

UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**Synchronization Algorithms for TDD Fronthaul in PTP-Unaware Networks**

**Camila Novaes Silva**

**DM: 08/23**

UFPA / ITEC / PPGEE  
Campus Universitário do Guamá  
Belém-Pará-Brasil  
2023



UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**Camila Novaes Silva**

**Synchronization Algorithms for TDD Fronthaul in PTP-Unaware Networks**

**DM: 08/23**

UFPA / ITEC / PPGEE  
Campus Universitário do Guamá  
Belém-Pará-Brasil  
2023

UNIVERSIDADE FEDERAL DO PARÁ  
INSTITUTO DE TECNOLOGIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**Camila Novaes Silva**

**Synchronization Algorithms for TDD Fronthaul in PTP-Unaware Networks**

Submitted to the examination committee in the graduate department of Electrical Engineering at the Federal University of Pará in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis in Telecommunications.

UFPA / ITEC / PPGEE  
Campus Universitário do Guamá  
Belém-Pará-Brasil

2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD  
Sistema de Bibliotecas da Universidade Federal do Pará  
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a)  
autor(a)

---

S586s Silva, Camila Novaes.  
Synchronization algorithms for TDD fronthaul in PTP-  
unaware networks / Camila Novaes Silva. — 2023.  
xxiv, 131 f. : il. color.

Orientador(a): Prof. Dr. Aldebaro Barreto da Rocha  
Kloutau Júnior  
Dissertação (Mestrado) - Universidade Federal do Pará,  
Instituto de Tecnologia, Programa de Pós-Graduação em  
Engenharia Elétrica, Belém, 2023.

1. Sincronização de relógio. 2. PTP. 3. Arranjos de  
lógica programável em campo. 4. Ethernet. 5. Fronthaul.  
I. Título.

---

CDD 384

**“SYNCHRONIZATION ALGORITHMS FOR TDD FRONTHAUL IN PTP-UNAWARE NETWORKS”**

AUTORA: **CAMILA NOVAES SILVA**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRA EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM: 17/03/2023

**BANCA EXAMINADORA:**



---

**Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior**  
(Orientador – PPGEE/UFPA)



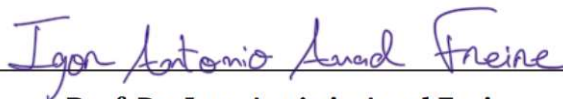
---

**Prof. Dr. Leonardo Lira Ramalho**  
(Avaliador Interno – PPGEE/UFPA)



---

**Prof. Dr. Eduardo Lins de Medeiros**  
(Avaliador Externo – ERICSSON)



---

**Prof. Dr. Igor Antônio Auad Freire**  
(Avaliador Externo – IFCOMM TELECOMUNICAÇÕES)

**VISTO:**

---

**Prof. Dr. Diego Lisboa Cardoso**

(Coordenador do PPGEE/ITEC/UFPA)

# Acknowledgments

First, I would like to thank my advisor, Prof. Aldebaro Klautau, for all the guidance and support over the last six years. I am very grateful for his insights, knowledge, and motivational stories that allowed me to evolve not only professionally but also personally.

Secondly, I would like to thank the committee members, Dr. Eduardo Medeiros, Dr. Igor Freire, and Prof. Leonardo Ramalho. Their experience, insights, and comments were invaluable to the final version of this work.

Furthermore, I would like to extend my thanks to my colleagues at LASSE whom I enjoyed spending a terrific time in a very inspiring and friendly atmosphere. I would like to thank all the members that worked in the synchronization team over the years, particularly Igor Freire, Pedro Bemerguy, Rodrigo Dutra, and Juan Lopes, for their contributions to the testbed and software tools used in this work. Additionally, I would like to thank Igor Almeida for sharing his experience and thoughts on various aspects of this work.

I would like to thank the sponsors of this work, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil, and the Innovation Center, Ericsson Telecomunicações, Brazil.

Finally, I would like to especially thank my family, Lucivane, Junior, Fernanda, and Rafaela. Without their love and support, I would not have come this far.

Thank you all for your invaluable contributions to my academic journey.

Camila Novaes

February 2023

# Glossary

4G	Fourth-generation. 1, 3, 22
5G	Fifth-generation. 1–3, 22
ADC	Analog To Digital Converter. 52
APTS	Assisted Partial-timing Support. 4, 82, 92
AVB	Audio Video Bridging. 49
AxC	Antenna Carrier. 50, 52–54, 60
BBU	Baseband Unit. 22, 23, 46–50, 52–55, 59, 60, 68
BC	Boundary Clock. 4, 14
BH	Backhaul. 21, 22
BMC	Best Master Clock. 13
BS	Base Station. 1, 3, 21, 22
C-RAN	Cloud Radio Access Network. 3, 21, 22
CA	Carrier Aggregation. 3
CBR	Constant Bit Rate. 50, 52
CDAE	Convolutional Denoising Autoencoder. 41
CoMP	Coordinated Multi-point. 3
CPRI	Common Public Radio Interface. 3
CRC	Frame Check Sequence. 51
cTE	Constant Time Error. 12
DL	Downlink. 22, 23, 51, 52, 54
DMA	Direct Memory Access. 53



dTE	Dynamic Time Error. 12
FDD	Frequency-division Duplex. 2
FFM	Flicker Frequency Modulation. 12
FH	Fronthaul. 3–6, 21–26, 28, 41–43, 45, 46, 50–53, 55, 59–61, 65–68, 70, 73, 78–80, 92, 93
FIFO	First-in First-out. 52–54
FPGA	Field-programmable Gate Array. 5, 46, 47, 50, 52
FPM	Flicker Phase Modulation. 12
FTS	Full-timing Support. 4
GbE	Gigabit Ethernet. 18, 46, 51, 59, 63
GMII	Gigabit Media-independent Interface. 50
GNSS	Global Navigation Satellite Systems. 1–4, 83
IoT	Internet Of Things. 1
IQ	In-phase And Quadrature. 50–55
KF	Kalman Filter. 6, 21, 33–38, 59, 80, 82–84, 87, 88, 91, 92, 94, 95, 97, 99
LP	Linear Programming. 33
LS	Least Squares. 33
m-t-s	Master To Slave. 15, 17–19, 29, 31, 33, 38–43, 45, 55, 56, 61, 63, 64, 66–70, 72, 74–76, 78, 91
MAC	Medium Access Control Layer. 17, 18, 51
max TE	Maximum Absolute Time Error. xviii, 12, 13, 36, 57, 59, 60, 78–88, 92
MIMO	Multiple-input Multiple-output. 3
MSE	Mean-square Error. 82, 94, 99

NTP	Network Time Protocol. 33
OC	Ordinary Clocks. 14
OCXO	Oven-controlled Crystal Oscillator. xvii, 48, 49, 60, 68, 72, 74–76, 78–80, 83, 84, 87, 88, 90, 92
PDV	Packet Delay Variation. 2, 4, 5, 14, 15, 20, 21, 24, 32, 38–41, 65, 66, 91, 92
PHY	Physical Layer. 17–19, 41, 50, 62, 63, 82
ppb	Parts Per Billion. 1
PPS	Pulse Per Second. 49
PPS-RTC	PPS-synchronized RTC. 55, 56, 60
PRC	Primary Reference Clock. 48
PRTC	Primary Reference Time Clock. 14
PTP	Precision Time Protocol. 2, 4–6, 13–15, 17–21, 23–26, 28–31, 33, 39–41, 45–50, 55–57, 59–68, 70, 72–74, 76, 78–80, 82, 83, 91–93
PTP-DAL	PTP Dataset Analysis Library. 5, 47, 56–58, 61, 70, 72, 80, 82, 87, 88
PTP-RTC	PTP-synchronized RTC. 55, 56, 60
PTS	Partial-timing Support. 4, 21, 41
QIA	Queuing-induced Asymmetry. 42, 45, 59, 76, 78, 80, 82, 84, 87, 91, 92
RAN	Radio Access Networks. 3, 21
RF	Radio Frequency. 1, 3
RRU	Remote Radio Unit. 22, 23, 46–54, 59, 60, 68
RTC	Real-time Clock. 9, 31, 48, 49, 55
RWFM	Random Walk Frequency Modulation. 12, 35

s-t-m	Slave To Master. 16–19, 29, 31, 33, 38–43, 45, 55, 56, 61, 63, 64, 67–70, 72, 74–76, 78, 91
SD	Standard Deviation. 61, 62, 64
SFD	Start Of Frame Delimiter. 49, 51
SFP	Small Form-factor Pluggable. 47
TAI	International Atomic Time. 9
TC	Transparent Clock. 4, 14
TDD	Time-division Duplex. 1, 3–5, 21–26, 28, 38, 39, 41, 42, 45, 49, 51–54, 59, 65, 66, 68, 70, 72, 74, 80, 91–93
TDM	Time-division Multiplexing. 3
TE	Time Error. 12
ToD	Time Of Day. 9, 14
Tx	Transmission. 52, 53
UE	User Equipment. 21, 23
UL	Uplink. 22, 23, 51, 52, 54, 68
UTC	Coordinated Universal Time. 9
VLAN	Virtual Local Area Network. xix, 47, 59, 61, 64, 65
WFM	White Frequency Modulation. 12, 35
WPM	White Phase Modulation. 12
XO	Crystal Oscillator. xvii, 48, 60, 68, 72, 74, 76–80, 84, 87, 90, 92

# Symbols

$A$	State transition matrix. 35, 94, 96, 97
$R_{line}$	Ethernet bit rate. 18, 51
$t'_{21}[n]$	Drift-compensated master-to-slave timestamp differences. 29, 30, 39
$t'_{43}[n]$	Drift-compensated slave-to-master timestamp differences. 29, 31, 39
$D$	Linear fractional frequency drift. 11, 12
$\gamma[n]$	Noise associated with the delay asymmetry. 16, 19, 20, 42
$\gamma_r[n]$	Random components from delay asymmetry. 20
$\gamma_s[n]$	Static components from delay asymmetry. 20
$d_{link}[n]$	Link level delay. 19
$d_{link}^{ms}[n]$	Link level delay in the master-to-slave direction. 18, 19
$d_{link}^{sm}[n]$	Link level delay in the slave-to-master direction. 18, 19
$\gamma_{phy}^m[n]$	Physical delay asymmetry from master node. 19
$\gamma_{phy}^s[n]$	Physical delay asymmetry from slave node. 19, 20
$d_{phy}^{rx,m}[n]$	Physical layer receive latency from master node. 18, 19
$d_{phy}^{tx,m}[n]$	Physical layer transmission delay from master node. 18, 19
$d_{phy}^{rx,s}[n]$	Physical layer receive latency from slave node. 18, 19
$d_{phy}^{tx,s}[n]$	Physical layer transmission delay from slave node. 18, 19
$d_{proc}[n]$	Processing delay. 19
$\gamma_{proc}[n]$	Switch Processing delay asymmetry. 19

$d_{proc}^{ms}[n]$	Switch processing delay in the master-to-slave direction. 19
$d_{proc}^{sm}[n]$	Switch processing delay in the slave-to-master direction. 19
$d_{prop}[n]$	Propagation delay. 19
$\gamma_{prop}[n]$	Propagation delay asymmetry. 19, 20
$d_{prop}^{ms}[n]$	Propagation delay in the master-to-slave direction. 19
$d_{prop}^{sm}[n]$	Propagation delay in the slave-to-master direction. 19
$d_{queue}[n]$	Queuing delay. 19
$\gamma_{queue}[n]$	Queueing delay asymmetry. 19
$d_{queue}^{ms}[n]$	Switch queuing delay in the master-to-slave direction. 19
$d_{queue}^{sm}[n]$	Switch queuing delay in the slave-to-master direction. 19
$d_{trans}[n]$	Transmission delay. 18, 19
$\gamma_{trans}[n]$	Transmission delay asymmetry. 19
$d_{trans}^{ms}[n]$	Transmission delay in the master-to-slave direction. 19
$d_{trans}^{sm}[n]$	Transmission delay in the slave-to-master direction. 19
$\Delta d_{ms}[n]$	First difference of delays in master-to-slave direction. 42, 43
$\Delta d_{sm}[n]$	First difference of delays in slave-to-master direction. 43
$\Delta t_{21}[n]$	First difference of delays in master-to-slave direction. 43
$\Delta t_{43}[n]$	First difference of delays in slave-to-master direction. 43
$d_{ms}[n]$	True master-to-slave PTP delay. 15, 16, 19, 42, 55, 56
$\hat{d}_{ms}[n]$	Estimated master-to-slave PTP delay. 15
$d_{sm}[n]$	True slave-to-master PTP delay. 16, 19, 42, 55, 56
$f(t)$	Instantaneous frequency. 10, 11
$f_{nom}$	Nominal frequency. 9–11
$f_s$	Sampling frequency. 54
<b>h</b>	Observation transition vector. 36
<b>H</b>	Observation transition matrix. 94, 95, 98, 99
$i_{fh}$	Period of FH packet. 50, 51
<b>I</b>	Identity matrix. 99
$l_{iq}$	IQ samples size. 51, 60

<b>K</b>	Kalman gain. 97–99
$\kappa_{ms}$	Total static delay in master-to-slave direction. 42, 43
$\kappa_{sm}$	Total static delay in slave-to-master direction. 42, 43
$\eta_a$	Number of antenna streams. 50, 60
$N_{bits}$	Number of bits. 18
$\eta_{oh}$	Ethernet overhead length. 51
$\eta_{spf}$	Number of IQ samples per frame. 50, 51, 60
$\nu$	Innovation vector. 97, 98
$\phi(t)$	Random phase deviation. 10–12
<b>P</b>	State estimation error covariance matrix. 37, 38, 96, 97, 99
$\psi[n]$	Time offset random noise. 35
$q_{ms}[n]$	Total queuing delay in master-to-slave direction. 42, 43, 45
$q_{sm}[n]$	Total queuing delay in slave-to-master direction. 42, 43, 45
<b>Q</b>	State noise covariance matrix. 36, 88, 90, 95, 97
<b>r</b>	Measurements noise covariance vector. 36
<b>R</b>	Measurement noise covariance matrix. 95, 99
<b>s</b>	State vector. 35–37, 94, 96, 97, 99
$\hat{s}$	Estimated State vector. 96–98
$\Delta_x[n]$	Time offset drift. 29, 40
$\hat{\Delta}_x[n]$	Time offset drift estimate. 29
$t_{fh}$	FH packet transmission delay. 51, 60, 67
$t_{ipg}$	Ethernet’s inter-packet gap. 51, 60, 67
$t_{ptp}$	PTP packet transmission delay. 60, 67
$T_s$	Sampling period. 50
$t_1[n]$	Sync departure timestamp. 15, 16, 47, 55, 56
$t_2[n]$	Sync arrival timestamp. 15, 16, 47, 55, 56
$t_{21}[n]$	Master-to-slave timestamp difference. 15, 16, 29, 40, 43, 45

$t_2^{pps}[n]$	Sync arrival timestamp according to the slave's PPS RTC.. 55, 56
$t_3[n]$	DelayReq departure timestamp. 15, 16, 47, 55, 56
$t_3^{pps}[n]$	DelayReq departure timestamp according to the slave's PPS RTC. 55, 56
$t_4[n]$	DelayResp arrival timestamp. 15, 16, 47, 55, 56
$t_{43}[n]$	Slave-to-master timestamp difference. 16, 29, 43, 45
$T(t)$	Clock time function. 11
$\Phi(t)$	Total instantaneous phase. 10, 11
$\hat{d}[n]$	Two-way delay measurement. 16
$V(t)$	Quasi-sinusoidal voltage. 10
$\mathbf{v}$	Measurement noise vector. 94, 95
$v[n]$	Time offset measurement noise. 36
$V_0$	Voltage amplitude. 10
$\sigma_v^2$	Time offset measurements noise variance. 37
$\mathbf{t}'_{21}[k]$	k-th vector of drift-compensated master-to-slave timestamp differences. 39
$\mathbf{t}'_{43}[k]$	k-th vector of drift-compensated slave-to-master timestamp differences. 39
$\mathbf{d}_{ms}[k]$	k-th vector of master-to-slave delays. 38
$\mathbf{d}_{sm}[k]$	k-th vector of slave-to-master delays. 38
$\mathbf{t}_{21}[k]$	k-th vector of master-to-slave timestamp differences. 38
$\mathbf{t}_{43}[k]$	k-th vector of slave-to-master timestamp differences. 38
$\mathbf{w}$	State noise vector. 35, 36, 94–97
$\omega(t)$	Instantaneous angular frequency. 10
$w_x[n]$	Time offset state noise. 35, 36
$w_y[n]$	Frequency offset state noise. 35, 36
$x[n]$	Discrete-time time offset. 15, 16, 29, 35, 55, 56
$x(t)$	Continuous-time time offset. 11, 12
$\hat{x}[n]$	Estimated discrete-time time offset. 38–40
$\tilde{x}[n]$	Noisy discrete-time time offset. 15, 16
$x_0$	Initial time offset. 11, 12

$y[n]$	Discrete-time frequency offset. 35
$y(t)$	Continuous-time frequency offset. 11, 12
$\hat{y}[n]$	Estimated discrete-time frequency offset. 40
$\tilde{y}[n]$	Noisy Discrete-time frequency offset. 16
$y_0$	Initial frequency offset. 11, 12
$\mathbf{z}$	Measurement vector. 94, 95, 97, 98
$z[n]$	Time offset measurements. 36



# List of Figures

2.1	Frequency, phase, and time synchronization. . . . .	8
2.2	Measurements of time error between two clocks based on [1]. . . . .	12
2.3	PTP hierarchical architecture. . . . .	13
2.4	PTP delay request-response mechanism. . . . .	14
2.5	Illustration of packet delays between the timestamp point at the master and the timestamp point at the slave node. . . . .	17
3.1	Traditional mobile architecture. . . . .	22
3.2	C-RAN architecture. . . . .	22
3.3	Representation of TDD frame DDDDDDDFUU. . . . .	23
3.4	Timing diagram for PTP transmit opportunity in FH (based on [2]). . . . .	24
3.5	Classification of PTP packets based on the experienced delays. . . . .	25
3.6	PTP cycle location with different exchange periods and TDD frame with 7 ms of downlink, 1 ms of gap and 2 ms of uplink. . . . .	27
3.7	Illustration of K-means classification process. . . . .	30
3.8	Illustration of the K-means outlier threshold. . . . .	31
3.9	Kalman filter loop. . . . .	37
3.10	Window-based filtering for time offset drift estimate. . . . .	40
3.11	Timestamp differences in the downlink direction. . . . .	44
3.12	Timestamp differences in the uplink direction. . . . .	44
4.1	Testbed overview. . . . .	47
4.2	Overview of the testbed connections and clock distribution paths. . . . .	48
4.3	Testbed timestamp position. . . . .	49
4.4	Illustration of the FH packet structure. . . . .	51
4.5	Testbed traffic pattern in CBR FH. . . . .	51

4.6	Tx design for controlling the FH traffic at the RRU. . . . .	52
4.7	Tx design for controlling the FH traffic at the BBU. . . . .	53
4.8	TDD State machine. . . . .	54
4.9	PTP delay request-response mechanism. . . . .	56
4.10	PTP-DAL architecture. . . . .	57
5.1	True delay distribution in the m-t-s and s-t-m direction using port-based VLAN in a one to four-hop configuration in absence of FH traffic. . . . .	62
5.2	True delay distribution in m-t-s and s-t-m direction using multiples physical switches in absence of FH traffic. The 1-hop configuration uses switch #4, 2-hops uses switches #4 and #1, 3-hops uses #4, #1 and #2 and 4-hops with all switches. . . . .	63
5.3	Delay statistics in m-t-s direction using multiple physical switches. . . . .	65
5.4	Delay distribution with FH traffic. . . . .	66
5.5	Delay distribution and PDF using VLAN-based hops. . . . .	67
5.6	Delay distribution with FH traffic. . . . .	68
5.7	Delay distribution and PDF using XO oscillator. . . . .	69
5.8	Delay distribution and PDF using OCXO oscillator. . . . .	69
5.9	PTP cycle location in the experiment with the XO oscillator. . . . .	71
5.10	K-means classification in the experiment with XO. . . . .	73
5.11	Confusion matrix in the experiment with XO. . . . .	73
5.12	TDD packet classification in the experiment with crystal oscillator (XO). . . . .	74
5.13	TDD packet classification in the experiment with oven-controlled crystal oscillator (OCXO). . . . .	75
5.14	Bar plot with the probability of each class calculated with the output from the K-means classifier. . . . .	75
5.15	Estimated delay difference in the experiment with XO oscillator. . . . .	76
5.16	Estimated QIA delay in the experiment with XO. . . . .	77
5.17	Time offset measurements error in the experiment with XO. . . . .	77
5.18	Estimated delay difference in the experiment with OCXO oscillator. . . . .	78
5.19	Estimated QIA in the experiment with OCXO oscillator. . . . .	78
5.20	Time offset measurement error in the experiment with OCXO oscillator. . . . .	79

5.21	maximum absolute time error ( $\max TE $ ) results from the experiments with four physical hops. . . . .	79
5.22	$\max TE $ results from the experiment with four physical hops and XO oscillator. . . . .	81
5.23	$\max TE $ violin plots from the experiment with four physical hops and XO oscillator. . . . .	83
5.24	$\max TE $ results from the experiment with four physical hops and OCXO oscillator. . . . .	85
5.25	$\max TE $ violin plots from the experiment with four physical hops and OCXO oscillator. . . . .	86
5.26	Cumulative drift estimation error on the experiment with four physical hops. . . . .	87
5.27	$\max TE $ in terms of window length using four physical hops and without bias correction. . . . .	88
5.28	Heatmap from KF Q matrix tuning using four physical hops and the XO oscillator. . . . .	89
5.29	Heatmap from KF Q matrix tuning using four physical hops and the OCXO oscillator. . . . .	89
A.1	Predict and update stages from Kalman filtering. . . . .	96
A.2	Pictorial representation of Kalman filtering process. . . . .	98

# List of Tables

2.1	Main sources of delay. . . . .	20
4.1	PTP-Unaware switches used in the testbed. . . . .	47
5.1	Transmission delay considering a PTP frame with 80 bytes and a GbE interface.	63
5.2	Average delays from switch #1 using port-based VLAN. . . . .	63
5.3	Average delays from switch #2 using port-based virtual local area network (VLAN). . . . .	64
5.4	Delays using multiples physical switches. . . . .	65
5.5	Ideal delay asymmetry from the experiment with XO oscillator calculated by PTP-DAL. . . . .	81
5.6	Static delay asymmetry from good packets from the experiment with the XO oscillator. . . . .	81
5.7	Ideal delay asymmetry from the experiment with OCXO oscillator. . . . .	84
5.8	Static delay asymmetry from good packets from the experiment with OCXO oscillator. . . . .	85

# Contents

<b>Acknowledgment</b>	<b>vi</b>
<b>Glossary</b>	<b>vii</b>
<b>Symbols</b>	<b>xi</b>
<b>List of Figures</b>	<b>xvi</b>
<b>List of Tables</b>	<b>xix</b>
<b>Contents</b>	<b>xx</b>
<b>Abstract</b>	<b>xxiii</b>
<b>Resumo</b>	<b>xxiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Outline . . . . .	5
1.3 Contributions . . . . .	6
1.4 Publications . . . . .	6
<b>2 Fundamentals of Time Synchronization in Packet-based Networks</b>	<b>7</b>
2.1 Synchronization Basics . . . . .	7
2.1.1 Types of Synchronization: Frequency, Phase and Time . . . . .	8
2.1.2 Clock Basics . . . . .	9
2.1.3 Measurements of Time Error . . . . .	12
2.2 IEEE 1588 Precision Time Protocol (PTP) . . . . .	13
2.2.1 PTP Timing-exchange Mechanisms . . . . .	14

2.2.2	Sources of Delay Asymmetry . . . . .	17
<b>3</b>	<b>Synchronization in TDD Fronthaul</b>	<b>21</b>
3.1	Overview . . . . .	21
3.2	PTP Classification Based on TDD cycle . . . . .	24
3.2.1	PTP Cycle Location . . . . .	25
3.2.2	K-means based Classifier . . . . .	28
3.3	Time and Frequency Offset Estimators . . . . .	31
3.3.1	Related Work . . . . .	31
3.3.2	Kalman Filtering . . . . .	34
3.3.3	Packet Filtering . . . . .	38
3.3.4	Time Offset Drift Estimation . . . . .	40
3.4	Delay Asymmetry Estimation . . . . .	40
3.4.1	Related work . . . . .	41
3.4.2	Queuing-Induced PTP Delay Asymmetry . . . . .	41
<b>4</b>	<b>Testbed</b>	<b>46</b>
4.1	Testbed Overview . . . . .	46
4.2	PTP Infrastructure . . . . .	49
4.3	Fronthaul Traffic . . . . .	50
4.4	TDD FH Implementation . . . . .	51
4.5	Acquisition of Labeled Datasets . . . . .	55
4.6	PTP-DAL . . . . .	56
<b>5</b>	<b>Results</b>	<b>59</b>
5.1	Experimental Setup . . . . .	59
5.2	Delay Analysis . . . . .	61
5.2.1	PTP without FH Traffic . . . . .	61
5.2.2	PTP with FH Traffic . . . . .	65
5.3	TDD Packet Classifier Experiments . . . . .	69
5.3.1	PTP Cycle location . . . . .	70
5.3.2	K-means Based Classifier . . . . .	72
5.4	Queuing Induced Asymmetry Experiments . . . . .	76
5.5	Synchronization for TDD FH Experiments . . . . .	80

<b>6 Conclusion and Future Work</b>	<b>91</b>
6.1 Conclusion . . . . .	91
6.2 Future Work . . . . .	93
<b>A Discrete Kalman Filter</b>	<b>94</b>
A.1 State-space Representation . . . . .	94
A.2 Prediction Stage . . . . .	96
A.3 Update Stage . . . . .	97
<b>Bibliography</b>	<b>100</b>

## **Abstract**

In fifth-generation (5G) mobile communications, clock synchronization is essential for the correct operation of the network. In this context, as 5G networks are moving to packet-based networks, the IEEE 1588 precision time protocol (PTP) is the industry-accepted protocol for flexibly distributing time synchronization. However, when PTP packets are transported in a cost-effective network without specialized equipment, two main issues impact the performance of PTP: packet delay variation (PDV) and delay asymmetries. In this type of network, the use of algorithms to filter the errors introduced by PDV and delay asymmetry is usually helpful. Thus, this work aims to present an analysis of PTP performance on a cost-effective network when time-division duplex (TDD) is used on the radio interface. We propose new algorithms to improve the PTP performance that take advantage of the fronthaul (FH) periodic utilization pattern when using TDD on the radio interface. Our evaluation is carried out on a realistic testbed composed of field-programmable gate arrays (FPGAs). The results indicate that the PTP performance can be improved on this type of network by using the knowledge about the FH periodicity to correct the queuing-induced asymmetry and more effectively filter the PDV.



## Resumo

Em redes móveis de quinta geração (5G), a sincronização do relógio é essencial para o correto funcionamento da rede. Nesse contexto, à medida que as redes 5G estão migrando para redes baseadas em pacotes, o protocolo de tempo de precisão IEEE 1588 (PTP) é o protocolo aceito pela indústria para distribuição flexível de sincronização de tempo. No entanto, quando os pacotes PTP são transportados em uma rede mais econômica, sem equipamento especializado, dois problemas principais afetam o desempenho do PTP: variação de atraso de pacotes (PDV) e assimetrias de atraso. Nesse tipo de rede, o uso de algoritmos para filtrar os erros introduzidos por PDV e assimetria de atraso costuma ser útil. Assim, este trabalho tem como objetivo apresentar uma análise do desempenho do PTP em uma rede econômica quando *duplex* por divisão de tempo (TDD) é usado na interface de rádio. Este trabalho propõe novos algoritmos para melhorar o desempenho do PTP que aproveitam o padrão de utilização periódica do *fronthaul* (FH) ao usar TDD na interface de rádio. Nossa avaliação é realizada em um ambiente de teste realista composto de arranjos de porta programável em campo (FPGAs). Os resultados indicam que o desempenho do PTP pode ser melhorado neste tipo de rede usando o conhecimento sobre a periodicidade do FH para corrigir a assimetria induzida pelo enfileiramento e filtrar o PDV de forma mais eficaz.

# Chapter 1

## Introduction

The synchronization of clocks in distributed systems has been an important long-standing problem. It enables applications to operate using the same notion of time, which enables key functions like determining the order of transactions in a financial application. Synchronization is a fundamental building block in different fields such as wireless communication, industrial automation [3], Internet of things (IoT) [4], financial applications [5] and many more [6, 7]. However, synchronizing two or more clocks is typically challenging due to many different aspects. One of them is that each clock tends to drift differently due to inherent instabilities and environmental factors such as temperature [8].

In mobile networks, time synchronization became crucial in the fourth-generation (4G) and fifth-generation (5G) networks. For instance, the frequency accuracy requirement within  $\pm 50$  parts per billion (ppb), needed for basic operations on base station (BS), has not changed in the recent mobile generations [9, Chapter 7]. For example, BS needs accurate frequency to transmit in the correct radio frequency (RF) spectrum and to support handover operation. In contrast, time synchronization becomes more important with the advance of cooperative transmission technologies in recent generations. For instance, 5G mostly rely on time-division duplex (TDD) technology [10], which requires time synchronization of the BS within  $\pm 1.5\mu\text{s}$  absolute time error [11].

In the majority of mobile network deployments, the primary method of synchronizing the clocks of distributed network devices was through the utilization of global navigation satellite systems (GNSS). However, the implementation of 5G mobile networks introduced several demands, such as the enhancement of cell density, indoor cell coverage, and deployment capabilities in areas that present geographical challenges such as tunnels, buildings, and factories

where satellite visibility is limited. As a consequence, the use of GNSS-based synchronization in these scenarios becomes difficult as the GNSS receiver must have a line of sight to the satellites. Thus, new techniques are being explored to meet the stringent time synchronization required by 5G networks and reduce deployment costs.

