B} \end{bmatrix}; B_c = \begin{bmatrix} 0 \\ \frac{1}{m_L} + \frac{1}{m_B} \\ 0 \\ -\frac{1}{m_B} \end{bmatrix}; C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \quad (6.19)$$

The parameters d_L, m_L, m_B, d_B, k_B are the parameters of the load and the base platform. The state vector, x_t , is represented as

$$x_t = \begin{bmatrix} x_L \\ \dot{x}_L \\ x_B \\ \dot{x}_B \end{bmatrix} \quad (6.20)$$

where x_L and x_B are the relative and absolute displacement of the load (L) relative to the base platform (B) respectively. \dot{x}_L and \dot{x}_B denote the speeds of the relative and absolute movements. The main objective of the load positioning system is to stabilize the states of the system to the origin. The sensor data from the sensor devices is sent to the remote controller via the uplink graph (U) in a multi-hop single-channel WirelessHART network. Similarly, control signals from the remote controller are sent to the actuators via the downlink graph (D) in the same WirelessHART network. The WirelessHART network is modeled in the Cooja simulator over 30 telosb motes. The four wireless control loops consist of eight flows — four data flows transmitting sensor data from the sensor devices to the remote controller and four control flows transmitting the control signals from the remote controller to the actuators. The routes of each flow is randomly generated between the source and the destination devices.

6.6.1 Experiments

Our experiments were conducted on two kinds of physical plants characterized by the same sets of differential equations as in Eq. (6.1) with different parameters. We considered the parameter values from [59]. The first plant, denoted by $PLANT_1$, has $d_L = 15$, $m_L = 100$, $d_B = 10$, $m_B = 10$, $k_B = 5$, $K_1 = [-1.9393 - 13.13730.0842 - 13.0264]$. The second plant, denoted by $PLANT_2$, has $d_L = 10$, $m_L = 15$, $d_B = 3$, $m_B = 5$, $k_B = 22$, $K_2 = [-1.0076 - 0.6317 - 0.1954 - 0.3814]$. $PLANT_2$ has shorter response time because of lower mass and damping compared to $PLANT_1$. In our experiments, two of the four physical plant models are of type $PLANT_1$ and the remaining two are of type $PLANT_2$. The state estimator and remote controller are modeled in SLDRT.

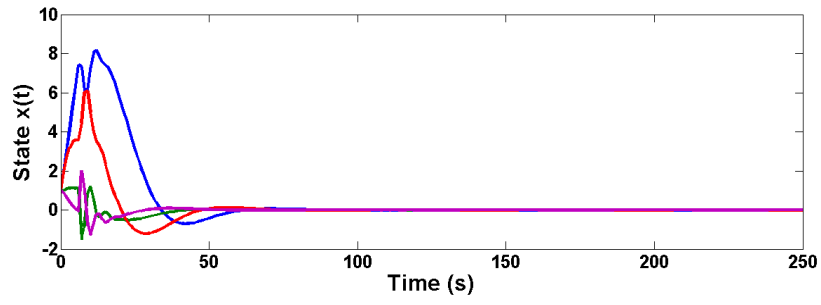
At the controller side, the continuous plant model is discretised with sampling time of 1s for the first two physical plants of type $PLANT_1$ and with sampling time of 500ms for the remaining two physical plants of type $PLANT_2$. The Kalman filter [119] runs after every

sampling period to estimate the state of the physical plant based on the current state and input. The measurement matrix H and the input matrix B of the Kalman filter (Refer Eq. (6.4) and Eq. (6.2)) are generated depending on the discretised state of the plant. The minimum periods of the real-time flows are typically chosen to be equal to the period of the controller. On adopting the period adaptation strategy, the sensor data (control signals) are transmitted at some periods which are multiples of the minimum period or base period associated with each control loop.

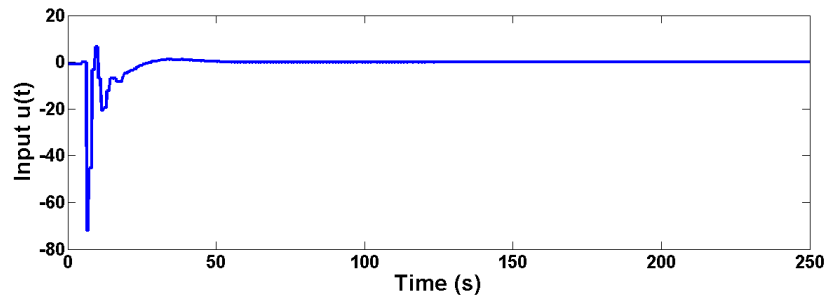
We consider two clusters, \mathcal{C}_1 and \mathcal{C}_2 , each with two control loops of type $PLANT_1$ and $PLANT_2$ respectively. The hyperperiod of the schedules in the WirelessHART network is considered to be 1s. On period adaptation, the periods associated with control loops L_1 and L_2 are given by $P_1 = P_2 = [1s, 2s, 4s]$ and that associated with control loops L_3 and L_4 are given by $P_3 = P_4 = [500ms, 1s, 2s]$. We use the common Lyapunov function (Eq. (6.15)) to determine the stability of each control loop in our system. We generate 24 random keys of varying lengths between 3 to 5 to form the set Key .

6.6.2 Control Performance

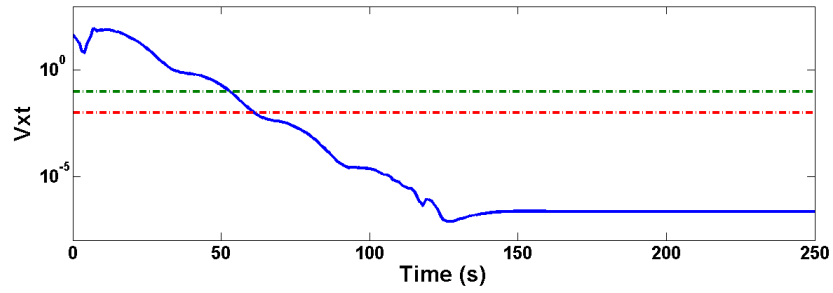
Figures. 6.4(a), 6.4(b) and 6.4(c) show the states, control input and the Lyapunov function of control loop L_1 of type $PLANT_1$ over a time period of 250s on executing the same base schedule \mathcal{S}_B every hyperperiod in the WirelessHART network. Figures. 6.5(a), 6.5(b) and 6.5(c) show the states, control input and the Lyapunov function of the same control loop L_1 over 250s on executing *DistSlotShuffler* at the WirelessHART network every hyperperiod. In both the cases, the remote controller runs the Kalman filter every 1s to estimate the state of the plant based on the current state and input. From Figures. 6.4(a), 6.4(b) and 6.4(c) and Figures. 6.5(a), 6.5(b) and 6.5(c), it can be concluded that there is no change in the control performance even on schedule randomization. This is because, *DistSlotShuffler* randomizes the slots in the schedules without violating the periods and the deadlines of the flows in the schedules. Thus, the sensor data (control signals) are delivered to the destination nodes before deadline even after schedule randomization guaranteeing the stability and hence safety of the system.



(a) States $x(t)$

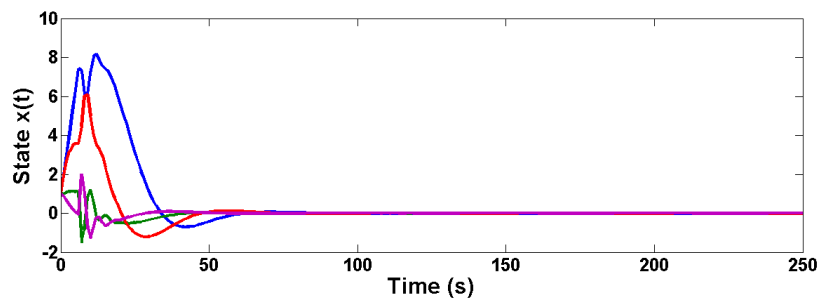


(b) Control Input $u(t)$



(c) Lyapunov function V_{xt}

FIGURE 6.4: (a) States $x(t)$ (b) Control Input $u(t)$ (c) Lyapunov function V_{xt} of $PLANT_1$ over 250s executing base schedule \mathcal{S}_B without schedule randomization



(a) States $x(t)$

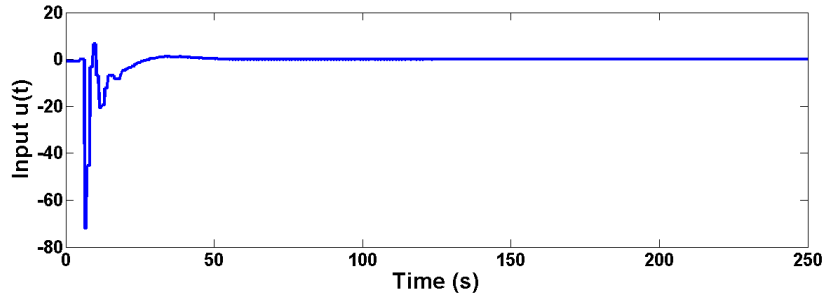
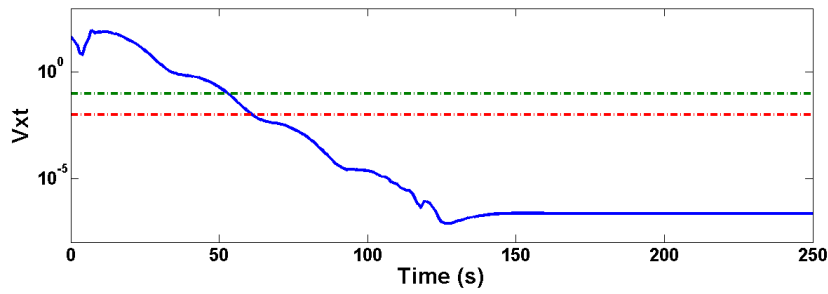
(b) Control Input $u(t)$ (c) Lyapunov function V_{xt}

FIGURE 6.5: (a) States $x(t)$ (b) Control Inputs $u(t)$ (c) Lyapunov function V_{xt} of $PLANT_1$ over 250s executing *DistSlotShuffler* at the WirelessHART network

Schedule randomization increases the uncertainty in predicting the slots of a hyperperiod schedule. Period adaptation strategy further increases the uncertainty in the slots of the schedules due to larger scheduling window associated with each flow instance in the network under normal conditions (in absence of any external disturbance). To observe the impact of schedule randomization along with period adaptation on the stability of the control loops, we run experiments on the four control loops under (i) normal condition (with no physical disturbance) (ii) with physical disturbance after the first period adaptation (iii) with physical disturbance after the second period adaptation. Figures. 6.6(a), 6.6(b), 6.6(c) and 6.6(d) show the states, control input, Lyapunov function and the period at which the sensor data (control signals) are transmitted in the WirelessHART network for control loop L_1 under normal condition (with no physical disturbance) over a time period of 300s (1) on running \mathcal{S}_B as the base schedule every hyperperiod (all plots to the left under Figure. 6.6) and (2) on running *DistSlotShuffler* every hyperperiod (all plots to the right under Figure. 6.6).

Algorithm 9 runs every 1s at the remote controller and calculates V_{xt} to decide the period at which the network devices in the WirelessHART network transmit the sensor data

(control signals). The period adaptation is triggered by the increasing and decreasing thresholds (V_{Inc} and V_{Dec}) which are dependent on some of the control parameters such as λ , τ , $\|se\|$, a_1 , a_2 , β . In our experiments, the parameter value λ is considered to be 0.1, τ is considered to be 50 and standard error $\|se\|^2$ is considered to be 0.1. a_1 , a_2 and β are derived from the physical plant equations. If the period associated with any control loop changes, a few slots are reserved at the beginning of every hyperperiod to transmit the adapted period and the control loop number associated with the adapted period. If the period remains the same, then the slot remains idle.

Our experiments show that under normal condition (with no physical disturbance) the first and the second period adaptation are triggered at 123s and 173s respectively in both the cases, *i.e.*, with and without schedule randomization for control loop L_1 (refer to both the plots under Figure. 6.6(d)). On triggering the first and the second period adaptation for control loop L_1 , the periods of the flows increase to 2s and 4s respectively. Table 6.2 shows the time at which the period adaptation is triggered in all the four control loops in our experiments. We found that for each of the four control loops, the period adaptation is triggered at the same time in both the cases, *i.e.*, with and without schedule randomization. On adopting the period adaptation strategy, the period and the deadline of the flows associated with L_k changes from p_k^1 to a longer period, say p_k^x , where x is any integer greater than 1. As a result, the scheduling window corresponding to each flow associated with control loop L_k increases in both the cases. Since, the period increases after the system attains stability for a sufficient amount of time, and the flows do not violate deadlines even after randomization, the control performance is not affected by the bounded jitters in the network due to randomization.

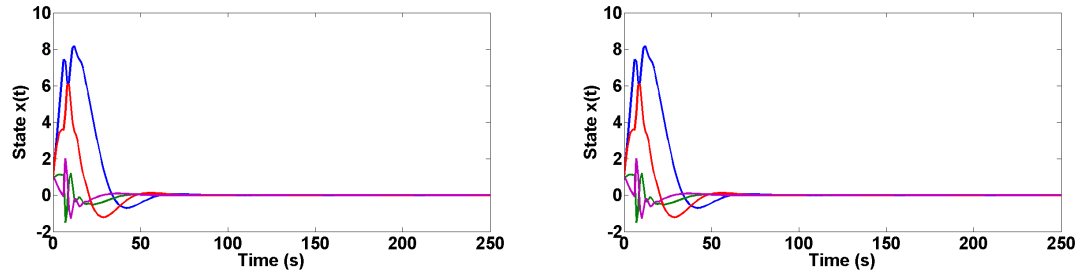
Figures. 6.7(b), 6.7(c), 6.7(d) and 6.7(e), show the states, control inputs, Lyapunov function and the period adaptation triggered at the control loop L_1 on applying a physical disturbance (noise) between 131s to 150s at the output of the physical plant (Figure. 6.7(a)) after the first period adaptation at 123s, upon executing (1) the same base schedule \mathcal{S}_B every hyperperiod (plots shown on the left side in each figure) (2) *DistSlotShuffler* (plots shown on the right side in each figure). Figures. 6.8(b), 6.8(c), 6.8(d) and 6.8(e) show the same on applying a physical disturbance (noise) between 176s to 195s (Figure. 6.8(a)) after the second period adaptation at 173s. From Figure. 6.7(e), it has been observed that the period of the flows belonging to control loop L_1 changes from 1s to 2s after 123s from the beginning of the simulation. However, due to application of physical disturbance between 131s to 150s, the period adaptation is triggered again at 134s

Control Loop	Period Adaptation Triggering Time (s)
L_1	123s (1 st period adaptation) 173s (2 nd period adaptation)
L_2	117s (1 st period adaptation) 167s (2 nd period adaptation)
L_3	92s (1 st period adaptation) 142s (2 nd period adaptation)
L_4	91s (1 st period adaptation) 141s (2 nd period adaptation)

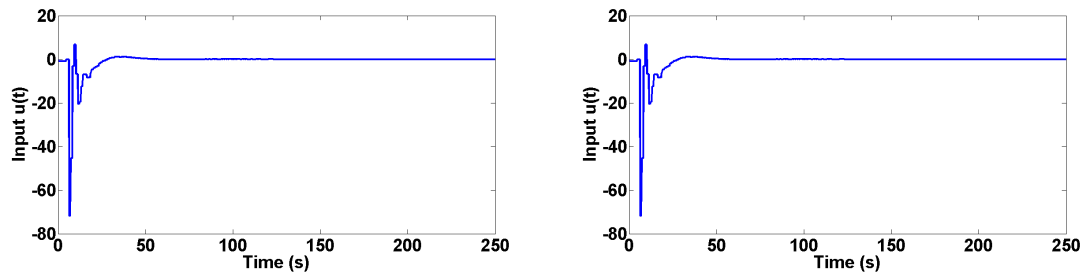
TABLE 6.2: Period Adaptation of the four control loops under normal condition (with no physical disturbance)

and the period of the flows decrease to 1s in both the cases *i.e.*, with and without randomization. The period adaptation is again triggered at 245s and 295s and increases the period to 2s and 4s respectively. Similarly, from Figure. 6.8(e), on applying a physical disturbance between 176s to 195s after the second period increase at 173s, the period adaptation is triggered at 179s from the beginning of the simulation and decreases the period from 4s to 1s in order to make the system stable. The period again increases to 2s after 295s from the beginning of the simulation. It has been observed that the control performance in both the cases, *i.e.*, with and without schedule randomization, are similar and the period adaptation is triggered at the same time in both the cases. This implies that the control performance is not affected on adopting the schedule randomization at the WirelessHART network.

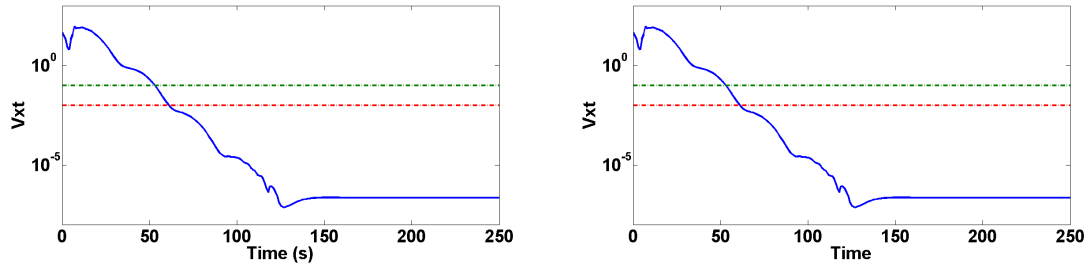
Table 6.3 shows the Mean Absolute Error (MAE) of the four control loops in each of these settings. It has been observed that the MAE is similar for both the cases (with and without schedule randomization) under normal conditions and increases for cases with physical disturbances. Note that, the physical disturbances are applied on control loop L_1 and L_2 . Hence, the MAE values for the other two control loops remain unaffected (row 5 to row 8 in Table 6.3 for L_3 and L_4). Also, the MAE is slightly higher, for cases with schedule randomization under physical disturbance, (around 0.006 and 0.002 higher after first and second period adaptation respectively for L_1 and around 0.001 higher for L_2 after the first and second period adaptation).



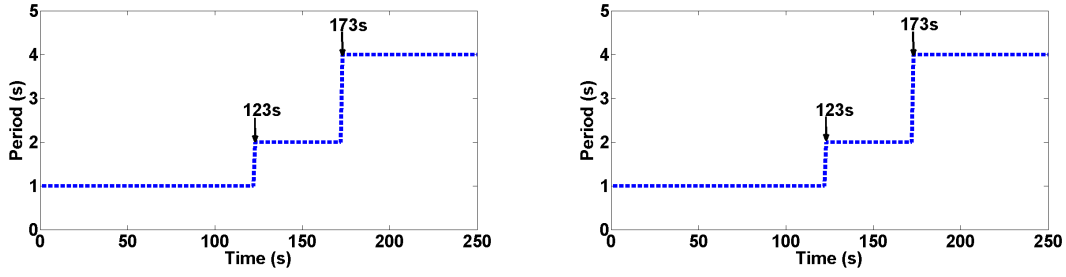
(a) States $x(t)$ without disturbance under (1) no randomization (plots to the left) (2) schedule randomization (plots to the right)



(b) Control Input $u(t)$ without disturbance under (1) no randomization (plots to the left) (2) schedule randomization (plots to the right)

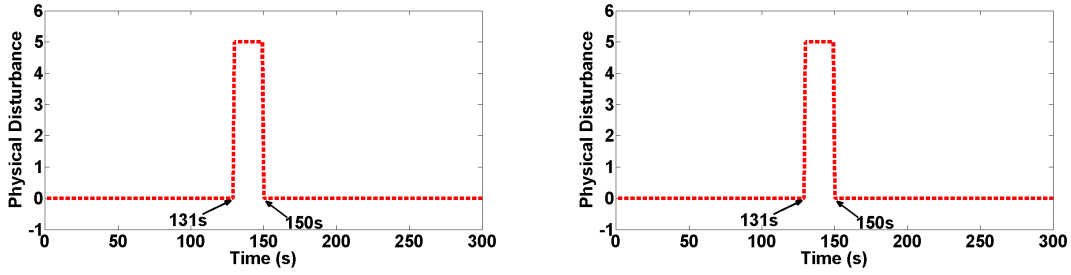


(c) Lyapunov function V_{xt} without disturbance under (1) no randomization (plots to the left) (2) schedule randomization (plots to the right)

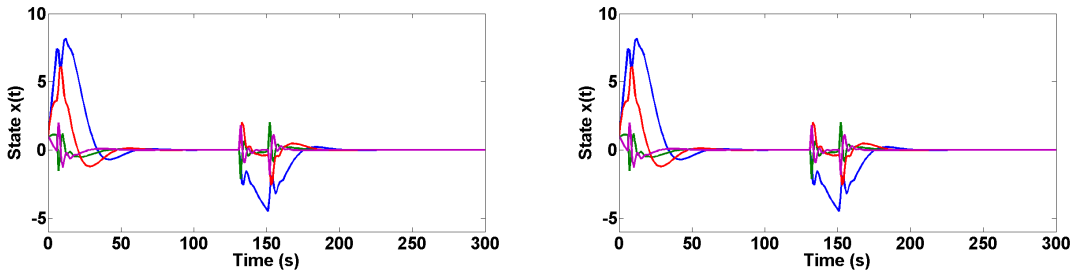


(d) Period Adaptation without disturbance under (1) no randomization (plots to the left) (2) schedule randomization (plots to the right)

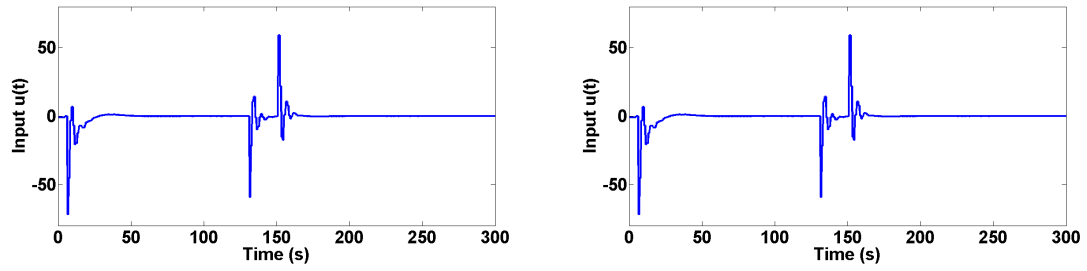
FIGURE 6.6: (a) States $x(t)$ (b) Control Inputs $u(t)$ (c) Lyapunov function Vx (d) Period Adaptation of $PLANT_1$ under normal condition (with no physical disturbance) over a time period of 250s executing (1) base schedule \mathcal{S}_B (plots to the left) and (2) *DistSlotShuffler* (plots to the right) over every hyperperiod



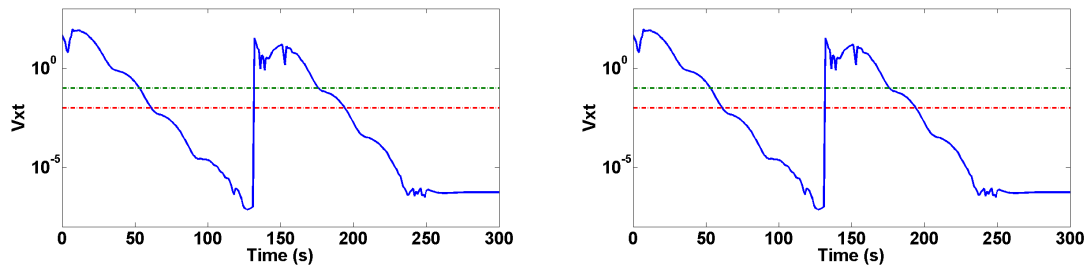
(a) Physical Disturbance applied to the output $y(t)$ between 131s and 150s



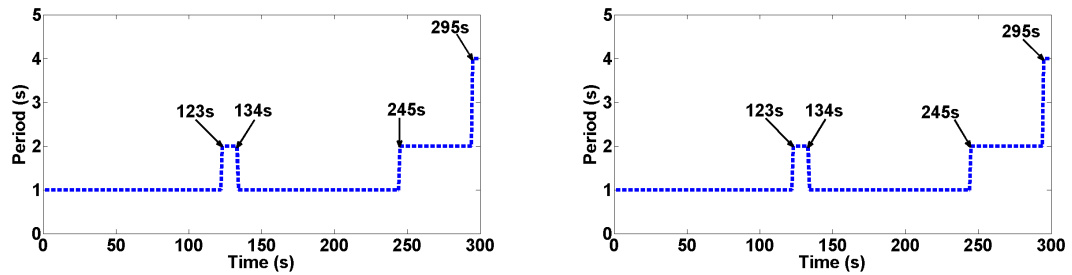
(b) States $x(t)$ with (1) no randomization (plots to the left) and (2) *DistSlotShuffler* (plots to the right)



(c) Control Input $u(t)$ with (1) no randomization (plots to the left) and (2) $DistSlotShuffler$ (plots to the right)

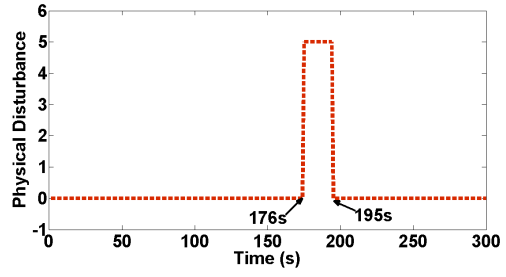
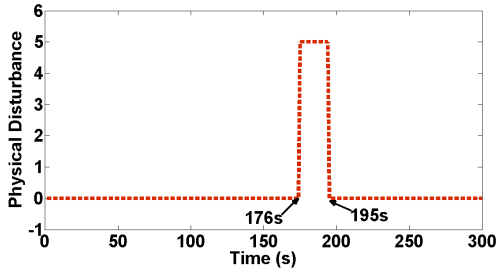


(d) Lyapunov function V_{xt} with (1) no randomization (plots to the left) and (2) $DistSlotShuffler$ (plots to the right)

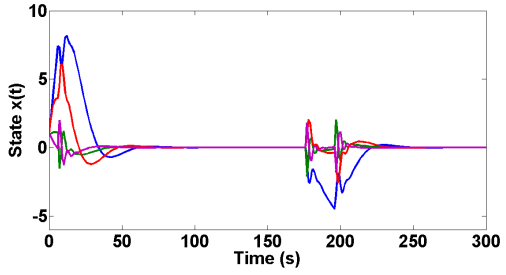
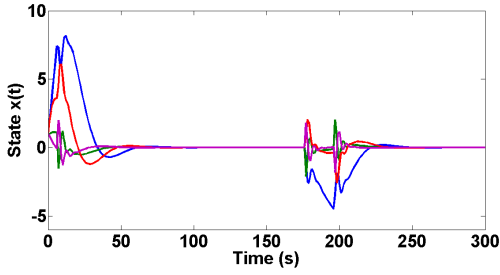


(e) Period Adaptation with (1) no randomization (plots to the left) and (2) $DistSlotShuffler$ (plots to the right)

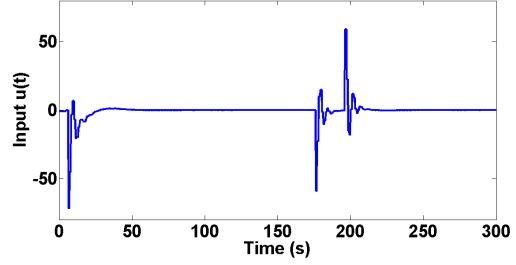
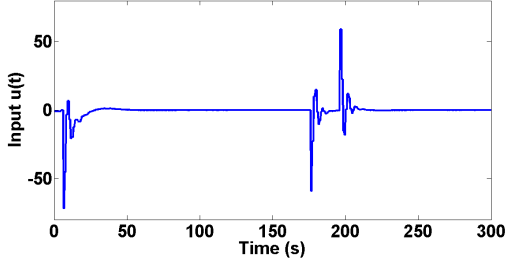
FIGURE 6.7: A physical disturbance is applied at the output of the physical plant from 131s to 150s (after first period increase at 123s). Corresponding States $x(t)$; Control Input $u(t)$; Lyapunov function V_{xt} ; Period Adaptation of control loop L_1 ; over a time period of 300s upon executing \mathcal{S}_B every hyperperiod (plots shown on the left side) and on executing $DistSlotShuffler$ (plots shown on the right side)



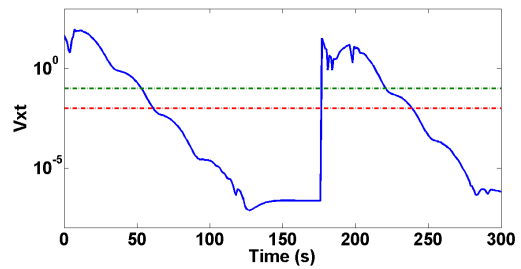
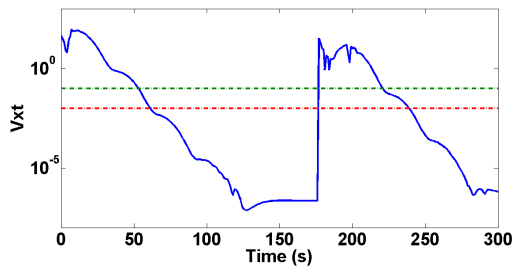
(a) Physical Disturbance applied to the output $y(t)$ between 176s and 195s



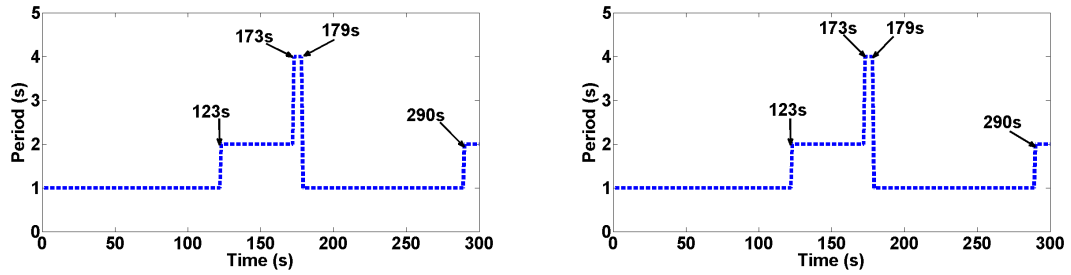
(b) States $x(t)$ with (1) no randomization (plots to the left) and (2) *DistSlotShuffler* (plots to the right)



(c) Control Input $u(t)$ with (1) no randomization (plots to the left) and (2) *DistSlotShuffler* (plots to the right)



(d) Lyapunov function (V_{xt}) with (1) no randomization (plot to the left) and (2) *DistSlotShuffler* (plots to the right)



(e) Period Adaptation on control loop L_1 with (1) no randomization (plots to the left); (2) *DistSlotShuffler* (plots to the right)

FIGURE 6.8: A physical disturbance is applied at the output of the physical plant from 176s to 195s (after second period increase at 173s). Corresponding States $x(t)$; Control Input $u(t)$; Lyapunov function Vx ; Period Adaptation of control loop L_1 ; over a time period of 300s on executing \mathcal{S}_B every hyperperiod (plots shown on the left side) and on executing *DistSlotShuffler* (plots shown on the right side)

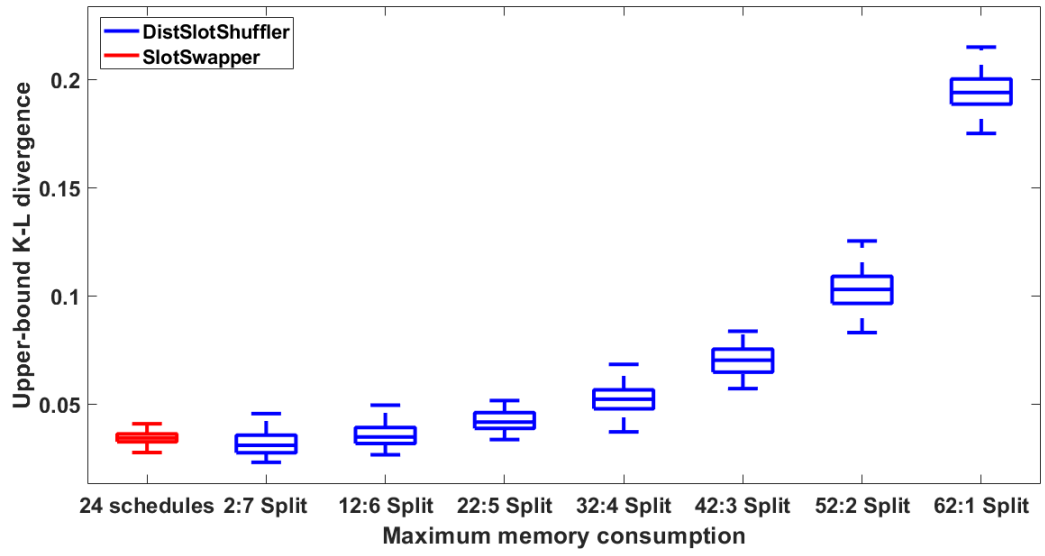
Different period adaptation and schedule randomization settings	Mean Absolute Error (MAE)			
	L_1	L_2	L_3	L_4
No Period Adaptation, No Schedule Randomization	0.1545	0.0830	0.0735	0.0610
No Period Adaptation, with Schedule Randomization	0.1545	0.0830	0.0735	0.0610
Period Adaptation (under normal condition), No Schedule Randomization	0.1534	0.0832	0.0613	0.0509
Period Adaptation (under normal condition), with Schedule Randomization	0.1535	0.0831	0.0613	0.0509
Period Adaptation, with Disturbance at L_1 and L_2 after 1 st period adaptation, No Schedule Randomization	0.5413	0.4135	0.0613	0.0509
Period Adaptation, with Disturbance at L_1 and L_2 after 1 st period adaptation with Schedule Randomization	0.5475	0.4148	0.0613	0.0509
Period Adaptation, with Disturbance at L_1 and L_2 after 2 nd period adaptation, No Schedule Randomization	0.5268	0.4143	0.0613	0.0509
Period Adaptation, with Disturbance at L_1 and L_2 after 2 nd period adaptation with Schedule Randomization	0.5287	0.4157	0.0613	0.0509

TABLE 6.3: Mean Absolute Error (MAE) of the four control loops under different conditions

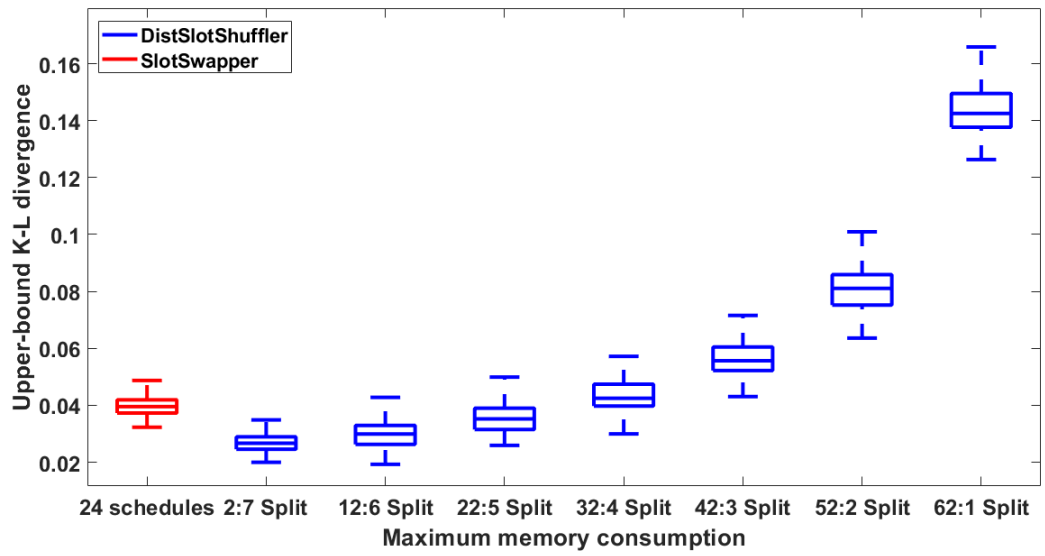
6.6.3 WirelessHART Network Performance

We compared the performance of the online distributed schedule randomization technique, the *DistSlotShuffler*, with the centralized offline schedule randomization technique, the *SlotSwapper*, described in Chapter 5 of this thesis.

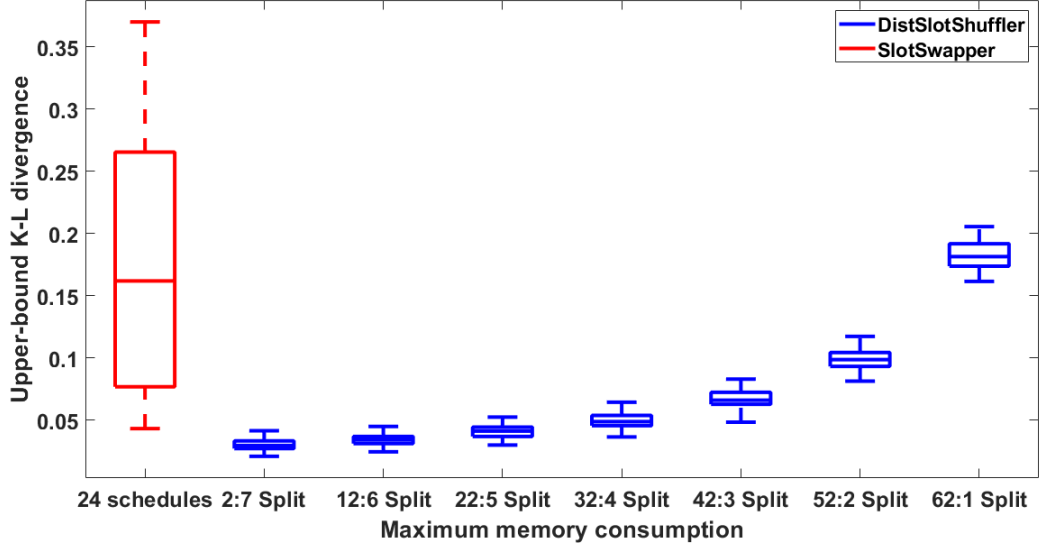
Evaluation by upper-bound K-L divergence: We use the *upper-bound K-L divergence* to measure the randomness in the slots of the schedules generated by both the schedule randomization techniques. We implemented both the techniques in *Python* and measured the divergence and the prediction probability in the schedules under varying slot utilization. We ran experiments on a Linux machine with 3.4GHz Intel Xeon 12 processors and 16GB RAM. We found that the amount of memory available to each telosb mote can only store 24 distinct random schedules for a single-channel WirelessHART network with a hyperperiod of 60 slots. We ran the schedule generation phase of the *SlotSwapper* for 10,000 iterations and selected 24 random schedules from the generated schedules. We ran the schedule selection phase of the *SlotSwapper* for 100 hyperperiods to select a random schedule from the set of 24 schedules at runtime in each hyperperiod. We calculated the upper-bound K-L divergence of the observed schedules in these 100 hyperperiods. We considered the same amount of available memory for the online schedule randomization technique, the *DistSlotShuffler*. We partitioned the available memory to store the number of base schedules and the number of keys. We varied the number of base schedules from 1 to 7 in increasing order and the number of keys from 62 down to 2 in decreasing order. For both the centralized offline and the distributed online techniques, we conducted our experiments under different slot utilization.



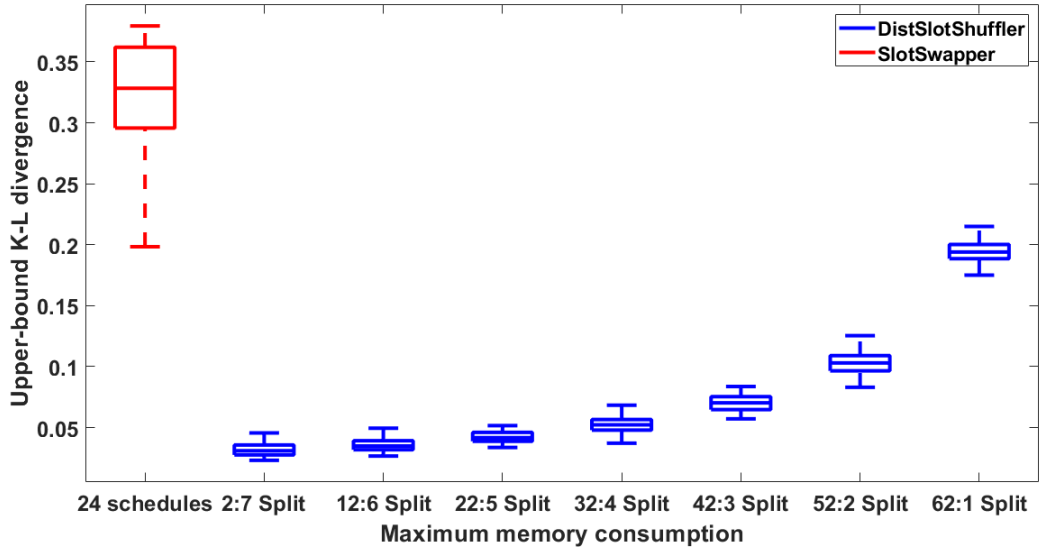
(a) slot utilization 33.3%



(b) slot utilization 50%



(c) slot utilization 66.6%



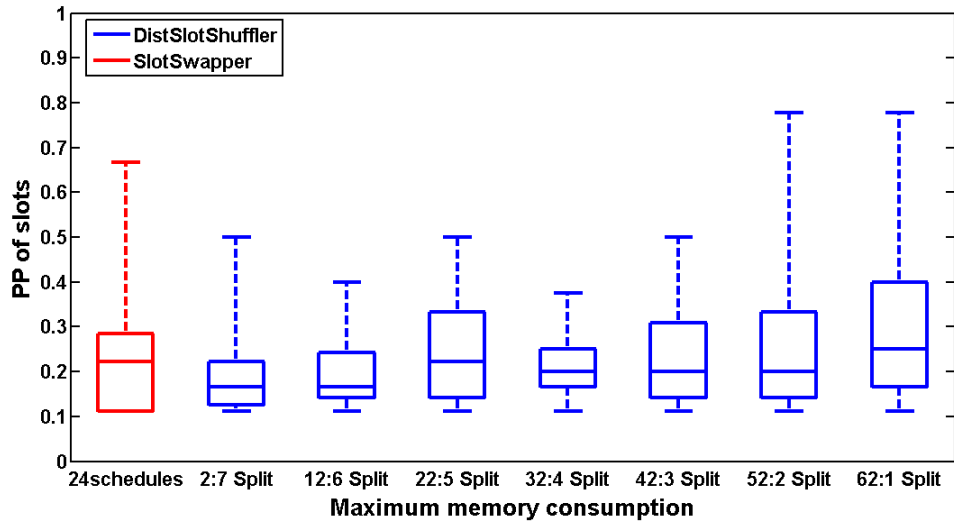
(d) slot utilization 75%

FIGURE 6.9: Upper-bound K-L divergence of (a) *SlotSwapper* (red) (b) *DistSlotShuffler* (blue) with slot utilization (i) 33.33% (ii) 50% (iii) 66.6% (iv) 75% over $hp = 60slots$. The X-axis shows the maximum memory consumed by the *SlotSwapper* (number of schedules) and *DistSlotShuffler* (number of keys: number of base schedules).

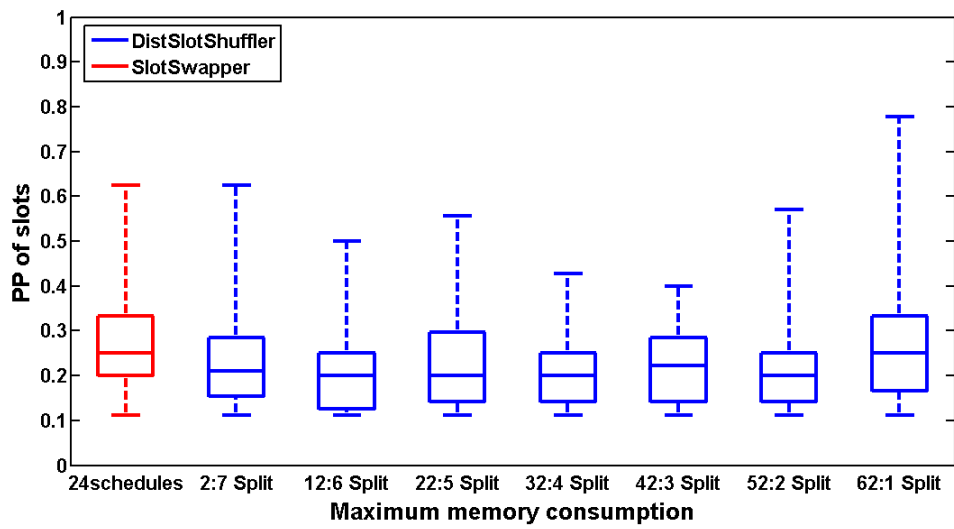
Figures. 6.9(a), 6.9(b), 6.9(c) and 6.9(d) show the upper-bound K-L divergence of the centralized offline schedule randomization technique, the *SlotSwapper* (plots in red) and

the online distributed schedule randomization technique, the *DistSlotShuffler* (plots in blue) for the set of schedules generated by each of them on consuming the maximum memory available on each telosb mote. The divergence values are calculated by varying the slot utilization in the schedules from 33.3% upto 75% over hyperperiods of 60 slots. Note that, the number of flows in each of these cases remain constant. The utilization of slots in the schedules is varied by varying the number of hops and the routes of the flows in the schedules. We observe that in all the cases (refer to Figures. 6.9(a),6.9(b),6.9(c),6.9(d)), the upper-bound K-L divergence is minimum for the *DistSlotShuffler*, with 2 keys and 7 random base schedules. The divergence values increase gradually as we increase the number of keys and decrease the number of base schedules in all the figures (Figures. 6.9(a),6.9(b),6.9(c),6.9(d)). This is because, with more base schedules, we have different combination of slots available to each cluster. The application of permutation cipher on each of these base schedules further increases the variation in the slots within each cluster in the hyperperiod schedules. Thus, the upper-bound K-L divergence value decreases with increase in the number of base schedules.