The industry is adopting a combination of GNSS and IEEE 1588 precision time protocol (PTP) [12]. Rather than deploying GNSS everywhere, these networks are increasingly using the IEEE 1588 packet-based network synchronization, where GNSS still provide the primary timing reference and PTP is responsible for distributing the time synchronization among the network devices. PTP enables synchronization by exchanging time information over packet-based networks between highly accurate devices, called master clocks, and multiple less accurate clocks, called slave clocks. The performance heavily depends on the networks that are being used to transmit the PTP packets. For example, in PTP-unaware networks, the PTP performance can be very affected depending on the nature of the network traffic being transmitted together with PTP, which can lead to high packet delay variation (PDV) and delay asymmetry. As clarified in the sequel, this work will focus on this scenario as this type of network, if properly designed, can be used to provide accurate synchronization in a very cost-effective manner.

In the remainder of this chapter, Section 1.1 describes the context and motivation for this work, presenting an overview of the use of PTP in timing-unaware networks and how the information about the nature of the traffic can benefit the synchronization performance in this scenario. Next, Section 1.2 outlines the content presented in the chapters of this work. Finally, Section 1.3 summarizes the research contributions.

## 1.1 Motivation

Mobile networks always required some kind of synchronization. Most legacy networks utilized frequency-division duplex (FDD) as the technique to separate the downlink and uplink channels. In FDD, both the uplink and downlink radios transmit simultaneously but at different frequencies. A predetermined offset is used to separate these frequencies and avoid any interference. As a consequence, when using FDD, frequency synchronization is necessary at the radio interface. This requirement for synchronization is necessary for many reasons. For example, as the local oscillator at the radio equipment is utilized as the source for the frequency being transmitted by the radio on the air, any deviation in the frequency of the local oscillator results

in the radio transmitting at an incorrect frequency. Therefore, one of the reasons for frequency synchronization is to ensure that the radios transmit at the correct RF.

Traditionally, frequency synchronization was delivered to the network nodes using time-division multiplexing (TDM) networks, which were naturally synchronous. For example, in 4G cloud radio access network (C-RAN), the fronthaul (FH) network was based on a point-to-point synchronous network using technology like common public radio interface (CPRI) [13]. However, motivated by the flexibility and cost-effectiveness of the Ethernet technology and to meet the demands of ever-growing bandwidth, the industry started to move the radio access networks (RAN) and the core network to packet-based networks. The problem is that, unlike CPRI that has frequency synchronization built-in, packet-based technologies like Ethernet are asynchronous and require other means for distributing synchronization.

Additionally, mobile networks also support technologies like TDD. In TDD mode, networks optimize spectral efficiency by sharing the same frequency for uplink and downlink radio, but at different time slots. As a consequence, TDD requires accurate time synchronization due to the time-sensitive nature of TDD operation. For example, to avoid unwanted interference between neighboring base stations and user equipment. In the case of more complex networks, even tighter time synchronization is required to enable more advanced technologies such as carrier aggregation (CA), distributed multiple-input multiple-output (MIMO) and coordinated multi-point (CoMP) [14, Chapter 6]. Therefore, a crucial requirement for deploying the recent generations of mobile networks in a packet-based network is the ability to provide highly accurate clock synchronization.

The most common method for achieving time synchronization is through GNSS. The GNSS receiver located within the BS synchronizes with the atomic clocks present in satellites, thereby providing a reliable and highly accurate synchronization. However, when it comes to 5G networks, several challenges must be addressed. For instance, the GNSS antenna must have a line of sight (LOS) to the satellites, which becomes increasingly difficult in dense urban areas and infeasible in indoor environments. Additionally, as the industry shifts towards the use of smaller, more densely distributed BS in 5G networks, the cost of installing GNSS equipment at all RAN sites becomes impractical. Besides, providing synchronization only using GNSS becomes problematic due to security issues such as *jamming*, where the GNSS signal is completely blocked and *spoofing*, where the GNSS signal is replaced by a similar signal but incorrect one [15]. Hence, as time synchronization becomes more important in telecommunication

networks, the need for distributing synchronization using packet networks becomes necessary.

In this context, the PTP is the industry-accepted protocol to distribute time synchronization in a packet network in a highly scalable way. However, one major issue is that the timing messages traveling between the master and slave node can encounter several intermediate switches and routers, accumulating random delays at each node. This randomness in the synchronization path is called PDV and affects the synchronization performance. Additionally, the PTP performance is also affected when the packet delay in the forward and reverse directions are unequal, known as delay asymmetry. The problem is that Ethernet was not designed to guarantee end-to-end latency. Thus, to achieve high synchronization accuracy, ITU-T has defined a full-timing support (FTS) [16] network to be used between the master and slave clocks.

In a FTS network, also known as PTP-aware network, all the networks elements are composed by boundary clock (BC) and transparent clock (TC) [17]. The BC and TC are specialized switches or routers that provide the functionality to mitigate the PDV and delay asymmetry that the PTP packets suffer when traversing the network. For instance, TC nodes are specialized nodes that can measure and compensate for the time taken for the PTP packet to pass through the device to alleviate the PDV. Hence, FTS networks provide a more reliable and accurate synchronization. However, it is more expensive.

Synchronizing clocks in a timing-unaware network, i.e., without using FTS networks, is a more complex challenge. This type of network is composed of PTP-unaware network elements, switches and routers that do not provide any specialized functionality for PTP. This common case is referred to as partial-timing support (PTS) [18]. The presence of PTP-unaware devices impacts synchronization performance due to the high PDV, caused by variable queuing delays over the network, and delay asymmetries. However, this scenario offers flexibility and cost-effectiveness. In this type of network, there is no need to use specialized equipment on the transport network, so network operators could use cheaper equipment or even already deployed networks without the need to make hardware modifications to accommodate PTP-based synchronization. In PTS networks, PTP can also be used as a second source of timing to backup GNSS, which is defined as assisted partial-timing support (APTS) [18].

In this context, the objective of this work is to present a detailed analysis of PTP performance in a cost-effective PTP-unaware network when TDD is being used to coordinate the transmission and reception of data on the radio interface. In this type of network, the FH network exhibits a periodic utilization pattern due to the use of TDD system on the radio interface.

By taking advantage of the periodic nature of the FH traffic, the PTP performance can be improved. This idea was proposed by our partners at Ericsson in [2]. Thus, this work focuses on extending and validating the ideas presented in [2]. To this end, this work uses a testbed based on field-programmable gate arrays (FPGAs) to provide a realistic analysis of how the FH periodic traffic affects PTP and also proposes new techniques to improve the synchronization performance by using the information about the TDD pattern to mitigate PDV and the asymmetry introduced by queuing delay using specialized algorithms.

## 1.2 Outline

This work is organized as follows:

- Chapter 2 provides the fundamental concepts used by this work. It presents some definitions regarding clocks, the noises that affect their accuracies and stability, and the metrics used for their characterization. It also describes the IEEE 1588 protocol, including a compact review of the noises that affect its performance. Lastly, it presents a discussion about the source of delay asymmetries that affects PTP.
- Chapter 3 discusses the key ideas about using the information about the periodicity from the FH traffic when using TDD on the radio interface to improve the PTP synchronization performance in a timing-unaware network. Additionally, the chapter presents a literature review of different synchronization algorithms. In particular, it presents different strategies to estimate the time and frequency offset. Moreover, Chapter 3 discusses a new method for estimating the bias induced by the queuing delay in such networks.
- Chapter 4 presents a brief description of the testbed used in this work. More specifically, the changes introduced to make a more realistic analysis in a FH that presents a periodic pattern. It also discusses how the datasets are acquired and how the evaluation is carried out using a processing tool called PTP dataset analysis library (PTP-DAL).
- Chapter 5 presents the experimental results using the datasets acquired from the testbed. It starts by presenting an analysis of the PTP delay distribution observed on the datasets used in this work. It also includes the analysis of two types of hop configurations: VLAN-based hops using one switch and multiple physical switches. It also describes the results

achieved by the methods used to classify the PTP packets based on the FH traffic. Furthermore, it includes a discussion about the results achieved by the proposed method for estimating the bias induced by queuing delay. Finally, it presents the achieved synchronization performance with different algorithms and oscillators.

- Lastly, Chapter 6 presents a summary of the results present in this work and provides directions for future work.

### 1.3 Contributions

The contributions of this work are:

- The analysis of PTP performance in a PTP-unaware network that presents periodic traffic using different types of synchronization algorithms such as Kalman filter (KF), sample minimum, maximum and average.
- The adaptations proposed to the KF and sample average algorithms to use the knowledge of the good and bad packets to improve the synchronization performance.
- The proposal of a new method to classify PTP packets using a K-means based classifier.
- The proposal of a new method to estimate the queuing-induced delay asymmetry in a network that presents periodic traffic.
- Discussion about delay distribution present in FH networks, such as the processing delay produced by different switches and the impact of these delays on the performance of PTP.
- A hardware-based evaluation of the impact of periodic FH traffic on the PTP performance.

### 1.4 Publications

The work in [19] was produced as an initial effort to understand the use of KF for the specific problem of clock synchronization. More specifically, the work focuses on how the clock model, in conjunction with the measurements provided by the PTP algorithm, can be expressed in the Kalman Filter form. It also presents the KF performance using a legacy Ethernet network with different network loads and two different oscillators.

## Chapter 2

# Fundamentals of Time Synchronization in Packet-based Networks

This chapter introduces the fundamental concepts used in this work and is divided into two major sections. In Section 2.1, the basic concepts related to clock synchronization are presented. For example, the different types of synchronization, how a clock device can be modeled mathematically, and which metrics are used for performance characterization. Next, Section 2.2 describes how the IEEE 1588 technology works.

### 2.1 Synchronization Basics

Maintaining many devices synchronized is a very important task in many applications. Generally, these applications need to have some kind of synchronization for various devices at different geospatial locations to perform some type of coordinated task. These devices usually perform timekeeping locally using a clock. The clock can be defined as a device composed of an oscillator and a counter. The oscillator is a device that produces a periodic electric signal called *clock signal*, and the counter is responsible for keeping track of the oscillator's cycles to keep track of the elapsed time.

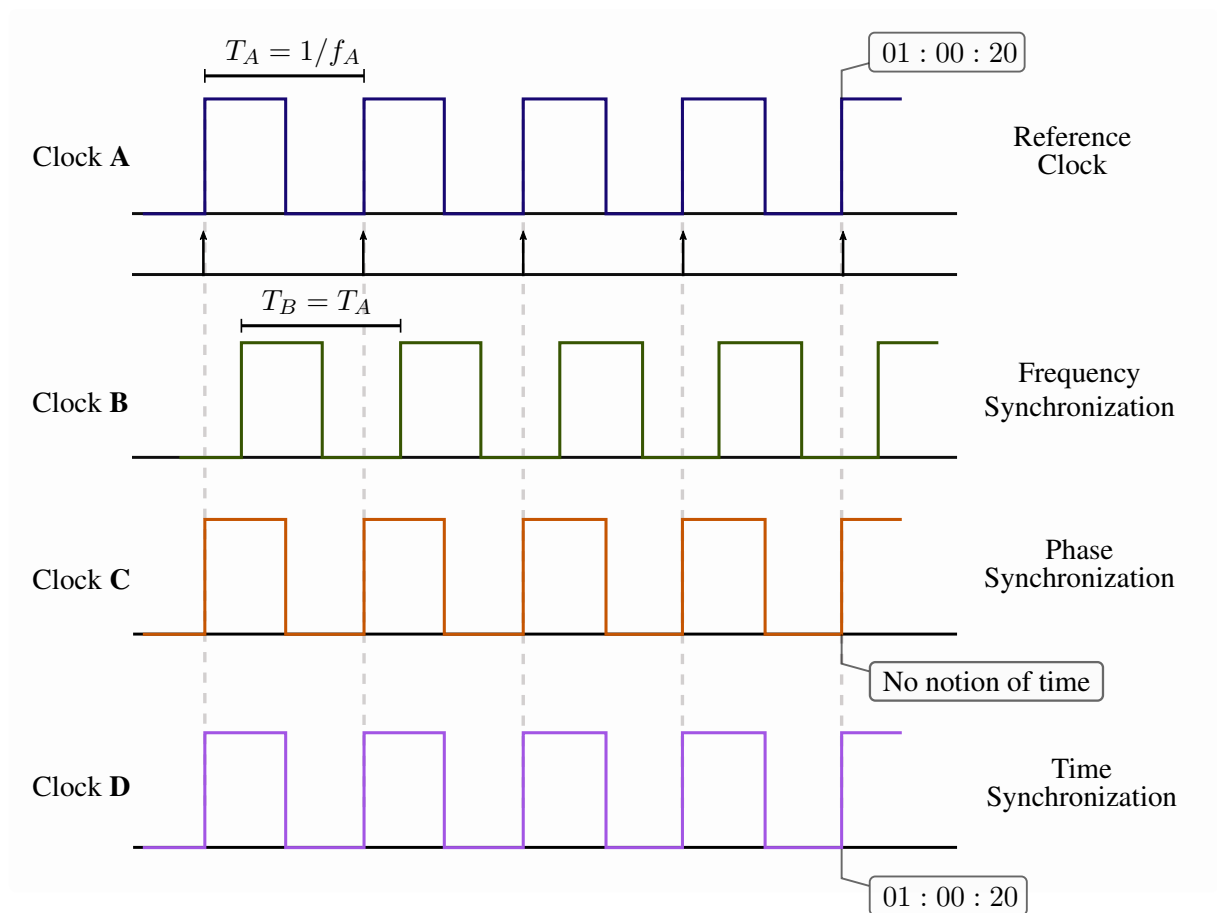
However, these timekeeping techniques are subject to errors that can accumulate over time due to the imperfection of the clock frequency generated by the oscillators. The clock frequency varies mainly with power, aging, and temperature [8]. Therefore, two similar oscillators are not guaranteed to oscillate with identical frequency. The magnitude of such errors depends on the oscillator's precision and stability, and most of the time the quality of such devices is



proportional to their prices. As a result, it becomes almost impossible to maintain any kind of synchronization between many devices only using the local clock without using any additional technique. Thus, there are many methods for providing synchronization to various devices across a distributed network without depending only on the local time clock.

### 2.1.1 Types of Synchronization: Frequency, Phase and Time

The word synchronization has multiple meanings. It can be mainly classified as frequency, phase or time synchronization. Thus, it is essential to distinguish these different types of alignment. Fig. 2.1 highlights the differences. The specific definitions presented below are based on the ITU-T Recommendations G.8260 [20] and G.810 [21].



**Figure 2.1:** Frequency, phase, and time synchronization.

The frequency synchronization is related to the alignment of clocks in frequency. It means that the significant instants, for example, the rising edges of different clocks, occur at the same

rate. In other words, they have the same number of pulses within the same period. This type of alignment is also referred to as *syntonization*.

Another type of synchronization is the so-called phase synchronization. When two or more clocks are said to be synchronized in phase, it means that significant events occur at the same instant. Note that, if the rising edge of two clocks occurs at the same instant, it also follows that they occur at the same rate. Therefore, phase synchronization implies syntonization.

Finally, time synchronization is closely related to phase synchronization. In this type of alignment, the clocks are synchronized in phase, and they also share a common time scale, e.g., each period of the reference time is marked and dated. Thus, the devices should share the same absolute time of day (ToD). For this type of synchronization, the devices should be equipped with a real-time clock (RTC), which is a module responsible for keeping track of the ToD.

Note that time synchronization implies phase synchronization, and phase synchronization implies frequency synchronization. Therefore, distributing time synchronization among different devices is one way to achieve both frequency and phase synchronization. This is the idea of the widely used IEEE 1588 protocol, which will be discussed later in this chapter.

## 2.1.2 Clock Basics

In the synchronization field, the word clock is most used to refer to a device that is capable of measuring time, i.e., an RTC. It is usually composed of an oscillator and a time counter. The oscillator is a device that produces a periodic electric signal, referred to as *clock signal*, which drives the time counter. The counter is responsible for keeping track of the oscillator cycles that have occurred, to keep track of the elapsed time. For example, if an RTC is driven by a clock signal that has a nominal frequency  $f_{nom} = 100$  MHz, at each rising edge the RTC would increment  $1/f_{nom} = 10$  ns. Thus, after  $f_{nom} \times (1/f_{nom})$  the RTC would accumulate a real-world second. Additionally, an RTC usually holds ToD information in a specific time standard, such as coordinated universal time (UTC) or international atomic time (TAI).

If a clock is ideal, it has a perfect frequency, which would yield a perfect measurement of the elapsed time. However, in practice, the clock is not ideal. That means the clock frequency that drives the counter is not perfect. Due to the random phase and frequency fluctuations that affect any oscillator. Hence, the time counter is strongly disturbed over time by the oscillator's deviation, which is caused by many factors such as temperature and aging effects [8].

The periodic signal produced by an oscillator could be modeled as a pure sinusoidal wave

if the clock was ideal. However, any real device is disturbed by unavoidable random noises, such as drifts due to frequency instability or environmental effects. Hence, many studies discuss a tractable mathematical model for the output of an oscillator. In the literature [22–24], a commonly adopted model is the quasi-sinusoidal voltage, mathematically represented by:

$$V(t) = V_0 \sin \Phi(t), \quad (2.1)$$

where  $V_0$  is the amplitude that, for simplicity, is assumed to be constant,  $f_{nom}$  is the nominal frequency and  $\Phi(t)$  is the total instantaneous phase, which is measured in radians. More specifically, the total instantaneous phase  $\Phi(t)$  can be given as follows:

$$\Phi(t) = 2\pi f_{nom}t + \phi(t), \quad (2.2)$$

where  $\phi(t)$  is the so-called random phase deviation. The term  $\phi(t)$  is a random process that models all the random deviations with respect to the nominal frequency  $f_{nom}$  [22].

In this model, the random phase deviation  $\phi(t)$  is typically the primary concern when dealing with oscillators. The phenomenon is well understood by the literature, and there are several analytical techniques for its characterization. In the sequel, we discuss some very useful measures used in the literature.

First, it is interesting to note that the timing information of the signal  $V(t)$  is carried by its total instantaneous phase  $\Phi(t)$ . Additionally, recall that the definition of instantaneous angular frequency is the rate of change of phase, as follows:

$$\omega(t) = 2\pi f(t) = \frac{d\Phi(t)}{dt}, \quad (2.3)$$

where  $\omega(t)$  is the instantaneous angular frequency given in radians per second and  $f(t)$  is the instantaneous frequency given in hertz. Thus, assuming (2.3) and (2.2), the *instantaneous frequency* can be expressed as defined in [25]:

$$\begin{aligned} f(t) &= \frac{1}{2\pi} \frac{d\Phi(t)}{dt} \\ &= f_{nom} + \frac{1}{2\pi} \frac{d\phi(t)}{dt}, \end{aligned} \quad (2.4)$$

where the second term on the right-hand side represents the instantaneous frequency fluctuation. In other words, it is the rate of phase change over time.

One should note that oscillators can only be characterized by measuring them against a reference oscillator. Thus, a useful definition is the so-called *fractional*, or *normalized frequency*

*offset* defined as:

$$y(t) = \frac{f(t) - f_{nom}}{f_{nom}}, \quad (2.5)$$

which is the instantaneous frequency fluctuation normalized to the nominal frequency. This quantity is dimensionless and is generally given in terms of parts per million (ppm) or parts per billion (ppb). The normalized frequency offset can also be interpreted in a more general way, where the  $f_{nom}$  is defined as the frequency of the reference clock oscillator.

Another important metric is the time offset  $x(t)$ , which is the difference between the local time and the reference (ideal) time, defined as follows:

$$x(t) = T(t) - t, \quad (2.6)$$

where  $T(t)$  is the local time measured by the clock at the "true" instant  $t$ . Mathematically, the time function  $T(t)$  is defined as:

$$T(t) = \frac{\Phi(t)}{2\pi f_{nom}} = t + \frac{\phi(t)}{2\pi f_{nom}}. \quad (2.7)$$

Thus, by substituting (2.7) in (2.6), the time offset can be interpreted as the random phase deviation converted into time and measured in seconds. Another way to calculate the time offset is by assuming (2.4) and integrating (2.5).

Note that the true instantaneous frequency  $f(t)$  is impossible to be measured since it will always involve some sample time, e.g., a window of time from which the oscillator is observed. Therefore, we can define an *average fractional frequency* over a period  $\Delta t$  as:

$$y(t) = \frac{x(t + \Delta t) - x(t)}{\Delta t}, \quad (2.8)$$

where the smaller the interval  $\Delta t$ , the better the approximation of instantaneous frequency. It can also be rewritten as:

$$x(t + \Delta t) = x(t) + y(t)\Delta t, \quad (2.9)$$

which yields a useful model to calculate the time offset in a recursive way.

In order to better model how the  $x(t)$  and  $y(t)$  evolve over time, these quantities are usually denoted by separating deterministic and random effects. For example, according to ITU-T G.810 [21], the time offset can be modeled as (see [26] for a detailed derivation):

$$x(t) = x_0 + y_0 t + \frac{D}{2} t^2 + \frac{\phi(t)}{2\pi f_{nom}}, \quad (2.10)$$

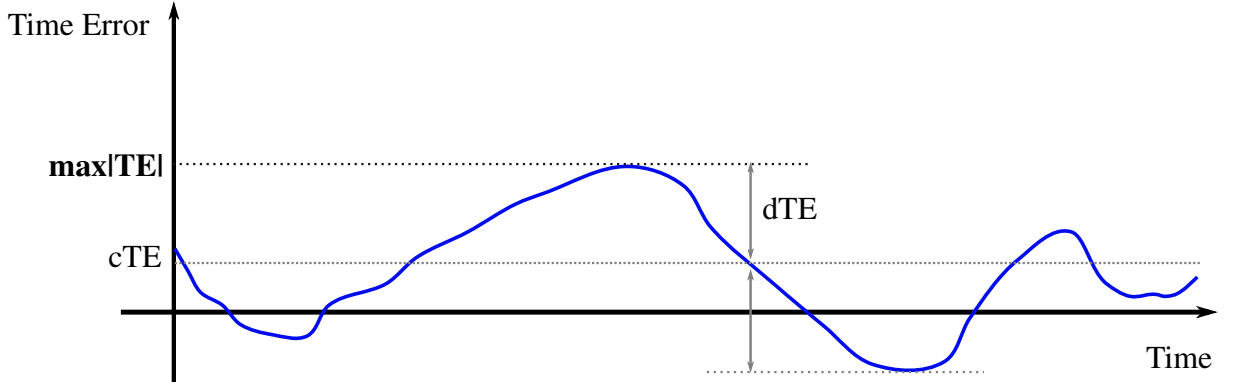
where  $x_0$  and  $y_0$  are the initial time and frequency offsets at the initial instant  $t_0 = 0$  and  $D$  represents the linear frequency drift.

In the literature, the random phase deviation  $\phi(t)$  is usually described as composed of five types of noise, together with their effects on  $x(t)$  and  $y(t)$ . These five noises are white phase modulation (WPM), flicker phase modulation (FPM), white frequency modulation (WFM), flicker frequency modulation (FFM) and random walk frequency modulation (RWFM). These random noises are generally described in the frequency domain through power spectral density  $S_y(f)$  [27] or, in the time domain, by the Allan variance  $\sigma_y^2(\tau)$  [22]. However, the analysis of  $\phi(t)$  in terms of  $S_y(f)$  and  $\sigma_y^2(\tau)$  is beyond the scope of this work.

### 2.1.3 Measurements of Time Error

As discussed in the last section, the time offset  $x(t)$  represents the difference between the time reported by a clock compared to the time reported by a reference clock at the same instant, using (2.6). In practical scenarios, the time offset is usually used to measure performance.

In this context, ITU-T G.8271.1 [28] specifies the  $x(t)$ , also called time error (TE), in terms of  $\max|TE|$ , constant time error (cTE) and dynamic time error (dTE), as shown in Fig. 2.2.



**Figure 2.2:** Measurements of time error between two clocks based on [1].

The cTE and the dTE represent the constant and the variable part of the time error, respectively. Therefore, the TE at any point may be expressed in terms of cTE and dTE, as follows:

$$TE(t) = cTE + dTE(t). \quad (2.11)$$

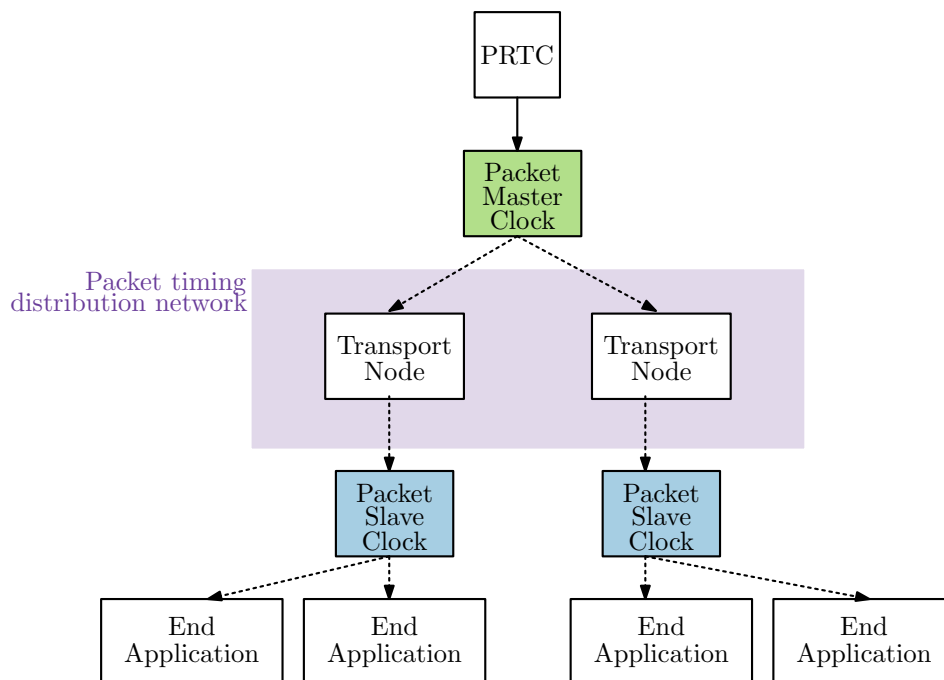
The  $\max|\text{TE}|$  metric represents the absolute time error, as:

$$\max|\text{TE}| = \max|c\text{TE} + d\text{TE}(t)|. \quad (2.12)$$

These metrics are used to constrain both the noise generated by an individual clock and the noise accumulated from a chain of clocks in a network. More specifically, the  $\max|\text{TE}|$  is used to specify the time error limit in a network, and it will be used as the main measurement to assert performance in this work.

## 2.2 IEEE 1588 Precision Time Protocol (PTP)

The IEEE 1588 standard, known as PTP [17], is the industry-accepted protocol for distributing precise time and frequency synchronization in a distributed packet network. Its first version was released in 2002 [12], and it has been further improved in 2008 [29], and in 2020 with [17], which adds many new features, such as the *high accuracy profile* to achieve nanoseconds accuracy.



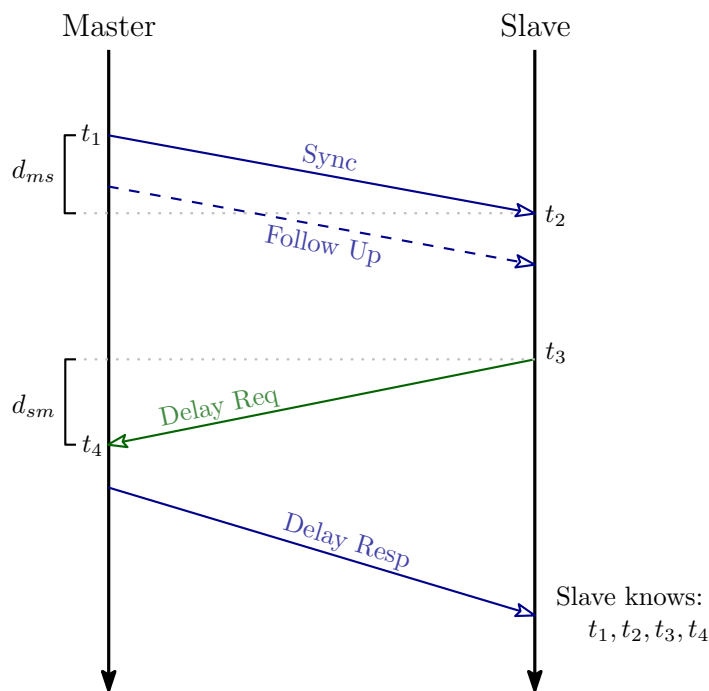
**Figure 2.3:** PTP hierarchical architecture.

At a high level, PTP works in a master-slave hierarchical architecture, as illustrated in Fig. 2.3. It starts defining the PTP instance at the top of the hierarchy using the so-called best master clock (BMC) algorithm, which configures the more precise clock in the network as the

master, and the others as the slaves. The packet master clock determines the reference time for the entire system and exchanges timing information with the slave clocks periodically and continuously. The slave clock can then adjust its local clock based on this timing information. Besides, the ToD reference comes from the so-called primary reference time clock (PRTC), which feeds a physical time/phase reference to the packet master clock.

Two types of clocks are defined in [12], BC and ordinary clocks (OC). The OC are end devices that only have one port and act as master or slave clocks. Another type of clock, called TC, was defined in [29]. The BC and TC clocks contain specialized IEEE 1588 functionality. These devices have mechanisms to mitigate PDV. For example, TC devices can measure the time taken for the PTP message to pass through the device, e.g., the *residence time* of the PTP message. The residence time measured by a TC device is added to the corresponding `correctionField` of the PTP message. Thus, the other devices in the system can use the delay measured by these specialized devices to compensate for the additional delay introduced by the network elements between the master and slave clock.