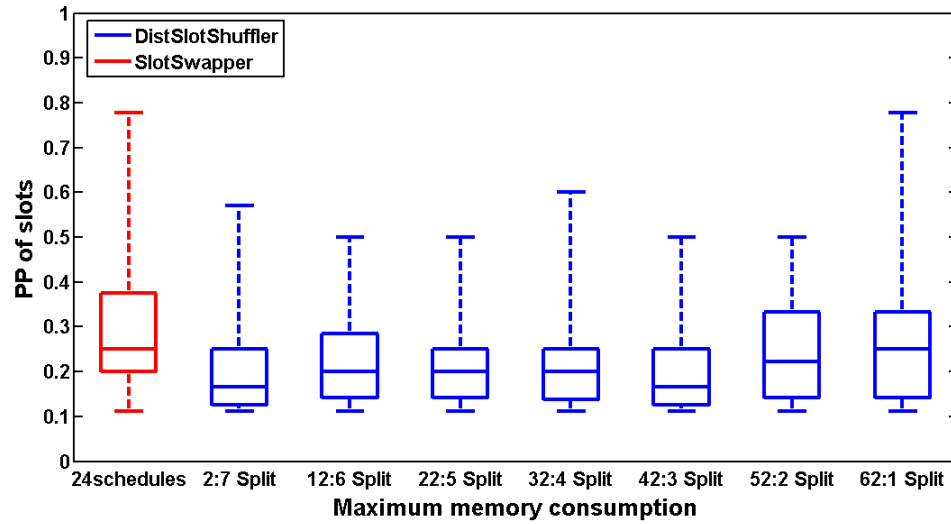
We also observe that the *SlotSwapper* performs better than the *DistSlotShuffler* for slot utilization of 33% and 50% with 0.12 and 0.14 less divergence on an average (refer Figures. 6.9(a),6.9(b)) when the number of base schedules available to the *DistSlotShuffler* is less than 4. This is because, although the number of keys for these cases is more than 42, the number of combination of slots available to each cluster is less due to less number of available base schedules in case of the *DistSlotShuffler*. However, at higher slot utilization of 66.6% and 75% (refer to Figures. 6.9(c),6.9(d)), the *DistSlotShuffler* performs far better with 0.05 and 0.14 less divergence on an average than the *SlotSwapper* even when the number of base schedules available to the *DistSlotShuffler* is very small. With increase in the utilization of slots in the schedules, the number of idle slots decreases which in turn reduces the scope of randomization in both the cases. However, with decrease in the number of base schedules for the *DistSlotShuffler*, the number of keys increases which in turn, increases the scope of randomization within each cluster thereby generating more random schedules compared to the *SlotSwapper*. Thus, the upper-bound K-L divergence value for the schedules generated by the offline centralized technique is more compared to the online distributed technique at higher slot utilization.



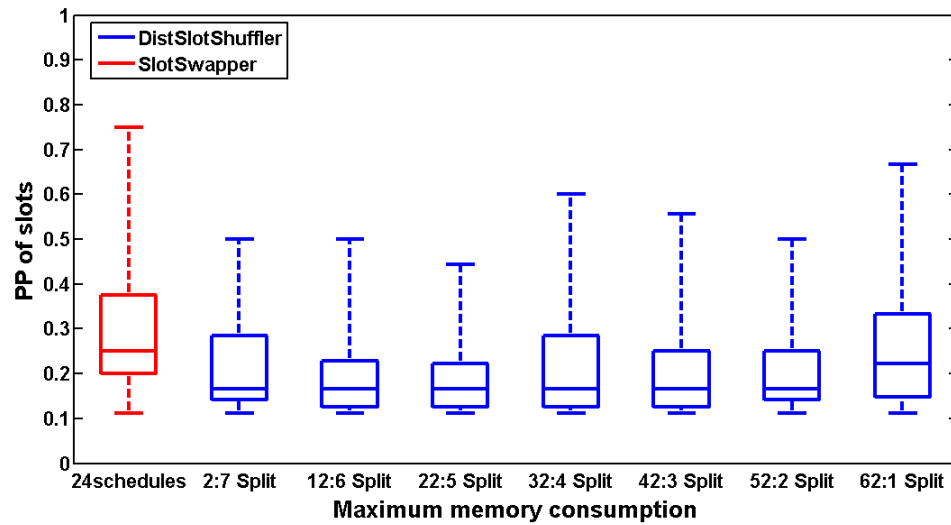
(a) slot utilization 33.3%



(b) slot utilization 50%



(c) slot utilization 66.6%



(d) slot utilization 75%

FIGURE 6.10: Prediction Probability of slots of (a) *SlotSwapper* (red) (b) *DistSlotShuffler* (blue) with slot utilization (i) 33.33% (ii) 50% (iii) 66.6% (iv) 75% over $hp = 60slots$. The X-axis shows the maximum memory consumed by the *SlotSwapper* (number of schedules) and *DistSlotShuffler* (number of keys: number of base schedules).

Evaluation by PP of slots: To compare the effectiveness of the random schedules generated by *DistSlotShuffler* and *SlotSwapper*, we collected the transmission schedules over 10 hyperperiods, i.e., over 600 slots at slot utilization of 33.3%, 50%, 66.6% and

75%. We calculated the PP of slots in the observed schedules based on the observations in the previous hyperperiods.

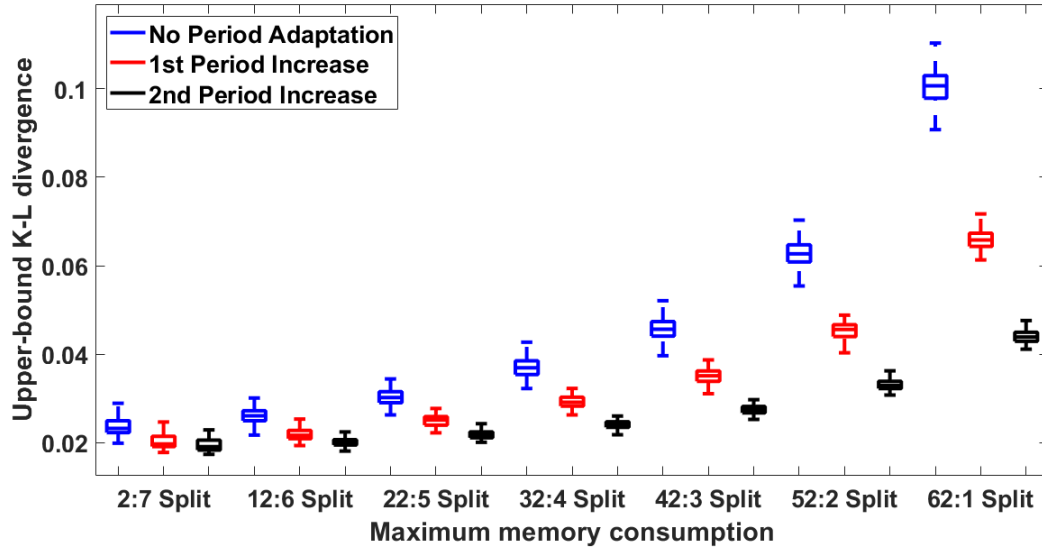
Figures. 6.10(a), 6.10(b), 6.10(c) and 6.10(d) show the non-zero prediction probabilities of the slots over 10 hyperperiod schedules (600 slots) generated by *DistSlotShuffler* (in blue) and *SlotSwapper* (in red) under slot utilization of 33.3%, 50%, 66.6% and 75% respectively. We observe that at slot utilization of 33.3%, the maximum PP of slots of the schedules generated by *SlotSwapper* is higher than that of *DistSlotShuffler*, when #base schedules ≥ 3 (Figure. 6.10(a)). Likewise, at slot utilization 50% and 66.6%, the maximum PP of slots of the schedules generated by *SlotSwapper* is higher than *DistSlotShuffler* when the #base schedules ≥ 1 . At slot utilization of 75%, the maximum PP of slots in the schedules generated by *SlotSwapper* is higher than *DistSlotShuffler* even with 1 base schedule. Besides, the mean PP of slots of the schedules generated by *SlotSwapper* is also higher than that of *DistSlotShuffler* for all key and base schedule combinations at slot utilization 66.6% and 75%. Hence, the schedules generated by *DistSlotShuffler* are less predictable than that of *SlotSwapper*, when #base schedules are high at lower slot utilization and under all conditions at higher slot utilization.

6.6.4 Performance after Period Adaptation

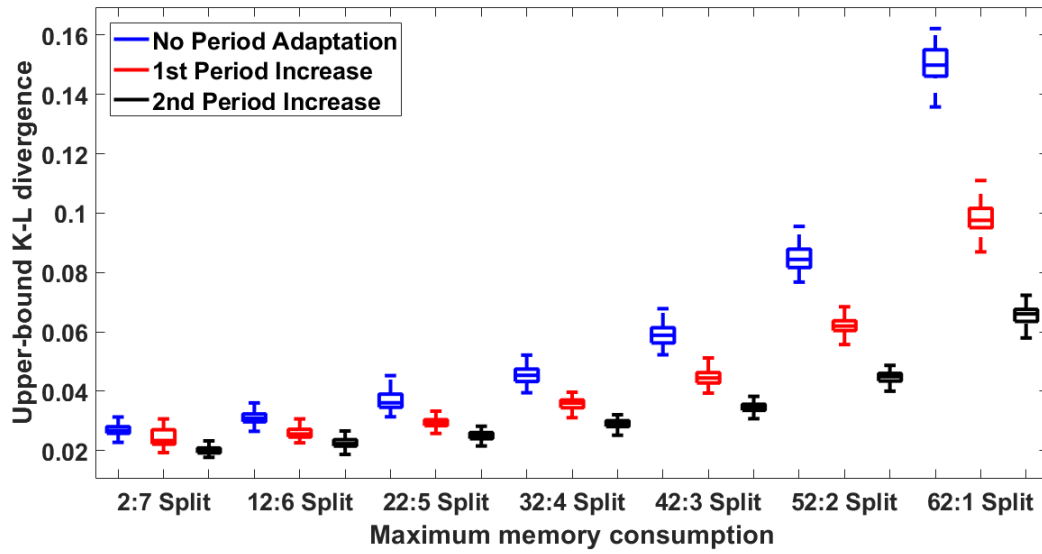
The plots under Figure. 6.6 show that the control performance is not affected under normal conditions on adopting period adaptation in our experiments. On adapting to a higher period, the scheduling window corresponding to the flow instances of control loop L_k increases and the slot utilization in the generated schedules decreases. It is to be noted that in the offline static schedule randomization technique, any change in the slots of the schedules involve recomputation of the schedules. Hence, it does not support period adaptation. In case of the online dynamic distributed schedule randomization technique, it can be observed from Figures. 6.9(a), 6.9(b), 6.9(c) and 6.9(d) that for a fixed number of flows, the upper-bound K-L divergence decreases with decrease in the slot utilization for every combination of the “number of keys” and the “number of base schedules”.

On adapting to a larger period, the scheduling window corresponding to all the flows of control loop L_k increases and the number of instances of that flow over a hyperperiod decreases. To compare the impact of period adaptation on upper-bound K-L divergence, we extend the hyperperiod of the schedules as the L.C.M. of the maximum periods of all control loops. Note that, this is done so that all the flow instances equally show

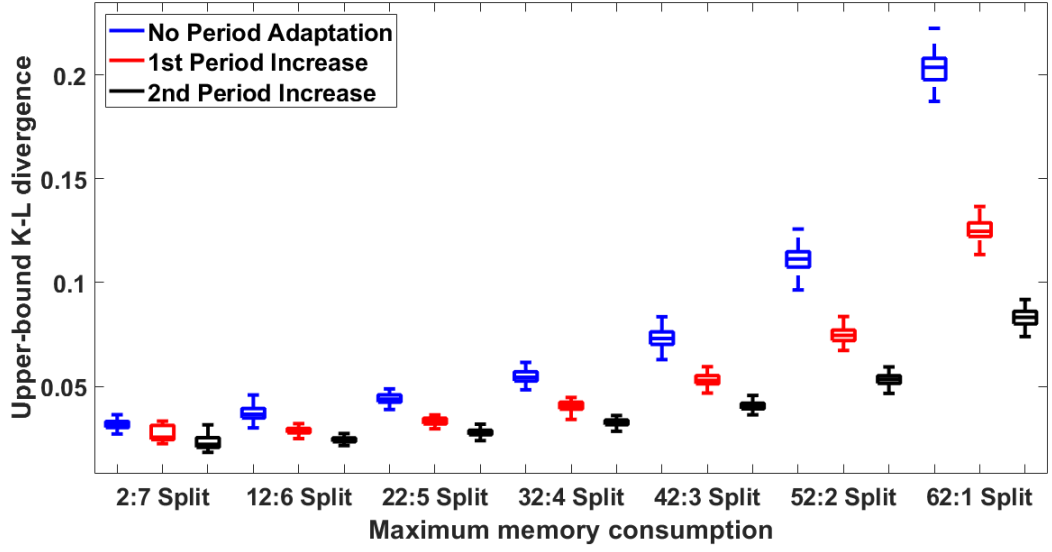
up in all the hyperperiod schedules. The length of the original hyperperiod hp is not affected by period adaptation as discussed in Section 6.5.5 of this chapter. In our experiments, we extend the hyperperiod upto 240 slots upon adopting period adaptation strategy. Figures. 6.11(a), 6.11(b) and 6.11(c) show the upper-bound K-L divergence of the *DistSlotShuffler* at three different slot utilization (25%, 50%, 75%) under normal condition (marked in blue) and after the first and second period increase due to period adaptation (marked in red and black respectively) for different combinations of keys and base schedules. From each of these figures, it can be observed that with increase in the period of the flows, the upper-bound K-L divergence decreases in each of these cases under different slot utilization. The lowest divergence values are observed for those cases where the number of base schedules is maximum. This is because, with increase in the number of base schedules, the extent of randomization increases, hence upper-bound K-L divergence decreases. With period increase, the slot utilization decreases which in turn further increases the extent of randomization, resulting in lower divergence. The difference in the divergence values increase as the number of base schedules decrease. The maximum difference in the divergence values (0.04, 0.07 and 0.1 decrease after first period adaptation and 0.02, 0.03 and 0.05 decrease after second period adaptation) is observed with 62 keys and 1 base schedule for all the three cases. This is because, with smaller number of base schedules, the extent of randomization in the slots of the schedules is less. However, after the first and second period increases due to period adaptation of the flows, the extent of randomization in the slots of the schedules increases due to longer scheduling windows, thereby decreasing the divergence values. ***Thus, apart from guaranteeing the control performance, period adaptation strategy also increases the uncertainty in the slots of the generated schedules making the system more resilient to timing attacks.***



(a) First and second Period Adaptation with initial slot utilization = 25%



(b) First and second Period Adaptation with initial slot utilization = 50%



(c) First and second Period Adaptation with initial slot utilization = 75%

FIGURE 6.11: The plots (blue) shows the upper-bound K-L divergence of the *DistSlotShuffler* under normal condition at slot utilization (a) 25% (b) 50% (c) 75%. The plots (red and black) show the same after first and second period increases with no disturbance. The X-axis shows maximum memory consumption represented as (number of keys: number of base schedules).

6.6.5 Power Analysis

We provide an overview of the estimated power consumed by all telosb motes in the WirelessHART network. We have measured the total estimated power using the TOSSIM simulator [35]. The motes operate at 1.8V and transmits signals at 0dBm. Table 6.4 shows the total power consumption of all the nodes in the network over 240 slots at different slot utilization on executing *SlotSwapper* and *DistSlotShuffler* at the wireless devices. Since, the schedules generated by the *SlotSwapper* are static schedules and *SlotSwapper* does not consider the interaction between the control loop and the real-time flows in the network, the *SlotSwapper* does not support period adaptation. On the other hand, the *DistSlotShuffler* supports period adaptation. Hence, we measure the power consumed by all the nodes on running *DistSlotShuffler* under — (a) normal condition operating with the minimum period, (b) after first period increase and (c) after second period increase.

In a WirelessHART network, the power consumed by the wireless devices is mainly incurred due to transmission/reception of packets in the network. The power consumed

Condition	Utilization	Total Power Consumption	
		<i>DistSlotShuffler</i>	<i>SlotSwapper</i>
Normal condition	25%	151.84mW	151.84mW + 18.57mW (every broadcast)
1 st period adaptation	12.5%	80.44mW + 18.57mW (period broadcast)	151.84mW + 18.57mW (every broadcast)
2 nd period adaptation	6.25%	44.76mW + 18.57mW (period broadcast)	151.84mW + 18.57mW (every broadcast)
Normal condition	50%	297.88mW	297.88mW + 18.57mW (every broadcast)
1 st period adaptation	25%	155.16mW + 18.57mW (period broadcast)	297.88mW + 18.57mW (every broadcast)
2 nd period adaptation	12.5%	83.8mW + 18.57mW (period broadcast)	297.88mW + 18.57mW (every broadcast)
Normal condition	75%	443.92mW	443.92mW + 18.57mW (every broadcast)
1 st period adaptation	37.5%	229.83mW + 18.57mW (period broadcast)	443.92mW + 18.57mW (every broadcast)
2 nd period adaptation	18.25%	122.78mW + 18.57mW (period broadcast)	443.92mW + 18.57mW (every broadcast)

TABLE 6.4: Total estimated power consumption of all the nodes over a hyperperiod of 240 slots executing the (1) *DistSlotShuffler* and (2) *SlotSwapper* at different slot utilization under (a) normal condition (b) after 1st period adaptation (c) after 2nd period adaptation

during computation operations is negligibly small. Hence, under normal condition, *i.e.*, with no period adaptation, the power consumed by *SlotSwapper* and *DistSlotShuffler* due to transmission/reception of packets is same in both the cases. An additional power of 18.57mW is consumed by the nodes at every broadcast of the schedules on executing the *SlotSwapper* at the network devices. Since, the *SlotSwapper* does not support period adaptation, the nodes continue to transmit/receive packets at the same period consuming the same amount of power over every hyperperiod. In case of the *DistSlotShuffler*, on adapting to a larger period, the number of transmissions reduces significantly which in turn reduces the total power consumed by all the nodes in the network. From Table 6.4 we found that for each period increase, the total transmission power reduces by approximately 46% on an average. However, an additional power of 18.57mW is consumed by the nodes to transmit the updated period in the network whenever period adaptation is triggered.

6.7 Conclusion

In this chapter, we proposed an online distributed schedule randomization strategy to mitigate timing attacks, such as selective jamming attacks in ICSs. Our schedule randomization strategy randomizes the slots in the hyperperiod schedules at each node in a distributed manner without violating the deadlines of the real-time flows while still guaranteeing the stability of the closed-loop controls in ICSs. We further adopted a period adaptation strategy that can adjust the periods of the flows based on the control performance at runtime. This further increases the scope of randomization of the slots in the schedules as well as reduces the power consumption of the system. We used Kullback-Leibler divergence as a metric to quantify the divergence in the solution space of our algorithm with respect to a truly random algorithm. We compared our online distributed schedule randomization strategy with the centralized offline schedule randomization strategy described in Chapter 5 of this thesis. We found that the online distributed schedule randomization strategy has performed much better than the offline centralized schedule randomization strategy at higher utilization ($\geq 66.6\%$). On adopting period adaptation strategy, the online distributed schedule randomization strategy produces at-least 0.01 less divergence and reduces the transmission power by at-least 46% on an average with no significant change in the control performance.

There are some scopes to improve the solution in the future. The current online distributed randomization strategy is implemented on a single-channel WirelessHART network. In the future, we would like to extend the distributed schedule randomization strategy for a multi-channel WirelessHART network. A multi-channel WirelessHART network will have collisions across the channels which will make the online schedule randomization harder.

Chapter 7

Online Schedule Randomization in 5G URLLC Communication

In this chapter, we present a novel online schedule randomization strategy as a countermeasure against timing attacks in 5G periodic ultra-reliable low-latency communication (URLLC). The URLLC communication service category of the fifth generation (5G) cellular network is expected to serve as one of the main communication standard in the future wireless sensor networks. A broad category of applications in industrial control systems (ICSs) are associated with periodic URLLC flows with strict deadlines. To satisfy the strict deadlines of the periodic URLLC flows, the base station (BS) pre-computes the communication schedule for such flows and repeats the same schedule over time. As a result, the slots in which the user equipment (UE) associated with periodic URLLC traffic communicates with the BS, becomes predictable over time. This provides an opportunity for the attacker to exploit the transmission slots and launch timing attacks in 5G networks. Our schedule randomization strategy tries to reduce the deterministic behavior of the schedule by randomizing the transmission slots and frequencies at runtime while ensuring the schedule feasibility.

In Section 7.1, we present the scheduling policy and the vulnerability of the scheduling policy for periodic URLLC traffic in 5G networks. In Section 7.2, we present some related works on attacks in 5G networks. Section 7.3 briefly describes our assumed network model. Section 7.4 briefly describes the threat model. In Section 7.5, we present the motivation for our proposed countermeasure. In Section 7.6, we describe our proposed countermeasure and the time complexity of running the proposed countermeasure.

In Section 7.7, we evaluate our proposed countermeasure against timing attacks. Section 7.8 concludes the chapter.

7.1 Introduction

In Chapter 2, Section 2.3.1, we described the main communication services in a 5G network. Among the three communication service categories in 5G networks, the URLLC service category is designed to support time-critical applications with very high reliability and very low latency. For example, ICSs fall under the URLLC service category, where each communication is typically associated with small packets requiring very high reliability (99.99%) and very low latency (1ms) [12]. Although 5G networks provide enhanced capacity to support a large number of UE belonging to different service categories, there are some pitfalls which can make the network vulnerable to attacks.

A typical 5G network consists of a set of UEs under the coverage area of a BS. As discussed in Section 2.3.3, the Third Generation Partnership Project (3GPP) has proposed grant-free access to serve the UEs with very low latency requirements [51]. A large number of applications in ICSs are periodic in nature with hard deadlines, implying that the packet transmissions associated with these applications are to be done within pre-specified time windows. Such network flows are known as periodic real-time URLLC flows as discussed in Section 2.3.3. For example, a closed-loop control application in motion control or factory automation involves URLLC services with deterministic sensing-actuation cycles of fixed period and very low latency and high reliability requirements [150]. As discussed in Section 2.3.3, the BS allocates dedicated radio resources (slots and frequencies) in advance for the UEs associated with periodic URLLC communication to provide guaranteed service within strict deadlines. As discussed in Section 2.3.3, the BS follows Semi-Persistent Scheduling (SPS) policy [51] for such UEs in which the BS computes the resource schedule in advance and informs the UEs about the generated schedules. Thus, all flow deadlines are met and the same schedule is repeated over time. However, due to this schedule repetition, the resource allocations used by each UE to transmit or receive packets become predictable. This information can then be exploited by an attacker to launch timing attacks, such as *selective jamming attacks*. An attacker can target a specific UE by listening to all the transmissions associated with it, and then be able to predict the slots in which specific radio resources will be

used by the UE. Thereafter, the attacker can selectively jam transmissions between the target UE and the BS in a stealthy manner and disrupt the system.

As a countermeasure against such attacks, in this chapter, we propose a novel *online schedule randomization strategy* that randomizes the slots and frequencies in which a UE transmits/receives data packets to/from the BS. This strategy ensures that the randomization does not impact the hard deadline requirements of periodic real-time URLLC flows, that is, it ensures schedule feasibility. The BS in 5G networks has sufficient resources to run this scheduling strategy periodically and compute the randomized schedules [80].

7.2 Related Works on other types of attacks in 5G Networks

5G is expected to serve as one of the main communication standards in future WSANs due to its support for high reliability and low latency. However, 4G LTE or 5G systems are not free from cyber attacks. Hussain *et al.* proposed privacy attacks in 4G/5G cellular paging protocol deployments and provided a countermeasure to such attacks by tracking the victim device within a geographical cell [151]. Fang *et al.* studied paging storm attacks where the attacker exhausts the network resource by injecting fake requests, thereby imposing denial of service attacks on legitimate users [152]. Roth *et al.* proposed a privacy attack where a vulnerability in the LTE system can be exploited by the attacker to get user location information [153]. Similar work in the context of LTE-A network using a feature called carrier aggregation has also been proposed [154]. Shaik *et al.* provided a brief overview of different vulnerabilities in 4G/LTE mobile communications that an attacker can exploit to launch attacks [155]. However, none of the above studies consider timing attacks, which are relevant for 5G periodic real-time URLLC traffic and the focus of this thesis.

7.3 System Model

In this chapter, we assume that our network model is represented by the 5G Network Model introduced in Chapter 2, Section 2.3.4. The transmissions in 5G are organized

into frames of $10ms$ duration which are sub-divided into sub-frames and slots as discussed in Section 2.3.2. The number of sub-frames per frame is represented by N_{sf} and the frame structure allows different sub-carrier spacing (SCS) to support different latencies as described in Section 2.3.2. The UEs in our system are assumed to be either static or mobile within a bounded range and is served under a single BS for the entire duration of service for which it is active. However, we allow admission of UEs in our system, implying a UE can join or leave the network at any point in time as assumed in Section 2.3.4. The UEs in our system operate at Frequency Division Duplexing (FDD) mode and each UE is associated with a set of operating frequencies as in Section 2.3.4. Based on these assumptions, we consider a group of n UEs, represented by $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$, that are at one hop distance from the BS B and operating with the same SCS as described in Section 2.3.4. The UE group require periodic URLLC service with hard deadline, use the same set of frequencies and associated with either an uplink or downlink flow of data as described in Section 2.3.4. The flows associated with the UEs are represented by the set \mathcal{F} introduced in Section 2.1. Hence, the flow parameters src_i , des_i , p_i , δ_i and $route_i$ have the same semantics as in Section 2.1. Each flow in our network is associated with one additional parameter e_i to represent the number of transmissions associated with F_i as considered in Section 2.3.4. The data size is assumed to be sufficiently small [53] to get transmitted in one slot. To support the low latency requirements, we assume that the UEs do not wait for acknowledgements [54]. Hence, one re-transmission slot is allocated for each data transmission as described in Section 2.3.4. Therefore, we have $e_i = 2$ (one slot for transmission and another slot for re-transmission of data) for all $F_i \in \mathcal{F}$.

As described in Section 2.3.4, our assumed system model also assumes that this UE group only uses α fraction of the sub-frames per frame in each of the m frequencies for periodic URLLC flows where $\alpha \geq 0.1$ and $\alpha \in [0.1, 1]$. Since, these α sub-frames are not uniformly distributed within a frame, we also assume that the flow periods are aligned at the frame boundaries as in Section 2.3.4. Hence, the minimum period of a flow in our system is of $10ms$ as assumed in Section 2.3.4. A feasible schedule \mathcal{S} for a 5G network with n periodic URLLC flows, m operating frequencies and α sub-frames per frame in each frequency has been defined in Definition 2.6 in Section 2.3.4 of Chapter 2.

7.4 Threat Model

In this chapter, we consider a threat model with the assumptions as in Chapter 4, Section 4.1. Our attacker also follows two additional assumptions specific to 5G networks as described under “Additional Assumptions” in Section 4.3. On the basis of the following assumptions, our attacker has some capabilities as described in Section 4.3 using which the attacker launches the attack. The attacker follows the three steps as described in Section 4.3 to launch the attack in the 5G network. Example 4.2 in Chapter 4 illustrates how an attacker can launch selective jamming attack in 5G networks. A discussion on the consequences of selective jamming attack in 5G networks has been provided in Section 4.4.

7.5 Motivation of our work in 5G

For a multi-channel WirelessHART network, illustration of timing attacks such as selective jamming attacks has already been presented [17]. In Chapter 5, we have already proposed a countermeasure against such attacks. Although the countermeasure proposed in Chapter 5 consider real-time flows with hard deadlines, the proposed countermeasure is an offline strategy. Moreover, the network in a WirelessHART is static and the schedules are determined at design time for a fixed number of flows. In contrast, 5G networks are highly dynamic in nature where a UE can join or leave the network at any point in time and the number of flows in the network also vary at runtime. Besides, only a portion of the network resources are typically allocated for periodic real-time URLLC flows in 5G. The remaining resources are reserved for other types of flows (sporadic URLLC, eMBB etc). Hence, any randomization strategy needs to guarantee flow deadlines at runtime using the partially allocated resources and with network dynamicity, which makes the schedule randomization problem harder in 5G.

7.6 Proposed Countermeasure for Attack

As a countermeasure to selective jamming attacks, we propose an *online schedule randomization strategy*, called *Period based Randomization (PerRand)* that considers a set

of n periodic real-time URLLC flows to be scheduled over m frequencies using α fraction of the sub-frames per frame in each frequency. It generates a *random schedule* \mathcal{S} of duration hp , satisfying the feasibility constraints of a schedule (refer Definition 2.6 in Section 2.3.4 of Chapter 2), in each hyperperiod.

7.6.1 Partitioning Scheduling Windows and Admission Control

In this section we adapt an existing technique from the real-time task scheduling domain to partition the scheduling window of each flow instance into non-overlapping sub-windows. This will not only facilitate scheduling (for property 1 in Definition 2.6 in Section 2.3.4 of Chapter 2), but also provide an efficient test which can be used for online admission control of new flows.

The flow scheduling problem is similar to the problem of scheduling periodic real-time tasks on a multi-core platform. Cores can be regarded as frequencies, tasks as flows and the schedule is required to satisfy Definition 2.6. Baruah *et al.* proposed a P-fair (proportionate fairness) scheduler to schedule n tasks on a system with m cores [156]. The main idea of this scheduler is to decompose the scheduling window of a task (flow) instance into non-overlapping sub-windows, one for each unit of execution (transmission slot). This ensures that the executions (transmission slots) of a task (flow) will never be scheduled on two cores (frequencies) at the same time. Techniques have been proposed to compute these sub-windows offline and deterministically schedule the resulting sub-tasks (sub-flows) online, while satisfying all task (flow) deadline requirements. A test to determine the feasibility for scheduling such tasks (flows) has also been derived.

In this paper, we adapt the above results for the admission control of new flows (*i.e.*, use the test), and to determine scheduling sub-windows for the admitted flows. We then develop a randomized scheduling strategy which satisfies Definition 2.6, while ensuring that each slot for each flow instance is only scheduled within its allocated sub-window. Thus, the transmissions of any flow F_i are divided into a sequence of unit-slot *sub-flows*, such that each sub-flow is allocated a dedicated and non-overlapping sub-window. Consider the j^{th} instance of flow F_i . Note, this instance requires two transmission slots ($e_i = 2$) in the scheduling window $W_{ij} = [r_{ij}, r_{ij} + p_i)$. We divide F_{ij} into two sub-flows, f_{ij} and \hat{f}_{ij} with scheduling sub-windows defined as follows.

Definition 7.1 (Pseudo-release/Pseudo-deadline). Pseudo-release and pseudo-deadline denote the start and end times, respectively, of the scheduling sub-windows. Sub-flow f_{ij} has a sub-window $[r_{ij}, \hat{\delta}_{ij})$ with pseudo-release r_{ij} and pseudo-deadline $\hat{\delta}_{ij} = \left\lfloor \frac{r_{ij} + \lfloor \frac{p_i}{2} \rfloor}{10} \right\rfloor \times 10$. Similarly, \hat{f}_{ij} has a sub-window $[\hat{\delta}_{ij}, \delta_{ij})$ with pseudo-release $\hat{\delta}_{ij}$ and pseudo-deadline $\delta_{ij} = r_{ij} + p_i$.

Note, the scheduling window is split into two approximately equal-sized sub-windows. To align all pseudo-deadlines with a frame boundary (multiple of 10), $\hat{\delta}_{ij}$ is defined as shown in Definition 7.1. This is applicable for flows with $p_i \geq 20ms$. Our assumed System Model in Section 7.3, derived from Section 2.3.4 of Chapter 2 already assumes that δ_{ij} is aligned with a frame boundary. As mentioned in the network model in Section 2.3.4 of Chapter 2.3.4, this alignment gives flexibility in the distribution of α fraction of the sub-frames within each frame without affecting deadline constraints. Note that the flows with $p_i = 10ms$, i.e., with a single frame within its scheduling window, cannot be split into equal sized sub-windows due to random allocation of α sub-frames inside each frame. Hence, flows with $p_i = 10ms$ are handled separately and is discussed in Section 7.6.5 of this chapter.

Example 7.1. Consider a flow F_i with $p_i = 30ms$. Flow instance F_{i1} consists of two sub-flows f_{i1} and \hat{f}_{i1} , with pseudo-releases $r_{i1} = 0ms$ and $\hat{\delta}_{i1} = 10ms$ and pseudo-deadlines $10ms$ and $\delta_{i1} = 30ms$, respectively. As one can see, $\hat{\delta}_{i1}$ has a value of $10ms$ instead of $15ms$ to align it with a frame boundary.

Admission control test. To enable online admission control of new flows in the system, it is necessary to have a time-efficient test that can evaluate whether a schedule satisfying Definition 2.6 in Section 2.3.4 of Chapter 2 can be generated for a set of flows. In the following lemma we derive such a test, adapting an existing test for the scheduling of real-time tasks on multi-core platforms.

Lemma 7.1. *Given a set of n periodic real-time URLLC flows \mathcal{F} , m frequencies with α fraction of sub-frame per frame in each frequency, and N_{sf} slots per sub-frame, there exists a feasible schedule using the partitioning technique of Definition 7.1 and satisfying Definition 2.6 if the following condition holds.*

$$\sum_{i=1}^n e_i / p'_i \leq \alpha \times N_{sf} \times m, \quad (7.1)$$

where $p'_i = p_i$ if $p_i/10$ is even and $p'_i = p_i - 10$ otherwise.

Proof. The statement of this lemma has been shown to be true for the case when $\alpha = 1$ and p'_i is the period of flow F_i for all i . This can be obtained directly from Theorem 1 of [156] by mapping a unit of task execution to a transmission slot of a flow, and by observing that from Definition 7.1 $\hat{\delta}_{ij} = r_{ij} + \lfloor \frac{p_i}{2} \rfloor$ when $p_i/10$ is even and $\hat{\delta}_{ij} = r_{ij} + \lfloor \frac{p_i-10}{2} \rfloor$ otherwise. Since $p'_i \leq p_i, \forall i$, it follows that if Eq. (7.1) holds for p'_i , then it also holds for p_i . This proves the lemma for the case when $\alpha = 1$ and p_i is the period of flow F_i for all i .

For $\alpha < 1$, it is easy to see that the lemma holds provided all pseudo-releases and pseudo-deadlines are aligned with frame boundaries (true for our case). This is because the available number of sub-frames in each frequency is guaranteed to be α fraction of m for any time duration starting and ending at frame boundaries. As such, this problem is equivalent to the case when p_i and frame duration are both scaled by α . \square

7.6.2 Schedule Randomization Strategy: *PerRand*

Algorithm 11: *PerRand*(\mathcal{F}, m, α)

```

1 if Eq. (7.1) is satisfied then
2   PreProcess( $\mathcal{F}, m, \alpha$ );
3   for current time = 0,  $hp$ ,  $2 \times hp$ , ... do
4     if a flow  $F'$  joins the network then
5       // online admission control phase
6       Admission_Control( $\mathcal{F}, F', m, \alpha$ );
7     if a flow  $F'$  leaves the network then
8       // online admission control phase
9       Admission_Control( $\mathcal{F}, F', m, \alpha$ );
10    // Online schedule generation phase
11     $\mathcal{S} = \text{GenSched}(\mathcal{F}, m, \alpha)$ ;

```

Our schedule randomization strategy has three phases all of which run at the base station B :

1. An offline feasibility checking phase which runs only once during network initialization. In this phase, B checks using Eq. (7.1) whether there exists a feasible schedule for the initial set of flows. If the condition is satisfied, then B runs pre-processing steps to initialize some data structures.

2. An online admission control phase that runs only when a UE joins or leaves the network. This will re-run the test and update the data structures accordingly.
3. An online schedule generation phase that runs at the beginning of each hyperperiod and generates a random feasible schedule.

Algorithm 11 shows the basic steps of our algorithm. If the feasibility condition in line 1 of Algorithm 11 is not satisfied, then B does not have enough resources to serve the flows in \mathcal{F} . Otherwise, we proceed to the pre-processing step (line 2), followed by the admission control (line 4) and online schedule generation (line 8) phases. *Admission_Control* runs only at the beginning of a hyperperiod if a flow joins or leaves the network.

Algorithm 12: *Preprocess*(\mathcal{F}, m, α)

```

1 Generate  $A$ ; // randomly select  $\alpha$  fraction of sub-frame per
   frame in each frequency over duration  $hp$ 
2  $\mathcal{F} = \text{Sort the flows in } \mathcal{F} \text{ in increasing order of their periods;}$ 
3 for ( $i = 1, 2, \dots, n$ ) do
4   for  $j = 1, 2, \dots, hp/p_i$  do
5     Compute and add  $r_{ij} * N_{sf}$  and  $\hat{\delta}_{ij} * N_{sf}$  to  $PR(i, j)$ , and  $\hat{\delta}_{ij} * N_{sf}$  and
        $\delta_{ij} * N_{sf}$  to  $PD(i, j)$ ;
6 Generate  $U$  over duration  $hp$ ;
   // Dictionary to store total number of required slots
   for flows in  $\mathcal{F}$ , with key equal to pseudo-deadlines

```

Algorithm 12 presents the offline pre-processing steps which run only once during network initialization. A list A consisting of (slot, frequency) pairs is generated by randomly selecting α fraction of the sub-frame per frame in each frequency over the duration hp (line 1). The flows in \mathcal{F} are sorted in ascending order of their periods (line 2). The pseudo-releases and pseudo-deadlines of the sub-flows are computed using Definition 7.1 and they are in ms . However, the duration of a slot in 5G NR depends on the SCS. Since the pseudo-deadlines are at frame boundaries, all pseudo-deadlines coincide with the time at which the last slot of the frame ends. Hence, we multiply all pseudo-deadlines and pseudo-releases by N_{sf} to represent them in terms of the number of slots from the beginning of the hyperperiod and store them in two lists PD and PR respectively (line 5). A dictionary called U is generated to store the total number of transmission slots required by all the flows in \mathcal{F} . This dictionary uses the pseudo-deadlines of flows as keys. For each key, say t , $U[t]$ stores the total number of slots needed by the sub-flows in \mathcal{F} with pseudo-deadlines equal to or lower than t (line 6).

$U[t]$ is used by the online schedule generator to track the number of slots available for scheduling sub-flows with pseudo-deadlines greater than t , in the slots preceding t .

Algorithm 13, *Admission_Control*, uses the test in Eq. (7.1) to determine flow set feasibility. We only allow a flow to join the network at the beginning of a hyperperiod, to ensure that the admitted flows are always guaranteed to meet deadlines. However, a flow can leave the network at any point in time.

Algorithm 13: *Admission_Control*($\mathcal{F}, F', m, \alpha$)

```

1 if  $F'$  left the network then
2    $\mathcal{F} = \mathcal{F} \setminus \{F'\}$ ;
3   Remove all pseudo-releases and pseudo-deadlines of  $F'$  from  $PR$  and  $PD$ ;
4 if  $F'$  joined the network then
5   if Eq. (7.1) is satisfied for  $(\mathcal{F} \cup \{F'\})$  then
6      $\mathcal{F} = \mathcal{F} \cup \{F'\}$ ;
7     Compute hyperperiod for  $\mathcal{F}$  and update  $hp$ ;
8     Add pseudo-releases and pseudo-deadlines of  $F'$  to  $PR$  and  $PD$  (lines 4-5 of
      Algorithm 2);

```

Algorithm 14, *GenSched*, presents the online random schedule generation phase. For each sub-flow of any instance of F_i , an eligible list of (slot,frequency) pairs ordered by slot numbers, called *elig*, is generated by considering all available (slot,frequency) pairs between its pseudo-release (*rel*) and pseudo-deadline (*dead*) (line 8). $[rel, dead)$ denotes the sub-window of the sub-flow of F_i . A random (slot,frequency) pair, (s_r, fr_r) from *elig*, is assigned to the sub-flow of F_i if there are sufficient slots to schedule the other sub-flows with pseudo-deadlines in $[rel, dead - 1)$ (based on the values in dictionary U). Function *Assign_elig*() shows the (slot,frequency) assignment procedure. If the randomly selected slot s_r has no pseudo-deadline between s_r and $dead - 1$, then s_r can be directly assigned to the sub-flow of F_i without updating U . However, if there is any pseudo-deadline between s_r and $dead - 1$, then the condition in line 24 is checked for all such pseudo-deadlines. If the condition is satisfied for all such pseudo-deadlines, then all entries in U between s_r and $dead - 1$ are updated to account for the transmission of sub-flow of F_i (line 33-34). Otherwise, a random (slot,frequency) pair is searched in a smaller sub-window after updating the list *elig* (line 36). We illustrate the steps of *GenSched* with an example.

Example 7.2. We consider the same flow setting as in Example 4.2 in Chapter 4. We consider three uplink flows F_1 , F_2 , and F_3 (marked in red, blue and brown respectively

Algorithm 14: $GenSched(\mathcal{F}, m, \alpha)$

```

1 Initialize  $\mathcal{S}$  to the empty schedule over  $hp$ ;
2 for ( $i = 1, 2, \dots, n$ ) do
3   for ( $t = 1, 2, \dots, hp * N_{sf}$ ) do
4     if ( $\exists j : t = PD(i, j)$ ) then
5        $rel = PR(i, j)$ ;
6        $dead = PD(i, j)$ ;
7        $elig = \{\}$ ; // empty list
8       Add all (slot, frequency) pairs in  $A$  between  $[rel, dead)$  to  $elig$ ;
9        $Assign\_elig(elig, i, rel, dead)$ ;
10 return  $\mathcal{S}$ ;
11 Function ( $Assign\_elig(elig, i, rel, dead)$ )
12  $done = false$ ;
13 while ( $NOT\ done$ ) do
14    $(s_r, f_r) = random(elig)$ ; // random (slot, frequency) pair
15   Set two flags,  $iter_1$  and  $iter_2$  to 0;
16   for ( $s = s_r, s_r - 1, \dots, rel$ ) do
17     if ( $s$  is a key in  $U$ ) then
18       Assign  $s$  to lower and  $U[s]$  to  $uprev$ ;
19       break;
20   for ( $s = s_r, s_r + 1, \dots, dead - 1$ ) do
21     if ( $s$  is a key in  $U$ ) then
22        $avail = (s - lower) * \alpha * m$ ;
23        $ucur = U[s]$ ;
24       if ( $avail - (ucur - uprev) > 0$ ) then
25         Set  $iter_1$  to 1;
26       else
27         Assign  $s$  to upper and set  $iter_2$  to 1;
28         break;
29   if ( $iter_2 == 0$ ) then
30     Assign  $(s_r, f_r)$  to  $F_i$  in  $\mathcal{S}$ ;
31     Remove  $(s_r, f_r)$  from  $elig$  and  $A$ ;
32      $done = true$ ;
33     if ( $iter_1 == 1$ ) then
34       Increment all entries in  $U$  between  $[s_r, dead)$  by 1;
35   else
36     Remove all (slot, frequency) pairs in  $[lower, upper)$  from  $elig$ ;
```

TABLE 7.1: Two schedules \mathcal{S}_1 and \mathcal{S}_2 over 6 frames and 2 frequencies with three uplink flows F_1 (red), F_2 (blue), F_3 (brown); source devices $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; destination devices $des_1 = des_2 = des_3 = B$; periods: $p_1 = 20ms$, $p_2 = 30ms$, $p_3 = 60ms$; The slot duration in the schedule is $1ms$.