### 2.2.1 PTP Timing-exchange Mechanisms



**Figure 2.4:** PTP delay request-response mechanism.

The synchronization between the master and slave clocks is achieved by exchanging

PTP messages. One of the PTP timing-exchange mechanisms is the so-called *Delay Request-Response*, summarized in Fig. 2.4. As defined in [17], this mechanism uses four different messages: *Sync*, and *Delay\_Req*, which are defined as *PTP event messages*, and *Follow\_Up* and *Delay\_Resp*, defined as *PTP general messages*. These messages are used in order to estimate the time and frequency differences between the master and slave nodes. For that, the master and slave devices generate timestamps on the departure and reception of PTP event messages. Then, general messages are used to carry additional information between the master and slave nodes. The description of the PTP exchange process is described next.

The exchange process begins with the master node transmitting a *Sync* message toward the slave. This message is timestamped on departure at instant  $t_1[n]$ . The timestamp itself can be placed in the *Sync* message on-the-fly as it is about to leave the device (in *one-step* mode [17, Section 9.5.9.4]), or the master can transmit the *Follow\_Up* message carrying the timestamp (in *two-step* mode [17, Section 9.5.9.5]) as shown in Fig. 2.4. At the slave node, the *Sync* message is timestamped on arrival at instant  $t_2[n]$ . Next, the slave node responds with a *Delay\_Req* message, which is timestamped on departure from the slave and arrival at the master node, at the instants  $t_3[n]$  and  $t_4[n]$ , respectively. Finally, the master node forwards a *Delay\_Resp* message with the  $t_4[n]$  timestamp. Ultimately, the slave node becomes aware of all the required timestamps  $t_1[n]$ ,  $t_2[n]$ ,  $t_3[n]$ , and  $t_4[n]$ .

After the PTP message exchange, the slave clock can use the timestamps to estimate the time and frequency difference (offsets) between its local clock and the master clock. The PTP protocol defines two ways to calculate the time offset. It can be either in *one-way* or *two-way* form. In the *one-way* approach, the time offset can be calculated by taking the difference between  $t_2[n]$  and  $t_1[n]$  on the slave as:

$$t_{21}[n] = t_2[n] - t_1[n] = x[n] + d_{ms}[n], \quad (2.13)$$

where  $d_{ms}[n]$  is the one-way delay experienced by the  $n$ -th *Sync* message from the master to slave (m-t-s) path. In order to calculate the true time offset, the true delay  $d_{ms}[n]$  is needed and, therefore, it must be estimated. However, this one-way delay is hard to be estimated under a high PDV. Thus, the calculation of the time offset using (2.13) will result in a noisy time offset  $\tilde{x}[n]$  due to the error between the estimated  $\hat{d}_{ms}[n]$  and the true delay  $d_{ms}[n]$ .

The second approach, called *two-way*, defines another method to calculate the time offset



using both exchanges present in Fig. 2.4 as follows:

$$\begin{cases} x[n] = t_2[n] - (t_1[n] + d_{ms}[n]) \\ x[n] = t_3[n] - (t_4[n] - d_{sm}[n]), \end{cases} \quad (2.14)$$

where  $d_{sm}[n]$  is the delay that the  $n$ -th DelayReq experiences in the slave to master (s-t-m) path. Note that, in both equations, the time offset is estimated as the difference between the slave timestamps ( $t_2[n]$  and  $t_3[n]$ ) and the master timestamps ( $t_1[n]$  and  $t_4[n]$ ) adjusted by the delays. However, the slave only knows the timestamps and does not have methods to estimate the individual delays. In particular, the slave only computes the following timestamps differences:

$$\begin{cases} t_{21}[n] = t_2[n] - t_1[n] = x[n] + d_{ms}[n] \\ t_{43}[n] = t_4[n] - t_3[n] = -x[n] + d_{sm}[n], \end{cases} \quad (2.15)$$

where both equations are calculated on the slave side and can be used to estimate  $x[n]$  as:

$$\tilde{x}[n] = \frac{t_{21}[n] - t_{43}[n]}{2}. \quad (2.16)$$

In practice, this computation produces biased estimates due to the delay asymmetry  $\gamma[n]$ :

$$\tilde{x}[n] = x[n] + \gamma[n] = x[n] + \frac{d_{ms}[n] - d_{sm}[n]}{2}. \quad (2.17)$$

The  $\gamma[n]$  can become one of the main sources of uncertainty as the difference between the delays becomes large. One should note that, even if the  $d_{ms}[n]$  and  $d_{sm}[n]$  were symmetric, there would still be other sources of uncertainty, such as clock stability and timestamping uncertainty.

The frequency offset can be measured based on the discrete-time difference of  $x[n]$  as:

$$\tilde{y}[n] = \frac{x[n] - x[n - N]}{t_1[n] - t_1[n - N]}, \quad (2.18)$$

where  $t_1[n] - t_1[n - N]$  is the time interval between the time offset samples. Moreover,  $N$  determines the size of the observation window. Note that the true instantaneous frequency is impossible to be measured since it will always involve some sample time, e.g., a window of time from which the RTC is observed.

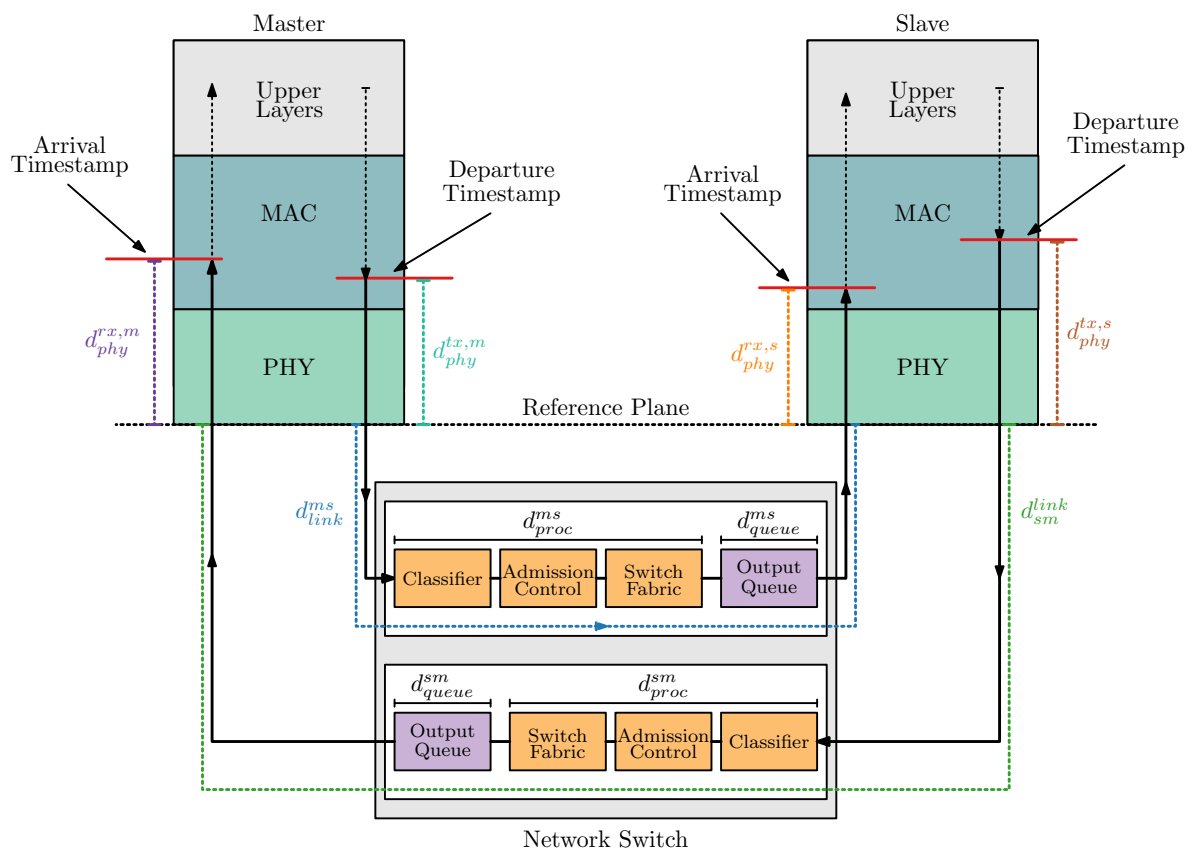
In addition to the time and frequency offset, an alternative linear combination of (2.15) can be used to calculate the so-called *two-way delay measurements*, such as:

$$\hat{d}[n] = \frac{t_{21}[n] + t_{43}[n]}{2} = \frac{d_{ms}[n] + d_{sm}[n]}{2}. \quad (2.19)$$

However, this estimate does not identify the individual delays  $d_{ms}[n]$  and  $d_{sm}[n]$ .

## 2.2.2 Sources of Delay Asymmetry

The PTP packets suffer from several sources of delays from the moment the departure timestamps are taken at the transmission node until the arrival timestamps are taken at the receiver node as illustrated in Fig. 2.5. The delays are usually composed of hardware physical layer (PHY) latency and network delays, i.e., propagation, processing, queuing and transmission delays. Consequently, the delay asymmetry affecting the PTP measurement, as shown in (2.17), will be originated from the difference between the delays in m-t-s and s-t-m directions.



**Figure 2.5:** Illustration of packet delays between the timestamp point at the master and the timestamp point at the slave node.

In a hardware timestamping implementation, the hardware PHY latency is denoted as the latency between the effective point where the timestamping is done and the reference plane, which is denoted as the boundary between the PHY and the network physical medium. Ideally, the timestamping mechanism would be placed at the reference plane, so that the departure and arrival time reflects the real moment in which the packet is transmitted or received. In practice, the timestamping is done at a higher layer, i.e., at the medium access control layer (MAC) or

PHY. Depending on the hardware implementation, the timestamping can also be corrected to reflect a time closer to the reference plane. For example, in Fig. 2.5, the timestamping is done at the MAC layer and if the latency between the timestamp point to the PHY layer, or even to the reference plane, is fixed and known, one could correct the timestamps so that they reflect a time closer to the reference plane. In Fig. 2.5, the latencies between timestamp point to the reference plane are denoted as  $d_{phy}^{tx,m}[n]$  and  $d_{phy}^{rx,m}[n]$  for egress and ingress, respectively, at the master node, and  $d_{phy}^{tx,s}[n]$  and  $d_{phy}^{rx,s}[n]$  for the egress and ingress at the slave node.

In addition to the delay latency introduced by the PHY, the PTP packet also suffers delays at the network medium, which is denoted as  $d_{link}^{ms}[n]$  and  $d_{link}^{sm}[n]$  in Fig. 2.5, for the m-t-s and s-t-m respectively. The link delay is defined as the time a packet takes to travel across the network from one endpoint to another. More specifically, the delay is measured from the reference plane from one node to the other node and it can be decomposed into propagation delay, transmission delay, processing delay and queuing delay.

The propagation delay is defined as the time required to propagate one bit along the link e.g., from the master to the slave node. It depends on two main factors, the distance between the nodes and the propagation speed associated with the particular physical medium. The propagation speed is usually defined in terms of the speed of light in the transmission medium.

The transmission delay is defined as the time required to transmit all the packet's bits in the physical medium. It is mathematically defined as:

$$d_{trans}[n] = \frac{N_{bits}}{R_{line}}, \quad (2.20)$$

where  $d_{trans}[n]$  is the transmission delay in seconds,  $N_{bits}$  is the number of bits and  $R_{line}$  is the Ethernet bit rate (in bits per second). For example, transmitting 80 bytes (640 bits) in a gigabit ethernet (GbE) interface ( $10^9$  bits/sec) would take  $0.64 \mu s$ .

In a packet-based network, the time required by each Ethernet switch to process a given packet is defined as processing delay. Fig. 2.5 shows three functional operations present in a typical switch that contributes to the processing delay which are classification, admission control and switching [30, Appendix I]. The classification step is required to identify the flow to which the frame belongs and determine the correct output port to which that packet should be forwarded. The admission control is responsible for flow management (e.g., policy, shaping). The switching step is responsible for forwarding the packet to the queue associated with a specific output port. Therefore, each of these steps takes time to be completed and may vary from switch to switch depending on how each step is implemented.

The queuing delay is defined as the time a packet stays on the switch output queuing. The queuing delay happens when a packet needs to wait in the output queue before it can be transmitted through the output port. The queuing delay depends on the network traffic load and queuing policies (e.g., packet priority) [31].

Therefore, the delays in the m-t-s and s-t-m directions can be written in terms of the sum of all the aforementioned delays components in the given direction, as:

$$\begin{cases} d_{ms}[n] = d_{phy}^{tx,m}[n] + d_{link}^{ms}[n] + d_{phy}^{rx,s}[n] \\ d_{sm}[n] = d_{phy}^{tx,s}[n] + d_{link}^{sm}[n] + d_{phy}^{rx,m}[n], \end{cases} \quad (2.21)$$

where PHY latencies due to the timestamp mechanism are denoted as  $d_{phy}^{tx,m}[n]$  and  $d_{phy}^{rx,m}[n]$  for the master node and  $d_{phy}^{tx,s}[n]$  and  $d_{phy}^{rx,s}[n]$  for the slave node. The  $d_{link}^{ms}[n]$  and  $d_{link}^{sm}[n]$  are the total link level delay for the m-t-s and s-t-m directions respectively.

As discussed, the total link level delay can be expressed as the sum of the propagation, processing, transmission and queuing delays:

$$d_{link}[n] = d_{prop}[n] + d_{proc}[n] + d_{trans}[n] + d_{queue}[n]. \quad (2.22)$$

Substituting (2.22) in (2.21) for each direction:

$$\begin{cases} d_{ms}[n] = d_{phy}^{tx,m}[n] + d_{proc}^{ms}[n] + d_{queue}^{ms}[n] + d_{trans}^{ms}[n] + d_{prop}^{ms}[n] + d_{phy}^{rx,s}[n] \\ d_{sm}[n] = d_{phy}^{tx,s}[n] + d_{proc}^{sm}[n] + d_{queue}^{sm}[n] + d_{trans}^{sm}[n] + d_{prop}^{sm}[n] + d_{phy}^{rx,m}[n]. \end{cases} \quad (2.23)$$

As defined in (2.17), the asymmetry that affects the PTP accuracy is defined as half of the difference between  $d_{ms}[n]$  and  $d_{sm}[n]$ . So, calculating the asymmetry based on (2.23), yields:

$$\begin{aligned} \gamma[n] &= \frac{d_{ms}[n] - d_{sm}[n]}{2} \\ &= \frac{d_{phy}^{tx,m}[n] - d_{phy}^{rx,m}[n]}{2} + \frac{d_{proc}^{ms}[n] - d_{proc}^{sm}[n]}{2} + \frac{d_{queue}^{ms}[n] - d_{queue}^{sm}[n]}{2} \\ &\quad + \frac{d_{trans}^{ms}[n] - d_{trans}^{sm}[n]}{2} + \frac{d_{prop}^{ms}[n] - d_{prop}^{sm}[n]}{2} - \frac{d_{phy}^{tx,s}[n] - d_{phy}^{rx,s}[n]}{2} \\ &= \gamma_{phy}^m[n] + \gamma_{proc}[n] + \gamma_{queue}[n] + \gamma_{trans}[n] + \gamma_{prop}[n] - \gamma_{phy}^s[n], \end{aligned} \quad (2.24)$$

where  $\gamma[n]$  can be viewed as the sum of different asymmetry components. The  $\gamma_{phy}^m[n]$  is the asymmetry component due to the difference between the PHY latency in the transmission and receive directions on the master node,  $\gamma_{proc}[n]$  is the asymmetry due to the different processing delays,  $\gamma_{queue}[n]$  is the asymmetry due to the queuing delay difference,  $\gamma_{trans}[n]$  is the

asymmetry caused by different transmission delays,  $\gamma_{prop}[n]$  is the asymmetry due to different propagation delay and  $\gamma_{phy}^s[n]$  is the asymmetry component due to the different latencies in the transmission and receive path on the slave node.

These delay components can be modeled with static and random nature, as summarized in Table 2.1. Consequently, the delay asymmetry that affects the PTP measurements derives from asymmetric static and random delay components as defined in (2.24).

**Table 2.1:** Main sources of delay.

Source	Nature
Queuing Delay	Random
Processing Delay	Random
Propagation Delay	Static
Transmission Delay	Static
Physical hardware	Static

Based on Table 2.1 and (2.24),  $\gamma[n]$  can be viewed as the sum of  $\gamma_s[n]$  and  $\gamma_r[n]$ , where  $\gamma_s[n]$  corresponds to the sum of the static component associated with the static delays, and  $\gamma_r[n]$  represents the random asymmetry components, associated with the random delay components. Hence, static asymmetry, generated by propagation<sup>1</sup>, transmission and physical latency, will be a static component that will contribute to the constant error present in the PTP measurements if not calibrated. On the other hand, the random components, based on queuing and processing delay, will be the ones contributing to the random error affecting the PTP measurements, which depend on the network traffic load and network topology. This random contribution can be a biased (non-zero mean) random variable dependent on the PDV.

In the scenarios discussed in this work, the asymmetry generated by the random delay components is the main source of error, as will be shown in Chapter 5. When dealing with PTP-unaware switches, the error associated with the processing and queuing components are not compensated as in PTP-aware networks. Therefore, mechanisms to correct or filter the delay asymmetry can have a big impact on the PTP accuracy in PTP-unaware networks.

<sup>1</sup>For simplicity, the propagation delay is assumed static in this work. However, in practice, the propagating delay is variable and depends on different factors such as environmental effects (e.g., temperature).

# Chapter 3

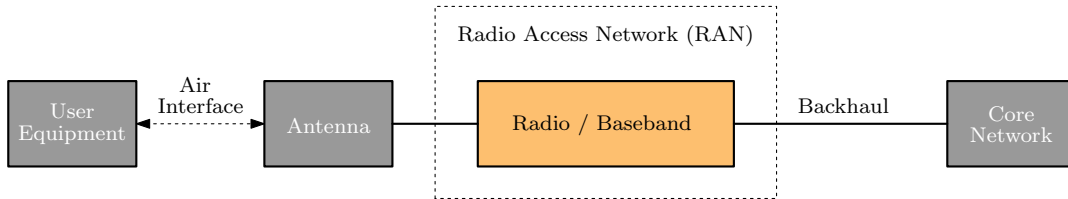
## Synchronization in TDD Fronthaul

This chapter describes how the information about the FH traffic pattern can be used to improve the synchronization performance from PTP in a PTS network. More specifically, how the information can be incorporated into the synchronization algorithms to help filter the PDV and correct the bias introduced by the delay asymmetry. First, Section 3.1 presents an overview of why the FH exhibits a periodic utilization pattern due to the use of TDD on the radio interface. Then, Section 3.2 discusses how the PTP packets can be classified based on the TDD FH traffic. Next, Section 3.3 discusses the packet filtering and KF algorithm, starting with a literature review followed by a discussion about how these algorithms are used in this work. Finally, Section 3.4 describes how the bias introduced by delay asymmetry due to queuing delay can be compensated when the FH presents a periodic utilization pattern.

### 3.1 Overview

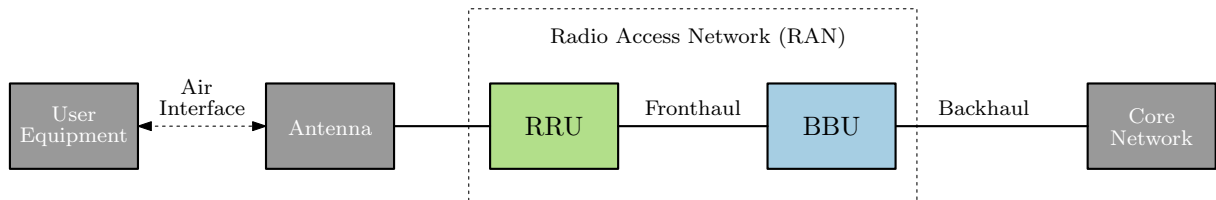
The mobile communication network is usually composed of a RAN that connects the user equipments (UEs) to the core network. The core network provides the bridge between the RAN, which is deployed in some geographical location, to the greater IP-based Internet. The RAN consists of a radio BS that provides radio coverage for a certain area, and it is connected to the core network via the so-called backhaul (BH) network, as illustrated in Fig. 3.1. In this architecture, the UEs are served by the BS and can transmit data over the wireless link to the BS in uplink and receive data from the BS in the downlink.

Several changes are being made to the RAN architecture with the evolution of mobile communication. For instance, the C-RAN, a well-investigated mobile architecture, is being



**Figure 3.1:** Traditional mobile architecture.

highly adopted in the newer mobile generations. In the C-RAN type of deployment [32], the original BS is divided into baseband unit (BBU), which contains all the baseband processing functions and remote radio unit (RRU) at the network edge, which contains radio functions. The BBU and RRU are connected by the so-called FH network, as illustrated in Fig. 3.2.



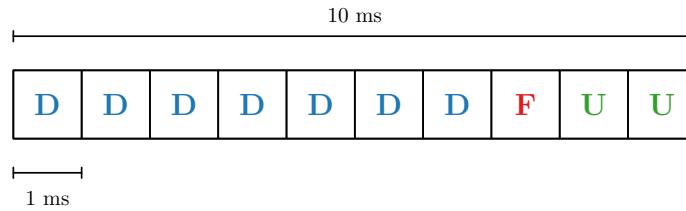
**Figure 3.2:** C-RAN architecture.

In the 4G and 5G mobile generation, motivated by the flexibility and cost-effectiveness of the Ethernet technology and to meet the demands of ever-growing bandwidth, the FH and BH are being moved to packet-based networks. For example, the FH will be composed of several intermediate switches and routers to transport the radio data.

Additionally, the newer mobile generations will be mostly based on the TDD scheme for increased flexibility as well as to make spectrum usage more efficient.

The TDD is a duplexing method used in telecommunication networks to coordinate the transmission and reception of data using the same frequency band. It allows for efficient use of resources by alternating the transmission and reception slots in a specific pattern. TDD systems have a configurable split, which means that some time slots are assigned to downlink (DL) and other time slots are assigned to uplink (UL). In 4G, the TDD split is semi-statically configured and essentially remains constant over time. In contrast, 5G can use a dynamic TDD, where the TDD slots are dynamically allocated as part of the scheduling decision [10].

There are many different configurable splits defined by the 3GPP for TDD systems. Fig. 3.3 illustrates a radio frame with the configuration split denoted as **DDDDDDDFUU**. A radio frame has a duration of 10 ms which consists of 10 subframes having 1 ms each. Each



**Figure 3.3:** Representation of TDD frame DDDDDDDFUU.

slot is used to transport data in some direction. For example, the slots with the symbol denoted as **D** are used for downlink, the slot with **F** is a flexible slot, and the slots denoted as **U** are used for the uplink. One of the reasons for having a flexible slot is to accommodate the guard period, which is necessary, for example, for the UEs switch from downlink reception to uplink transmission. More specifically, a UE needs additional time to receive and process data received on the downlink slot and switch from reception to transmission before the uplink slot starts.

The FH network exhibits a periodic utilization pattern when using a TDD system, similar to the air interface. In such a system, the FH traffic in the DL direction will have a high utilization before the start of the DL slot on the radio interface, followed by a period of silence. Similarly, the FH traffic on the UL direction will have a high utilization right after the UL slot finishes in the radio interface, followed by a period of silence. Fig. 3.4 illustrates the FH utilization based on the TDD pattern on the radio interface.

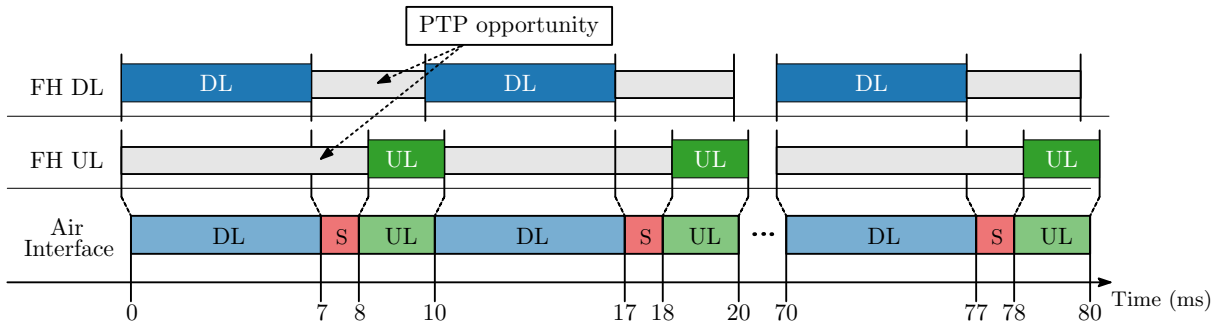
Fig. 3.4 covers the timing on the air interface as well as the usage pattern on the FH interface for both downlink and uplink direction. Note that before starting the downlink slot on the air interface (in this case from 0 to 7 ms), the radio data to be transmitted is first transported in the FH network in the downlink direction (from the BBU to the RRU). After the flexible slot, assumed here that it is fully used for the guard period, the uplink slot starts at 8 ms. After some time, the RRU starts to receive data and transmit it to the BBU in the uplink direction. Note that, because the TDD system on the air interface is half-duplex, the FH, which is full-duplex exhibits a high usage followed by a low utilization. In this work, these periods of low utilization in the DL and UL directions are called *opportunity windows*.

In this scenario, the authors of [2] proposed the idea of using the information about the periodic utilization pattern in the FH to improve the PTP performance. The authors proposed two ways for taking advantage of such systems. The first one consists of filtering the PTP messages based on the information about the TDD slot format. For example, only using the *Sync* messages that were transmitted in the moments of low activity in the FH, e.g., on the



periods of silence in the downlink direction. The second method consists of controlling the departure of PTP messages to coincide with the moments of low activity in the FH.

In this work, we focus on the scenario where the PTP transmissions are not controlled to use the TDD opportunities. Thus, the slave nodes in the network attempt to improve the overall synchronization performance by identifying whether the transmitted and received messages passed through the opportunity windows. By doing so, the slave can use those packets to estimate the time and frequency offset and correct the local clock. The rationale is that these packets will have small latency and PDV, so by using those packets, the PTP performance will be improved.

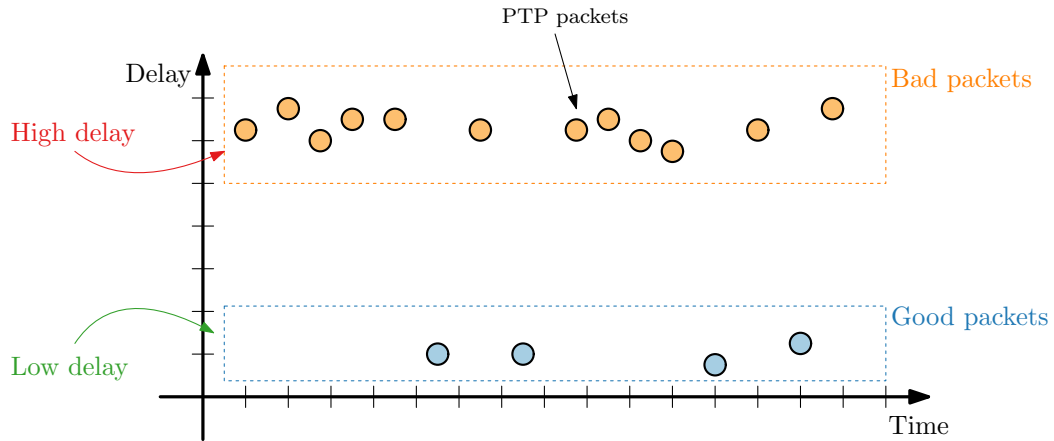


**Figure 3.4:** Timing diagram for PTP transmit opportunity in FH (based on [2]).

From now on, the term *TDD fronthaul* will be used to refer to a packet-based FH interface that presents periodic traffic, with high utilization followed by a period of silence, due to the periodic behavior in the radio interface when using a TDD system.

## 3.2 PTP Classification Based on TDD cycle

In a PTP-unaware network, the PTP performance will be affected by the traffic being transmitted together with PTP. In this scenario, the PTP messages can be classified as "good" or "bad" depending on the delay experienced when traversing the FH. For example, if a PTP packet is transmitted on the downlink in the so-called opportunity windows, then this packet has a high probability of passing through the FH without any queuing delay. Thus, considering the PTP event messages (*Sync* and *Delay-Req*) described in Section 2.2.1, if a *Sync* message is transmitted when the downlink is not being used by the TDD system, then it has a high probability of traversing the network with low latency and PDV. If that is the case, the *Sync* message is classified as "good". Otherwise, it would be classified as "bad". On the other hand,



**Figure 3.5:** Classification of PTP packets based on the experienced delays.

the delay experienced by the *Delay-Req* depends on the FH utilization in the uplink direction. Similarly, if the *Delay-Req* is transmitted when the uplink is not being used by the TDD system, it is classified as "good", or "bad" otherwise. Fig. 3.5 illustrates this classification.

Hence, one way to classify the PTP packets is based on the information about the TDD system. For instance, a PTP packet can be classified based on where it is located on the timing from the TDD system. The next section presents a discussion about using the modulo operator to classify the packets.

### 3.2.1 PTP Cycle Location

Modular arithmetic is used to get the remainder of a division of one number by another. For example, "4 mod 2", where "mod" is used to denote the modulo operator, is 0 because 2 divides 4 perfectly. However, if we calculate "5 mod 2" the result will be 1, because 1 is the remainder of 5 divided by 2. This type of operation is very helpful when dealing with periodic patterns as will be clarified in the sequel.

For example, a familiar use of modular arithmetic is in the 12-hour clock, in which the day is divided into 12-time slots (from 0 to 11). When counting on a 12-hour clock, you count up to 11 and then wrap back to 0, and the same pattern repeats. For instance, if you want to determine in which location 15 hours would be on the clock, you can simply calculate  $15 \bmod 12 = 3$ , which means that 15 is located at position 3. Similarly, if we add 24 to 15, which would be equivalent to two full cycles in the 12-hour clock, and calculate the modulo of 12, the result would still be 3.

Thus, we can define modular arithmetic as follows:

$$m(t, P) = t \bmod P, \quad (3.1)$$

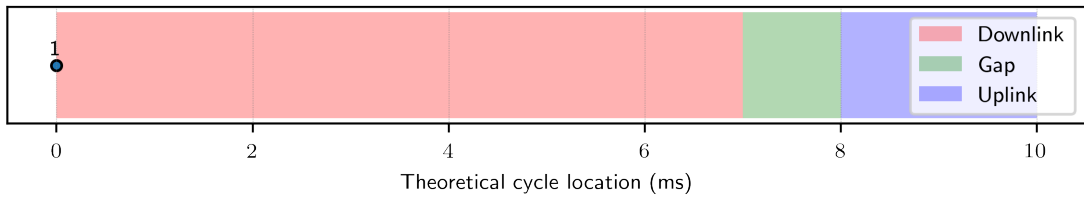
where "mod" denotes the modulo operator,  $t$  is the time,  $P$  is the period.

Now, let us consider the TDD cycle from 0 to 9 ms ( $P = 10$  ms) to be similar to a 10-hour clock, where the downlink, uplink and flexible slots are allocated on this period of 10 ms, and then repeat the same pattern over time. Based on that, we could use the modular arithmetic to find in which location on the TDD cycle a PTP message would be located if it were transmitted at time  $t$ . For example, if a PTP message is transmitted at time  $t = 125$  ms, calculating the modulo with (3.1) results in  $m(125, 10) = 5$ , which means that the message would be located at position 5 ms. If we consider the TDD configuration from Fig. 3.3, the position 5 ms is allocated for downlink operation. Therefore, if the PTP message is transmitted in downlink at  $t = 125$  ms, it would be located at the downlink slot from the TDD system. In this scenario, with high probability, this message will suffer a high queuing delay, because it will need to compete with the FH radio frames for the same outbound link in each intermediate network switch. On the other hand, if this PTP message were transmitted in the uplink, with high probability, this message would pass through the network without any queuing delay.

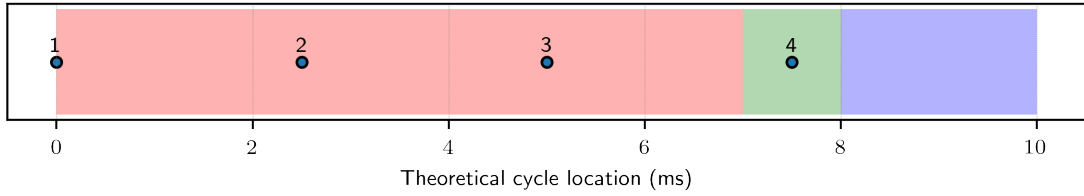
Using this approach, we can theoretically classify the PTP packets based on the position it would be located on the TDD cycle. One property from PTP systems that help with this analysis is that PTP is transmitted with a specific periodicity. Fig. 3.6 illustrates the position of PTP messages with different configurations on a TDD cycle of 10 ms considering that PTP starts to be transmitted as soon as the TDD cycle starts. The number above the samples denotes the order in which the messages were transmitted. For example, the first transmitted PTP packet is denoted as 1, the second as 2 and so on. Note that the message position depends on the periodicity that it is transmitted. For instance, the departure timestamp from PTP *Sync* message and *Delay-Req* can be used to calculate the PTP cycle location in the downlink and uplink, respectively, as follows:

$$\begin{cases} m(t_1, 10) = t_1 \bmod 10 \\ m(t_3, 10) = t_3 \bmod 10. \end{cases} \quad (3.2)$$

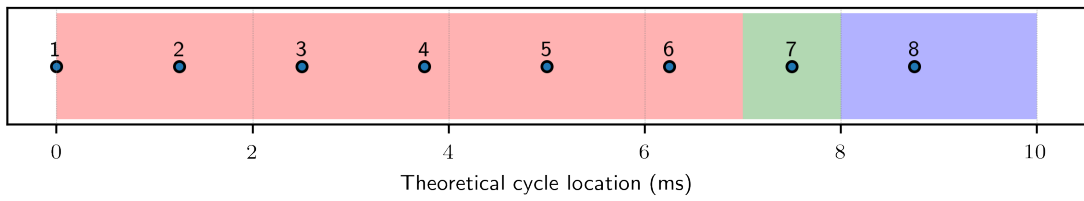
Fig. 3.6a considers that PTP messages are being transmitted with a periodicity of 4 exchanges per second ( $\Delta t = 250$  ms). If the first PTP message is transmitted at 0 ms, the second would be at 250 ms, the third at 500 ms and so on. In this case, if we calculate  $m(250, 10) = 0$ ,



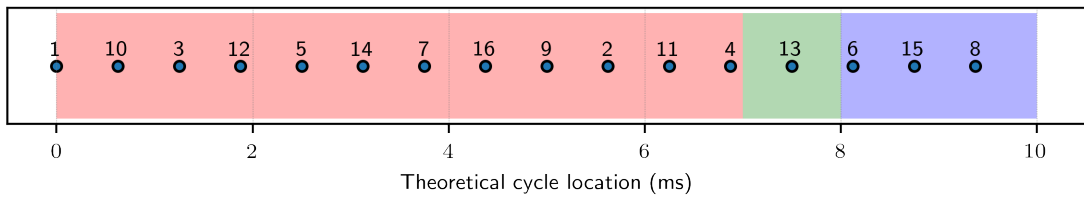
(a) With 4 exchanges per second.



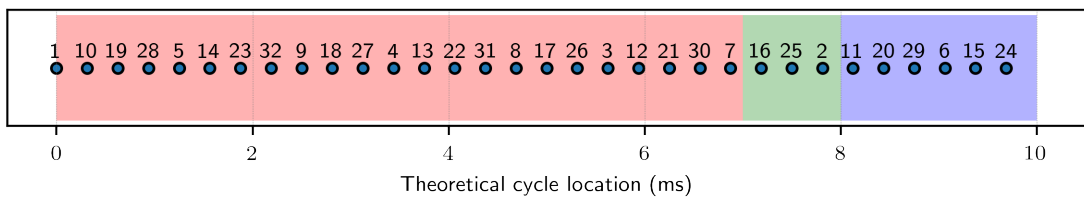
(b) With 16 exchanges per second.



(c) With 32 exchanges per second.



(d) With 64 exchanges per second



(e) With 128 exchanges per second

**Figure 3.6:** PTP cycle location with different exchange periods and TDD frame with 7 ms of downlink, 1 ms of gap and 2 ms of uplink.

we can see that the message transmitted at 250 ms would always be located at 0 ms. Actually, transmitting with a period of 250 ms is equivalent to always transmitting a message after 25 TDD full cycles. Therefore, when transmitting 4 PTP messages per second, the message will be always located at 0 ms.

Fig. 3.6b considers 16 exchanges per second, which is equivalent to transmitting with period  $\Delta t = 62.5$  ms. In this case, the theoretical transmission instants for the five first messages are 0 ms, 62.5 ms, 125 ms, 187.5 ms and 250 ms. Calculating the modulo considering a period of 10 ms for the TDD cycle, results in the positions 0 ms, 2.5 ms, 5 ms, 7.5 ms and 0 ms. Because of the PTP periodicity, the messages will be located in the same positions.

Fig. 3.6c, Fig. 3.6d and Fig. 3.6e present the PTP positions when using 32, 64 and 128 exchanges per second, respectively. Note that, as the PTP period decreases, more samples are transmitted at the gap and uplink time slots.

In addition to the classification, this type of analysis is useful for asserting the percentage of PTP packages on the good, bad and gap slots, regardless of the alignment between the PTP packets and FH data streams.

### 3.2.2 K-means based Classifier

An alternative way to classify the PTP packets can be based on the estimated delay experienced by the PTP packets. The expectation is that the good packets have a lower delay when compared with the bad packets. Thus, using clustering methods we could learn the delay levels from the good and bad packets and classify them accordingly. More specifically, we propose using a modified version of the K-means clustering algorithm.

The K-means is a well-known clustering algorithm used for data classification. Given a set of observations, K-means aims to partition the  $n$  observations into  $k$  ( $k \leq n$ ) clusters to minimize the sum of the squared distance between each data point and its closest cluster center (or centroid). The algorithm usually starts by selecting randomly  $k$  clustering centers from the data points. Then, the next steps are divided into two phases. In the first phase, the distance between the observation and all cluster centers is calculated, and the observation is assigned to the cluster associated with the minimum distance. This distance measure is usually assumed to be the squared Euclidean distance. Then, in the second phase, the centers of the clusters are recalculated based on the all data assigned to them. After that, these two phases repeat until the process stabilizes.

In this work, we propose to use K-means to organize the data into two clusters ( $k = 2$ ), and then define a binary classifier based on thresholds established with the boundary values of the clusters. The objective is to divide the data into a cluster of good packets and another cluster of bad packets. Thus, an individual classification task is performed for the packets in the m-t-s and s-t-m direction. Based on the estimated delays, the algorithm classifies the packets into specific clusters.

The initialization process starts by accumulating 1-minute of PTP exchanges and selecting the higher and lower experienced delay as the initial levels for the two clusters. The expectation is that, by initializing the algorithm like so, the samples with lower delay will be assigned to the cluster initialized with the lower level, denoted as  $C_{min}[0]$ . Similarly, the packets with a high delay will be assigned to the cluster initialized with the higher delay level, denoted as  $C_{max}[0]$ . Fig. 3.7 illustrates the process. Next, the algorithm can classify the new samples and update the levels from the two clusters by assigning the new observation to the nearest cluster and updating the cluster level based on the mean delay experienced by all data assigned to them.

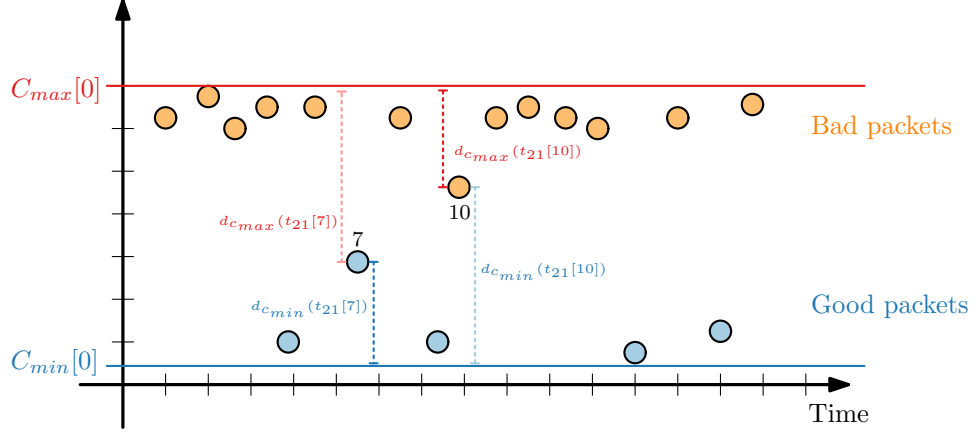
The one-way delay is estimated based on the timestamp differences calculated with (2.15). However, note that the delay estimate using (2.15) is corrupted by the time offset term  $x[n]$ . Thus, we use an adjusted version of the timestamp difference defined as:

$$\begin{cases} t'_{21}[n] = t_{21}[n] - \sum_{j=0}^n \hat{\Delta}_x[j] \\ t'_{43}[n] = t_{43}[n] + \sum_{j=0}^n \hat{\Delta}_x[j], \end{cases} \quad (3.3)$$

where  $\hat{\Delta}_x[n]$  is the estimate of the time offset drift, defined as  $\Delta_x[n] = x[n] - x[n-1]$ . The discussion about how the time offset drift is calculated is postponed to Section 3.3.4. By substituting (2.15) in (3.3), note that the compensated timestamps differences are approximately:

$$\begin{cases} t'_{21}[n] \approx d_{ms}[n] + x_0 \\ t'_{43}[n] \approx d_{sm}[n] - x_0, \end{cases} \quad (3.4)$$

where  $x_0$  is the initial time offset. The accumulative sum from the time offset drift in (3.3) is biased by the initial time offset  $x_0$ . It is important to note that equation (3.4) can not be used for an accurate estimation of individual one-way delays. However, the idea of using the compensated timestamp differences is that we can effectively remove the effect of the time offset component over time and bring the timestamps to a more constant level. The main objective here is to classify the packets into "good" and "bad", and for this purpose, a precise estimation of the one-way delays is not necessary.



**Figure 3.7:** Illustration of K-means classification process.

The assigned process is defined as follows. Based on the one-way delay estimates  $t'_x[n]$  using (3.3), the distance between each sample and the two cluster levels are calculated as:

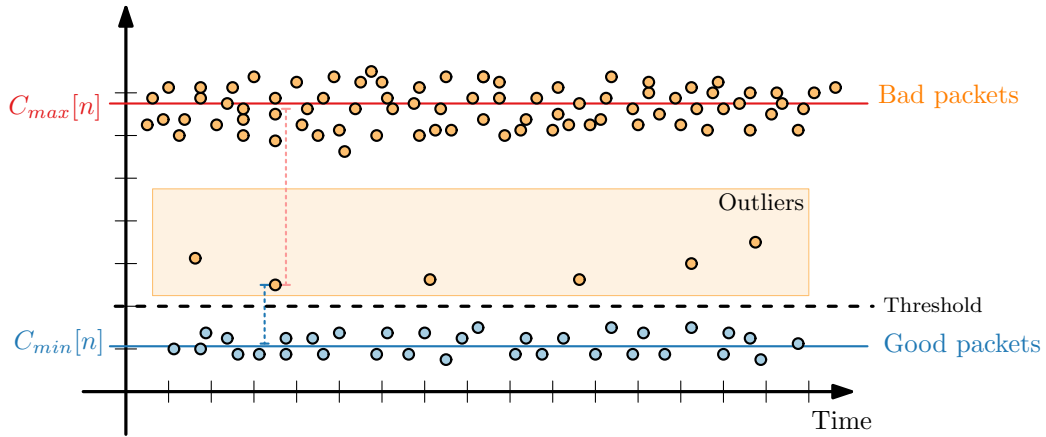
$$\begin{cases} d_{c_{max}}(t'_x[n]) = |C_{max}[n] - t'_x[n]| \\ d_{c_{min}}(t'_x[n]) = |C_{min}[n] - t'_x[n]|, \end{cases} \quad (3.5)$$

where  $d_{c_{max}}(\cdot)$  defines the absolute distance between the timestamp difference sample and the bad packet cluster, and  $d_{c_{min}}(\cdot)$  defines the absolute distance from the good packet cluster. Thus, the PTP packet is assigned to the bad packet cluster if  $d_{c_{max}}[n] \leq d_{c_{min}}[n]$ , or to the good packet cluster otherwise.

However, as shown in Fig. 3.7, if a packet experienced transitory delays such as queueing delay, it would be assigned to the closest cluster, which could be the minimum delay cluster. Thus, to prevent this case, an additional step is used to filter the outliers from the "good" packets cluster. The filtering process is based on the standard deviation. Assuming that the distribution of "good" PTP packets have a Gaussian shape because they are not affected by queueing delay, we can filter out the samples that are closer to the cluster with minimum delays but have a delay greater than 3.5 standard deviations from the cluster level. This way, if a packet experienced a transitory delay level, even if it is closer to the cluster of good packets, it is classified as bad if it has a delay greater than the defined outlier threshold. The threshold is calculated based on the estimated variance, which is recursively calculated as:

$$v[n] = \frac{1}{N} \sum_{k=0}^{N-1} (d_{c_{min}}(t'_x[k]))^2 - \left[ \frac{1}{N} \sum_{k=0}^{N-1} d_{c_{min}}(t'_x[k]) \right]^2, \quad (3.6)$$

where the vector  $t'_x[k]$ ,  $0 \leq k \leq N - 1$ , represents the adjusted timestamp differences  $t'_{21}[n]$  for



**Figure 3.8:** Illustration of the K-means outlier threshold.

m-t-s direction and  $t'_{43}[n]$  for s-t-m, and  $N$  is the number of packets. Both values are based on the PTP packets classified as good. Hence, the outlier threshold is calculated as:

$$\tau_{th}[n] = C_{min}[n] + 3.5\sqrt{v[n]} \quad (3.7)$$

In the end, we expect to have a classification as illustrated in Fig. 3.8, where the  $n$ -th PTP packet is only classified as "good" if it is closer to  $C_{min}[n]$  and it has a delay less than  $\tau_{th}[n]$ .

### 3.3 Time and Frequency Offset Estimators

In the context of IEEE 1588 networks, we are interested in estimating the true time and frequency offsets by using the measurements provided by the PTP protocol to correct the RTC from the slave clock. However, as discussed in Section 2.2.1, the measurements using (2.17) and (2.18) can be very noisy depending on network conditions and other sources of uncertainty. Hence, in this scenario, estimation algorithms are heavily adopted to alleviate the noise in the measurements and consequently improve the time accuracy. Section 3.3.1 presents a literature review and Section 3.3.2 and Section 3.3.3 discusses how the algorithms are modeled.

#### 3.3.1 Related Work

The algorithms presented in the literature to estimate the time and frequency offset are usually classified into two main groups: *window-based* and *model-based*. The former group is mainly composed of the so-called *packet-selection* algorithms, which consist of partitioning the time offset measurements in windows and, for each window, selecting the measurement



based on some statistical metric, e.g., minimum, maximum, mean, mode or median [31, 33–36]. In contrast, the latter group aims to improve the estimation accuracy by incorporating the knowledge of how the time and frequency offsets evolve over time.

In [31], the authors provided a formal statistical analysis of the commonly used packet selection process. More specifically, the paper analyzes the performance from the sample-minimum algorithm, where the time is partitioned into non-overlapping windows and, for each window, the method selects only the one that had the smallest packet delay. The authors modeled the network delay as the sum of propagation, processing, transmission and queuing delay. The transmission and propagation were modeled as constant delays so that the PDV was mainly composed of the processing and queuing delays, which were modeled as two biased random variables. More specifically, the authors modeled the queuing delay at each hop with an exponential distribution, so the sum of these independent's exponentially distributed random variables has an Erlang distribution. For simplicity, the processing delay was omitted from the statistical formulation. The results show that the effectiveness of the packet-selection algorithms depends on the shape of the packet delay distribution, which changes depending on the network loads. For instance, the sample-minimum presented a better performance for light load networks, where the delay distribution presented an exponential-like shape. For heavy-load networks, the sample-mean had better performance because the higher loads produced a delay distributing similar to higher-order erlang's densities, resembling Gaussian shape. Finally, the sample-maximum had a better performance with in-line traffic.

Based on the analysis presented in [31], the work in [33] proposed a dynamic mechanism to select between the sample-minimum, sample-mean and sample-maximum based on the delay distribution. The objective is to select the algorithm that yields better performance in real-time depending on the current delay distribution. For that, the algorithm that outputs the minimum variance for the given window of delay measurements is selected.

In the same way, the works in [34, 35] propose the use of the sample-mode as a filtering technique. The idea is to use the statistical mode to select the measurements with the delay that most appears in a window of observations. For example, in a higher load network, the delay distribution would be more concentrated on the maximum. Thus, using the sample mode bin, we would also select the measurements close to the maximum from the distribution. Similarly, for a light load network where the delay distribution would be more concentrated on the minimum delays, the mode of the operator would also select measurements close to the minimum. Using

simulation, the authors show that sample-mode could efficiently adapt to the delay distribution. However, the authors do not present a good description of the implementation details. For instance, how to select the bin width of each histogram bin. Additionally, the authors do not describe the weaknesses of the technique. For example, the performance heavily depends on how narrow the distribution is around the theoretical mode and the performance will be affected if there is more than one mode, i.e., in a dual-mode distribution.

None of the works in [31, 33–35] took into consideration the bias associated with the packet-selecting algorithms, which differ significantly depending on the packets that are selected. As discussed, the sample-minimum could be used to select the packets with the smallest delays in the m-t-s and s-t-m direction. Intuitively, we could think this is the best approach in a light network load. However, if the minimum delays in the m-t-s have a big difference when compared with the minimum delays in the s-t-m direction, when calculating (2.17), the bias introduced by this asymmetry becomes problematic for the PTP performance. Therefore, the estimation of the delay asymmetry also has a very important role in the estimation techniques for the IEEE 1588 networks and will be discussed later in this chapter.

In addition to the packet selection algorithms, some algorithms incorporate the knowledge about how the time and frequency offset evolves over time, which are the so-called model-based algorithms. One example is the well-known KF that can be used to estimate both time and frequency based on a set of noisy measurements. Many works analyzed the use of KF for the specific problem of clock synchronization [19, 37–41].

In [37], the author presented a performance comparison between KF, linear programming (LP), and the moving average algorithm for the problem of clock synchronization. The work is based on numerical simulation and uses the network time protocol (NTP), but it can also be interpreted for PTP without problems. The KF was modeled using a scalar-state vector-measurement model, where the state was designed to be the inter-arrival interval from the NTP messages between the master and the slave clock. Subsequently, the time and frequency offsets can be calculated by comparing this value with the nominal inter-departure interval. It was shown that KF outperforms both least squares (LS) and moving average algorithms in a scenario where the delays experienced by the NTP messages were modeled as white-gaussian random variables. In contrast, LS was superior in a scenario with bursty traffic. In both scenarios, the moving average was less accurate than the other two approaches.

In [38], KF was modeled as a vector-state vector-measurement, where the system state is

composed of both time and frequency offsets, and both states are also observed in the measurements. The authors analyze the influence of oscillator noises and timestamping uncertainties in the synchronization accuracy, although only standard deviations of the time and frequency offset were given. Also, this work does not consider packet delay uncertainties, which in PTP-unaware networks would be the main source of the noise. In these conditions, it was shown that KF only helps under high timestamping uncertainty, e.g., software-based, whereas the KF performance approaches the raw PTP measurements under a low timestamping uncertainty, e.g., hardware-based.

The authors of [38], under the same assumptions and simulation environment, extended the analysis of KF in [39]. They emphasize that the accuracy level depends on the timestamping uncertainty. Thus, in [39] they proposed a combined algorithm consisting of two Kalman filters with the objective to detect outliers and improve the KF robustness. The idea is that the update stage should only be calculated if the time and frequency offsets measurement are reliable. The method is based on the innovation value, which means that the update state is only used if the new measurement is not much discrepant from the estimated one. The second KF runs in parallel and serves as a fallback, for example, if the reference time source fails. The approach is shown to improve robustness and achieve good accuracy.

A more practical analysis of KF was presented in [41]. The authors proposed a more energy-efficient and less computationally complex version from [38] with a vector-state scalar-measurements model. The proposed solution only runs the update stage when requested. This work also discusses that the KF parameters need to be carefully initialized, especially the state noise covariance matrix, to maintain a good synchronization accuracy.

### 3.3.2 Kalman Filtering

The KF is a widely used estimation algorithm for linear filtering and prediction in different applications. It was first introduced by Rudolf E. Kálmán in [42]. The filter provides an efficient recursive way to estimate the state of the linear dynamic system by combining the mathematical system model with a series of noisy measurements observed over time. The idea is that, by combining these two pieces of information, we can estimate the system state better than the estimate produced by either information alone.

The KF overall goal is to find the best estimate of the system state at each time step using the value predicted by the system model and using the information provided by the noisy

measurements. For that, it starts by predicting the system's behavior at the next time step (a priori estimate). Then, it adjusts the value predicted using the available measurements (a posteriori estimate). Thus, the KF process can be described in terms of two stages, predict and update, as shown in Fig. 3.9. Appendix A presents the basic theory regarding KF.

We start the KF design by defining the state variables of the system. The state vector  $\mathbf{s}$ , corresponds to the vector of states that can be used to describe the future states for a given dynamic system. In the context of clock synchronization, a two-state clock model can be adopted, as shown in [38, 41], where the state is composed of the time and frequency offset, as follows:

$$\mathbf{s}[n] = [x[n], y[n]]^T. \quad (3.8)$$

According to ITU-T G.810 [21], the time offset can be modeled as a piecewise linear function of time:

$$x[n] = x[n-1] + y[n-1]\Delta t + \psi[n], \quad (3.9)$$

where  $y[n]$  is the frequency offset,  $\Delta t$  is the PTP exchange period, and  $\psi[n]$  is a random variable that models all the deviations that affect the time offset and frequency offsets.

The random noise  $\psi[n]$  comes primarily from the random phase deviations, as discussed in Section 2.1.2. As shown in [41], a simple model can be used considering only WFM, which causes random-walk in time, and RWFM. In this case,  $\psi[n]$  can be denoted as a random process with increments represented by stationary zero-mean random variables with finite variance. Hence, the model becomes:

$$\begin{cases} x[n] = x[n-1] + y[n-1]\Delta t + w_x[n-1] \\ y[n] = y[n-1] + w_y[n-1], \end{cases} \quad (3.10)$$

where  $w_x[n]$  and  $w_y[n]$  are the random walk noise terms that affect the time and frequency offset, respectively.

The state process  $\mathbf{s}$  can be modeled using (3.10) and (3.8) as:

$$\mathbf{s}_k = \mathbf{A}\mathbf{s}_{k-1} + \mathbf{w}_{k-1}, \quad (3.11)$$

where  $\mathbf{A}$  is the *state transition matrix* and can be defined as:

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \quad (3.12)$$

and  $\mathbf{w}[n] = [w_x[n-1], w_y[n-1]]^T$  is the *state noise vector*.

In this formulation, the state noise vector  $\mathbf{w}[n]$  describes the random-walk process in time and frequency. This process is modeled as a zero-mean Gaussian vector with covariance  $\mathbf{Q} = E[\mathbf{w}[n]\mathbf{w}^T[m]]$ . In other words, the  $\mathbf{Q}$  matrix describes the intrinsic stochastic uncertainty present in the system state model. Furthermore, the formulation also assumes that  $\mathbf{Q} = 0$  for  $n \neq m$ , and that  $w_x[n]$  and  $w_y[n]$  are uncorrelated so that  $\mathbf{Q}$  is a diagonal matrix [38, 41].

In the context of clock synchronization, clock noises are usually estimated in terms of Allan variances [22]. Many works discuss ways to find a relationship between the Allan variances and the clock noise parameters, as shown in [43]. However, these methods require individual analysis for each clock device. Differently, in this work, we tried to automate this process by testing a range of values in the  $\mathbf{Q}$  matrix and choosing the one that minimizes the overall  $\max|\text{TE}|$  of the system.

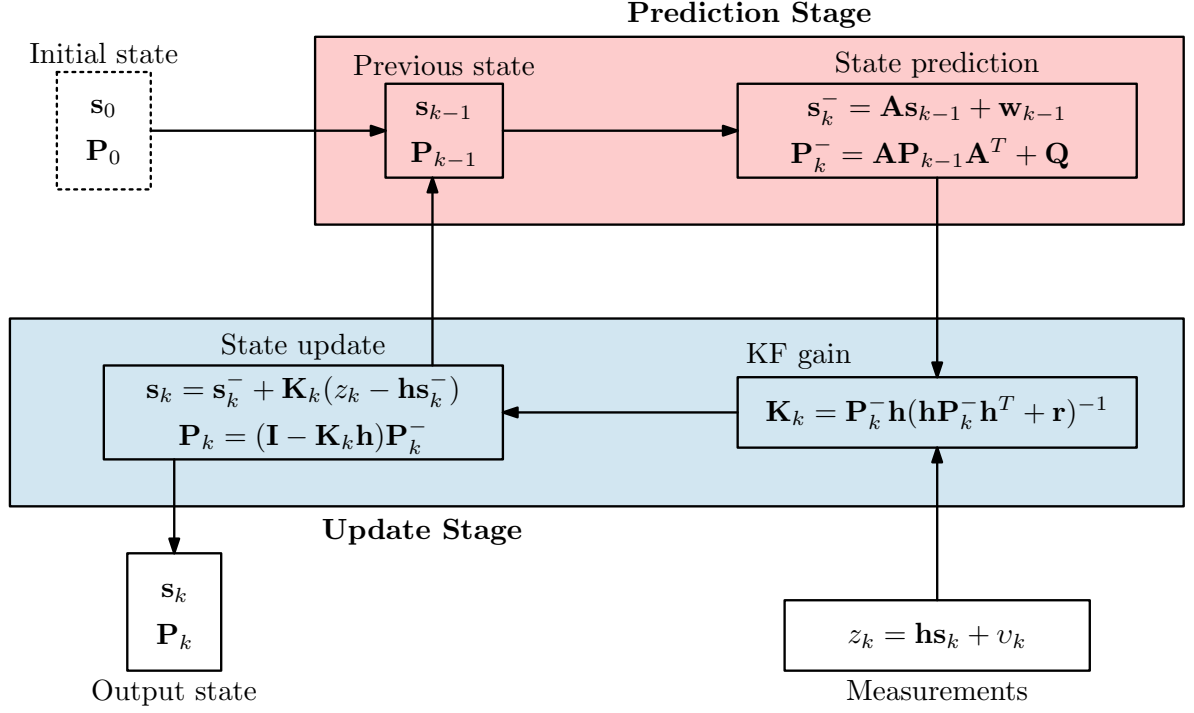
In addition to the state model, the KF also uses the measurement model. In [41], the adopted strategy only relies on the time offset. Differently, [38] adopted the measurements composed of both time and frequency offsets. One should note that depending on the adopted measurement model, the resulting state variables can be directly measured or not. Sometimes, the parameters we choose to represent a system are intrinsic to the problem but impossible to measure in practice. In these scenarios, as the KF is a state observer, the internal states of the system can be inferred using the information provided by the others states if the system is classified as observable.