Schedule	Freq.	Slots											
\mathcal{S}_1		2	5	14	18	27	29	31	32	48	49	54	58
	fr_1	F_1		F_2		F_1			F_3	F_1		F_2	
		1	3	12	15	25	28	31	32	43	49	52	55
	fr_2		F_1		F_2		F_1	F_3			F_1		F_2
\mathcal{S}_2		2	3	14	17	27	28	31	32	43	49	52	55
	fr_1		F_1	F_2		F_1	F_3		F_1				F_1
		5	7	16	18	23	29	32	35	41	48	55	59
	fr_2	F_2			F_1				F_2		F_1	F_2	F_3

in Table 7.1), corresponding to three UEs. The source devices are $src_1 = U_1$, $src_2 = U_2$, $src_3 = U_3$; All the flows have a common destination device, $des_1 = des_2 = des_3 = B$. Let $\alpha = 0.2$ and $N_{sf} = 1$. The pseudo-deadlines (in number of slots from the start of the hyperperiod) are $\{F_1 : 10, 20, 30, 40, 50, 60\}$, $\{F_2 : 10, 30, 40, 60\}$, $\{F_3 : 30, 60\}$. The (key:value) pairs in the dictionary are $U = \{10 : 2, 20 : 3, 30 : 6, 40 : 8, 50 : 9, 60 : 12\}$. Consider the generation of schedule \mathcal{S}_2 in Table 7.1 using *GenSched*.

Consider f_{11} , the first sub-flow of the first instance of F_1 with pseudo-release at slot 0 and pseudo-deadline at slot 10; $rel = 0$ and $dead = 10$. The eligible (slot,frequency) pairs for f_{11} in \mathcal{S}_2 are $elig = \{(2, fr_1), (3, fr_1), (5, fr_2), (7, fr_2)\}$. Consider function *Assign_elig()*. Let the randomly selected (slot,frequency) pair be $(3, fr_1)$. On scanning the slots from 3 to 0, we get $lower = 0$ and $uprev = 0$. On scanning the slots from 3 to 9, we do not get any (key:value) in U . Hence, $iter_1 = iter_2 = 0$. We assign $(3, fr_1)$ to f_{11} in \mathcal{S}_2 . In a similar manner, (slot,frequency) pairs are allocated to all the sub-flows of other instances of F_1 in \mathcal{S}_2 .

Consider \hat{f}_{21} , the second sub-flow of the first instance of F_2 with pseudo-release at slot 10 and pseudo-deadline at slot 30; $rel = 10$ and $dead = 30$. Note that all the sub-flows of F_1 and the first sub-flow of the first instance of F_2 are already scheduled in \mathcal{S}_2 . The eligible (slot,frequency) pairs for \hat{f}_{21} are given by $elig = \{(14, fr_1), (17, fr_1), (28, fr_1), (16, fr_2), (23, fr_2), (29, fr_2)\}$. Let $(14, fr_1)$ be the randomly selected (slot,frequency) pair. On scanning the slots from 14 to 10, we get $lower = 10$ and $uprev = 2$. On scanning the slots from 14 to 29, we get $upper = 20$ and $ucur = 3$. The total available slots between slot 10 and slot 20 are $avail = (20 - 10) * 0.2 * 2 = 4$. Since, the expression in line 24 of

GenSched is satisfied, i.e., $(4 - (3 - 2)) > 0$, we can allocate $(14, fr_1)$ to \hat{f}_{21} . However, we need to update the value corresponding to key 20 in U , i.e., $U[20] = 4$.

7.6.3 Time Complexity of *PerRand*

The time to check the feasibility condition in Eq. (7.1) for all $F_i \in \mathcal{F}$ is $\mathcal{O}(n)$ (line 1 of *PerRand*). The time to generate the list A over hp with m frequencies takes $\mathcal{O}(hp \times m)$ (line 1 of *PreProcess*). The time to sort all flows in \mathcal{F} in ascending order of their periods is $\mathcal{O}(n)$ (line 2 of *PreProcess*). For each $F_i \in \mathcal{F}$, the time to compute the pseudo-releases and pseudo-deadlines over hp and adding them to the lists PR and PD respectively takes $\mathcal{O}(hp)$. Since the process repeats over each flow in \mathcal{F} , the total time to generate PR and PD is $\mathcal{O}(hp \times n)$ (line 3-5 of *PreProcess*). For each pseudo-deadline in hp , each entry in U calculates the total number of sub-flows having pseudo-deadline at or before hp . The time taken to calculate each entry in U is $\mathcal{O}(hp \times n)$. Since we assume that the pseudo-deadlines occur only at frame boundaries, there are at-most $\frac{hp}{10}$ frames over hp . Therefore, the total time to generate U is $\mathcal{O}(n \times (hp)^2)$. Hence, the total time taken by *PreProcess* is $\mathcal{O}(n \times (hp)^2)$.

If a flow leaves the network, the time to update PR and PD in *Admission_Control* is $\mathcal{O}(hp)$ (line 3). If a flow joins the network, the time to check Eq. (7.1) in *Admission_Control* is $\mathcal{O}(n)$ (line 5). The time to compute the updated hyperperiod (line 7) and to add the entries of F' in PR and PD (line 8) takes $\mathcal{O}(hp)$ each. Hence, the total time complexity to run *Admission_Control* is $\mathcal{O}(n + hp)$.

For each F_i , the time to generate *elig* is $\mathcal{O}(hp \times m)$. For *Assign_elig()*, in the worst-case $p_i = hp$. Then, the time to select the previous (lines 16-19) and next (lines 20-28) pseudo-deadlines is $\mathcal{O}(hp)$. If the else condition in line 35 is satisfied, then the time to remove all the (slot, frequency) pairs from *elig* is $\mathcal{O}(hp \times m)$. Since the entire process runs over hp and is repeated for each flow in \mathcal{F} , the total time complexity of *GenSched* is $\mathcal{O}(nm(hp)^2)$.

Hence, the total time complexity to run *PerRand* is $\mathcal{O}(nm(hp)^2)$.

7.6.4 Online Schedule Distribution

During a hyperperiod, B can generate the schedule for the next hyperperiod and send the (slot,frequency) pairs to the UEs through the downlink control channel. Since the attacker has no access to the network encryption keys and it does not know the exact time at which the packet with scheduling information is transmitted, it cannot jam this packet.

In the following theorem we prove that any schedule generated by *PerRand* is guaranteed to be a feasible schedule.

Theorem 7.2. *Any schedule \mathcal{S} generated by *PerRand* satisfies the three properties of a feasible schedule (refer to Definition 2.6 in Section 2.3.4 of Chapter 2).*

Proof. Any feasible schedule for a set of periodic URLLC flows in a 5G network should satisfy three properties as described in Definition 2.6 in Section 2.3.4 of Chapter 2. We need to show that any schedule \mathcal{S} generated by *PerRand*, also satisfies these three properties of a feasible schedule.

Property 1: The scheduling window W_{ij} of a flow instance F_{ij} is partitioned into two non-overlapping sub-windows (line 5 of Algorithm 12), and each transmission of F_{ij} is scheduled within its sub-window (line 8 of Algorithm 14).

Property 2: The random (slot,frequency) pair is eliminated from A after being assigned to a transmission of F_{ij} (line 31 of Algorithm 14).

Property 3: The two transmissions of F_{ij} are always scheduled within their respective sub-windows (line 8 of Algorithm 14) which are part of the original scheduling window W_{ij} .

Thus, \mathcal{S} satisfies Definition 2.6 in Section 2.3.4 of Chapter 2. □

In the following theorem we prove that *GenSched* can always find a feasible schedule.

Theorem 7.3. *Line 30 of *GenSched* will always be executed for any call to *Assign_elig()*.*

Proof. Consider any sub-flow of F_{ij} , say f_{ij}/\hat{f}_{ij} . Let s_1 and s_2 be the pseudo-release and pseudo-deadline (in number of slots from the start of the hyperperiod) of f_{ij}/\hat{f}_{ij} . Let $\rho_1, \rho_2, \dots, \rho_t$ be t slots with pseudo-deadlines of other sub-flows, such that $s_1 \leq \rho_1, \rho_2,$

$\dots, \rho_t < s_2$. Let s_r be a randomly selected slot along any frequency to schedule the transmission of f_{ij}/\hat{f}_{ij} (line 14 of *GenSched*). Note, the required slots for f_{ij}/\hat{f}_{ij} have already been considered in $U[s_2]$ (Algorithm 12). If $\rho_t < s_r < s_2$, then there is no pseudo-deadline in $[s_r, s_2)$, i.e., the condition in line 21 fails. In this case, f_{ij}/\hat{f}_{ij} can be scheduled in s_r . If $s_1 \leq s_r \leq \rho_t$, then the transmission of f_{ij}/\hat{f}_{ij} has not been considered in any $U[\rho_x]$, where $\rho_x \in [s_r, \rho_t]$ and $1 \leq x \leq t$. The pseudo-deadline immediately before s_r is stored in variable *lower* (line 18). For each ρ_x , line 22 computes the total number of slots between *lower* and each ρ_x . For each ρ_x , the condition at line 24 holds true if there is at-least one available slot in $[lower, \rho_x)$ after reserving slots for all sub-flows (may or may not be scheduled yet) with pseudo-deadline at ρ_x . If one such slot exists, then f_{ij}/\hat{f}_{ij} will be scheduled in s_r . Otherwise, random slots are searched in a smaller sub-window, $[\rho_x, s_2)$ and the process is repeated until the sub-window $[\rho_t, s_2)$ is reached, where f_{ij}/\hat{f}_{ij} will surely get a slot as shown below.

Consider any sub-flow say f_{xy}/\hat{f}_{xy} with pseudo-deadline after s_2 that was scheduled in $[\rho_t, s_2)$ before f_{ij}/\hat{f}_{ij} was considered in *GenSched*. Since the transmission of f_{ij}/\hat{f}_{ij} has already been considered in $U[s_2]$, the transmission of f_{xy}/\hat{f}_{xy} was scheduled only if a slot for f_{ij}/\hat{f}_{ij} is available in $[\rho_t, s_2)$. Line 23 accounts for the transmission of f_{ij}/\hat{f}_{ij} at the time of scheduling f_{xy}/\hat{f}_{xy} and line 24 ensures that at-least one slot is available to schedule the transmission of f_{ij}/\hat{f}_{ij} . Hence, f_{ij}/\hat{f}_{ij} can always get a slot in $[\rho_t, s_2)$. \square

7.6.5 Scheduling of flows with 10ms period

If \mathcal{F} contains flows with period 10ms, then such flows can be separately scheduled before executing *GenSched*. This is because such flows cannot be assigned pseudo-deadlines at a frame boundary. Two distinct slots along any frequency are required per frame for each such flow. Such slots can be randomly obtained from the α fraction of allocated slots. If the last such scheduled flow does not have two distinct slots in some frame, then it could swap a slot with another randomly selected 10ms flow. This is always feasible if $N_{sf} > 1$ or $\alpha > 0.1$. For the case when $N_{sf} = 1$ and $\alpha = 0.1$, there are only 2 slots per frame and then we need to ensure that those two slots are distinct. Note, we have implemented these aspects and considered such flows in our experiments.

7.7 Experiments and Evaluation

The main objective of *PerRand* is to generate a *random feasible schedule* \mathcal{S} in each hyperperiod such that the inherent predictability in the scheduled slots is obfuscated to an attacker. The uncertainty of a flow to occur in a specific slot and frequency comes from the random selection of (slot,frequency) pairs from the list *elig* in Algorithm 14. The more this uncertainty, the more difficult it becomes for an attacker to predict transmissions. In Chapter 2, Section 2.4, we introduced *Kullback-Leibler divergence* or *K-L divergence* to measure the uncertainty in the slots of the generated schedules. In this chapter, we use the same metric to measure the performance of our proposed randomization strategy in terms of its ability to randomize schedules in comparison to a hypothetical truly random strategy.

Definition 7.2. Given an algorithm \mathbb{A}_1 and a truly random algorithm \mathbb{A}_2 , both scheduling a set of $n+1$ periodic real-time URLLC flows ($\mathcal{F} \cup F_0$, where F_0 represents a virtual flow corresponding to the idle slots in the schedule) over a 5G network with m frequencies and α fraction of the sub-frames per frame in each frequency, the K-L divergence of \mathbb{A}_1 with reference to \mathbb{A}_2 measures the divergence in the probability distribution of flows across the slots and frequencies in the schedules generated by \mathbb{A}_1 with reference to \mathbb{A}_2 .

$$\mathcal{KL}(\mathbb{A}_1 || \mathbb{A}_2) = \sum_{j=1}^{hp} \sum_{k=1}^m \sum_{i=0}^n Pr(a_{jk} = i) \log \frac{Pr(a_{jk} = i)}{Pr(a'_{jk} = i)}$$

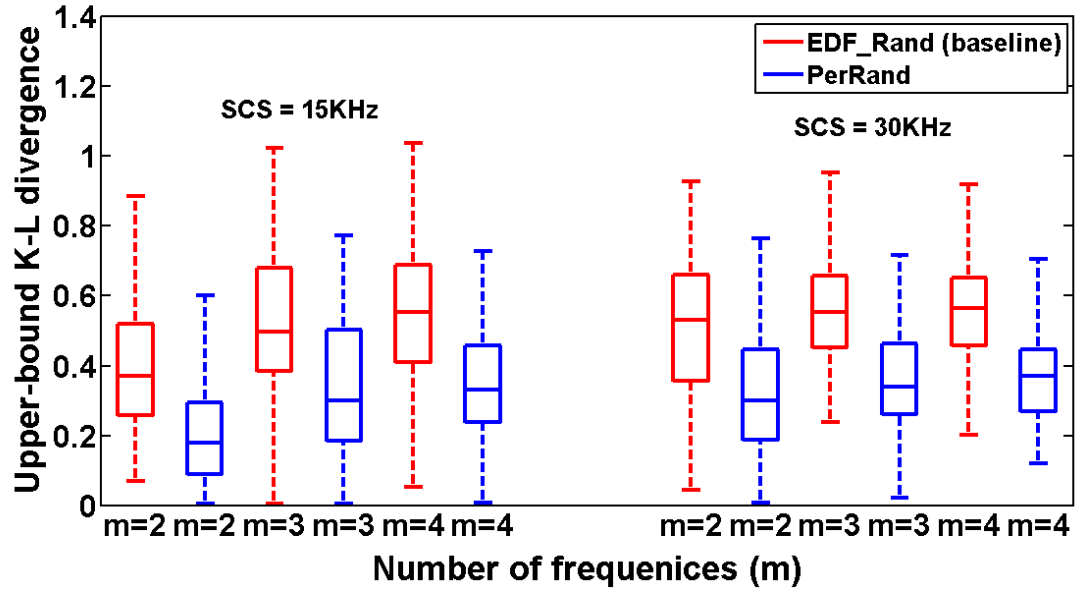
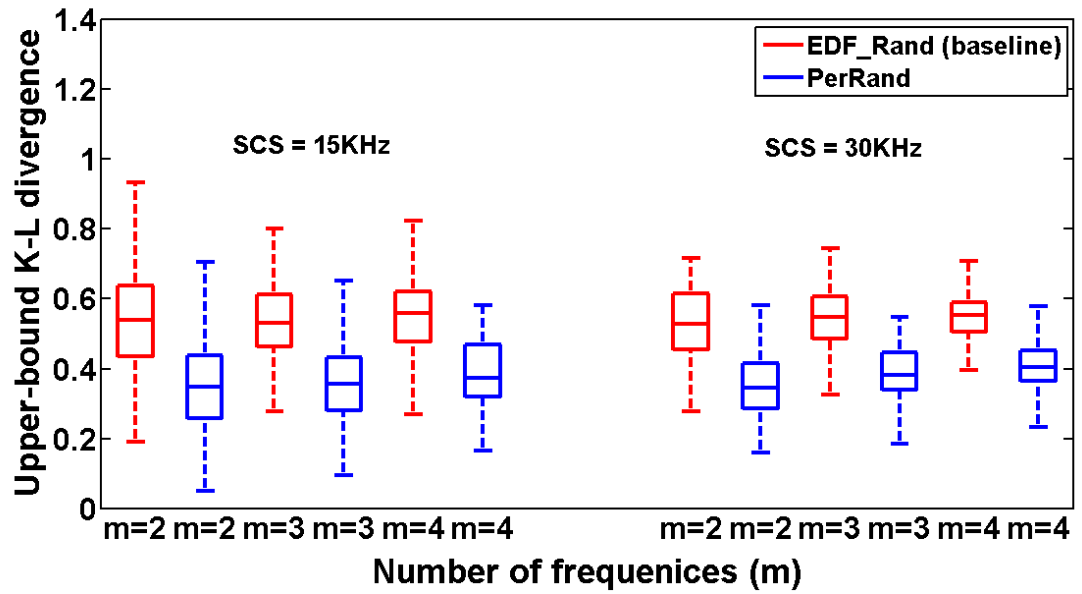
where a_{jk} and a'_{jk} are discrete random variables and $Pr(a_{jk}=i)$ and $Pr(a'_{jk}=i)$ are the probability mass functions of the i^{th} flow occurring in the k^{th} frequency of the j^{th} slot in the schedules generated by \mathbb{A}_1 and \mathbb{A}_2 , respectively.

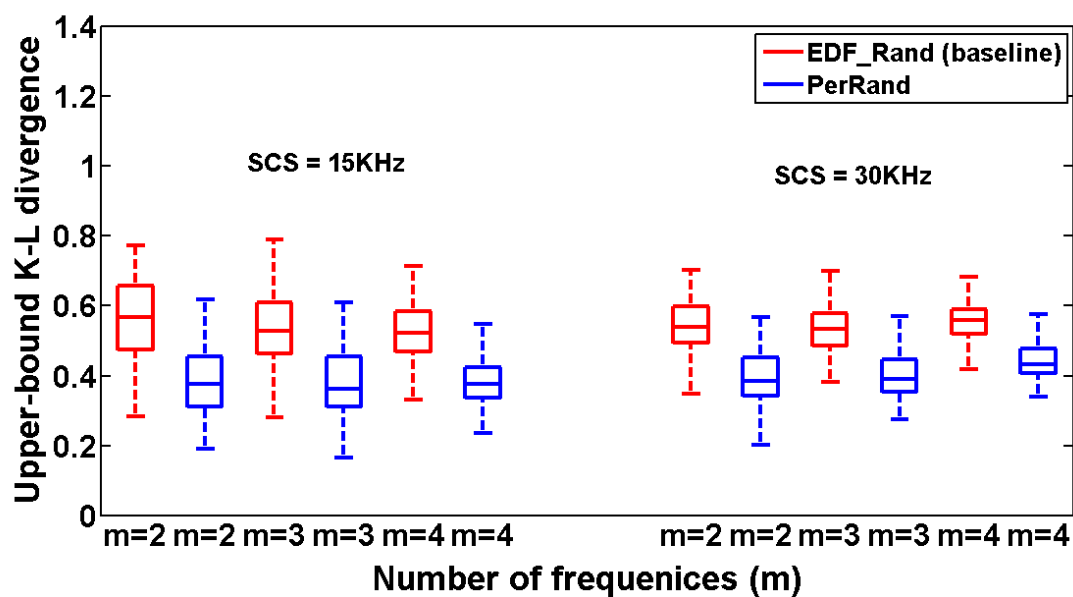
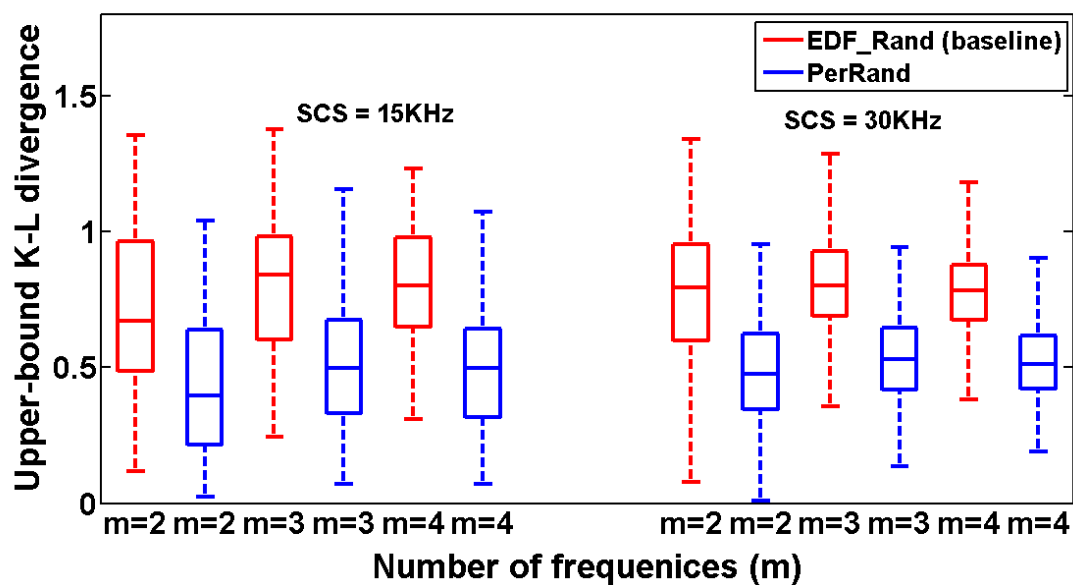
The computation of all possible feasible schedules for a given flow set is not feasible (we need to generate all possible schedules and then check for their feasibility). Hence, we calculate *upper-bound K-L divergence* instead, by ignoring schedule feasibility in algorithm \mathbb{A}_2 with probability mass function $Pr(a'_{jk}=i) = \frac{2}{\alpha * p_i} * \frac{1}{\alpha * hp * m}$ and by assuming F_i appears with this probability in every slot and frequency in the generated schedules. Note, an algorithm with lower value of K-L divergence is considered better as it indicates less divergence in the solution space of that algorithm with reference to the truly random algorithm.

In Chapter 5, Definition 5.4, we defined *Prediction Probability (PP) of slots* to measure the security provided by our randomization technique in terms of the attacker's ability to predict the transmissions in the slots of the schedules based on observations in the previous hyperperiod schedules. In this chapter, we use the same metric to perform security analysis of the schedules generated by our proposed randomization technique.

7.7.1 Experiments

In our experiments, we compared the performance of our algorithm, *PerRand*, with a simple baseline strategy, called *EDF_Rand*, that initially generates an earliest deadline first schedule based on pseudo-deadlines and then randomizes the allocated slots and frequencies only within windows that do not contain any pseudo-releases or pseudo-deadlines. This strategy is adopted from an existing work in the domain of periodic real-time task scheduling [19]. We use the upper-bound K-L divergence of each strategy against the reference algorithm \mathbb{A}_2 , to compare and contrast their performance. We also use Prediction Probability of slots in the schedules and compare the security provided by the schedules generated by the two algorithms. We have implemented these strategies in Python and performed the experiments on a Linux machine with 3.4GHz Intel Xeon having 12 processors and 16GB RAM. We conducted our experiments for two SCS, 15KHz ($N_{sf} = 1$) and 30KHz ($N_{sf} = 2$). For each SCS, we varied the number of frequencies m between 2 and 4, and $\alpha \in \{0.1, 0.3, 0.5\}$. For each value of m and α we generated 100 different slot allocations by randomly selecting α fraction of sub-frames per frame in each frequency. Finally, for each m and α , we varied the flow set utilization ($\sum_{i=1}^n e_i / (p_i * \alpha * N_{sf} * m)$) between (40%-70%) and (70%-100%) and generated 100 different flow sets for each utilization range (in total 3600 different flow sets). We selected the flow periods randomly between 10ms to 1000ms such that the hyperperiod of the flow set is no larger than 1000ms (typical hyperperiod values in ICS [58, 59]). To compare the performance of *PerRand* with *EDF_Rand*, we ran both the strategies for 1000 hyperperiods on each of these flow sets.

(a) Utilization: 40% – 70%, $\alpha : 0.1$ (b) Utilization: 40% – 70%, $\alpha : 0.3$

(c) Utilization: 40% – 70%, $\alpha : 0.5$ (d) Utilization: 70% – 100%, $\alpha : 0.1$

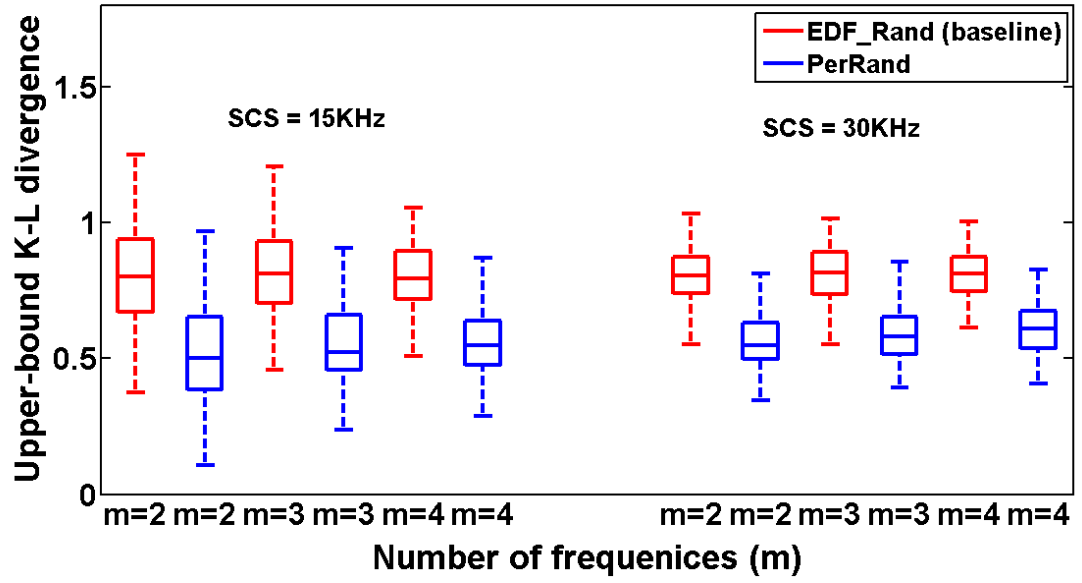
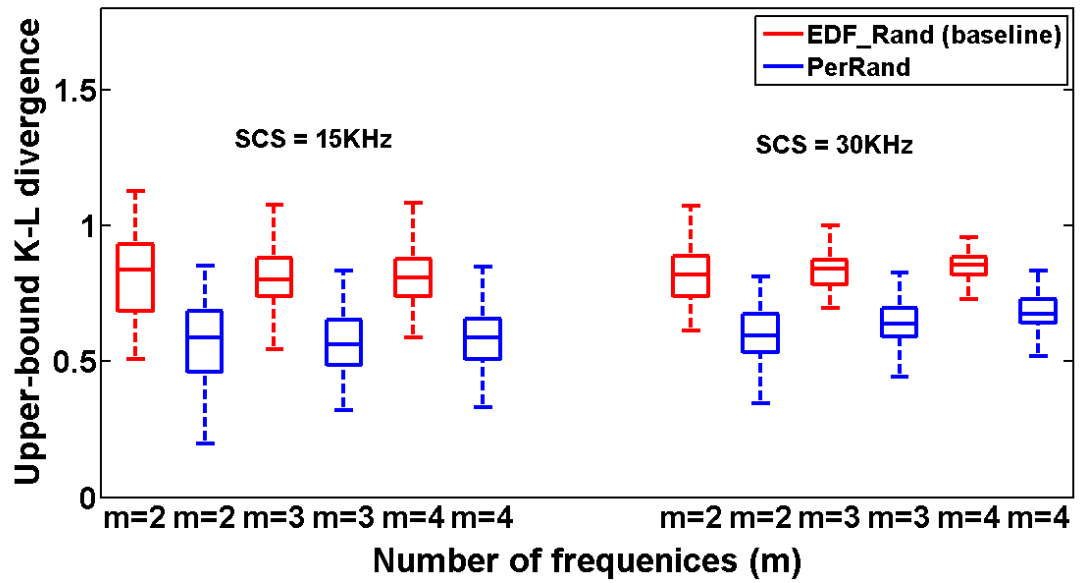
(e) Utilization: 70% – 100%, $\alpha : 0.3$ (f) Utilization: 70% – 100%, $\alpha : 0.5$

FIGURE 7.1: Upper-bound K-L divergence. SCS: 15KHz (graphs on the left in each sub-figure), 30KHz (graphs on the right). Divergence values of *EDF_Rand* and *PerRand* are marked in red and blue, respectively.

7.7.1.1 Evaluation by K-L divergence :

Figures. 7.1(a), 7.1(b), 7.1(c), 7.1(d), 7.1(e) and 7.1(f) show the upper-bound K-L divergence of the two strategies for various parameter settings. The plots in red (blue) represent the divergence values of *EDF_Rand* (*PerRand*). The plots to the left (right) show the divergence values for SCS 15KHz (30KHz) in all the sub-figures of Figure. 7.1. It can be observed that *PerRand* performs better than *EDF_Rand* under all parameter settings. The divergence values of *PerRand* are **25%** less than *EDF_Rand* on an average, with minimum and maximum gain in divergence being **23%** and **47%**, respectively, over all the settings.

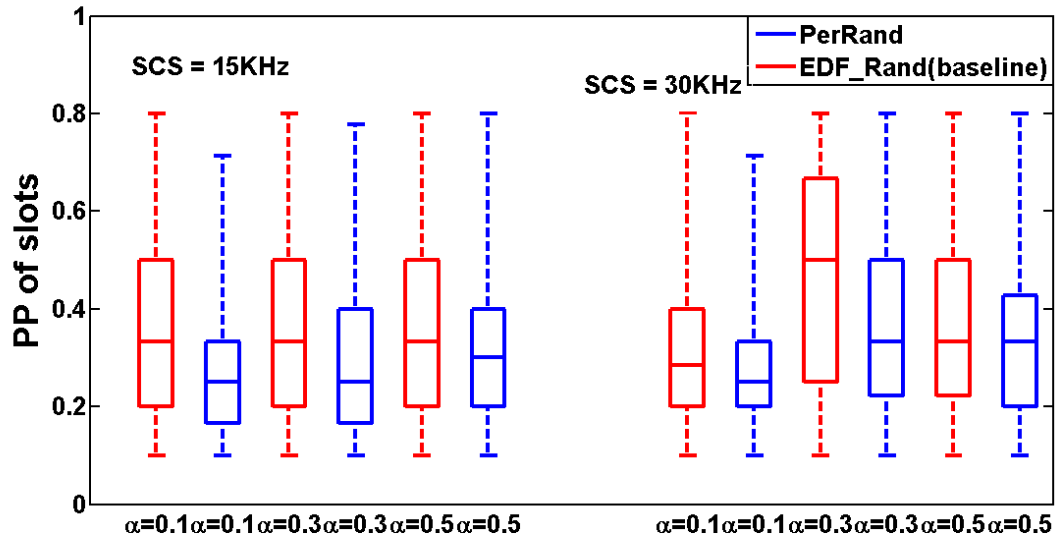
For a fixed SCS, α and flow set utilization, the number of available slots to schedule a flow increases with an increase in m . However, the number of flows also increase in parallel to keep utilization constant. To satisfy pseudo-deadlines, each sub-flow is scheduled within its sub-window, which reduces the extent of randomization possible. Hence, K-L divergence increases by 0.1 under 40% – 70% slot utilization at SCS=15KHz and by 0.06 under 70% – 100% slot utilization at SCS=30KHz on an average as the number of flows increase. Similarly, the K-L divergence increases by 0.08 under 40% – 70% slot utilization at SCS=30KHz and by 0.06 on an average as m increases (see each sub-figure of Figure. 7.1 under a fixed SCS and different m).

For a fixed m , SCS and utilization, the number of available slots per frame increases with an increase in α . The number of flows increase in parallel to keep utilization constant. Hence K-L divergence increases in this case as well (see plots under a fixed SCS and m in Figures. 7.1(a), 7.1(b), 7.1(c) and in Figures. 7.1(d), 7.1(e), 7.1(f)). However, the average divergence values increase by 0.1 and 0.03 on an average as we increase α at SCS=15KHz and 30KHz respectively, keeping other parameters fixed.

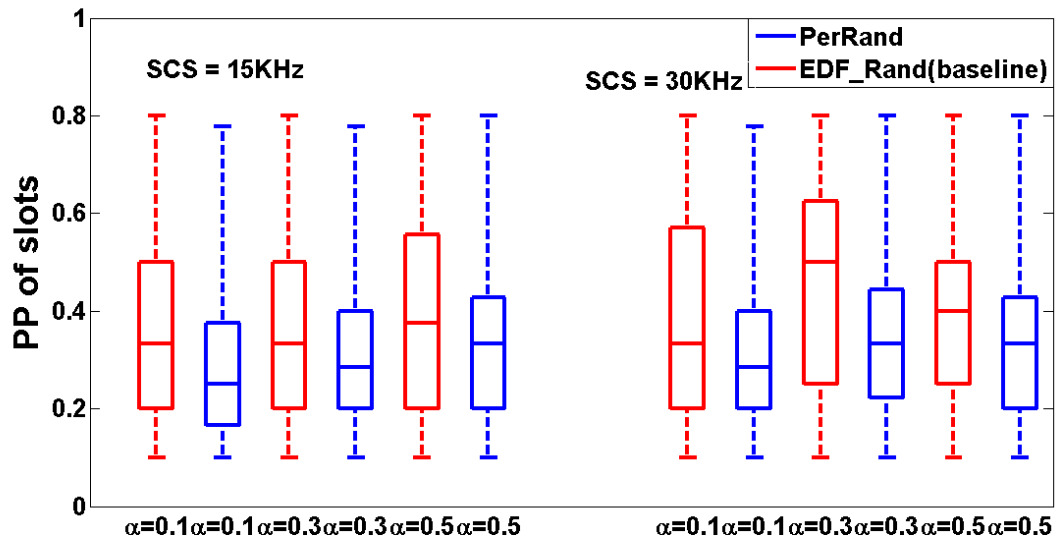
Similarly, for a fixed m , utilization and α , the number of slots per frame increases with an increase in SCS. To keep utilization constant, the number of flows increase, thereby increasing K-L divergence (see plots to the left and right for a fixed m in each sub-figure of Figure. 7.1). In this case, the divergence increases by 0.1 on an average for $m = 2$, keeping other parameters fixed and by 0.06 on an average for $m = 3$ and $m = 4$ keeping other parameters fixed.

Finally, for a fixed m , SCS and α , the number of flows increase with an increase in utilization, which increases K-L divergence (see plots for a fixed m , SCS in Figs. 7.1(a)

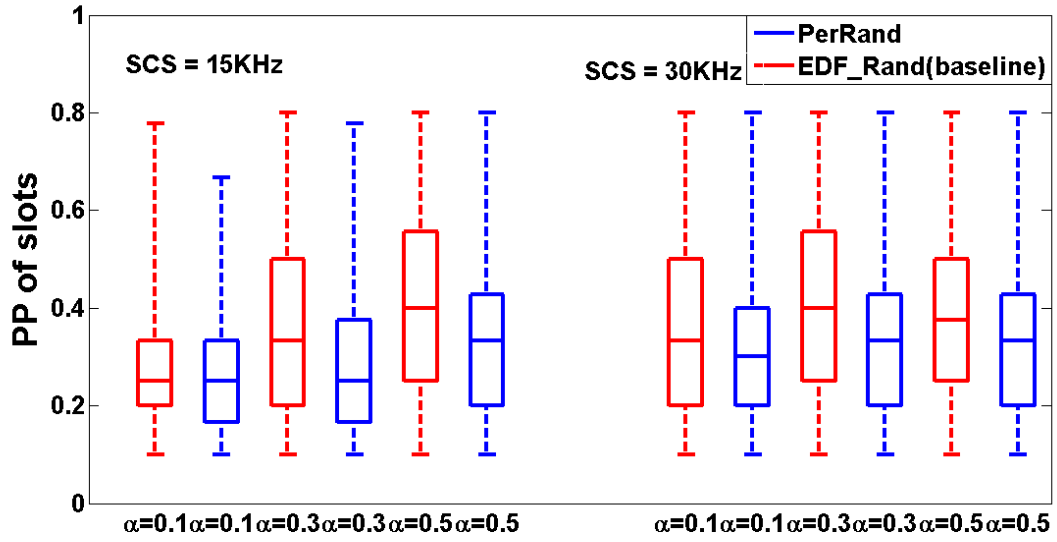
and 7.1(d); Figs. 7.1(b) and 7.1(e); Figs. 7.1(c) and 7.1(f)). In this case, with increase in the utilization, the divergence values increases by 0.2 on an average for $m = 2$ and $m = 3$, keeping other parameters fixed and by 0.1 on an average for $m = 4$, keeping other parameters fixed.



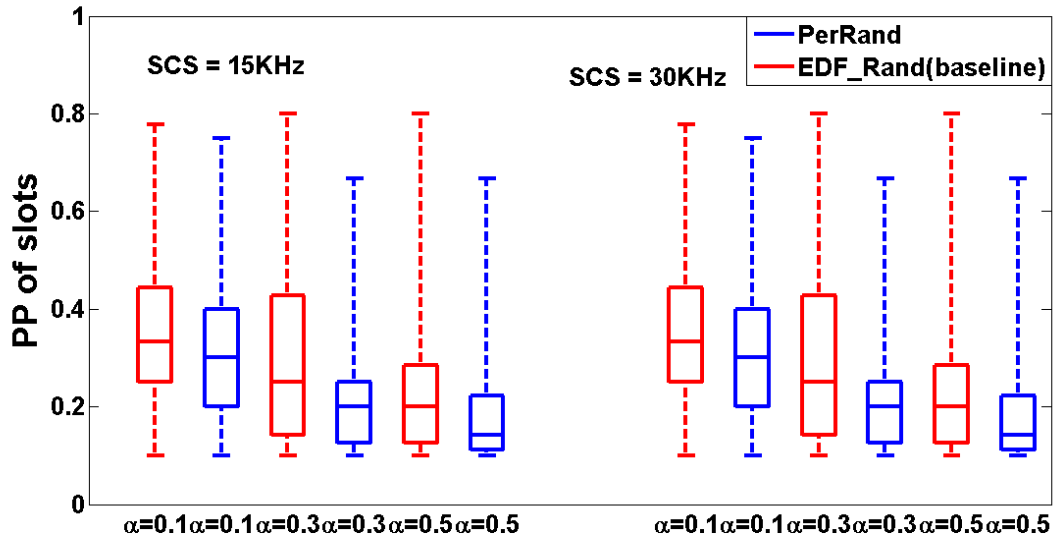
(a) Utilization: 40% – 70%, $\alpha : 0.1$, number of frequencies (m) = 2



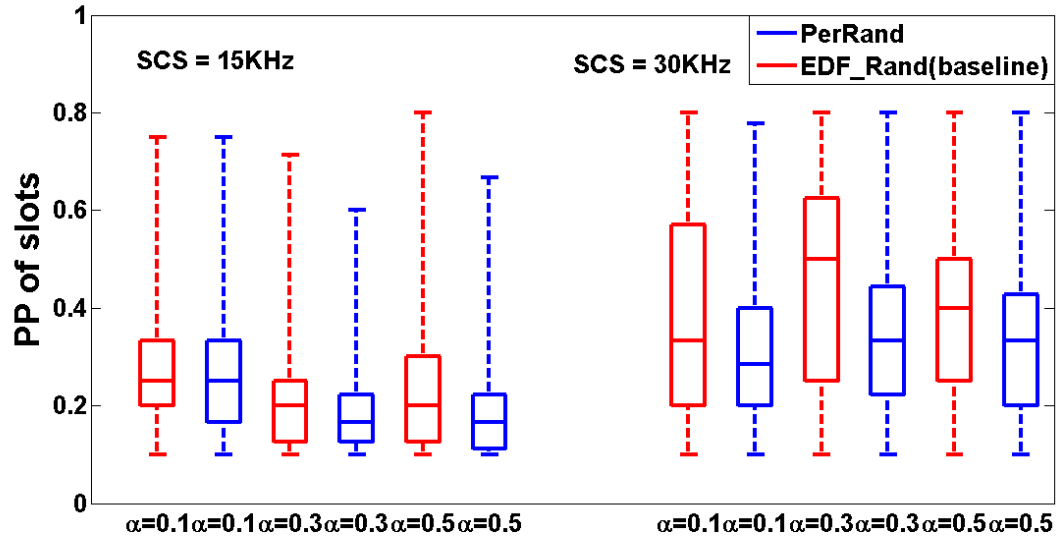
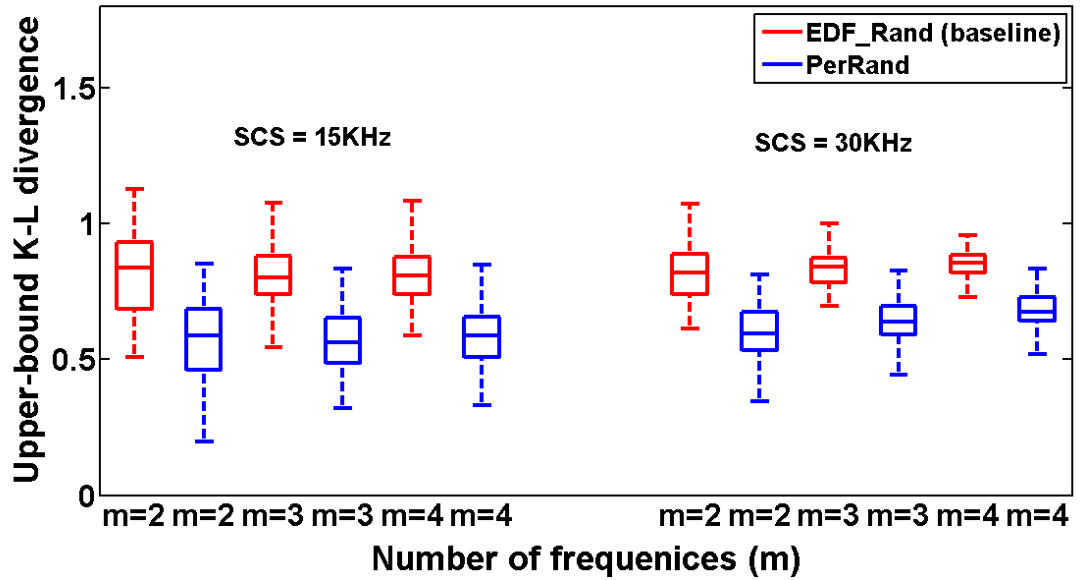
(b) Utilization: 40% – 70%, $\alpha : 0.3$, number of frequencies (m) = 3



(c) Utilization: 40% – 70%, $\alpha : 0.5$, number of frequencies (m) = 4



(d) Utilization: 70% – 100%, $\alpha : 0.1$, number of frequencies (m) = 2

(e) Utilization: 70% – 100%, $\alpha : 0.3$, number of frequencies (m) = 3(f) Utilization: 70% – 100%, $\alpha : 0.5$, number of frequencies (m) = 4FIGURE 7.2: PP of slots. SCS: 15KHz (graphs on the left in each sub-figure), 30KHz (graphs on the right). Divergence values of *EDF_Rand* and *PerRand* are marked in red and blue, respectively.

7.7.1.2 Evaluation by Prediction Probability of slots :

To compare the strength of the schedules generated by *PerRand* with those generated by *EDF_Rand*, we use PP of slots on 100 different flowsets over an observation period of 10 hyperperiods. Figure. 7.2 shows the non-zero prediction probabilities of slots starting from the second hyperperiod upto the eleventh hyperperiod for the schedules generated by both the strategies under various parameter settings. Note that there is no observation available in the first hyperperiod. As a result, PP of slots for the first hyperperiod is 0 for both the strategies and hence not shown in the plots under Figure. 7.2. The plots in red (blue) represent the PP of *EDF_Rand* (*PerRand*), the plots to the left (right) show the PP for sub-carrier spacing of 15KHz (30KHz) in all the sub-figures of Figure. 7.2. It can be observed that the mean PP of slots in the schedules generated by *PerRand* is less than *EDF_Rand* by 0.4 under all settings. Moreover, the PP of slots in the schedules generated by *PerRand* is 27% less on an average compared to those generated by *EDF_Rand* under all settings which ensures that the schedules generated by our proposed strategy are more secure compared to the baseline strategy.

7.7.1.3 Computation time

Figure. 7.3 shows the average time (in *ms*) to compute a single random feasible schedule for a hyperperiod of 1000*ms* by *EDF_Rand* (marked in pyramids) and *PerRand* (marked in circles), over varying parameter values. The average computation time is recorded by running both the strategies 100 times in each case. *EDF_Rand* takes significantly more time compared to *PerRand* in all the cases (plots in Figures. 7.3(a) and 7.3(b)). In the worst-case *EDF_Rand* takes 2900*ms* more time compared to *PerRand* (Figure. 7.3(b) for $m = 4$ and 70 – 100% flow utilization). *PerRand* generates schedules in even less time than the duration of a hyperperiod for all the parameter combinations. This shows that *PerRand* is highly efficient in terms of computation time, and hence can be used as an online strategy (schedule for the next hyperperiod can be generated during the current hyperperiod).