Here, we adopt the measurement model similar to [41]. The measurement is only composed of the time offset. This way, given that our system state is composed of both the time and frequency offsets, as defined in (3.10), the frequency offset is estimated internally by the filter. Thus, the measurement can be modeled as:

$$z[n] = \mathbf{h}\mathbf{s}[n] + v[n], \quad (3.13)$$

where  $v[n]$  represents all the noises present in the PTP measurements and is defined as the *measurement noise*, modeled as a zero-mean Gaussian random variable with variance  $\mathbf{r}$ , and  $\mathbf{h} = [1, 0]$  corresponds to the *transition vector*. Here, as the measurements are only composed of time offset measurements, the transition is a  $(2 \times 1)$  vector. The variance from the measurement noise is defined as:

$$\mathbf{r} = \sigma_v^2 = [\text{Var}\{v[n]\}]. \quad (3.14)$$



**Figure 3.9:** Kalman filter loop.

We assume that the noise present in the PTP measurements is mainly composed of PDV, unlike [38], which models the measurement noise mainly composed of timestamp uncertainties and propagation delays. Thus, based on (2.17) and (2.19), and assuming that the master-to-slave and slave-to-master delays are independent, wide-sense stationary (WSS) and white discrete-time random process, the measurements noise variance  $\sigma_v^2$  can be defined as:

$$\sigma_v^2 = \frac{\text{Var}\{d_{ms}[n]\} + \text{Var}\{d_{sm}[n]\}}{4} = \text{Var}\{\hat{d}[n]\}. \quad (3.15)$$

A big advantage of this measurement model is that the variance from (3.15) coincides with the variance of (2.19), which the slave can compute in practice. Also, the KF formulation assumes that the noise that affects the measurements is composed of a zero-mean Gaussian random variable. Thus, we keep these assumptions to make the model more tractable.

In contrast, when assuming the vector-measurement model, where the observation vector consists of both time and frequency offsets, as shown in [44], the measurement noise covariance requires the knowledge of the individual delay variances, master-to-slave and slave-to-master, which the slave does not have in practice.

For initializing the filter, KF assumes the knowledge of the *initial state estimation error covariance*, denoted as  $\mathbf{P}_0$ , and the initial state  $\mathbf{s}_0$ . To initialize  $\mathbf{s}_0$  we opt to use the first two PTP exchanges. More specifically, the time offset is initialized with the estimate from (2.16),

and the frequency offset is estimated using (2.18). Note that the filter must wait for the first two exchanges before starting. Most of the time, it is preferable to use this approach than to initialize with a random state, as the filter tends to start closer to the real initial state. In addition, the  $\mathbf{P}_0$  is initialized with arbitrarily large values due to the lack of knowledge of the first state's variance. The idea is that by initializing this way, the filter will start with a high KF gain. In other words, it will start with low confidence in the state, favoring the observed measurements. Finally, Fig. 3.9 summarizes the equations and computational sequences of KF.

In terms of using the knowledge of the TDD system, one way is to perform the update stage only if the exchange is considered good. In this case, the KF model described above remains the same, and KF continues to estimate the system state in the prediction stage on every PTP exchange. The only change is that the state is predicted only using the system model until there is a good PTP exchange to adjust the predicted value using the good PTP measurement.

### 3.3.3 Packet Filtering

This section discusses the strategies such as sample average, sample minimum and sample maximum. As discussed in Section 3.3.1, these methods are helpful for filtering the PDV. These algorithms work by filtering the timestamp differences, from (2.15) in the m-t-s and s-t-m direction, to estimate the time offset. The observed timestamp differences are usually partitioned into windows, where the samples within the window are used to estimate the time offset. In this work, we use an overlapping sliding window where only one sample differs from the current window to the next window.

As defined in [36], the packet selection algorithms estimate the time offset for the  $k$ -th window as follows:

$$\hat{x}[n] = \frac{\xi\{\mathbf{t}_{21}[k]\} - \xi\{\mathbf{t}_{43}[k]\}}{2} \quad (3.16)$$

where  $\xi\{\mathbf{t}\}$  denotes either the minimum, maximum or average of the elements of the vector  $\mathbf{t}$ .

Note that, under the assumption that the time offset is constant within the observation window  $k$ , the estimation from (3.16) can be seen as:

$$\hat{x}[n] \approx x[kN] + \frac{\xi\{\mathbf{d}_{ms}[k]\} - \xi\{\mathbf{d}_{sm}[k]\}}{2} \quad (3.17)$$

where  $N$  denotes the size of the observation window,  $x[kN]$  is the constant time offset from the  $k$ -th window, and  $\mathbf{d}_{ms}[k]$  and  $\mathbf{d}_{sm}[k]$  are the vectors with the observed delays within the  $k$ -th window in the m-t-s and s-t-m direction, respectively.

From (3.17), it is easy to understand the rationale from the packet selection algorithm. The objective is to select the PTP packets that have symmetric delays in the m-t-s and s-t-m so that the delay terms cancel each other. Thus, if the delays are symmetric, the final estimate is closer to the true time offset, consequently, filtering out the PDV.

For increasing the chances of finding symmetric delays in both directions, we could use a very large size  $N$  for the observation windows. By doing so, the assumption of a constant time offset within the observation window no longer holds and the performance would be affected. We can conclude that these algorithms are sensitive to the size of the observation window.

In this work, we use the method proposed in [36] that consists of adding a preprocessing step to adjust the values of the timestamp differences. Thus, instead of (2.15), the compensated version from (3.3) is used instead.

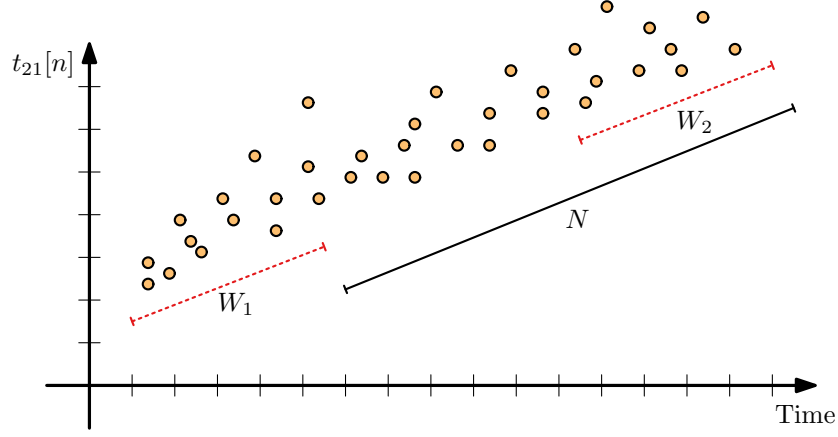
In this case, equation (3.16) needs to be adjusted to use the drift-compensated  $t'_{21}[n]$  and  $t'_{43}[n]$  as:

$$\hat{x}[n] = \frac{\xi\{t'_{21}[k]\} - \xi\{t'_{43}[k]\}}{2} + \sum_{j=0}^{N-1} \hat{\Delta}_x[j] \quad (3.18)$$

where  $t'_{21}[k]$  and  $t'_{43}[k]$  are vectors with the drift-compensated samples within the  $k$ -th window and the summation term is used to reintroduce the estimated time offset drift so that the final result produces a time offset estimation. The objective of using the drift-compensated version from the timestamp difference is to compensate for the time offset drift so that the time offset remains constant for a bigger period of time. As a result, this enables the observation window to be bigger and consequently improves the time offset estimation.

In terms of using the knowledge from the TDD system, the packet filtering as described above can be performed only in the PTP packets that are considered to be good. For instance, the sample average can be modified to only process the packets that are classified as good. The same rationale can be extended to sample minimum and sample maximum. However, the expectation is that the sample minimum will naturally select the samples that traverse the network with the minimum delay, i.e., the "good" packets. On the other hand, the sample maximum will naturally select the samples that traverse the network with the maximum delays which we define as the bad packets. Thus, only the sample average is modified to use the TDD information in this work.





**Figure 3.10:** Window-based filtering for time offset drift estimate.

### 3.3.4 Time Offset Drift Estimation

The time offset drift estimation used in Sections 3.2.2 and 3.3.3 is defined as follows:

$$\Delta_x[n] = \hat{y}[n](t_1[n] - t_1[n - 1]), \quad (3.19)$$

where  $\hat{y}[n]$  is the frequency offset estimate. The frequency offset, in turn, is estimated with (2.18), where the time offset  $\hat{x}[n]$  term is estimated based on the one-way formulation using the timestamp difference  $t_{21}[n]$ . Thus, the frequency offset is estimated as:

$$\hat{y}[n] = \frac{t_{21}[n] - t_{21}[n - N]}{t_1[n] - t_1[n - N]} \quad (3.20)$$

To help alleviate the PDV noise on the  $\hat{y}[n]$  estimation, the minimum values of  $t_{21}[n]$  are filtered using a sliding window as illustrated in Fig. 3.10. Differently from Section 3.3.3, the process relies on two windows of  $W$  samples, spaced by  $N$  samples. In the end, the time offset drift is estimated as:

$$\hat{y}[n] = \frac{\min_{0 \leq i \leq W_2} \{t_{21}[n - i]\} - \min_{0 \leq j \leq W_1} \{t_{21}[n - N - j]\}}{t_1[n] - t_1[n - N]}. \quad (3.21)$$

## 3.4 Delay Asymmetry Estimation

The PTP formulation for calculating the time offset, assumes that the delay in the m-t-s and s-t-m are symmetric, as discussed in Section 2.2.1. So, the PTP measurement from (2.17) produces biased estimates due to the delays difference in the forward and backward paths. As discussed in Section 2.2.2, and demonstrated in (2.24), there are many sources of delay

asymmetry, such as PHY latencies and network delays, i.e., propagation, processing, queuing and transmission.

Many works in the literature proposed methods to estimate the delay asymmetry that affects the PTP measurements that could be used in conjunction with the algorithm used to estimate the time and frequency to enhance the overall performance, some of these methods are discussed in Section 3.4.1.

### 3.4.1 Related work

The authors of [45] assumed that the random delay experienced by the PTP packets conformed to Gamma distribution, based on the results from [31,33]. Based on this key assumption, the authors derived a recursive method to find the distribution parameters. However, in practice, the delay distribution depends on many factors, like the network topology and network load. Thus, it is difficult to fit the delay with a single distribution as proposed, which can lead to estimation error.

In [46], the proposed algorithm estimates the delay asymmetry based on the difference between the queueing delay of each packet and the minimum observed queueing in a window. However, this approach is sensitive to the minimum observed queueing delay because it requires a symmetric value in both m-t-s and s-t-m directions. Furthermore, it requires that the windows are large enough to contain minimum delayed packets.

Based on [46], the authors of [47] proposed a method to estimate the path asymmetry on PTP-unaware packet-switched networks using convolutional denoising autoencoders (CDAEs). The CDAEs are used to filter the noisy PTP information at the slave side to reconstruct the path asymmetry information. However, the downside of this approach is that the CDAE needs to be trained to denoising known asymmetry patterns, which change depending on the network, which might not be practical.

### 3.4.2 Queuing-Induced PTP Delay Asymmetry

The two major challenges in PTS networks are the PDV and delay asymmetry. The algorithms presented in Section 3.3 are usually used to filter the PDV and other methods need to be used to estimate the bias introduced by the delay asymmetry.

In the case of TDD, the FH traffic will be most of the time asymmetric in the m-t-s and s-t-m directions. However, different from the other scenarios, in a TDD FH the packets will

experience two main levels of delay. Thus, this work proposes a mechanism that operates continuously (in runtime) and utilizes the knowledge about the TDD cycle provided by the method described in Section 3.2. The mechanism uses information such as the TDD class that each packet belongs, good or bad clusters, to compensate for the bias introduced by the different queuing delay in the m-t-s and s-t-m directions.

Recall from Section 2.2.1 that the slave clock can estimate the time difference between the local clock and the master clock using the timestamp differences in (2.15). However, this calculation produces a biased estimate due to the delay asymmetry  $\gamma[n]$  as shown in (2.17).

The delay asymmetry is defined as half of the difference from the delay in the m-t-s and s-t-m, as follows:

$$\gamma[n] = \frac{d_{sm}[n] - d_{ms}[n]}{2}, \quad (3.22)$$

where each delay component ( $d_{ms}[n]$  and  $d_{sm}[n]$ ) can be modeled as a sum of dynamic and static components, as discussed in Section 2.2.2. Rewriting (3.22) in terms of the dynamic and static components, yields:

$$\gamma[n] = \frac{(q_{ms}[n] + \kappa_{ms}) - (q_{sm}[n] + \kappa_{sm})}{2} \quad (3.23)$$

$$= \frac{q_{ms}[n] - q_{sm}[n]}{2} + \frac{\kappa_{ms} - \kappa_{sm}}{2}, \quad (3.24)$$

where the first term is the queuing-induced asymmetry (QIA) caused by the different end-to-end delay experienced by the PTP packets in the downlink and uplink directions due to the different FH utilization. The QIA is generally the most significant by far in the TDD scenario. The second term is the asymmetry caused by the static components of the delay.

The method relies mainly on the first difference of the delays experienced by PTP messages defined as:

$$\Delta d_{ms}[n] = d_{ms}[n] - d_{ms}[n-1]. \quad (3.25)$$

When the TDD is on the uplink or gap slot, the PTP messages in the downlink will experience low delays, with null queuing delays in the network. That is,  $q_{ms}[n] = 0$ . In contrast, when the TDD is in the downlink slot, the PTP messages in downlink will have to share the network with the FH traffic, and in this case, the delay will vary with a random realization of  $q_{ms}[n]$ . Thus, the goal is to capture the difference between packets that traverse the network with a high delay (transmitted in the downlink slot) and those with a low delay (transmitted in

the uplink or gap slots). Ultimately, this difference results in the asymmetry caused by the FH traffic and can be used to compensate final time offset estimated by the slave clock. The same analysis is used for the packets in the uplink direction. The difference is that on the uplink, the higher delays will happen when in the uplink slots and the lower in the downlink and gap slots.

The goal is to estimate the bias affecting the bad packets due to the queuing delay. Thus, if we capture the difference of the delay in the m-t-s direction, experienced by packets classified as good and the delay of a packet classified as bad, the delay difference becomes:

$$\Delta d_j^{GB}[n] = \begin{cases} q_{ms}[n] + (\kappa_{ms} - \kappa_{ms}), & \text{for } j=ms \\ q_{sm}[n] + (\kappa_{sm} - \kappa_{sm}), & \text{for } j=sm, \end{cases} \quad (3.26)$$

where the subscript  $GB$  denotes a transition from a good to a bad packet, and the subscript  $j$  denotes the direction (m-t-s or s-t-m). The term  $(\kappa_j - \kappa_j)$  is considered to be equal to zero, as the constants cancel each other out. Thus, equation (3.26) becomes:

$$\Delta d_j^{GB}[n] \approx \begin{cases} q_{ms}[n] & \text{for } j=ms \\ q_{sm}[n] & \text{for } j=sm. \end{cases} \quad (3.27)$$

As discussed in Section 2.2.1, the slave only has the knowledge from the timestamps. So, by taking the first differences of  $t_{21}[n]$  and  $t_{43}[n]$  from (2.15), we obtain the following metrics that can be used to estimate the  $\Delta d_j^{BG}$ :

$$\begin{cases} \Delta t_{21}[n] = \Delta x[n] + \Delta d_{ms}[n] \\ \Delta t_{43}[n] = -\Delta x[n] + \Delta d_{sm}[n], \end{cases} \quad (3.28)$$

where  $\Delta x[n]$  represents the time offset drift. If the time offset does not change significantly between the consecutive PTP messages, the true time offset drift can be assumed to be negligible.

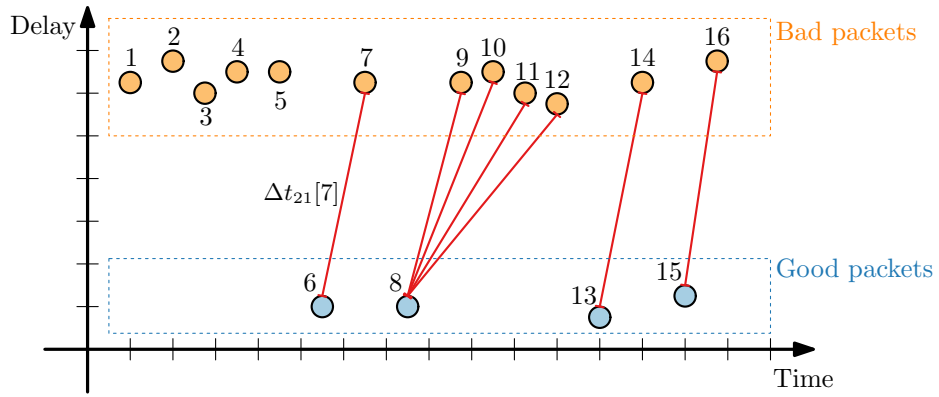
Therefore, for simplicity, we will assume that  $\Delta x[n] = 0$ .

From (3.28), the slave clock can estimate the delays as follows:

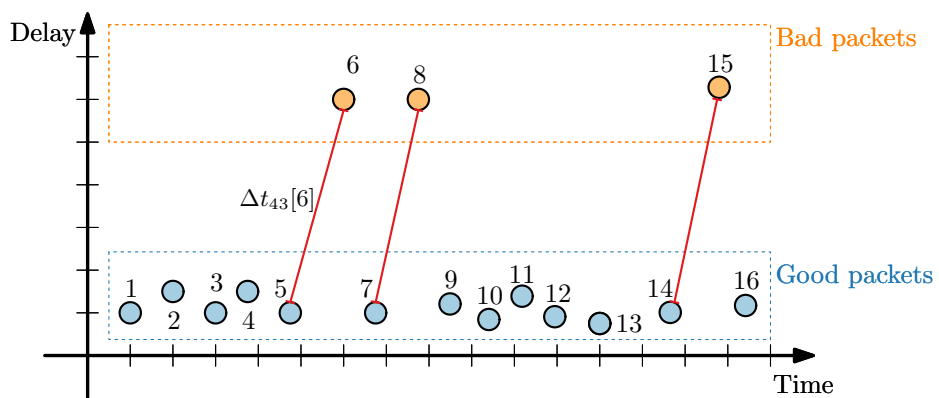
$$\begin{cases} \tilde{\Delta} d_{ms}[n] = \Delta t_{21}[n] - \Delta x[n] \\ \tilde{\Delta} d_{sm}[n] = \Delta t_{43}[n] + \Delta x[n], \end{cases} \quad (3.29)$$

However, as discussed previously, the only transitions of interest are the ones from good to bad packages, as the only packets that need to be corrected are the ones that suffer queuing delay. Therefore, the slave can estimate  $\Delta d_{ms}^{GB}$  and  $\Delta d_{sm}^{GB}$  for the  $n$ -th packet as follows:

$$\begin{cases} \tilde{\Delta} d_{ms}^{GB}[n] = t_{21}[n] - t_{21}[k] \\ \tilde{\Delta} d_{sm}^{GB}[n] = t_{43}[n] - t_{43}[k], \end{cases} \quad (3.30)$$



**Figure 3.11:** Timestamp differences in the downlink direction.



**Figure 3.12:** Timestamp differences in the uplink direction.

where  $k$  is the sample position of the last good packet ( $k < n$ ), and  $t_{21}[k]$  and  $t_{34}[k]$  represents the difference of the timestamps from the last good packet in the m-t-s and s-t-m, respectively. Fig. 3.11 and Fig. 3.12 illustrates the operation.

Finally, the QIA can be estimated by calculating the first difference from the good to bad transitions in both directions (m-t-s and s-t-m), as follows:

$$\hat{b}[n] = \frac{\tilde{\Delta}d_{ms}^{GB}[n] - \tilde{\Delta}d_{sm}^{GB}[n]}{2} = \frac{q_{ms}[n] - q_{sm}[n]}{2}. \quad (3.31)$$

However, note that when considering an  $n$ -th exchange that happens on a downlink slot on the TDD FH, the PTP packet on the m-t-s will have a high delay and will be classified as bad. On the other hand, the packets on the s-t-m will have the network with a low network load. Thus, the packets in the s-t-m direction will be classified as good. In this situation, there is no estimation for the  $\tilde{\Delta}d_{sm}^{GB}[n]$ , so (3.31) should consider  $\tilde{\Delta}d_{sm}^{GB}[n] = 0$  to calculate the bias for the  $n$ -th packet. The same idea holds for the downlink.

The slave can compensate for the bias on the time offset measurement from (2.16), as follows:

$$\hat{x}[n] = \tilde{x}[n] - \hat{b}[n]. \quad (3.32)$$

The corrections can also be applied asynchronously, on the difference of the timestamps, as follows:

$$\begin{cases} \hat{t}_{21}[n] = t_{21}[n] - \alpha_{ms}[n]\tilde{\Delta}d_{ms}^{GB}[n] \\ \hat{t}_{43}[n] = t_{43}[n] - \alpha_{sm}[n]\tilde{\Delta}d_{sm}^{GB}[n], \end{cases} \quad (3.33)$$

where  $\alpha_j[n]$  is 1 if the packet is classified as bad or 0 otherwise. We assume that the slave can detect the class for each packet using the methods shown in Section 3.2.

Once the slave compensates for the bias using (3.32) or (3.33), the residual bias is expected to approach the bias that exists when the PTP is transmitted without FH traffic, aside from the estimation error.

# Chapter 4

## Testbed

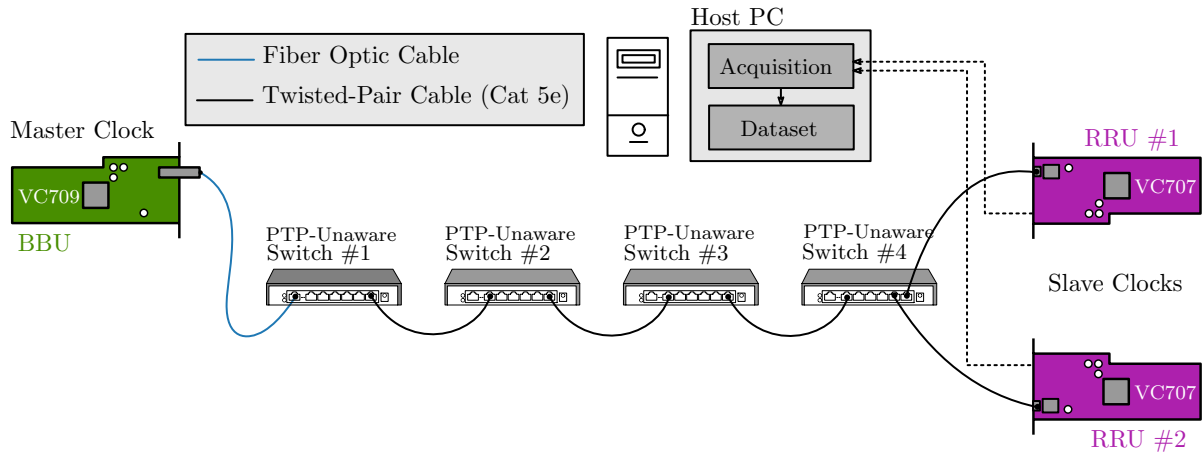
This chapter describes the testbed used to generate the results present in this work. The use of a testbed to extract realistic information helps to understand the complexity and problems that are present in timing systems. The testbed's primary goal is to support the acquisition of labeled timestamps datasets with high precision for offline processing in a realistic environment. The collected datasets can then be used to evaluate multiple synchronization algorithms, find patterns and extract further insights from the data.

This chapter is organized as follows. In Section 4.1, an overview of the current testbed is described. Section 4.2 highlights some important points related to the PTP implementation. Subsequently, Section 4.3 discusses how the FH packets are encapsulated and periodically transmitted. Section 4.4 details the changes introduced on the testbed to make the FH traffic mimic the periodic pattern described in Section 3.1. Finally, Section 4.5 describes the acquisition of labeled datasets, and Section 4.6 presents the framework used for processing the datasets.

### 4.1 Testbed Overview

The testbed used in this work was initially described in [26, 48, 49] and further improved in [36, 44]. Given the complexity of the current testbed, the details described in this work will be concentrated on the key components used to generate the experiments described in Chapter 5.

Fig. 4.1 presents an overview of the current testbed. It is composed of three Xilinx Virtex-7 FPGAs, among which two Xilinx VC707 boards act as PTP slave clocks (RRUs) and one Xilinx V709 board act as the PTP master clock (BBU), where the FH network are composed of PTP-unaware GbE switches in a tree topology. In the testbed, the master and slave clocks



**Figure 4.1:** Testbed overview.

exchange PTP messages, as described in Section 2.2.1. For each exchange, the slave clocks collect  $t_1[n]$ ,  $t_2[n]$ ,  $t_3[n]$ ,  $t_4[n]$  and some auxiliary timestamps. All the collected timestamps are then forwarded to the host PC. Finally, the host PC saves all the exchanges in a dataset to be later processed by PTP-DAL. The details about the auxiliary timestamps and PTP-DAL will be clarified later in this chapter.

The PTP-unaware network can be configured in two ways. The first is using one switch, where a different number of hops are configured using port-based VLANs. The second option is to use four physical switches, whose details are described in Table 4.1. The analysis of the different processing delays in both configurations is described in Section 5.2.

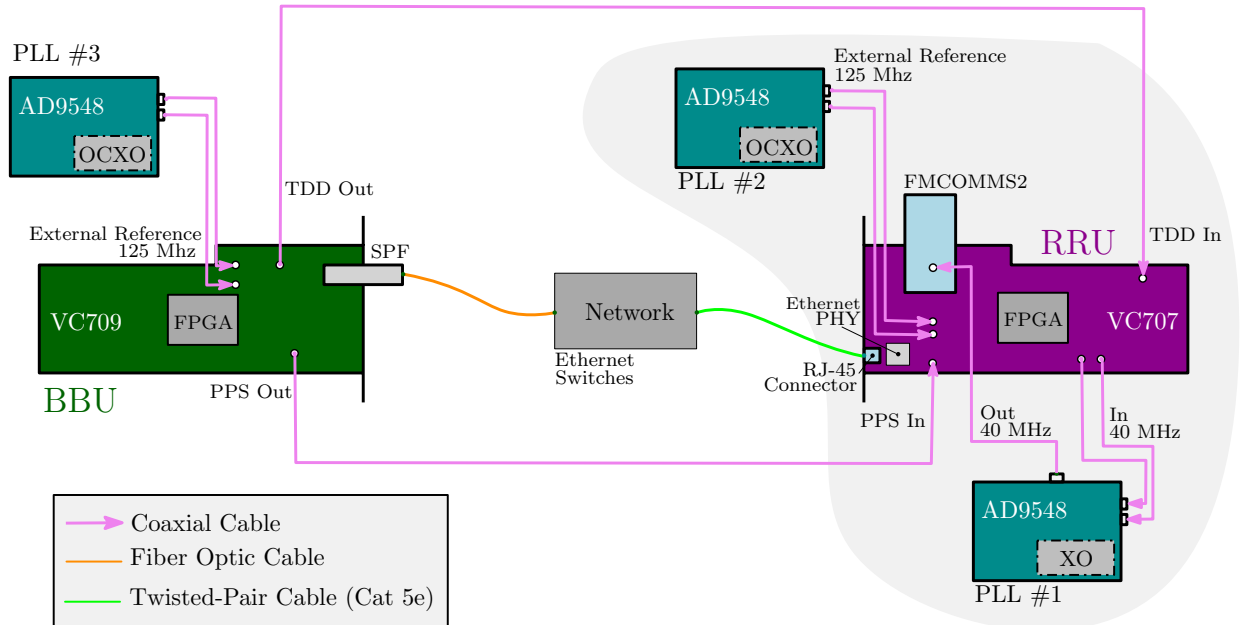
**Table 4.1:** PTP-Unaware switches used in the testbed.

Label	Model
Switch #1	Intelbras SG 2404 MR
Switch #2	TP-Link T2600G-28TS
Switch #3	TP-Link TL-SG3210
Switch #4	TP-Link TL-SG3210

The BBU communicates with the Ethernet Switch through a 1000BASE-SX (optical) small form-factor pluggable (SFP) transceiver, and the RRUs adopt a 1000BASE-T (copper), using the onboard Marvel 88E1111 GbE PHY. Despite the different PHY interfaces, all the FPGAs implement the same Ethernet MAC, which consists of the Xilinx Tri-Mode Ethernet



MAC (EMAC) IP [50].



**Figure 4.2:** Overview of the testbed connections and clock distribution paths.

Fig. 4.2 describes the connections and some of the clock distribution present in the testbed (for simplicity it only describes one BBU and one RRU). Starting with the BBU, Fig. 4.2 shows an external reference with a high-stability 125 MHz differential clock signal, which comes from an AD9548 evaluation board [51] (labeled as PPL#3). The PPL #3 was customized with a Connor-Winfield's OH300 stratum 3E OCXO, which has frequency stability of  $\pm 5$  ppb. It is important to note that this external reference feeds the master RTC. Since all the other PTP clocks are derived with respect to the master RTC, this external clock determines the primary reference clock (PRC) in the testbed.

Next, the BBU is connected to a chain of Ethernet Switches and distributes its clock via PTP. The RRU synchronizes its local RTC based on the PTP packets. As detailed in [49], the RRU uses the synchronized time to generate a synchronized frequency that is used to feed an Analog Devices FCOMMS2 board. More specifically, the RRU first outputs a 40 MHz reference that is connected to an AD9548 (labeled as PLL #1). The PPL provides a cleaner signal which is finally used to feed the FCOMMS2 board, the RF frontend.

The RRU provides two types of oscillators used to control its local RTC. The first one is an XO, which comes from the onboard XO and has frequency stability of  $\pm 50$  ppm. The second one is an OCXO and has a similar setup to the one used in the BBU, which comes from

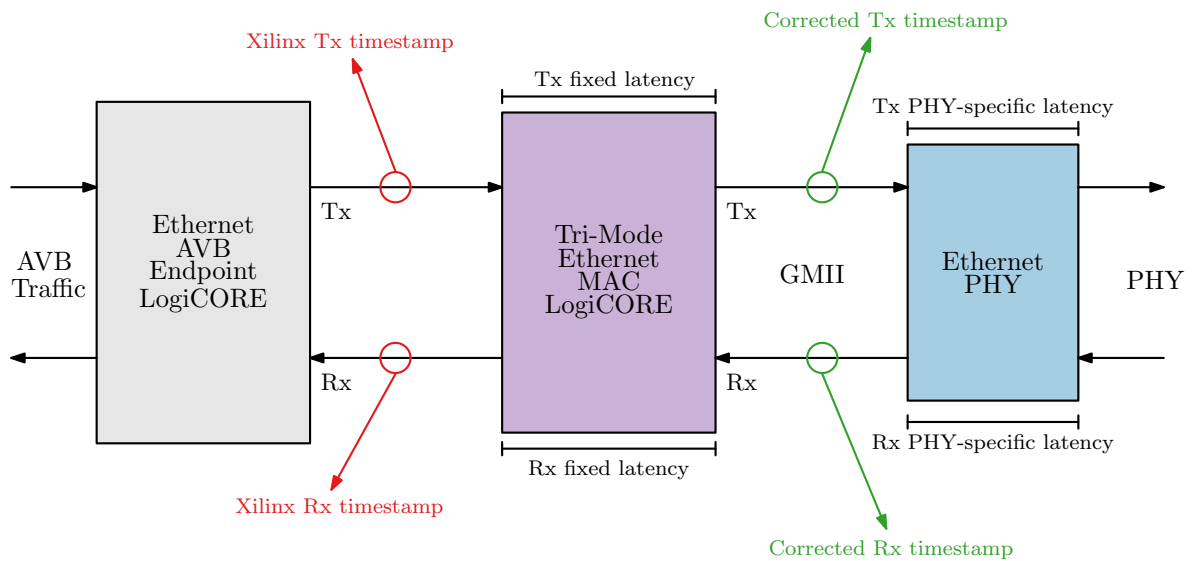
an AD9548 with a custom OCXO with a frequency stability of  $\pm 5$  ppb. In the experiments presented in Chapter 5, both oscillators will be used to compare the PTP performance when the slave clock uses oscillators with different frequency stability specifications.

The pulse per second (PPS) signal and the TDD signal that connects the BBU and the RRUs, as illustrated in Fig. 4.2, will be described in Section 4.5 and Section 4.4, respectively.

## 4.2 PTP Infrastructure

The PTP implementation used in the testbed comes from the adopted EMAC [50], which is instantiated using the so-called Ethernet audio video bridging (AVB) endpoint option [52]. This implementation provides PTP hardware timestamping and the RTC from which the PTP packet departure and arrival timestamps are taken.

In terms of timestamping granularity, the RTC from the EMAC implementation, is driven by a 125 MHz clock. Thus, the time increment is  $1/f_{nom} = 8$  ns. As a result, the quantization noise for each timestamp on the testbed is uniformly distributed from 0 to 8 ns.



**Figure 4.3:** Testbed timestamp position.

Regarding the timestamp mechanism, as discussed in Section 2.2.2, the EMAC implementation takes the timestamping on PTP arrival and departure when the first symbol after the start of frame delimiter (SFD) from the PTP event messages, crosses the red line highlighted in Fig. 4.3. Then, to reduce the latency introduced by the timestamping mechanism, the implementation corrects the fixed latency introduced by the Tri-Mode Ethernet MAC block for each

timestamp by subtracting or adding the known MAC latency. As a result, the final timestamps correspond to the time at the gigabit media-independent interface (GMII) interface, highlighted in green in Fig. 4.3.

Note that, the latency introduced by the external PHYs will still introduce a variable latency that will contribute to the total delay experienced by the PTP packet. More specifically, it will introduce delay asymmetry that impacts the PTP performance.

### 4.3 Fronthaul Traffic

The testbed is composed of 2 RRUs served by a central BBU. Each RRU has two antenna carriers (AxCs). Thus, the FH traffic at the BBU is configured to transmit 4 independent LTE 5 MHz baseband streams, two for each RRU. Similarly, each RRU transmits 2 streams, one for each AxC, in the uplink to the BBU.

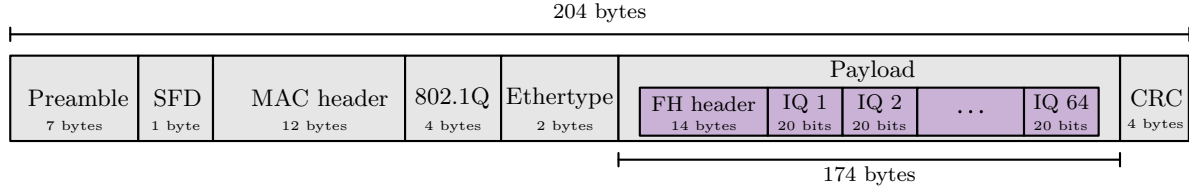
The testbed adopts the FH functional split E from [53], also called raw in-phase and quadrature (IQ) samples, where the IQ samples are encapsulated into the FH packets. Additionally, the FH frames have fixed length, i.e., the number of encapsulated IQ samples per frame and the IQ size in bits are set to fixed values. Finally, the frames are transmitted periodically in time with a constant period  $i_{fh}$ . Thus, the FH traffic generated by the FPGAs are called constant bit rate (CBR) FH traffic.

For instance, if the Ethernet frame is configured to carry 32 IQ samples per AxC, the total IQ samples per packet will be 64 (for each RRU). With this configuration, as the devices process the samples for each AxC in parallel, it takes 32 sample periods  $T_s$  for the devices to fill the FH packet. The FH packets are transmitted as soon as they are filled up with the configured number of samples per packet. Each frame is transmitted periodically with the interval defined as:

$$i_{fh} = \frac{\eta_{spf} \times T_s}{\eta_a}, \quad (4.1)$$

where  $T_s$  is the sampling period,  $\eta_{spf}$  is the number of IQ samples encapsulated per frame, and  $\eta_a$  is the number of antenna streams in the destination RRU. Also, note that  $\eta_{spf}$  includes IQ samples to (from) all antennas of a RRU. For example, considering  $T_s = 1/f_s$ , where  $f_s = 7.68$  MHz,  $\eta_{spf} = 64$  and  $\eta_a = 2$ , the FH packet inter-departure interval is approximately  $4.16 \mu\text{s}$ .

In terms of encapsulation, Fig. 4.4 illustrates a FH frame. Additionally to the IQ samples, the payload of the FH packet also includes a header with 14 bytes of metadata. Besides, the Ethernet header adds more 30 bytes of overhead, which is composed of 7 bytes for the Ethernet

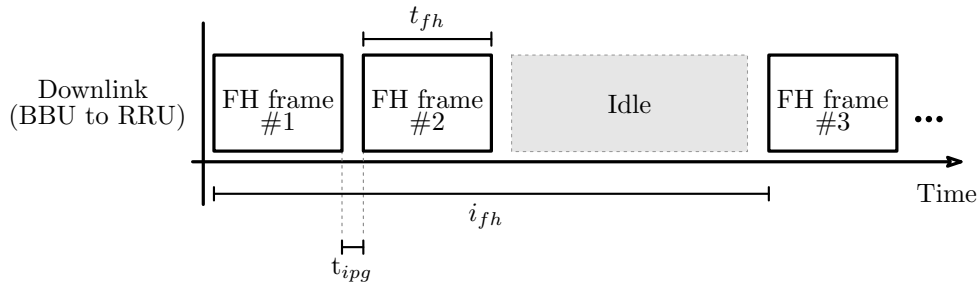


**Figure 4.4:** Illustration of the FH packet structure.

preamble, 1 byte for SFD, 12 bytes for MAC header (destination and source MAC address), 4 bytes for the VLAN 802.1Q tag, 2 bytes for the Ethertype and 4 bytes for the frame check sequence (CRC). Thus, the transmission delay from the FH packets  $t_{fh}$  is given by:

$$t_{fh} = \frac{l_{iq} \times \eta_{spf} + \eta_{oh}}{R_{line}}, \quad (4.2)$$

where  $l_{iq}$  is the IQ sample size in bits,  $\eta_{oh}$  is the Ethernet overhead length and  $R_{line}$  is the Ethernet bit rate. For example, if we consider  $\eta_{spf} = 64$ ,  $l_{iq} = 20$  bits, and a GbE Ethernet, the transmission delay will be  $1.632 \mu\text{s}$ .



**Figure 4.5:** Testbed traffic pattern in CBR FH.

Finally, Fig. 4.5 illustrates the traffic pattern in the FH. In summary, in the DL, two FH frames are transmitted (one of each RRU) every  $i_{fh}$ . In UL, each RRU will transmit one FH frame. The  $t_{ipg}$  is the minimum inter-departure gap of 12 bytes of idle line state required by the Ethernet protocol before transmitting the next packet. Finally, every FH packet has a transmission delay of  $t_{fh}$ .

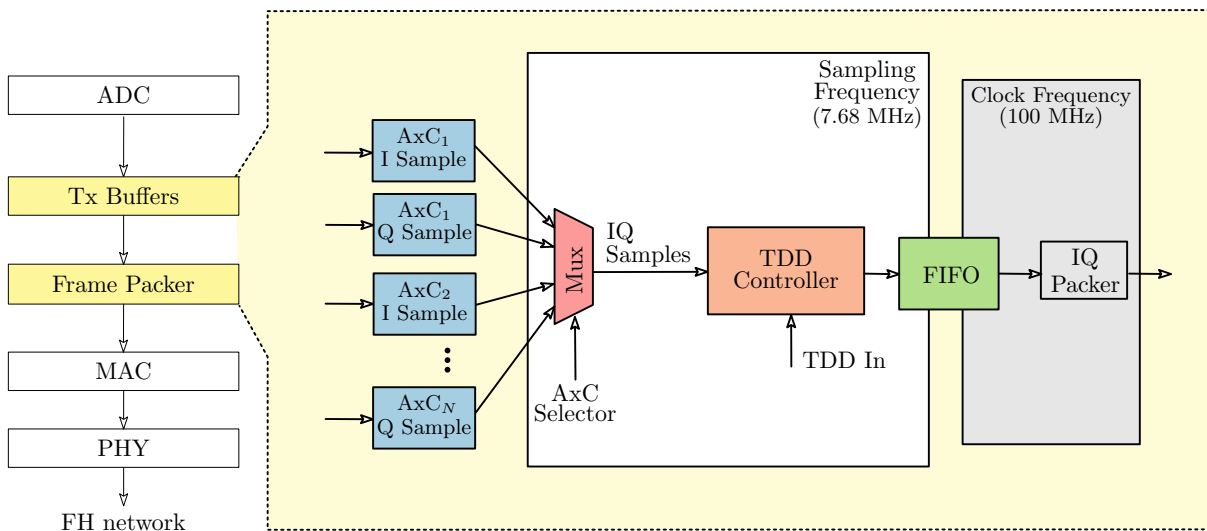
## 4.4 TDD FH Implementation

The testbed used in this work uses an updated version of the one described in [44]. This work introduces a new mechanism to control the FH traffic to mimic a TDD FH, as described

in Section 3.1. However, instead of implementing a TDD system on the air interface, we opt to only mimic the FH traffic’s periodic behavior by controlling how the FH packets are transmitted between the BBU and RRUs in the DL and UL directions.

As described in Section 4.3, the testbed implements a CBR traffic, where the BBU transmits two FH packets, one for each RRU, with a constant period. Similarly, each RRU transmits one FH frame with a constant period. To implement this behavior, the FPGA design of both the BBU and RRU first stores the IQ samples to be transmitted over the FH in the transmission (Tx) buffers. Then, it encapsulates the samples on Ethernet frames and transmits them in the FH network. More specifically, these buffers consist of dual-port first-in first-out (FIFO) queues with independent clocks for read and write operation. By using this architecture, the production rate of FH packets can be controlled.

At the RRU, the transmission path is illustrated in Fig. 4.6. First, the IQ samples are acquired from the analog to digital converter (ADC). Then, at each sampling clock, the IQ sample for each AxC is buffered at the FIFO block. In other words, the write-side of the Tx FIFO is clocked with the sampling clock. For example, using two AxCs, it would take  $32T_s$  periods to write 64 IQ samples in the green FIFO. At the FIFO reader-side, the *IQ packer* block, responsible for forming the FH packets, reads the samples more quickly, with a clock rate of 100 MHz. However, using this dual-port FIFO, the production rate is controlled by the slower operation, which in this case is the sampling frequency.

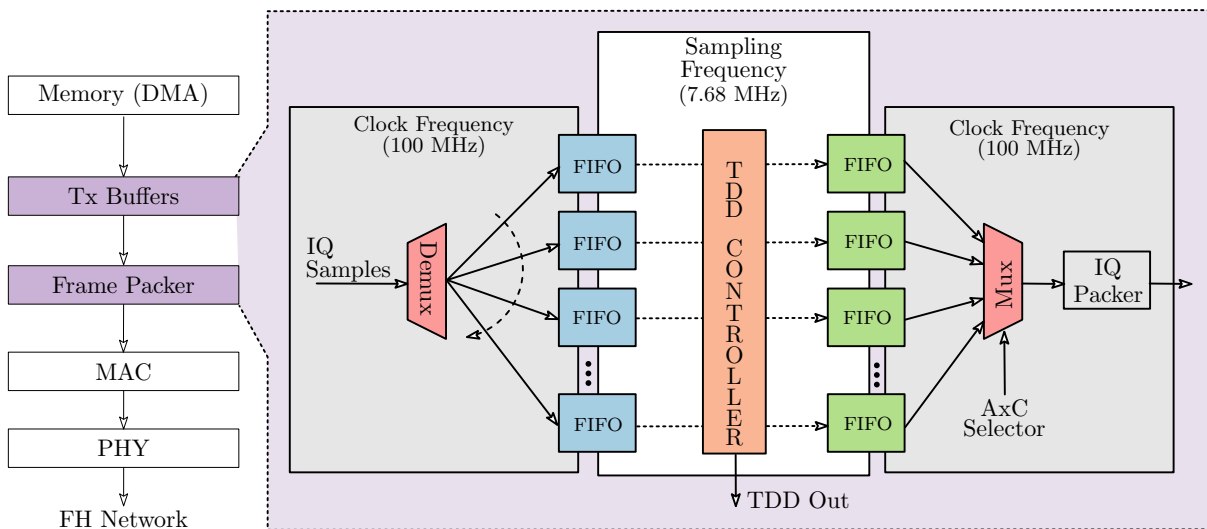


**Figure 4.6:** Tx design for controlling the FH traffic at the RRU.

Given the design implemented in the RRU, we introduced the *TDD Controller* block to

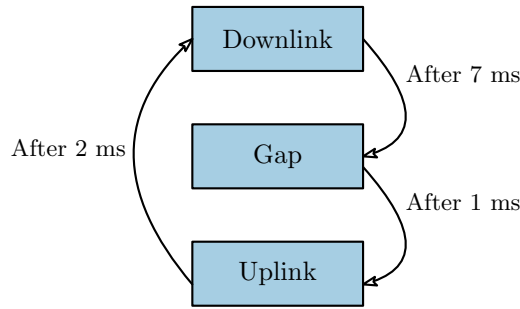
add one more level of control for the FH packet production. The TDD controller is responsible for enabling and disabling the write-side of the green FIFO illustrated in Fig. 4.6. By doing so, the FH traffic, transmitted by the RRU, can be disabled for a precise period of time.

Differently from the RRU, at the BBU side, the Tx FIFOs reads the IQ samples from the memory through a direct memory access (DMA) engine, which operates at 100 MHz. In this case, the mechanism used to control the production rate is based on a two-level dual FIFO architecture, as illustrated in Fig. 4.7. The input has a stream of IQ samples ordered with alternated samples for each AxC. Then, the IQ samples are demultiplexed alternately to a number of parallel two-stage FIFO paths, which is equal to the number of AxC served by the BBU. The write-side of this first-level FIFOs (blue) operates with 100 MHz. Then, the intermediate path conveys the IQ samples from the first-level FIFOs to the second-level FIFOs (green) with a rate equal to the sample rate of the radio interface. As a result, the *IQ packer* block needs to wait for the samples to be ready at the second-level FIFOs, which is controlled by the sampling rate. Similarly to TDD controller implemented at the RRUs, a *TDD controller* block implemented at the BBU can enable/disable the FH production operation by controlling the read-side from the first-level FIFOs.



**Figure 4.7:** Tx design for controlling the FH traffic at the BBU.

To mimic a TDD FH as described in Section 3.1, the TDD controller at the BBU implements a state machine composed of three stages: downlink, gap and uplink. At the downlink stage, the controller illustrated in Fig. 4.7 enables the intermediate path to get only enough samples to form FH packets that can be transmitted within the downlink time slot. In other words,



**Figure 4.8:** TDD State machine.

at the downlink stage, the controller counts enough IQ samples to be transmitted in the defined period of time. Similarly, the TDD controller at the RRU controls the number of samples that can be transmitted within the uplink time slot. As DL and UL in a TDD system need to be synchronized, the controller block implemented in the BBU is the one that controls the overall system. For that, the BBU produces a *TDD out* signal that is connected in both RRUs (*TDD in*) using a coaxial cable, as illustrated in Fig. 4.2.

For example, if the TDD controller at the BBU is configured to operate with the TDD slot configuration **DDDDDDDFUU**, the state machine will have the behavior described in Fig. 4.8, where the downlink is enabled for 7 ms, then the gap is enabled for 1 ms and finally, the uplink is enabled for 2 ms. In this configuration, the first stage will be the downlink, where the TDD controller at the BBU will enable the buffers to make the IQ samples flow and it will disable the buffers at the RRU to stop the traffic during this time slot. Then, at the gap time slot, the controller will disable the traffic from both BBU and RRUs. Finally, at the uplink time slot, the controller will enable only the traffic from both RRUs for 2 ms.

To facilitate the explanation, let us consider a numerical example. First, recall that the IQ samples from each AxC are processed in parallel by each green FIFO, as illustrated in Fig. 4.7. Thus, considering that the BBU are serving 4 AxCs (two for each RRU), there will be 4 FIFOs. In this setup, the TDD controller counts up to 53760 IQ samples for each FIFO and then it disables the buffers. That is because, considering the sampling frequency of  $f_s = 7.68$  MHz, in 7 ms the BBU can only produce 53760 samples per AxC. After that, the state machine will transit to the gap stage, where using the same rationale, the TDD controller will count up to 7680. Each increment is controlled by the sampling period. As a result, it is equivalent to waiting for 1 ms. Finally, the controller will start the uplink stage, for that, the TDD signal will be enabled and the TDD controller at the RRU will enable the traffic, similar to the BBU, until

the limit of 15360 IQ samples, which is the number of samples produced in 2 ms. Then, the cycle restarts at the BBU.

Therefore, using this implementation, the FH traffic can be controlled to have periodic traffic, similar to the one described in Section 3.1.

## 4.5 Acquisition of Labeled Datasets

A key feature of the testbed is that it can generate datasets with PTP information with high accuracy for offline analysis. As commented in previous sections, those datasets are composed of the common PTP timestamps  $t_1[n]$ ,  $t_2[n]$ ,  $t_3[n]$  and  $t_4[n]$ , and auxiliary timestamps. The auxiliary timestamps are used to calculate the true slave time offset ( $x[n]$ ) and the real one-way packet delays in the m-t-s direction ( $d_{ms}[n]$ ) and the s-t-m direction ( $d_{sm}[n]$ ) over each PTP exchange. This feature allows for the offline analysis of different synchronization algorithms under the same environmental conditions once we know the true time offset, such that we can assert the quality of such algorithms. It also enables the analysis of delays suffered by the PTP packets in different network conditions.

To support the acquisition of such datasets, the testbed uses two distinct RTCs in the slave node, the PTP-synchronized RTC (PTP-RTC) and the PPS-synchronized RTC (PPS-RTC). The PTP-RTC comes with the EMAC IP and is the source from which the departure and arrival PTP timestamps are taken, as discussed in Section 4.2. On the other hand, the PPS-RTC is responsible for the auxiliary timestamps, which are used to determine the ground truth labels. The design aspects and hardware details of the PPS-RTC implementation is out of the scope of this work but can be found in [36, 44].

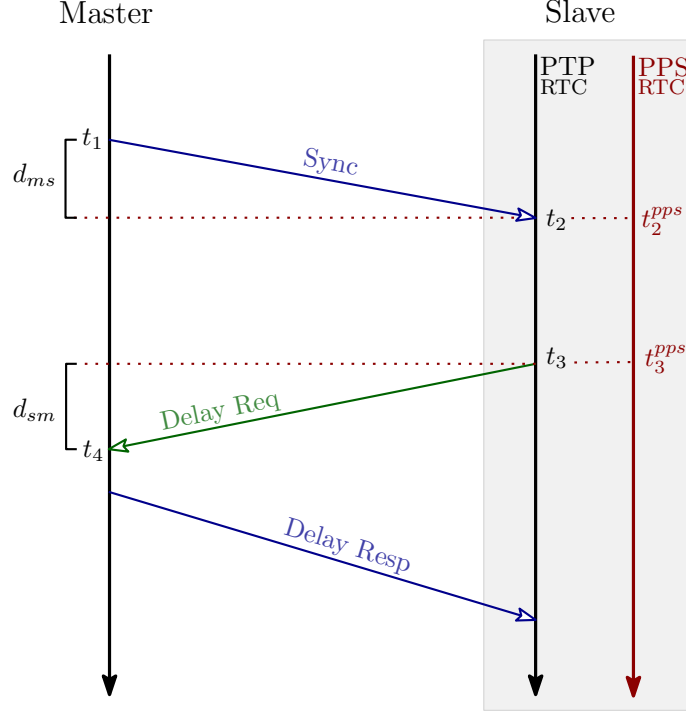
The acquisition architecture is described at a high level in Fig. 4.9. The master clock produces a PPS signal from its RTC, which is connected through a coaxial cable to the slaves. This signal is used to synchronize with a high precision the PPS-RTC present in the slave devices. In other words, the PPS-RTC shares the reference time from the master clock. In addition to the PPS-RTC, the slave also has the PTP-RTC, which is used for the normal PTP operation. Therefore, for each PTP exchange, the slave generates the timestamps  $t_2[n]$  and  $t_3[n]$ , and almost at the same time <sup>1</sup> it also takes the auxiliary timestamps ( $t_2^{pps}[n]$  and  $t_3^{pps}[n]$ ), which

---

<sup>1</sup>The sampling of the two RTCs happens at the same clock cycle but it has a sophisticated mechanism for matching the timestamps as described in [44].



correspond to the  $t_2[n]$  and  $t_3[n]$  at the master reference time, as shown in Fig. 4.9.



**Figure 4.9:** PTP delay request-response mechanism.

In the end, the acquisition procedure generates six timestamps: the common PTP timestamps taken from the PTP-RTC ( $t_1[n]$ ,  $t_2[n]$ ,  $t_3[n]$  and  $t_4[n]$ ) and the auxiliary timestamps taken from the PPS-RTC ( $t_2^{pps}[n]$  and  $t_3^{pps}[n]$ ). With the timestamps from the PPS-RTC, it becomes possible to calculate the following ground truth labels:

$$x[n] = t_2[n] - t_2^{pps}[n] \quad (4.3)$$

$$d_{ms}[n] = t_2^{pps}[n] - t_1[n] \quad (4.4)$$

$$d_{sm}[n] = t_4[n] - t_3^{pps}[n] \quad (4.5)$$

where  $x[n]$  is the true slave time offset with respect to the master clock,  $d_{ms}[n]$  is the true delay experienced by the PTP packet in the m-t-s direction and  $d_{sm}[n]$  is the true delay experienced in the s-t-m direction.

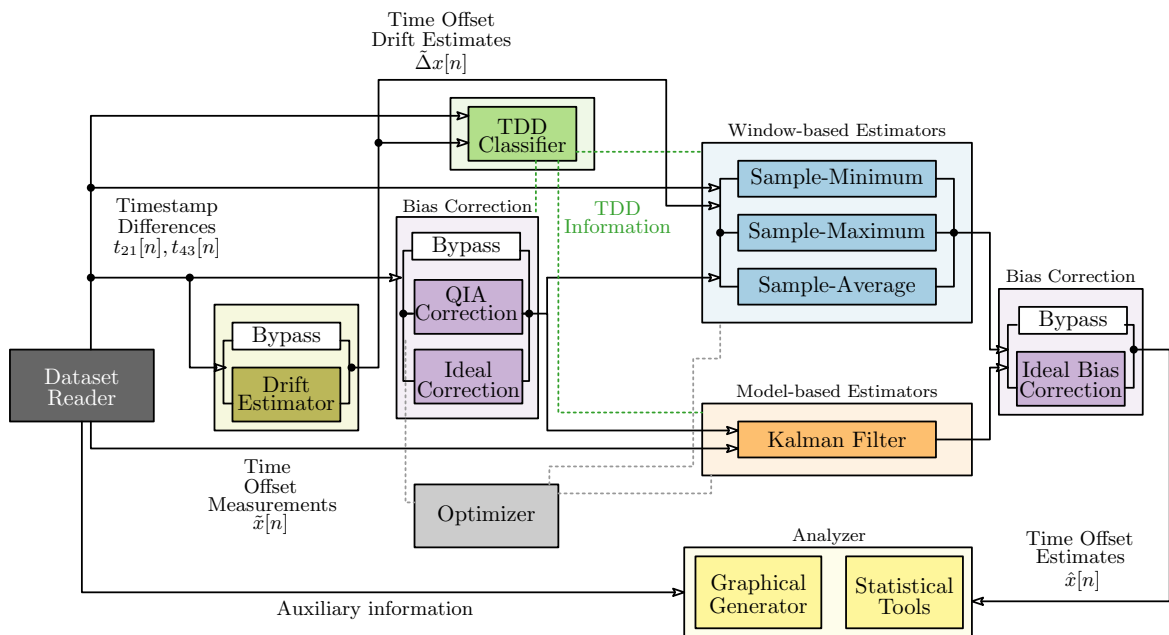
## 4.6 PTP-DAL

The offline processing of the dataset acquired from the testbed described in the last section is carried out by the PTP-DAL, an open-source library proposed in [36]. The library is imple-

mented in Python and presents several synchronization algorithms discussed in the literature. It also provides many graphical and statistical assessment tools to compare different algorithms under the same conditions.

Fig. 4.10 illustrates the PTP-DAL architecture used in this work. The processing chain starts by reading the dataset acquired from the testbed at the *dataset reader* block. Next, the framework processes all the information available to calculate the true labels, using (4.3), (4.4), (4.5), and also the PTP measurements using (2.17) and (2.18) to estimate the time and frequency offset, respectively. The true labels and the resulting estimates become available to be used by the other algorithms.

The algorithm blocks (e.g., *TDD classifier*, *window-based estimators* and *model-based estimators*, *QIA correction* and *drift estimator*) are fully implemented with the formulation presented in Chapter 3. Besides, the *optimizer* block implements a grid search optimizer to adjust the algorithm parameters based on the  $\max|\text{TE}|$ .



**Figure 4.10:** PTP-DAL architecture.

PTP-DAL also implements a way to calculate and correct the ideal bias present in the dataset due to the delay asymmetry. The bias correction is calculated based on the true minimum, mean and maximum delay experienced by the PTP packet in each direction. Based on these measurements, the delay asymmetry is calculated and based on this value, the time offset estimates are shifted. For instance, the output from the sample minimum estimate produced

with (3.16) is corrected by compensating the minimum asymmetry calculated based on the true delays. On the other hand, the output from the sample-average and sample-maximum are compensated using the average and maximum delay asymmetry, respectively. In summary, the bias correction depends on the samples used by the algorithm. If the algorithm uses the minimum delayed packets, the correction will be made based on the minimum delays.

Finally, based on all the estimates produced by the algorithms and the information available on the acquired datasets, PTP-DAL generates graphical and statistical results.

# Chapter 5

## Results

This chapter presents the experimental results from the algorithms presented in Chapter 3. The experiments explore the testbed described in Chapter 4. The main goal is to evaluate the achievable PTP performance in a practical TDD FH network.

The chapter is organized as follows. First, Section 5.1 describes all the parameters used to generate the experiments. Next, Section 5.2 analyzes the delay suffered by the PTP packets when only PTP is being transmitted on the network and when PTP shares the network with FH traffic that has a periodic behavior as described in Section 3.1. Afterwards, Section 5.3 analyzes the TDD packet classifiers. Then, Section 5.4 presents the performance of the proposed mechanism to compensate QIA delay. Finally, the performance of the sample minimum, sample maximum, sample average and KF are analyzed in terms of  $\max|\text{TE}|$  in Section 5.5.

### 5.1 Experimental Setup

In terms of network configuration, the FH network used in this work is composed of GbE network switches, where the master (BBU) and slave (RRU) clocks are connected in a tree topology with both RRUs connected to the same aggregation node, as shown in Fig. 4.1. Besides, as discussed in Section 4.1, the testbed can be configured with two types of network hops using the switches described in Table 4.1. The first option is a dedicated port-based VLAN configured using only one PTP-unaware switch, in a way that one switch acts as multiple hops. The second option is based on multiple physical switches, where each hop is indeed an individual switch. The former option was used in previous works [19, 36]. However, it will be shown that the total processing delay differs between the configurations.

In the experiments, both RRUs are served by the central BBU. Each RRU has two antenna carriers  $\eta_a = 2$ . Thus, the FH traffic at the BBU is configured to transmit 4 independent LTE 5 MHz baseband streams, two for each RRU. Similarly, each RRU transmits 2 streams, one for each AxC, in the uplink to the BBU. Additionally, the FH packets are transmitted in unicast and are configured with  $\eta_{spf} = 64$  and  $l_{iq} = 20$  bits. Consequently, the transmission delay is  $t_{fh} = 1.632\mu s$ , with an interpacket gap  $t_{ipg} = 96$  ns.

The PTP is configured with a periodicity of 64 exchanges per second, where each packet has 80 bytes, 54 bytes for the PTP information and 26 bytes used for the Ethernet header, resulting in a transmission delay of  $t_{ptp} = 640$  ns. The PTP measurements, also called raw measurements in this chapter, are calculated using (2.16).

The testbed is flexible and presents many types of configurations. For instance, the experiments presented in this work use two different types of oscillators on the slave clock. The first is the XO, with frequency stability of  $\pm 50$  ppm, and the second is the OCXO, with frequency stability of  $\pm 5$  ppb. One should note that, as discussed in [44], the error associated with the ground truth labels (from the PPS-RTC) depends on the chosen oscillator due to the RTC implementation. Therefore, the error associated with the true labels ranges within  $\pm 4$  ns when using the OCXO oscillator, and  $\pm 8$  ns when using the XO.