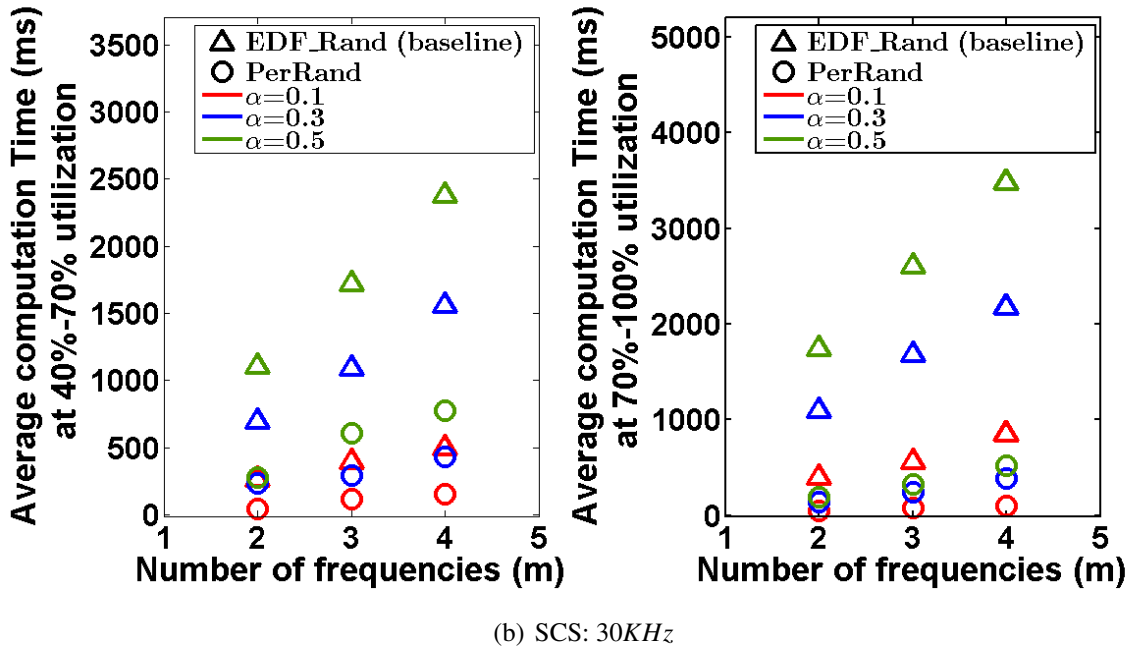
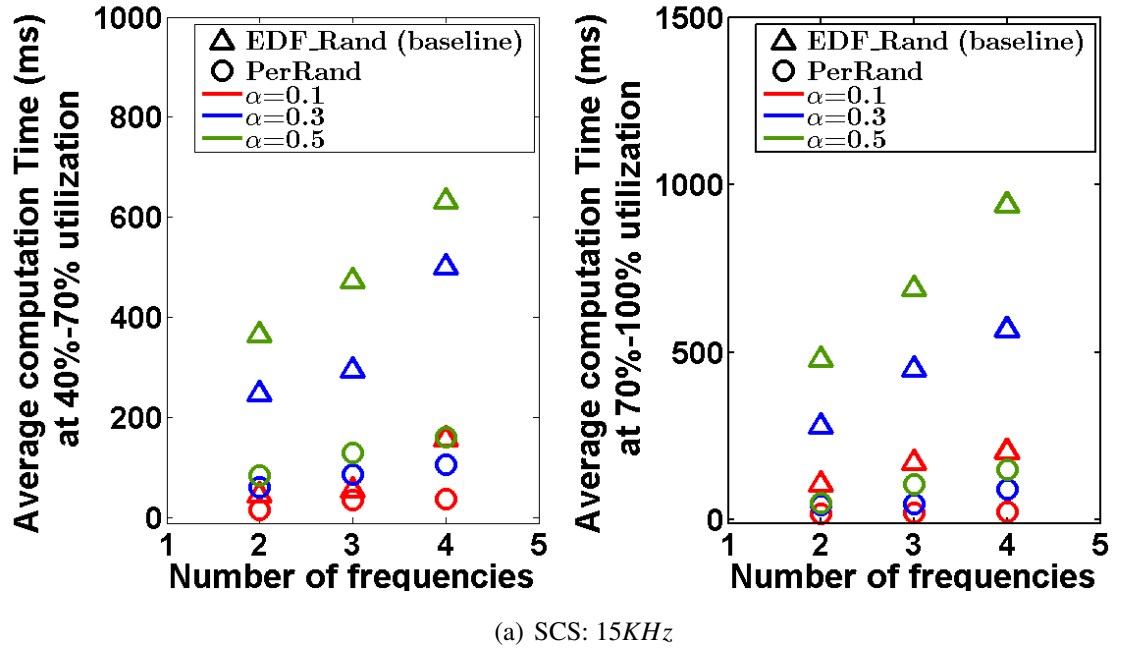


FIGURE 7.3: Average computation time (ms) of *EDF_Rand* (plots in pyramids) and *PerRand* (plots in circles).

7.8 Conclusion

In this chapter, we proposed an *online schedule randomization strategy* to mitigate timing attacks, such as selective jamming attacks in periodic real-time URLLC flows for 5G networks. We prove that our strategy can generate feasible schedules efficiently, while satisfying all the flow deadlines. When compared to a baseline strategy using earliest-deadline first scheduling, our strategy performs at-least 23% better in terms of upper-bound K-L divergence under all parameter settings. However, in the current system model, we considered flows periods to be at the frame boundaries. We would like to explore solutions for online schedule randomization that can support flow periods within a frame boundary and flow periods less than $10ms$ in the future.

Chapter 8

Conclusion and Future Works

In this chapter, we summarize the contributions of this thesis and present some possible future research directions. Section 8.1 presents the conclusion of our work. In Section 8.2, we identify the shortcomings in the state of the art research and some open problems, and present some future research direction on attacks in wireless networks.

8.1 Conclusions

Real-time cyber-physical systems such as the industrial control systems (ICSs) are associated with a large number of control applications. The communication in these applications are periodic with hard deadlines. Among the IEEE 802.15.4 based battery powered wireless modules, the WirelessHART is the most suitable protocol that provides reliable and guaranteed service within hard deadlines. To ensure deadline satisfaction of real-time flows, the radio resources in a WirelessHART network are preallocated and the schedules are pre-computed offline at the time of network initialization. The same schedule is repeated over every hyperperiod which makes the system vulnerable to timing attacks.

Several countermeasures against timing attacks exist in the literature. However, most of these countermeasures cannot guarantee the timeliness requirements of the system. Schedule randomization serves as a notable countermeasure against timing attacks. Two recent works on schedule randomization in wireless sensor networks exist in the literature [27, 28]. However, these countermeasures do not guarantee flow deadlines. Recent

works on schedule randomization as a countermeasure against timing attacks for uniprocessor platforms also exist in the literature [20, 19]. However, such countermeasures are not applicable to our system which is similar to a multiprocessor system. Our system is equivalent to a multiprocessor system where m channels and n flows correspond to m processors and n tasks in a multiprocessor setup. Hence, existing works on schedule randomization cannot provide solution to our system.

Towards addressing this problem, in Chapter 5, we proposed a schedule randomization strategy, the *SlotSwapper*, as a countermeasure against timing attacks in WirelessHART networks. Our proposed countermeasure randomizes the slots in the hyperperiod schedules to reduce the predictability in the schedules. Schedule randomization is performed in such a way that the flow deadlines and the schedule feasibility are not violated in the randomly generated schedules. We used upper-bound Kullback-Leibler divergence or K-L divergence as a metric to measure the performance of our randomization strategy. K-L divergence measures the randomness in the slots of the generated schedules with reference to a truly random solution. We also performed security analysis of the generated random schedules by using Prediction Probability of slots in the schedules. Experimental results have shown that the upper-bound K-L divergence for single and multi-channel WirelessHART networks never exceed 0.41 and 0.3 respectively. Since, this schedule randomization strategy is a centralized strategy, we also provided some analysis on the memory and power consumption overheads in storing and broadcasting the schedules respectively.

Towards addressing the drawbacks of the schedule randomization strategy in Chapter 5, we proposed an online distributed schedule randomization strategy for a single-channel WirelessHART network. The randomization strategy proposed in Chapter 5 is a centralized randomization technique. The random schedules are to be broadcasted periodically to all the nodes in the network which incurs a lot of energy in transmitting the schedules. Moreover, any change in the network topology due to addition or removal of nodes in the network or any change in the route of a flow in the network involves recomputation of the schedules at the centralized network manager, followed by redistribution of the schedules. The distributed dynamic online schedule randomization strategy proposed in Chapter 6 addresses all these drawbacks and randomizes the slots of a schedule in a distributed manner at each node in the network. The proposed randomization strategy also considers the control in the loop. Based on the control performance, this strategy incorporated a period adaptation strategy that can adapt the period of transmission of

the flows in the network depending on the control performance. The period adaptation strategy further increased the extent of randomization of slots in the schedules when the controller is stable under normal conditions. Besides, the period adaptation strategy also reduced the transmission power of the system under normal conditions. We compared the performance of the offline schedule randomization strategy with the online distributed schedule randomization strategy using K-L divergence. We observed that the distributed strategy generates 0.05 and 0.14 less divergence on an average compared to the offline strategy under 66.6% and 75% slot utilization respectively. Further, with period adaptation, our distributed strategy generated much lower divergence with 46% less power consumption on an average.

The fifth generation (5G) cellular network is supposed to dominate the future wireless sensor networks. 5G networks support ultra-reliable low-latency communications (URLLC) for time-critical applications with very high reliability (99.99%) and very low latency (1ms) [12]. To serve the periodic URLLC flows, 5G networks allocate radio resources (slots and frequencies) in advance and follows a semi-persistent scheduling policy [51] in which the resource schedules are computed by the base station and the same schedule repeats over time. As a result, the transmissions associated with the periodic URLLC flows become predictable in nature. This predictability in the schedules makes a 5G network vulnerable to timing attacks. Although we proposed countermeasures against timing attacks in WirelessHART networks in Chapter 5 and Chapter 6 of this thesis, these countermeasures are not applicable for a 5G network. A WirelessHART network is a static network where the schedules are decided and fixed during design time for a fixed number of flows. However in case of a 5G network, the number of flows are highly dynamic. Besides a fraction of the resources are only allocated to for periodic URLLC flows. Hence, schedule randomization in a 5G network is much more challenging.

Towards addressing this problem, in Chapter 7, we proposed an online schedule randomization strategy, that randomizes the slots and the frequencies online without violating the flow deadlines as well as guaranteeing the resource schedule feasibility at runtime. We have proved that the generated schedules are always feasible. We compared our strategy with a baseline strategy using earliest-deadline first scheduling. Experimental results have shown that our strategy performed at-least 23% better in terms of the upper-bound K-L divergence under all parameter settings.

8.2 Future Research Directions

In Section 8.1, we presented the contributions of this thesis related to schedule randomization based countermeasures against timing attacks in real-time wireless networks. Nevertheless, several important research problems still remain open and we present some of them below.

Cyber attacks in WirelessHART networks if the attacker is a compromised node in the network:

The Threat Model in WirelessHART Networks presented in Section 4.2 of Chapter 4 considered the attacker to be a battery powered device deployed in industrial control system setups. Since, the attacker is not part of the network, *i.e.*, it is not an authenticated node in the network, it has no access to the cryptographic keys such as the network keys and session keys in the network. Hence, the attacker can only eavesdrop the communication between any two nodes in the network. It has no access to the content of the packets that has been transmitted. However, if the attacker would have been a compromised node in the network, then it would have access to the cryptographic keys such as the network keys, session keys in the network. Proposing countermeasures to timing attacks under such scenarios would have been much more difficult and challenging. Moreover, timing attacks under such scenarios would have led to other types of attacks such as resource availability attacks, denial of service attacks, integrity attacks, etc. Thus, I want to explore such attacks and countermeasures to such attacks in the future.

Cyber attacks during handover of UEs from one base station to another in 5G networks:

Unlike ICSs where the UEs are static or semi static in nature, the UEs associated with other URLLC applications such as V2X communications in automotives, involve mobile UEs. To provide uninterrupted service to such mobile UEs, handover of UEs from one base station to another base station takes place. Although some security protocols run during this handover phase, there are still a lot of vulnerabilities that an attacker can exploit to launch attacks at the time of handover of the UEs. These attacks are not limited to timing attacks alone. Other types of cyber-attacks can also occur during this handover phase. For instance, a vulnerability in the applications running at the network service provider can lead to Denial of Service attack in the system that can result in interruption of the services in the critical infra-structures during the handover phase,

leading to complete disruption of the critical infra-structures. An attacker can even forge as a fake base station and can interrupt services or launch integrity attacks during the handover phase. Although a few works exist in the literature to address these issues in mobile communication networks, they are still in its infancy in the context of 5G networks. We want to explore such attacks in the future.

Extension of the distributed online schedule randomization strategy with period adaptation under different controller settings:

In Chapter 6, the period adaptation strategy incorporated with the distributed online schedule randomization strategy considers linear state feedback controller. The period adaptation strategy in Chapter 6 showed promising results without any loss in the control performance. In future, we would like to explore the period adaptation strategy with schedule randomization under different controller framework, (eg. a PID controller). The randomization technique may differ depending on the characteristic of the controller. This will help us to come up with different schedule randomization techniques for each class of controllers.

Schedule randomization with resource sharing across UE groups in 5G networks:

In Chapter 7, we divided the UEs into groups of UEs such that each UE group uses the same set of frequencies and uses the same sub-carrier spacings. Our randomization strategy in Chapter 7 randomizes the schedule for one such UE group. However, it would be interesting if there are overlaps in the frequencies, allocated to different UE groups. Due to hard deadlines of the flows, the resource schedule feasibility will be harder for such case. Online schedule randomization will be much more challenging. In the future we want to explore schedule randomization strategies with multiple groups of UEs that allows frequency sharing across groups while still ensuring resource schedule feasibility.

List of Author's Publications and Awards

Publications:

- Ankita Samaddar, Arvind Easwaran and Rui Tan, “Work Already Published: A Schedule Randomization Policy to Mitigate Timing Attacks in WirelessHART Networks”, Brief-Presentations Session of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021.
- Ankita Samaddar, Arvind Easwaran and Rui Tan, “A Schedule Randomization Policy to Mitigate Timing Attacks in WirelessHART Networks”, Springer Real-Time Systems, Volume 56, Pages 452-489, October 2020.
- Ankita Samaddar, Arvind Easwaran and Rui Tan, “SlotSwapper: A Schedule Randomization protocol for Real-Time WirelessHART Networks”, Real-Time Networks (RTN), 2019.
- Ankita Samaddar, Arvind Easwaran, “Online Distributed Schedule Randomization to Mitigate Timing Attacks in Industrial Control Systems”, ACM Transactions on Embedded Computing Systems (under review).
- Ankita Samaddar, Arvind Easwaran, “Online Schedule Randomization to Mitigate Timing Attacks in 5G Periodic URLLC Communication”, ACM Transactions on Sensor Networks (under review).

Awards:

- First Runner-Up in Graduate Student Research Symposium, 2019 under School of Computer Science and Engineering, NTU.

Bibliography

- [1] Insup Lee and Oleg Sokolsky. Medical Cyber Physical Systems. In *CPS Demystified Session, Design Automation Conference (DAC)*, 2010.
- [2] J.W.S. Liu. Real-Time Systems. In *Upper Saddle River, NJ : Prentice Hall*, 2000.
- [3] Military Standard 882C: System Safety Program Requirements. US Department of Defense, January 1993.
- [4] Easwaran, Arvind and Chattopadhyay, Anupam and Bhasin, Shivam. A systematic security analysis of real-time cyber-physical systems. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 206–213, 2017.
- [5] Alessandro Mascellino. Switching your industrial control systems to wireless. In *Control Automation*, January 2020. <https://control.com/news/switching-your-industrial-control-systems-to-wireless/>.
- [6] Rainer Matischek, Thomas Herndl, Christoph Grimm, and Jan Haase. Real-time wireless communication in automotive applications. In *2011 Design, Automation Test in Europe*, pages 1–6, 2011.
- [7] P. McDermott-Wells. What is Bluetooth? *IEEE Potentials*, 23(5):33–35, 2005.
- [8] Stanislav Safaric and Kresimir Malaric. ZigBee wireless standard. In *Proceedings ELMAR 2006*, pages 259–262, 2006.
- [9] Deji Chen, Mark Nixon, Song Han, Aloysius K. Mok, and Xiuming Zhu. WirelessHART and IEEE 802.15.4e. In *2014 IEEE International Conference on Industrial Technology (ICIT)*, pages 760–765, 2014.
- [10] Industrial Control Systems. <https://www.trendmicro.com/vinfo/in/security/definition/industrial-control-system>.
- [11] Installed networks exceed 8,000 at major manufacturing sites worldwide. 2012. <https://bit.ly/2SiwSbu>.
- [12] 5G; Service requirements for next generation new services and markets (3GPP TS 22.261 version 15.5.0 Release 15). July, 2018.
- [13] Introduction to Cellular V2X. <https://www.qualcomm.com/media/documents/files/introduction-to-c-v2x.pdf>.

- [14] Oliver Eigner, Philipp Kreimel, and Paul Tavorato. Attacks on Industrial Control Systems - Modeling and Anomaly Detection. In *ICISSP*, 2018.
- [15] Vijay K. Garg and Yih-Chen Wang. 7 - Wireless Network Access Technologies. In WAI-KAI CHEN, editor, *The Electrical Engineering Handbook*, pages 1005–1009. Academic Press, Burlington, 2005. <https://www.sciencedirect.com/science/article/pii/B978012170960050075X>.
- [16] Alejandro Proano and Loukas Lazos. Selective jamming attacks in wireless networks. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–6. IEEE, 2010.
- [17] Xia Cheng, Junyang Shi, Mo Sha, and Linke Guo. Launching Smart Selective Jamming Attacks in WirelessHART Networks. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.
- [18] The CIA triad: Definition, components and examples. <https://www.csoonline.com/article/3519908/the-cia-triad-definition-components-and-examples.html>.
- [19] Ke Jiang, Petru Eles, Zebo Peng, Sudipta Chattopadhyay, and Lejla Batina. SPARTA: A scheduling policy for thwarting differential power analysis attacks. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 667–672, 2016.
- [20] Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. TaskShuffler: A Schedule Randomization Protocol for Obfuscation against Timing Inference Attacks in Real-Time Systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016.
- [21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: The Case of AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, pages 1–20, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [22] Joseph Bonneau and Ilya Mironov. Cache-Collision Timing Attacks Against AES. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 201–215, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [23] Saman Feghhi and Douglas J. Leith. An efficient web traffic defence against timing-analysis attacks. *IEEE Transactions on Information Forensics and Security*, 14(2):525–540, 2019.
- [24] Jing Deng, R. Han, and S. Mishra. Countermeasures Against Traffic Analysis Attacks in Wireless Sensor Networks. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM’05)*, pages 113–126, 2005.
- [25] Darshana Jayasinghe, Roshan Ragel, and Dhammika Elkaduwe. Constant time encryption as a countermeasure against remote cache timing attacks. In *2012 IEEE 6th International Conference on Information and Automation for Sustainability*, pages 129–134, 2012.

- [26] Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 104(5):1013–1024, 2015.
- [27] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K. Das. JAMMY: A Distributed and Dynamic Solution to Selective Jamming Attack in TDMA WSNs. *IEEE Transactions on Dependable and Secure Computing*, 14(4):392–405, 2017.
- [28] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K. Das. DISH: DIStributed SHuffling Against Selective Jamming Attack in IEEE 802.15.4e TSCH Networks. *ACM Transactions on Sensor Networks*, 15(1):1–28, dec 2018.
- [29] Fiana Raiber and Oren Kurland. Kullback-leibler divergence revisited. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 117–124. ACM, 2017.
- [30] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local computer networks, proceedings 2006 31st IEEE conference on*. IEEE, 2006.
- [31] Paramasiven Appavoo, Ebram Kamal William, Mun Choon Chan, and Mobashir Mohammad. Indriya2: A Heterogeneous Wireless Sensor Network (WSN) Testbed. In *International Conference on Testbeds and Research Infrastructures*, pages 3–19. Springer, 2018.
- [32] Ankita Samaddar, Arvind Easwaran, and Rui Tan. SlotSwapper: A Schedule Randomization Protocol for Real-Time WirelessHART Networks. *SIGBED Rev.*, 16(4):32–37, January 2020. <https://doi.org/10.1145/3378408.3378413>.
- [33] Ankita Samaddar, A. Easwaran, and Rui Tan. A schedule randomization policy to mitigate timing attacks in WirelessHART networks. *Real Time Syst.*, 56:452–489, 2020.
- [34] B. Aminian, J. Araújo, M. Johansson, and K. H. Johansson. GISOO: A virtual testbed for wireless cyber-physical systems. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pages 5588–5593, 2013.
- [35] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys '03*, page 126–137, New York, NY, USA, 2003. Association for Computing Machinery. <https://doi.org/10.1145/958491.958506>.
- [36] Nils Vreman, Anton Cervin, and Martina Maggio. Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. <https://www.ecrts.org/>.

- [37] Roy, Debayan and Hobbs, Clara and Anderson, James H. and Caccamo, Marco and Chakraborty, Samarjit. Timing Debugging for Cyber-Physical Systems. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1893–1898, 2021.
- [38] WirelessHART Security Overview-Emerson. <https://www.emerson.com/documents/automation/white-paper-wirelesshart-security-overview-by-hcf-en-42578.pdf>.
- [39] Rajeev Alur, Alessandro D’Innocenzo, Karl H Johansson, George J Pappas, and Gera Weiss. Modeling and analysis of multi-hop control networks. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 223–232. IEEE, 2009.
- [40] WirelessHART. <https://www.cse.wustl.edu/~lu/cse521s/Slides/wirelesshart.pdf>.
- [41] Gabriella Fiore, Valeria Ercoli, Alf J Isaksson, Krister Landernäs, and Maria Domenica Di Benedetto. Multihop multi-channel scheduling for wireless control in WirelessHART networks. In *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1–8. IEEE, 2009.
- [42] Deji Chen, Mark Nixon, and Aloysius Mok. *WirelessHART: Real-Time Mesh Network for Industrial Automation*. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [43] Jianping Song, Song Han, Al Mok, Deji Chen, Mike Lucas, Mark Nixon, and Wally Pratt. WirelessHART: Applying wireless technology in real-time industrial process control. In *IEEE real-time and embedded technology and applications symposium*, pages 377–386. IEEE, 2008.
- [44] Venkata Prashant Modekurthy, Abusayeed Saifullah, and Sanjay Madria. DistributedHART: A Distributed Real-Time Scheduling System for WirelessHART Networks. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 216–227. IEEE, 2019.
- [45] Xiang-Yang Li, Kousha Moaveni-Nejad, Wen-Zhan Song, and Wei-Zhao Wang. Interference-aware topology control for wireless sensor networks. In *2005 Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005.*, pages 263–274. IEEE, 2005.
- [46] Jianping Song, Song Han, Xiuming Zhu, Aloysius K Mok, Deji Chen, and Mark Nixon. A complete wirelessHART network. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008.
- [47] Waziha Kabir. Orthogonal Frequency Division Multiplexing (OFDM). In *2008 China-Japan Joint Microwave Conference*, pages 178–184, 2008.