As discussed in Section 4.5 the experiments are based on datasets acquired from the testbed. However, before starting the acquisition process, an initialization procedure is performed. The slave clocks estimate and correct the frequency offset by running a real-time mechanism until the frequency offset remains within  $\pm 59.6$  ppb (i.e.,  $\pm 59.6$  ns/sec). After the initial procedure, the PTP-RTC operates in a free-running mode so that there are no corrections to the timestamps in the hardware. In this way, we can capture the natural evolution from the timestamps and only correct (apply the synchronization procedure) in software using the algorithms described in Chapter 3. The adopted datasets consist of 45 minutes of acquisition. For fairness, the initial 10 minutes are discarded to ignore the transient state of some algorithms. Thus the performance is evaluated based on the last 35 minutes.

Finally, the performance of the algorithm used to estimate the time offset is evaluated in terms of the  $\max|TE|$  metric from (2.12). More specifically, the  $\max|TE|$  is calculated over a one-minute observation window, i.e., there is one value for every minute of the experiment.

## 5.2 Delay Analysis

This section analyzes the delay experienced by the PTP messages over the experiments analyzed in this work. In PTP-unaware networks, the synchronization performance is heavily impacted by the characteristics of the delays experienced by the PTP packets. Hence, it is helpful to analyze the delay distribution to understand the performance achieved by the raw PTP measurements and the algorithms discussed in Chapter 3.

The delays presented in this section are based on the true delay experienced by the PTP packets. PTP-DAL calculates the true delay using the auxiliary timestamps, as discussed in Section 4.5. More specifically, the true delay in the m-t-s and s-t-m are calculated using (4.4) and (4.5), respectively.

In this section, the PTP delay distribution is analyzed using the two types of hop configuration: port-based VLAN and physical hops through multiple physical switches. In addition to the hop configuration, Section 5.2.1 analyzes the distinct delay experienced by the PTP packets when using a network exclusively for transmitting PTP packets, and Section 5.2.2 analyzes the delay when PTP shared the network with FH traffic.

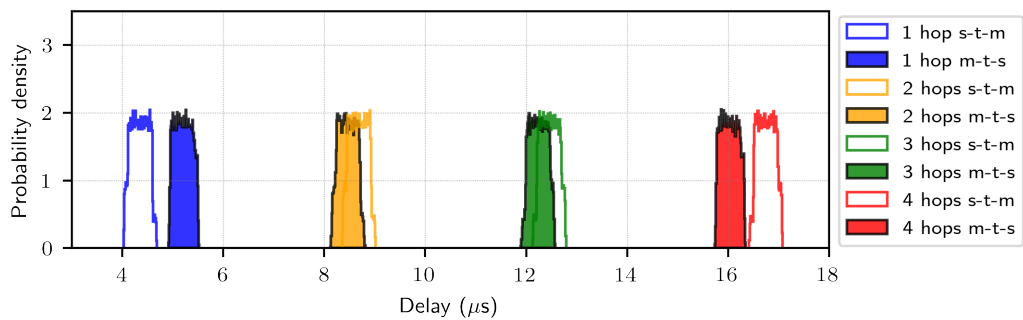
### 5.2.1 PTP without FH Traffic

Fig. 5.1 shows the PTP delay distribution from one to four hops in the m-t-s and s-t-m directions. The distribution of delays in the m-t-s direction is shown with the filled histograms, while the unfilled ones show the delay in the s-t-m direction. Fig. 5.1a and 5.1b show the delay distribution with a port-based VLAN setup using switches #1 and #2 from Table 4.1, respectively. The experiments with switches #3 and #4 were omitted because the delay distribution is very close to the ones from switch #2, since switches #2, #3, and #4 are similar models from the same manufacturer, which have very close characteristics. Note that the delay distribution in all hop configurations presents a shape that resembles the uniform distribution.

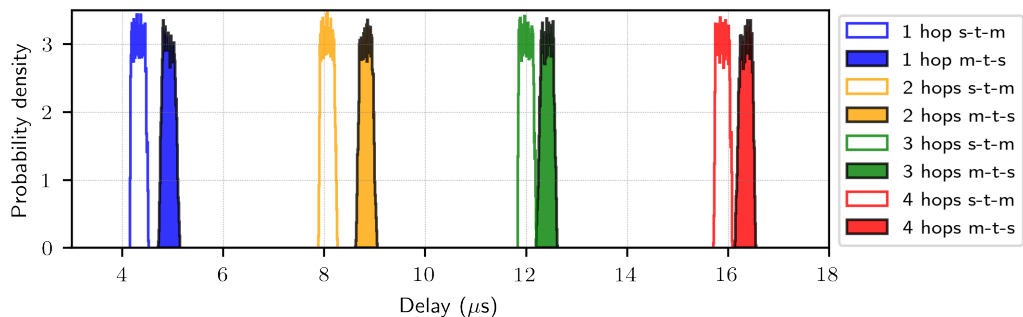
Furthermore, Fig. 5.1 shows that different switches introduce delays with specific characteristics depending on their internal components and implementation. Switch #1 has a distinct delay distribution when compared with the delays from switch #2. It also introduces a different delay asymmetry per hop as summarized in Table 5.2 and Table 5.3. One of the main differences between switch #1 and #2 is on the dispersion of the distribution. The delay distribution from switch #1 presents approximately a standard deviation (SD) of 160 ns and the distribution span

between 600 and 700 ns for all hop configurations. On the other hand, the delay distribution from switch #2 presents approximately a SD of 100 ns and a span between 380 ns and 430 for all hop configurations.

As discussed in Section 2.2.2, the total delay that a PTP event message suffers through the network is composed of different components. Furthermore, the delay components can be divided between static and dynamic. The delay components modeled as a realization from a random variable are the queueing and processing delay, as summarized in Table 2.1. One should note that Fig. 5.1 shows the delay when only PTP packets are present in the network, so there is no queueing delay contribution. Therefore, the processing delay is the predominant source of random noise and determines the shape of the delay distribution. Thus, we can use such results to understand the contribution of the processing delay on the PTP packets.



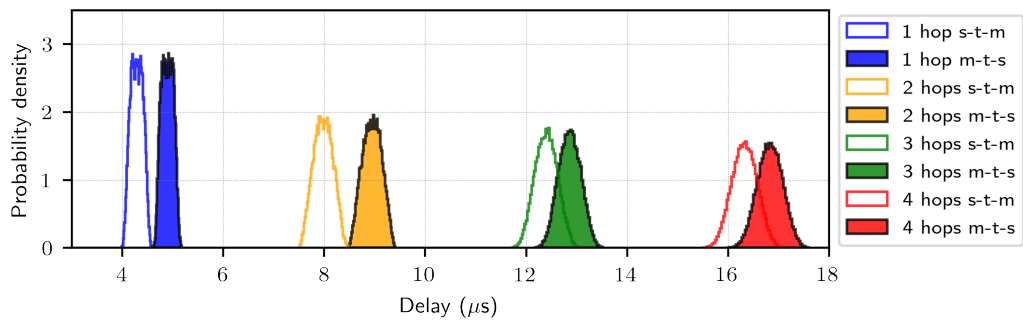
(a) With switch #1 using VLAN-based network.



(b) With switch #2 using VLAN-based network (similar distribution with switches #3 and #4).

**Figure 5.1:** True delay distribution in the m-t-s and s-t-m direction using port-based VLAN in a one to four-hop configuration in absence of FH traffic.

In the case where the queueing delay is zero, the total delay left is given by the propagation, transmission, processing and PHY latencies from the master, slave and switch nodes. As



**Figure 5.2:** True delay distribution in m-t-s and s-t-m direction using multiples physical switches in absence of FH traffic. The 1-hop configuration uses switch #4, 2-hops uses switches #4 and #1, 3-hops uses #4, #1 and #2 and 4-hops with all switches.

discussed in Section 5.1, the PTP packet has 80 bytes, so the transmission delay using a GbE link can be calculated using (2.20). The transmission delay from one to four hops is summarized in Table 5.1. For simplicity, we can consider the propagation delay as zero, as it has a negligible contribution due to the small length of the Ethernet cables used in the testbed.

**Table 5.1:** Transmission delay considering a PTP frame with 80 bytes and a GbE interface.

Number of hops	1	2	3	4
PTP Transmission delay ( $\mu s$ )	0.64	1.28	1.92	2.56

**Table 5.2:** Average delays from switch #1 using port-based VLAN.

Number of hops	1	2	3	4
Delays in m-t-s ( $\mu s$ )	5.21	8.46	12.24	16.03
Delay in s-t-m ( $\mu s$ )	4.34	8.67	12.46	16.75
Asymmetry (ns)	865.46	-218.25	-219.42	-721.37

Thus, if we subtract the constant transmission delay, summarized in Table 5.1, from the total delay experienced in each direction, summarized in Table 5.2 and Table 5.3, the delay left is the sum of the processing and PHY latencies. If we consider that the PHY latencies are constant and calculate the difference between the hops, for the experiments with switch #1 and



**Table 5.3:** Average delays from switch #2 using port-based VLAN.

Number of hops	1	2	3	4
Delays in m-t-s ( $\mu s$ )	4.92	8.83	12.41	16.35
Delay in s-t-m ( $\mu s$ )	4.32	8.07	12.01	15.89
Asymmetry (ns)	594.97	765.87	407.68	456.85

#2, we can conclude that each hop is adding approximately a processing delay of  $3.3 \mu s$  and  $3.2 \mu s$  on average, respectively.

However, one thing to note is that if the processing delay of each hop is modeled as an independent random variable, both the mean and the variance must scale in a chain of hops. This is because the total delay would be the sum of the random variables. In theory, using the central limit theorem, if we consider the processing delay at each hop as uniformly distributed, as we start to add more hops, the sum of the independent random variables will tend to a Gaussian, with its mean being the sum of the processing delay at each hop and its variance being the sum of the variance of the processing delay of each hop. However, this behavior is not observed in the delays from Fig. 5.1. Note that, only the mean scales with the number of hops, and the variance stays the same, which indicates the switch may be processing the frame more efficiently when the hops consist of internal VLANs.

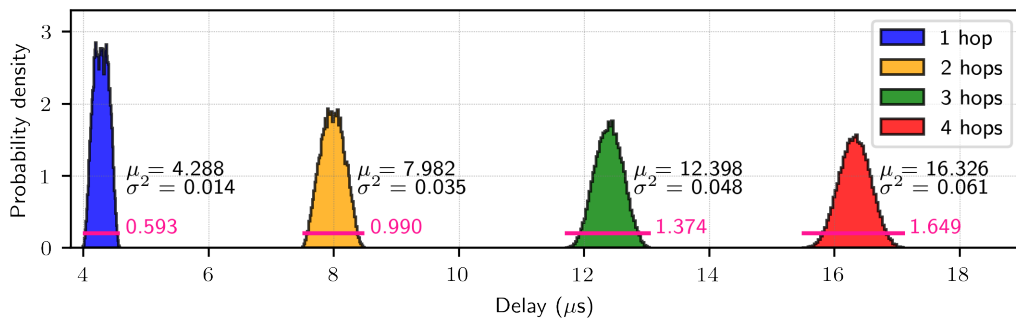
Fig. 5.2 shows the delay distribution when using multiple physical switches and the average delay and asymmetry are summarized in Table 5.4. Note that, compared to the results from Fig. 5.1, the average delays in all hop configurations stay roughly the same. However, the delay distribution completely changes in a configuration with more than one physical hop. Here, the central limit theorem applies and at each hop configuration, we can see that the final distribution is the sum of the random processing delays. For instance, Fig. 5.3 shows the statistics (mean, SD and span) from the delay distribution in the m-t-s direction. Note that, both mean and SD scales with the number of hops.

Therefore, we can conclude that the switches used in this work, when configured with the port-based VLAN setup, are smart enough to detect that the same data is being transmitted between isolated VLANs and the processing step that has a variable latency only happens once.

In summary, the number of hops, in experiments using VLAN-based switches, only increases the mean delay experienced by the PTP packets. The dispersion of the packets stays the

**Table 5.4:** Delays using multiples physical switches.

Number of hops	1	2	3	4
Average delay in m-t-s ( $\mu$ s)	4.89	8.95	12.86	16.83
Average delay in s-t-m ( $\mu$ s)	4.29	7.98	12.40	16.33
Average asymmetry (ns)	606.86	966.50	464.38	504.28

**Figure 5.3:** Delay statistics in m-t-s direction using multiple physical switches.

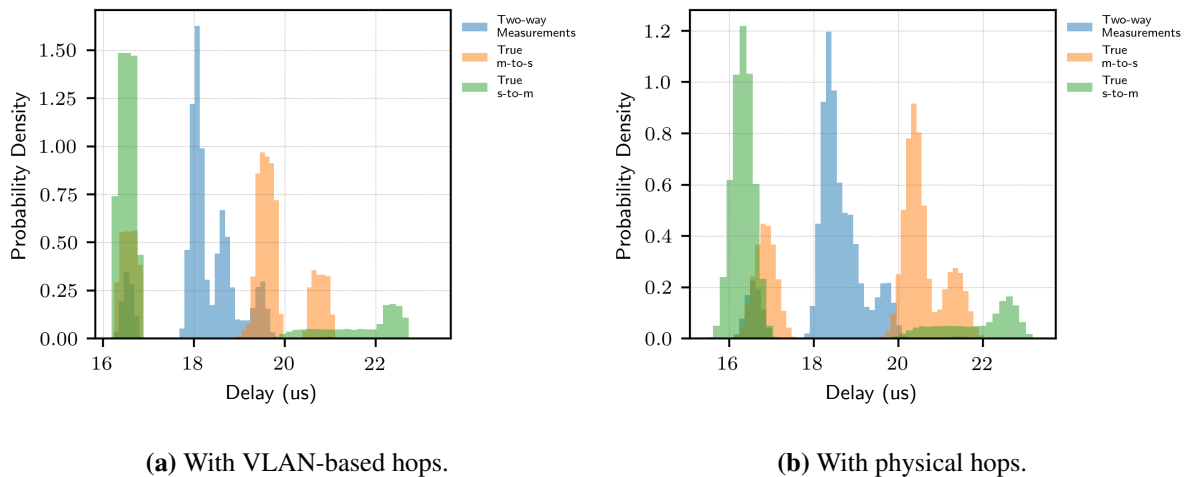
same because the processing step that has a variable latency only seems to happen once. Thus, in these scenarios, the processing delay has distribution close to the uniform independent of the number of hops. On the other hand, when using multiple physical switches, the number of hops contributes to the mean and also the variance of the total delay experienced by the PTP packets. Thus, in this scenario, the number of hops contributes to the PDV, and as the number of hops increases, the PDV also increases. The processing delay in this case will have a Gaussian distribution, where the mean and variance will depend on the number of hops.

## 5.2.2 PTP with FH Traffic

Fig. 5.4 shows the delay distribution when PTP shares the network with TDD FH traffic. Comparing with Fig. 5.1 and Fig. 5.2, note that the PDV is substantially higher in Fig. 5.4. This is caused by the high competition between the PTP packets and the FH packets for the same outbound link, which causes a high queueing delay in this scenario.

Before proceeding with the analysis of the delay distribution shape when PTP shares the network with TDD FH traffic, first, the impact of the processing delay is presented. Fig. 5.4a shows the distribution in a port-based VLAN configuration and Fig. 5.4b shows the distributing

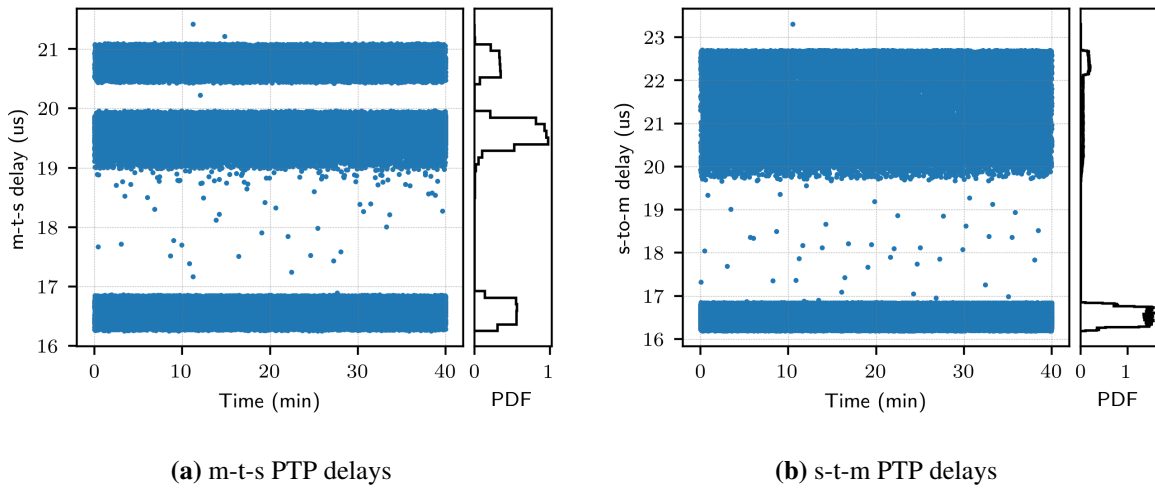
when using multiple physical switches. In both figures, a four-hop configuration is used. Note that the peaks on the delay distribution in both experiments are similar. However, if we look at the individual components of the distribution, in the experiment with multiple switches, as the processing delay has a more Gaussian shape, the individual components also have a more Gaussian shape. More specifically, if we compare the delays over time experienced by the PTP packets in Fig. 5.5 and Fig. 5.8, the maximum and minimum delays are more well-behaved when using the VLAN-based setup because all the hops together are introducing a uniformly distributed processing delay. Thus, the impact from the queuing delay determines the delay distribution when PTP shares the network with FH traffic. However, the processing delay also needs to be considered as it adds a random component contributing to the final PDV.



**Figure 5.4:** Delay distribution with FH traffic.

Fig. 5.5a shows the delay distribution in the m-t-s direction using the VLAN-based setup. Note that there are three levels of delay. The first level between 16  $\mu\text{s}$  and 17  $\mu\text{s}$ , the second between 19  $\mu\text{s}$  and 20  $\mu\text{s}$  and the third above 20  $\mu\text{s}$ . The first level is similar to the delay distribution when only PTP is being transmitted on the network, in a 4-hop configuration, as shown in Fig. 5.2. The reason for this is the behavior of the TDD system. In the time slots that the FH are not used for transmitting radio data in the downlink direction, the PTP packets in the downlink traverse the network without experiencing any queuing delay. On the other hand, the second and third levels are the ones when PTP suffers queuing delay, i.e., when PTP is transmitted with the TDD FH packets.

The queuing delay introduced by the FH packets comes from the store-and-forward mech-



**Figure 5.5:** Delay distribution and PDF using VLAN-based hops.

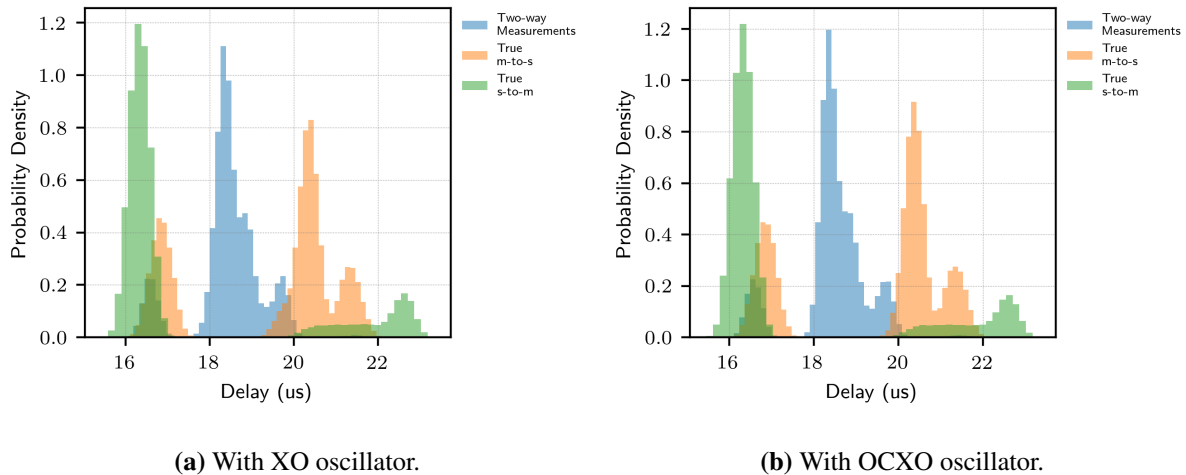
anism on the network switches. For instance, consider that the FH packet and PTP have a transmission delay of  $t_{fh}$  and  $t_{ptp}$ , respectively. In this case, if the PTP packet is transmitted right after the FH packet, i.e. separately only by the interpacket gap  $t_{ipg}$ , by the time the PTP packet is completely stored on the input switch port, the FH packet can still be in the outgoing port (being transmitted). Thus, the PTP packet has to wait before it can be transmitted. More specifically, this can happen when  $t_{fh}$  is greater than  $t_{ptp}$ , which is true most of the time. In this case, the worst-case queuing delay can be calculated by considering that the PTP packet will have to wait for  $(t_{fh} - t_{ptp})$  on every hop.

As discussed in Section 5.1, the FH packets are sent as unicast-addressed messages. Thus, the PTP packets on the m-t-s direction can depart behind a unicast FH packet sent to slave #1 or behind a packet sent to slave #2. In the case where it goes behind an FH packet sent to the slave where the acquisition runs (slave #1), it experiences an extra queuing delay on the last hop, namely  $(t_{fh} - t_{ptp})$ . In this situation, the expected delay lies on the third delay level shown in Fig. 5.5a. In contrast, when it goes behind the FH packet set to the other slave, it does not experience an extra queuing delay on the aggregation node, given that the FH packet is forwarded through another port. Thus, the expected delay lies on the second delay level.

By comparing the delays experienced by the packets in the s-t-m direction to the delays in the m-t-s, in Fig. 5.6, note that the s-t-m direction experiences higher delays. This is caused by the aggregation node present in the adopted tree topology (see Fig. 4.1). The PTP and FH packets from both slaves compete for the single outbound port to the master, which causes

an extra delay. On the other hand, the packets in the m-t-s direction experience lower delays because they do not experience contention.

Similarly to the distribution on the m-t-s direction, the distribution on the s-t-m have different levels of delay. Fig. 5.5b shows two levels. The first level, between  $16\mu\text{s}$  and  $17\mu\text{s}$ , is the delay experienced by the PTP packets when traversing the network without queueing delay, which occurs when the PTP packets are transmitted in the time slots that the FH is not being used by the TDD system in the UL direction. The second level, above  $17\mu\text{s}$ , happens when PTP packets suffer queueing delay. The peak of the second level happens when there is a collision on the aggregation node between two FH packets and the PTP packet has to wait for the two FH packets, one from each RRU.

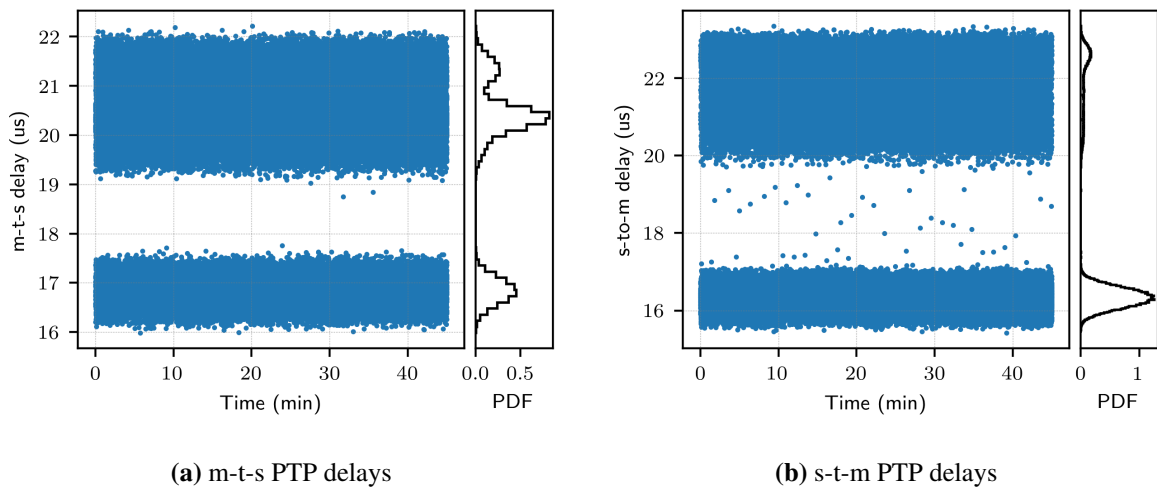


**Figure 5.6:** Delay distribution with FH traffic.

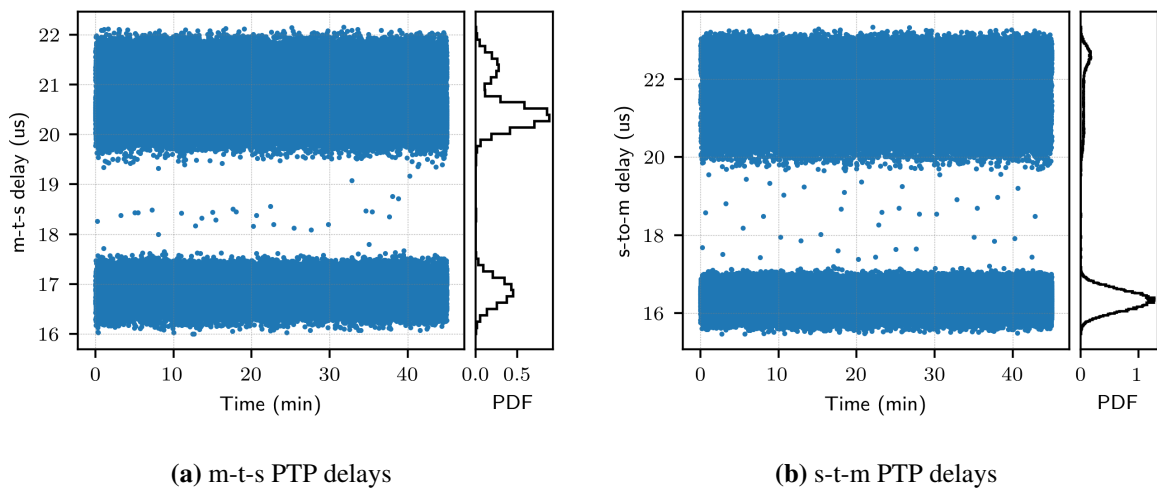
The distribution in the experiments with XO and OCXO shown in Fig. 5.6a and Fig. 5.6b are very similar. This is because the mechanism discussed in Section 3.1 is being used to control the FH streams from all devices (BBU and RRUs) synchronously. In the end, we can expect to have the same distribution in the experiments with the same characteristics independent from the oscillator used.

Fig. 5.7 and Fig. 5.8 are the delays over time experienced in the experiments with XO and OCXO, and multiples physical switches. The discussion about the distribution shape is similar to the one given for Fig. 5.5. All the levels in the m-t-s and s-t-m have the same explanation, the only thing that changes is the shape of the individual components. Instead of well-behaved and uniformly distributed levels, as shown in Fig. 5.5, the experienced delays when using multiples

switches have a more Gaussian shape, with tapered probabilities on the extremes (maximum and minimum delays).



**Figure 5.7:** Delay distribution and PDF using XO oscillator.



**Figure 5.8:** Delay distribution and PDF using OCXO oscillator.

### 5.3 TDD Packet Classifier Experiments

This section presents the results from the two methods described in Section 3.2, to classify the PTP packets into good and bad packets. The good packets are the ones that traverse the network without suffering any queuing delay and the bad packets are the ones that suffer queuing.

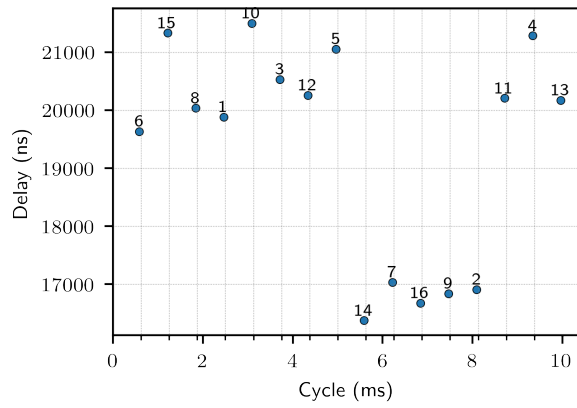
Section 5.3.1 presents a discussion regarding the method based on the modulo operator, which is useful to understand how the delay from PTP packets are impacted based on the TDD cycle location. Finally, Section 5.3.2 evaluates the performance of the K-means-based classifier.

### 5.3.1 PTP Cycle location

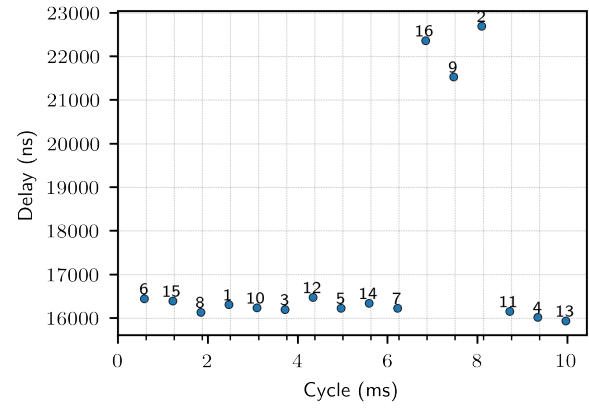
As discussed in Section 5.1, the experiments presented in this chapter use a TDD cycle as illustrated in Fig. 3.3, where the downlink is active for 7 ms, the gap has 1 ms and the uplink has 2 ms. The experiments use 64 PTP exchanges per second. Thus, we expect to have a PTP configuration as described in Fig. 3.6d, where, in a window of 16 samples, 11 samples are located at the downlink time slot, 1 sample at the gap and 3 samples at the uplink. If the PTP starts to be transmitted aligned with the FH cycle, we would also expect to have the 13th sample located in the gap, the 6th, 15th and 8th samples located in the uplink and all the other samples are located in the downlink time slot.