- [48] Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Release description; Release 15 (3GPP TR 21.915 version 15.0.0 Release 15).
- [49] LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (3GPP TS 36.213 version 14.2.0 Release 14). Mar, 2017.
- [50] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch. Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.
- [51] Semi-Persistent Scheduling for 5G New Radio URLLC. R1-167309, 3GPP TSG-RAN WG1 #86, Aug, 2016.
- [52] D. Segura, E. J. Khatib, J. Munilla, and R. Barco. 5G Numerologies Assessment for URLLC in Industrial Communications. *Sensors*, 21(7):2489, 2021.
- [53] Matthew Weiner, Milos Jorgovanovic, Anant Sahai, and Borivoje Nikolić. Design of a Low-Latency, High-Reliability Wireless Communication System for Control Applications. In *IEEE International Conference on Communications (ICC)*, pages 3829–3835, 2014.
- [54] Renato Abreu, Preben Mogensen, and Klaus I. Pedersen. Pre-Scheduled Resources for Retransmissions in Ultra-Reliable and Low Latency Communications. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–5, 2017.
- [55] Salah Eddine Elayoubi, Patrick Brown, Matha Deghel, and Ana Galindo-Serrano. Radio Resource Allocation and Retransmission Schemes for URLLC Over 5G Networks. *IEEE Journal on Selected Areas in Communications*, 37(4):896–904, 2019.
- [56] Chao Wang, Yan Chen, Yiqun Wu, and Liqing Zhang. Performance Evaluation of Grant-Free Transmission for Uplink URLLC Services. In *IEEE Vehicular Technology Conference (VTC Spring)*, pages 1–6, 2017.
- [57] 5G; NR; Physical layer procedures for data (3GPP TS 38.214 version 15.3.0 Release 15). Jan, 2018.
- [58] Fabian Mager, Dominik Baumann, Romain Jacob, Lothar Thiele, Sebastian Trimpe, and Marco Zimmerling. Feedback Control Goes Wireless: Guaranteed Stability over Low-Power Multi-Hop Networks. In *ACM International Conference on Cyber-Physical Systems (ICCPS)*, pages 97–108, New York, NY, USA, 2019. <https://doi.org/10.1145/3302509.3311046>.
- [59] Yehan Ma and Chenyang Lu. Efficient Holistic Control over Industrial Wireless Sensor-Actuator Networks. In *IEEE International Conference on Industrial Internet (ICII)*, pages 89–98, 2018.

- [60] Jonathon Shlens. Notes on Kullback-Leibler divergence and likelihood. *arXiv preprint arXiv:1404.2000*, 2014.
- [61] Lecture Notes: Information Theory and Statistics, 2005. <https://www.stat.berkeley.edu/~binyu/summer08/L1P1.pdf>.
- [62] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- [63] Janaka Alawatugoda, Darshana Jayasinghe, and Roshan Ragel. Countermeasures against Bernstein’s remote cache timing attack. In *2011 6th International Conference on Industrial and Information Systems*, pages 43–48, 2011.
- [64] Zhen Hang Jiang, Yungsi Fei, and David Kaeli. A complete key recovery timing attack on a GPU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 394–405, 2016.
- [65] Reza Montasari, Richard Hill, Amin Hosseinian-far, and Farshad Montaseri. Countermeasures for timing-based side-channel attacks against shared, modern computing hardware. *International Journal of Electronic Security and Digital Forensics*, 11(3):294–320, jul 2019.
- [66] Tom Van Goethem, Christina Pöpper, Wouter Joosen, and Mathy Vanhoef. Timeless Timing Attacks: Exploiting Concurrency to Leak Secrets over Remote Connections. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1985–2002. USENIX Association, August 2020. <https://www.usenix.org/conference/usenixsecurity20/presentation/van-goethem>.
- [67] S. Ramesh and D. T. S. Prakash. An effective attack analysis and defense in web traffic using only timing information. 2017.
- [68] Xi Luo, Xu Ji, and Myong-Soon Park. Location Privacy against Traffic Analysis Attacks in Wireless Sensor Networks. In *2010 International Conference on Information Science and Applications*, pages 1–6, 2010.
- [69] Tianhui Meng, Xiaofan Li, Sha Zhang, and Yubin Zhao. A Hybrid Secure Scheme for Wireless Sensor Networks against Timing Attacks Using Continuous-Time Markov Chain and Queueing Model. *Sensors*, 16(10), 2016. <https://www.mdpi.com/1424-8220/16/10/1606>.
- [70] Roberta Daidone, Gianluca Dini, and Marco Tiloca. A solution to the GTS-based selective jamming attack on IEEE 802.15. 4 networks. *Wireless networks*, 20(5):1223–1235, 2014.
- [71] Hossein Pirayesh and Huacheng Zeng. Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey. *ArXiv*, abs/2101.00292, 2021.

- [72] Anthony D Wood, John A Stankovic, and Gang Zhou. DEEJAM: Defeating energy-efficient jamming in IEEE 802.15. 4-based wireless networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE, 2007.
- [73] Alejandro Proano and Loukas Lazos. Packet-hiding methods for preventing selective jamming attacks. *IEEE Transactions on dependable and secure computing*, 9(1):101–114, 2012.
- [74] Youness Arjoune and Saleh Faruque. Smart Jamming Attacks in 5G New Radio: A Review. In *Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1010–1015, 2020.
- [75] New Services & Applications With 5G Ultra-Reliable Low-Latency Communications. <https://www.5gamericas.org/new-services-applications-with-5g-ultra-reliable-low-latency-communications/>.
- [76] Kanthakumar Pongaliur, Zubin Abraham, Alex X Liu, Li Xiao, and Leo Kempel. Securing sensor nodes against side channel attacks. In *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*. IEEE, 2008.
- [77] David Brumley and Dan Boneh. Remote timing attacks are practical. In *12th USENIX Security Symposium (USENIX Security 03)*, Washington, D.C., August 2003. USENIX Association. <https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical>.
- [78] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The Clock is Still Ticking: Timing Attacks in the Modern Web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, page 1382–1393, New York, NY, USA, 2015. Association for Computing Machinery. <https://doi.org/10.1145/2810103.2813632>.
- [79] Ramazan Algin, Huseyin O. Tan, and Kemal Akkaya. Mitigating Selective Jamming Attacks in Smart Meter Data Collection Using Moving Target Defense. In *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet '17*, page 1–8, New York, NY, USA, 2017. Association for Computing Machinery. <https://doi.org/10.1145/3132114.3132127>.
- [80] Manli Qian, Yuanyuan Wang, Yiqing Zhou, Lin Tian, and Jinglin Shi. A Super Base Station based Centralized Network Architecture for 5G Mobile Communication Systems. *Digital Communications and Networks*, 1(2):152–159, 2015.
- [81] Sofiane Takarabt, Alexander Schaub, Adrien Facon, Sylvain Guilley, Laurent Sauvage, Youssef Souissi, and Yves Mathieu. Cache-Timing Attacks Still Threaten IoT Devices. In Claude Carlet, Sylvain Guilley, Abderrahmane Nitaj, and El Mamoun Souidi, editors, *Codes, Cryptology and Information Security*, pages 13–30, Cham, 2019. Springer International Publishing.

- [82] MA Settana, AS Naila, HY Yaseen, and TE Huwaida. Cache-Timing Attack against AES Crypto-Systems Countermeasure Using Weighted Average Masking Time Algorithm. *Journal of Information Warfare*, 15(1):104–114, 2016. <https://www.jstor.org/stable/26487484>.
- [83] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: The Case of AES. CT-RSA'06, Berlin, Heidelberg, 2006. Springer-Verlag. https://doi.org/10.1007/11605805_1.
- [84] Chao Su and Qingkai Zeng. Survey of CPU Cache-Based Side-Channel Attacks: Systematic Analysis, Security Models, and Countermeasures. In *Security and Communication Networks*, vol. 2021, Article ID 5559552, 15 pages. <https://doi.org/10.1155/2021/5559552>.
- [85] Yangdi Lyu and Prabhat Mishra. A Survey of Side-Channel Attacks on Caches and Countermeasures. *Journal of Hardware and Systems Security*, 2(1):33–50, 2018. <https://app.dimensions.ai/details/publication/pub.1093029882>.
- [86] Alexandres Andreou, Andrey Bogdanov, and Elmar Tischhauser. Cache timing attacks on recent microarchitectures. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 155–155, 2017.
- [87] Qian Ge, Y. Yarom, David Cock, and G. Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, 8:1–27, 2016.
- [88] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss. PLATYPUS: Software-Based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, Los Alamitos, CA, USA, may 2021. IEEE Computer Society. <https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00063>.
- [89] M. Randolph and William Diehl. Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman. *Cryptogr.*, 4:15, 2020.
- [90] Kristin Krüger, Marcus Volp, and Gerhard Fohler. Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems. *LIPICs-Leibniz International Proceedings in Informatics*, 106:22, 2018.
- [91] C. Chen and S. Mohan and R. Pellizzoni and R. B. Bobba and N. Kiyavash. A Novel Side-Channel in Real-Time Schedulers. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 90–102, Los Alamitos, CA, USA, apr 2019. IEEE Computer Society. <https://doi.ieeecomputersociety.org/10.1109/RTAS.2019.00016>.
- [92] Songran Liu and Nan Guan and Dong Ji and Weichen Liu and Xue Liu and Wang Yi. Leaking your engine speed by spectrum analysis of real-Time scheduling sequences. *Journal of Systems Architecture*, 97:455–466, 2019. <https://www.sciencedirect.com/science/article/pii/S1383762118302224>.

- [93] Liu, Songran and Yi, Wang. Task Parameters Analysis in Schedule-Based Timing Side-Channel Attack. *IEEE Access*, 8:157103–157115, 2020.
- [94] Xian Zhang, Kerong Ben, and Jie Zeng. Cross-Entropy: A New Metric for Software Defect Prediction. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 111–122, 2018.
- [95] David J Galas, Gregory Dewey, James Kunert-Graf, and Nikita A Sakhanenko. Expansion of the Kullback-Leibler divergence, and a new class of information metrics. *Axioms*, 6(2):8, 2017.
- [96] Xia Cheng, Junyang Shi, and Mo Sha. Cracking the Channel Hopping Sequences in IEEE 802.15.4e-Based Industrial TSCH Networks. In *Proceedings of the International Conference on Internet of Things Design and Implementation, IoTDI '19*, page 130–141, New York, NY, USA, 2019. Association for Computing Machinery. <https://doi.org/10.1145/3302505.3310075>.
- [97] Shahid Raza, Adriaan Slabbert, Thiemo Voigt, and Krister Landernas. Security considerations for the wirelessHART protocol. pages 1 – 8, 10 2009.
- [98] Anne Remke and Xian Wu. WirelessHART modeling and performance evaluation. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.
- [99] Bauer, Kevin and McCoy, Damon and Anderson, Eric and Breitenbach, Markus and Grudic, Greg and Grunwald, Dirk and Sicker, Douglas. The Directional Attack on Wireless Localization -or- How to Spoof Your Location with a Tin Can. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, pages 1–6, 2009.
- [100] Yiqiao Wei, Seung-Hoon Hwang, and Ivan Marsa-Maestre. Optimization of Cell Size in Ultra-Dense Networks with Multiattribute User Types and Different Frequency Bands. 2018.
- [101] Gordon J. Sutton, Jie Zeng, Ren Ping Liu, Wei Ni, Diep N. Nguyen, Bee-shanga A. Jayawickrama, Xiaojing Huang, Mehran Abolhasan, Zhang Zhang, Eryk Dutkiewicz, and Tiejun Lv. Enabling Technologies for Ultra-Reliable and Low Latency Communications: From PHY and MAC Layer Perspectives. *IEEE Communications Surveys Tutorials*, 21(3):2488–2524, 2019.
- [102] 5G; NR; Overall description; Stage-2 (3GPP TS 38.300 version 15.8.0 Release 15). Jan, 2020.
- [103] I. Pavić and H. Džapo. Optimal Harmonic Period Assignment With Constrained Number of Distinct Period Values. *IEEE Access*, 8:175697–175712, 2020.
- [104] Xin Lou, Cuong Tran, Rui Tan, David KY Yau, and Zbigniew T Kalbarczyk. Assessing and mitigating impact of time delay attack: a case study for power grid frequency control. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 207–216, 2019.

- [105] Kanika Grover, Alvin Lim, and Qing Yang. Jamming and anti-jamming techniques in wireless networks: a survey. *International Journal of Ad Hoc and Ubiquitous Computing*, 17(4):197–215, 2014.
- [106] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2005.
- [107] Mitra Nasri, Thidapat Chantem, Gedare Bloom, and Ryan M. Gerdes. On the Pitfalls and Vulnerabilities of Schedule Randomization Against Schedule-Based Attacks. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 103–116, 2019.
- [108] François Koeune. Pseudo-random number generator. pages 485–487, 2005. https://doi.org/10.1007/0-387-23483-7_330.
- [109] Anna N Kim, Fredrik Hekland, Stig Petersen, and Paula Doyle. When HART goes wireless: Understanding and implementing the WirelessHART standard. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 899–907. IEEE, 2008.
- [110] Song Han, Xiuming Zhu, Aloysius K Mok, Deji Chen, and Mark Nixon. Reliable and real-time communication in industrial wireless mesh networks. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 3–12. IEEE, 2011.
- [111] Michal Holčápek and Tomáš Tichý. Strong law of large numbers for random variables valued by gradual numbers and closed intervals of gradual numbers. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2019.
- [112] WirelessHART.2019, 2019. <https://fieldcommgroup.org/technologies/hart/hart-technology>.
- [113] Indriya2 Testbed at NUS, 2019. <https://indriya.comp.nus.edu.sg/>.
- [114] Rafael Lajara, José Pelegrí-Sebastiá, and Juan J Perez Solano. Power consumption analysis of operating systems for wireless sensor networks. *Sensors*, 10(6):5809–5826, 2010.
- [115] Christine Jardak. The storage and data processing in wireless sensor networks. 2012.
- [116] Stig Petersen and Simon Carlsen. Performance evaluation of WirelessHART for factory automation. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–9. IEEE, 2009.
- [117] A. Cervin, B. Lincoln, J. Eker, K. Årzén, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. 2004.

- [118] Yehan Ma, Dolvara Gunatilaka, Bo Li, Humberto Gonzalez, and Chenyang Lu. Holistic Cyber-Physical Management for Dependable Wireless Control Systems. *ACM Trans. Cyber-Phys. Syst.*, 3(1), September 2018. <https://doi.org/10.1145/3185510>.
- [119] Bruno Sinopoli, Luca Schenato, Massimo Franceschetti, Kameshwar Poolla, Michael I Jordan, and Shankar S Sastry. Kalman filtering with intermittent observations. *IEEE transactions on Automatic Control*, 49(9):1453–1464, 2004.
- [120] M. Mazo and P. Tabuada. On event-triggered and self-triggered control over sensor/actuator networks. In *2008 47th IEEE Conference on Decision and Control*, pages 435–440, 2008.
- [121] Manuel Mazo and Paulo Tabuada. Decentralized event-triggered control over wireless sensor/actuator networks. *IEEE Transactions on Automatic Control*, 56(10):2456–2461, 2011.
- [122] Manuel Mazo Jr, Adolfo Anta, and Paulo Tabuada. An ISS self-triggered implementation of linear controllers. *Automatica*, 46(8):1310–1314, 2010.
- [123] Z. Li and M. Chow. Adaptive Multiple Sampling Rate Scheduling of Real-time Networked Supervisory Control System - Part II. In *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, pages 4615–4620, 2006.
- [124] D. Goswami, M. Lukasiewicz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1227–1232, 2012.
- [125] B. Demirel, Z. Zou, P. Soldati, and M. Johansson. Modular co-design of controllers and transmission schedules in WirelessHART. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 5951–5958, 2011.
- [126] Dohwan Kim, Yuchang Won, Seunghyeon Kim, Yongsoon Eun, Kyung-Joon Park, and Karl H. Johansson. Sampling Rate Optimization for IEEE 802.11 Wireless Control Systems. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, New York, NY, USA, 2019. Association for Computing Machinery. <https://doi.org/10.1145/3302509.3311045>.
- [127] A. Saifullah, C. Wu, P. B. Tiwari, Y. Xu, Y. Fu, C. Lu, and Y. Chen. Near Optimal Rate Selection for Wireless Control Systems. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 231–240, 2012.
- [128] B. Li, Y. Ma, T. Westenbroek, C. Wu, H. Gonzalez, and C. Lu. Wireless Routing and Control: A Cyber-Physical Case Study. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pages 1–10, 2016.
- [129] P. Christofides. Control of nonlinear distributed process systems. 2005.

- [130] Merid Ljesnjanin, D. Quevedo, and D. Nešić. Packetized MPC with dynamic scheduling constraints and bounded packet dropouts. *Autom.*, 50:784–797, 2014.
- [131] L. A. Montestruque and P. Antsaklis. Stability of model-based networked control systems with time-varying transmission times. *IEEE Transactions on Automatic Control*, 49(9):1562–1572, 2004.
- [132] M. Pajic, J. Le Ny, S. Sundaram, G. J. Pappas, and R. Mangharam. Closing the loop: A simple distributed method for control over wireless networks. In *2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 25–36, 2012.
- [133] J. Bai, E. P. Eyisi, F. Qiu, Y. Xue, and X. D. Koutsoukos. Optimal Cross-Layer Design of Sampling Rate Adaptation and Network Scheduling for Wireless Networked Control Systems. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 107–116, 2012.
- [134] X. Koutsoukos, N. Kottenstette, J. Hall, P. Antsaklis, and J. Sztipanovits. Passivity-Based Control Design for Cyber-Physical Systems. 2008.
- [135] D. Bernardini and A. Bemporad. Energy-aware robust Model Predictive Control based on wireless sensor feedback. In *2008 47th IEEE Conference on Decision and Control*, pages 3342–3347, 2008.
- [136] B. Demirel, A. Aytekin, D. E. Quevedo, and M. Johansson. To wait or to drop: On the optimal number of retransmissions in wireless control. In *2015 European Control Conference (ECC)*, pages 962–968, 2015.
- [137] M. Franceschelli, M. Egerstedt, and A. Giua. Motion probes for fault detection and recovery in networked control systems. In *2008 American Control Conference*, pages 4358–4363, 2008.
- [138] Yehan Ma, Chenyang Lu, and Yebin Wang. Efficient Holistic Control: Self-Awareness across Controllers and Wireless Networks. *ACM Trans. Cyber-Phys. Syst.*, 4(4), jun 2020. <https://doi.org/remotexs.ntu.edu.sg/10.1145/3371500>.
- [139] André Preumont. Controllability and Observability. In *Vibration Control of Active Structures*, pages 289–312. Springer, 2018.
- [140] Youngjoo Kim and Hyochoong Bang. Introduction to Kalman Filter and Its Applications. In Felix Govaers, editor, *Introduction and Implementations of the Kalman Filter*, chapter 2. IntechOpen, Rijeka, 2019. <https://doi.org/10.5772/intechopen.80600>.
- [141] Mohammed Dahleh, Munther A Dahleh, and George Verghese. Lectures on dynamic systems and control. *A + A*, 4(100):1–100, 2004.
- [142] R.B. Lee, Z.J. Shi, Y.L. Yin, R.L. Rivest, and M.J.B. Robshaw. On permutation operations in cipher design. In *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, volume 2, pages 569–577 Vol.2, 2004.

- [143] Hai Lin and Panos J. Antsaklis. Stability and stabilizability of switched linear systems: A survey of recent results. *IEEE Transactions on Automatic Control*, 54(2):308–322, 2009.
- [144] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. Van Den Bosch. Analysis of event-driven controllers for linear systems. *International Journal of Control*, 81(4):571–590, 2008.
- [145] Manuel Mazo, Adolfo Anta, and Paulo Tabuada. An ISS self-triggered implementation of linear controllers. *Automatica*, 46(8):1310–1314, 2010. <https://www.sciencedirect.com/science/article/pii/S0005109810002153>.
- [146] Adolfo Anta and Paulo Tabuada. To Sample or not to Sample: Self-Triggered Control for Nonlinear Systems. *IEEE Transactions on Automatic Control*, 55(9):2030–2042, 2010.
- [147] Telos Rev B (Low Power Wireless Sensor Module). <https://www2.ece.ohio-state.edu/~bibyk/ee582/telosMote.pdf>.
- [148] Y. Jiang, Y. Wang, S. A. Bortoff, and Z. Jiang. Optimal Codesign of Nonlinear Control Systems Based on a Modified Policy Iteration Method. *IEEE Transactions on Neural Networks and Learning Systems*, 26(2):409–414, 2015.
- [149] V. Shilpiekandula, S. A. Bortoff, J. C. Barnwell, and K. E. Rifai. Load positioning in the presence of base vibrations. In *2012 American Control Conference (ACC)*, pages 6282–6287, 2012.
- [150] Petar Popovski, Čedomir Stefanović, Jimmy J. Nielsen, Elisabeth de Carvalho, Marko Angelichinoski, Kasper F. Trillingsgaard, and Alexandru-Sabin Bana. Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC), 2018.
- [151] S. R. Hussain, Mitziu Echeverria, Omar Chowdhury, N. Li, and E. Bertino. Privacy Attacks to the 4G and 5G Cellular Paging Protocols Using Side Channel Information. In *Network and Distributed System Security Symposium (NDSS)*, 2019.
- [152] Kaiming Fang and Guanhua Yan. Paging Storm Attacks against 4G/LTE Networks from Regional Android Botnets: Rationale, Practicality, and Implications. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, page 295–305, 2020.
- [153] J. D. Roth, M. Tummala, J. C. McEachen, and J. W. Scrofani. Location Privacy in LTE: A Case Study on Exploiting the Cellular Signaling Plane’s Timing Advance. 2017. <https://calhoun.nps.edu/handle/10945/55165>.
- [154] N. Lakshmanan, N. Budhdev, M. S. Kang, M. C. Chan, and J. Han. A Stealthy Location Identification Attack Exploiting Carrier Aggregation in Cellular Networks. In *USENIX Security Symposium*, 2021.

- [155] A. Shaik, R. Borgaonkar, N. Asokan, V. Niemi, and J.-P. Seifert. Practical Attacks Against Privacy and Availability in 4G/LTE Mobile Communication Systems. *CoRR*, abs/1510.07563, 2015. <http://arxiv.org/abs/1510.07563>.
- [156] Sanjoy Baruah, N. Cohen, C. Plaxton, and Donald Varvel. Proportionate Progress: A Notion of Fairness in Resource Allocation. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1993.