Fig. 5.9 shows the packet delay over the TDD cycle location calculated using the modular operation with a different number of samples. Thus, the horizontal axis shows the PTP location calculated with equation (3.2) and the vertical axis shows the true packet delay calculated by PTP-DAL using equations (4.4) and (4.5) for the m-t-s and s-t-m directions respectively. Recall from Section 3.2.1 that if the PTP packets are transmitted in the m-t-s direction, the samples located at the TDD downlink slot have a high probability of experiencing high delay caused by the queueing delay. On the other hand, the samples located at the gap and uplink slots have a high probability of experiencing zero queueing delays, resulting in a low packet delay. Similarly, the PTP packets transmitted in the s-t-m direction have a high probability of experiencing high delays only when transmitted in the uplink slot.

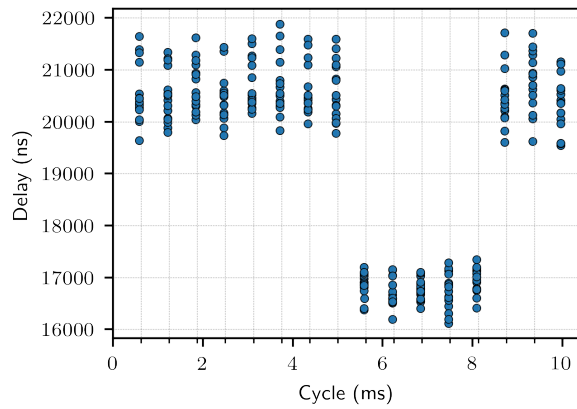
A window for 16 samples is shown in Fig. 5.9a and Fig. 5.9b. Note that the location of the samples does not match with Fig. 3.6d. The reason is that the PTP stream and the FH stream do not start at the same moment, which means that the first PTP sample is not transmitted at 0 ms. Consequently, there is a shift in the position of PTP samples in relation to the FH cycle. Furthermore, Fig. 5.9b shows that in a window of 16 samples in the s-t-m direction, 3 samples appear with a high packet delay, which matches with the theoretical analysis presented in Fig. 3.6d. However, Fig. 5.9a shows that in this window of 16 samples in the m-t-s direction, 5 samples experienced lower packet delay instead of 4 as shown in 3.6d. This happens because the samples close to the boundary between the TDD slots can be located at the next time slot



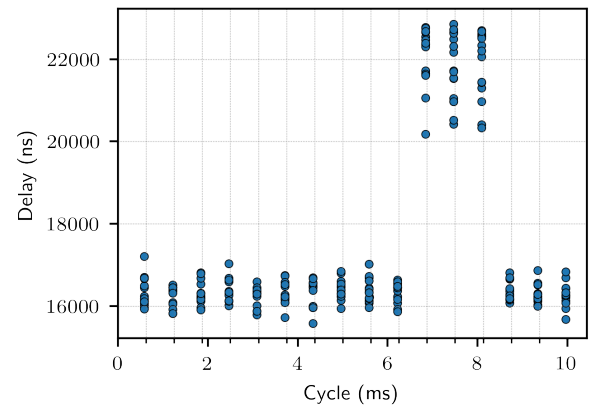
(a) m-t-s direction with 16 packets per second.



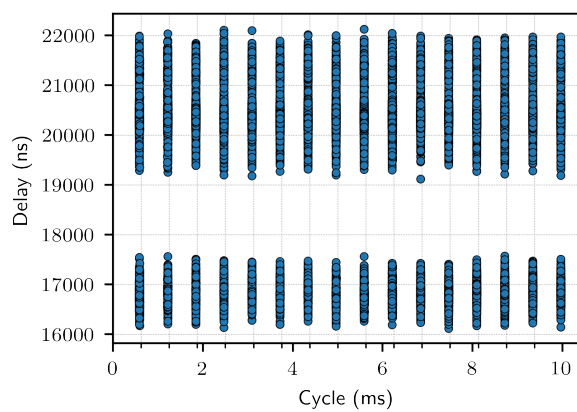
(b) s-t-m direction with 16 packets per second.



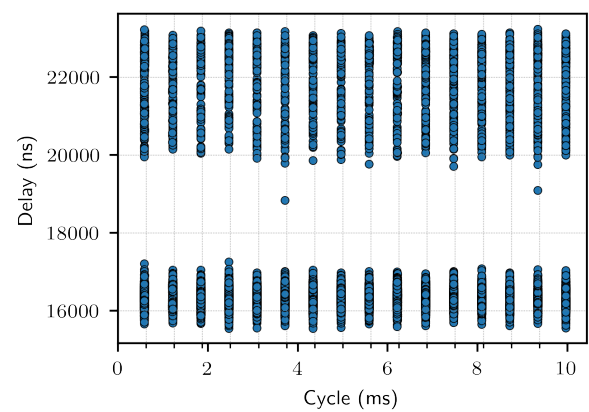
(c) m-t-s direction with 256 packets per second.



(d) s-t-m direction with 256 packets per second.



(e) m-t-s direction with 16384 packets per second.



(f) s-t-m direction with 16384 packets per second.

**Figure 5.9:** PTP cycle location in the experiment with the XO oscillator.



depending on the moment the PTP stream starts. For instance, in Fig. 3.6d, if the first PTP is transmitted at 0.2 ms, all the other samples would have a shift of 0.2 ms to the right, thus, the 4th sample would be located at the gap slot. In this situation, there would be 5 samples with low packet delay. In this case, the PTP samples on the uplink would continue to have only 3 samples with high delay.

Fig. 5.9c and Fig. 5.9d show the same plot with 256 samples. Note that, we could classify the good and bad packets based on the cycle location. For instance, the good samples on the downlink would be the ones located between 6 and 8 ms, and the ones located between 7 and 8 ms for the uplink. However, as time passes, and we increase the number of samples on the plot, the samples start to move to the right. In the end, becomes difficult to differentiate the good and bad packets based on their location as shown in Fig. 5.9e and Fig. 5.9f. This error accumulated over time is primarily related to the fact that the size of the TDD cycle, on the implementation described in Section 4.4, is not exactly 10 ms, it has an error close to 648 ns. Thus, over time this error accumulates and the samples start jumping positions.

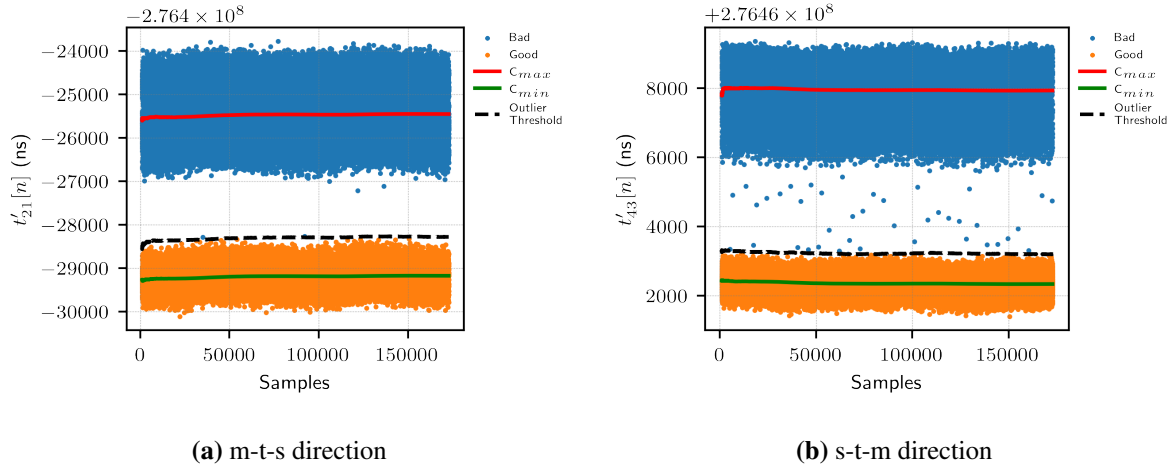
The analysis of the error associated with the TDD cycle is beyond the scope of this work. Therefore, this method can be used to classify the packets over a small window of samples. In the next section, the results from the K-means-based classifier are presented.

### 5.3.2 K-means Based Classifier

Next, the goal is to evaluate the performance of the K-means-based classifier discussed in Section 3.2.2. Fig. 5.10 shows the classification process, where the figure shows the level from the two clusters, denoted as  $C_{max}$  and  $C_{min}$ , the outlier threshold and finally how each sample is classified. Fig. 5.10a shows the classification process in the m-t-s direction and Fig. 5.10b shows the classification process in the s-t-m direction in the experiment with XO oscillator. The result with the OCXO oscillator is very similar so it is omitted from the text.

Note that in both cases, visually, the method has been able to classify the packets in the two clusters correctly. Besides, the proposed way to filter the outliers from the good packet cluster by relying on the standard deviation also works as expected. However, the testbed has no programmatic way to confirm that a packet traverses the network without suffering any queuing delay. So, to assert the classifier performance, we use the true delays calculated by PTP-DAL and the information about the true delay distribution presented in Section 5.2.1 and Section 5.2.2. Thus, by knowing the delay distribution experienced by the PTP packets when

they do not share the network with FH traffic in a configuration with 4 hops, as shown in Fig. 5.3, we can confirm that the method can classify the packets correctly with an accuracy of 99%.



**Figure 5.10:** K-means classification in the experiment with XO.

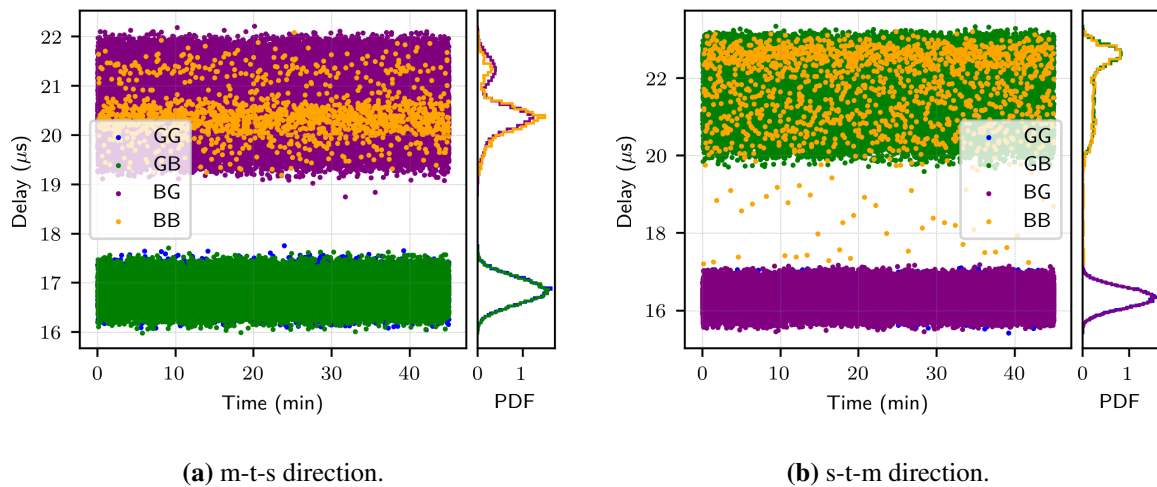
True label	GG	100.000	0.000	0.000	0.000
	GB	0.003	99.997	0.000	0.000
	BG	0.000	0.000	100.000	0.000
	BB	0.000	0.000	0.000	100.000
		GG	GB	BG	BB
		Predicted label			

**Figure 5.11:** Confusion matrix in the experiment with XO.

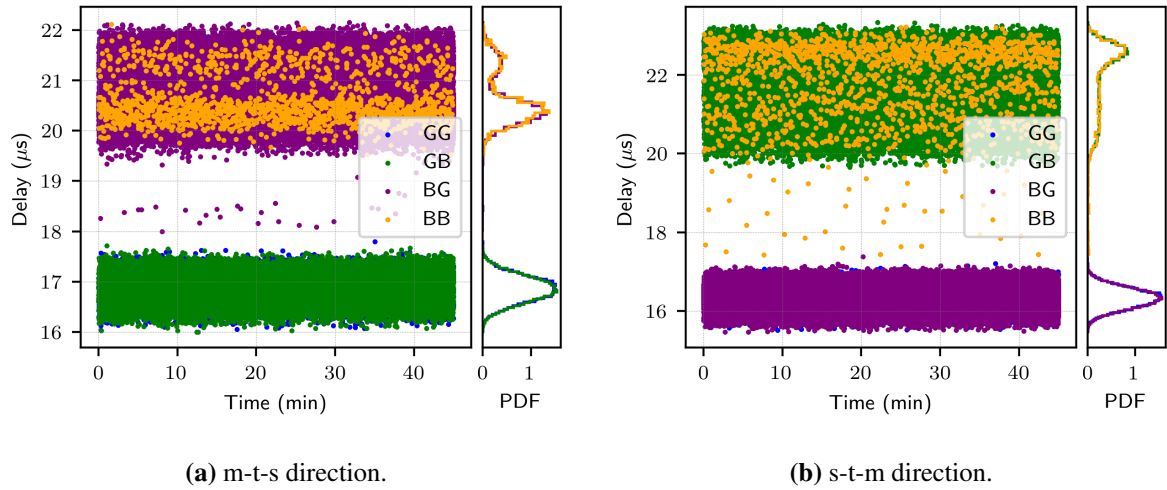
Additionally, instead of classifying the packets individually in each direction, we could classify a complete PTP exchange in four classes "GG", "GB", "BG" and "GG". Where the first letter is related to the classification of the *Sync* message, and the second is for the classification of the *Delay-Req* message. Fig. 5.11 shows a confusion matrix, which provides the model performance in terms of the percentage of correctly and incorrectly predicted samples for each class. Note that, the model correctly predicted the "GG", "BG" and "BB" classes. The model errors were concentrated on the "GB" class.

Additionally, Fig. 5.12 and Fig. 5.13 show the final classification for both experiments, with XO and OCXO oscillators. Fig. 5.14 presents a bar plot with the probability of each class calculated with the output from the K-means classifier.

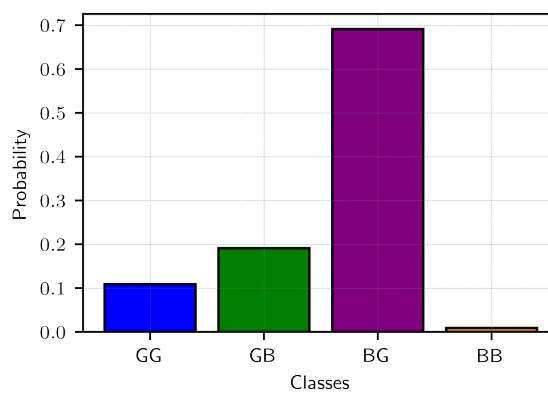
Note that the class "GG" happens approximately 10% of the time. Similarly, the classes "GB", "BG" and "BB" happens approximately 20%, 69% and 1%, respectively. This is expected given the characteristics of the TDD cycle configuration. For instance, as the traffic in the downlink will be active 70% of the cycle (7 ms), the PTP packets transmitted in this direction will be affected, and consequently, these packets will be classified as bad. Meanwhile, the PTP packets transmitted in the uplink direction will experience zero queuing delay. Therefore, the PTP exchange with bad *Sync* and good *Delay-Req* (denoted as BG) are expected to happen approximately 70% of the time. The probability of the "GG" and "GB" classes can be explained similarly. Note that, the class "BB" occurred approximately 1% of the time. This means that the PTP packets in both directions suffered queuing delay, which happens mostly on the transitory delays as showed in Fig. 5.12 and Fig. 5.13.



**Figure 5.12:** TDD packet classification in the experiment with XO.



**Figure 5.13:** TDD packet classification in the experiment with OCXO.

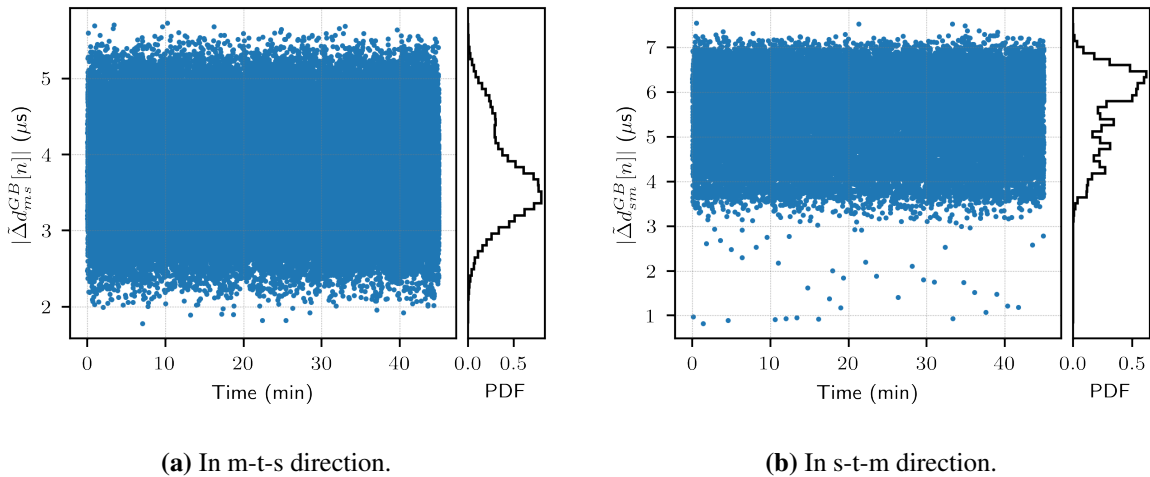


**Figure 5.14:** Bar plot with the probability of each class calculated with the output from the K-means classifier.

## 5.4 Queuing Induced Asymmetry Experiments

In this section, we evaluate the results from the proposed QIA mechanism presented in Section 3.4.2. The PTP packets are classified between good and bad based on the K-means classifier presented in the last section. Thus, the QIA mechanism uses the packet classification as presented in Fig. 5.12, for the experiment with XO, and Fig. 5.13, for the experiment with the OCXO oscillator.

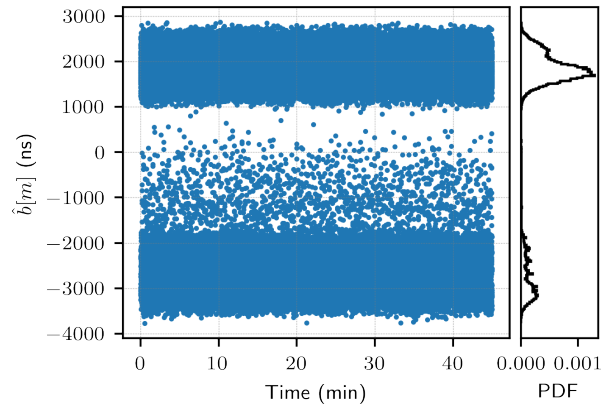
The independent estimates,  $\tilde{\Delta}d_{ms}^{GB}[n]$  and  $\tilde{\Delta}d_{sm}^{GB}[n]$ , calculated using (3.30) are showed in Fig. 5.15a and Fig. 5.15b. Fig. 5.16 shows the calculated bias with (3.31). Recall that the objective is to use the estimates from (3.31) to compensate for the bias on the time offset measurements from (2.16).



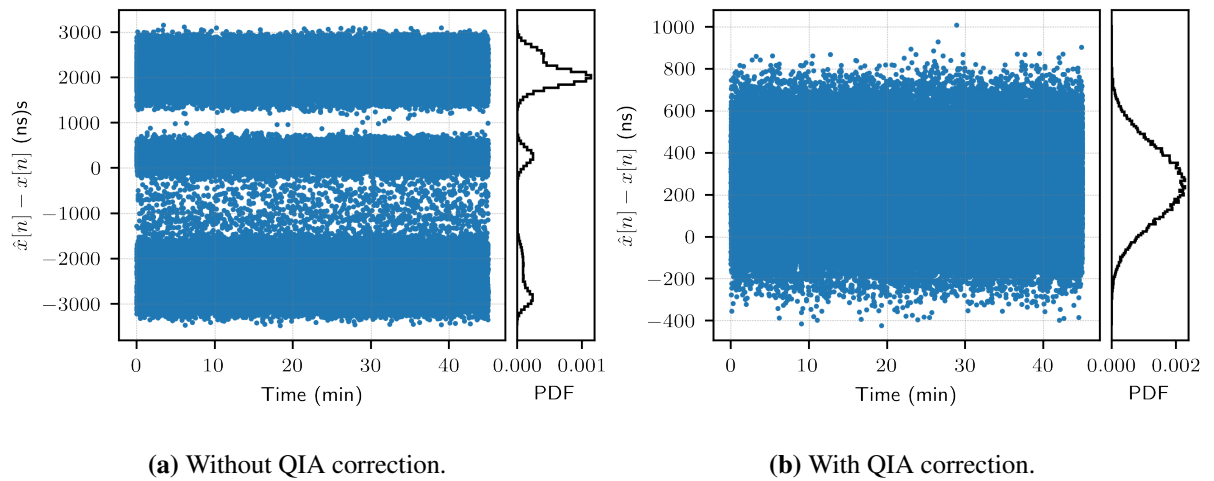
**Figure 5.15:** Estimated delay difference in the experiment with XO oscillator.

Fig. 5.17a shows the error of the time offset measurements calculated with (2.16). Those errors are mostly from the delay asymmetry, as discussed in Section 5.2. Comparing those errors with the estimated bias from Fig. 5.16, we can see that the delay distribution on the higher and lower levels matches very well with the levels shown in Fig. 5.16. The only level that does not appear in Fig. 5.16 is the one close to 0. This is expected because this level is associated with the packets that are transmitted without queuing delay. Hence, the queuing delay is zero for those packets.

Fig. 5.17b shows the time offset measurement error after applying the QIA correction. The correction is made using (3.32). We can see that the residual bias affecting the time offset



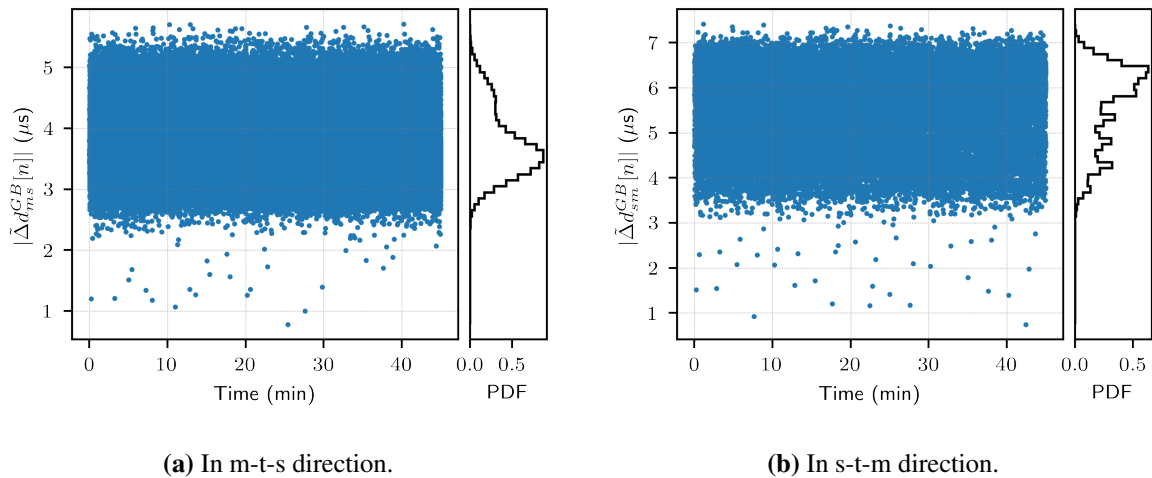
**Figure 5.16:** Estimated QIA delay in the experiment with XO.



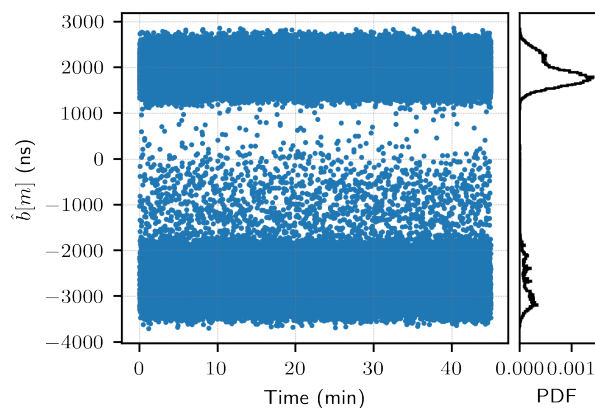
**Figure 5.17:** Time offset measurements error in the experiment with XO.

measurement is the bias that exists when the PTP is transmitted without FH traffic as showed in Section 5.2.1.

Fig. 5.18a and Fig. 5.18b show the estimated delay differences for the experiment with the OCXO oscillator. As the delay distribution on the XO and OCXO experiments does not change, the results are very similar. Fig. 5.19 shows the estimated QIA. Finally, Fig. 5.20a shows the time offset error from the PTP measurements before the QIA correction and Fig. 5.20b shows the error after correction.

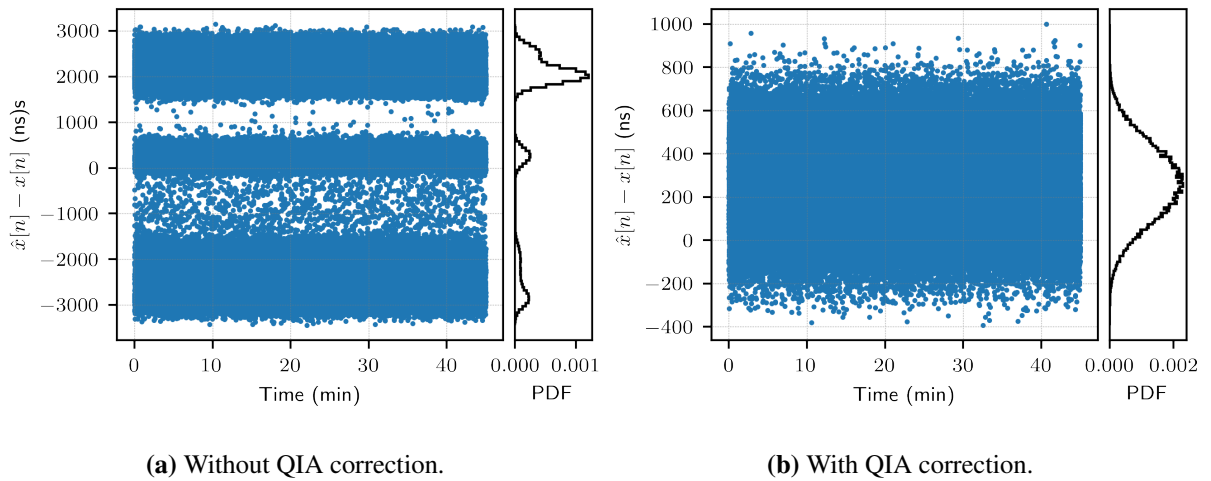


**Figure 5.18:** Estimated delay difference in the experiment with OCXO oscillator.



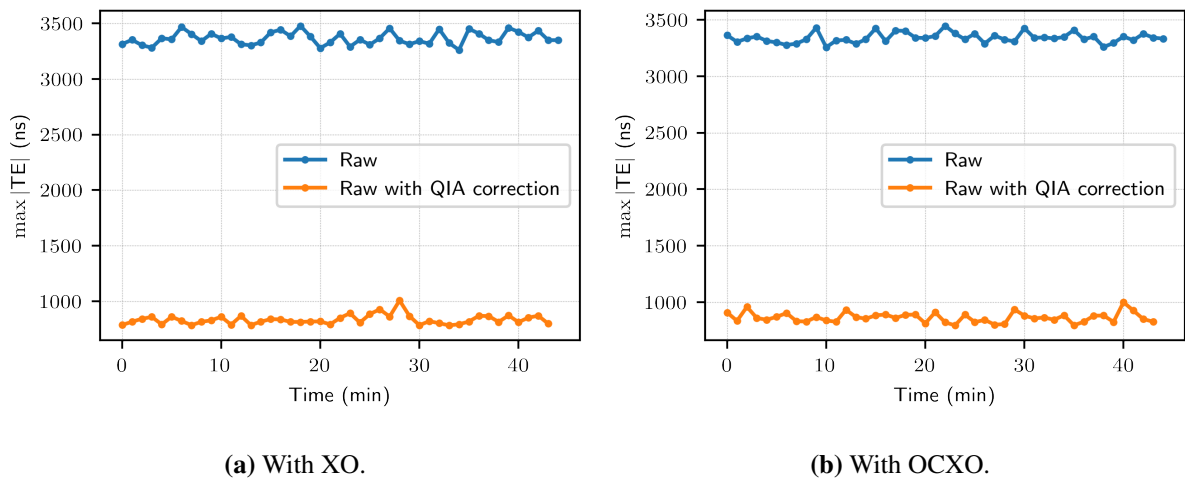
**Figure 5.19:** Estimated QIA in the experiment with OCXO oscillator.

Fig. 5.21 present the  $\max|\text{TE}|$  results with and without the proposed QIA mechanism for both oscillators. In both experiments, there is a gain on the PTP performance by correcting the



**Figure 5.20:** Time offset measurement error in the experiment with OCXO oscillator.

queuing delay affecting the PTP packets that are transmitted with FH traffic. More specifically, the  $\max|\text{TE}|$  without correction presents on average 3364.23 ns, and after correcting the QIA, the  $\max|\text{TE}|$  goes to 835.95 ns on the experiment with XO. In the experiment with OCXO, the  $\max|\text{TE}|$  without correction presented on average 3340.55 ns and 863.54 ns with correction. Therefore, by using the proposed method, we obtained an improvement of approximately 75% in both cases. The analysis of using the proposed method to correct the QIA delay as a preprocessing step for the time offset estimate algorithms is postponed to the next section.



**Figure 5.21:**  $\max|\text{TE}|$  results from the experiments with four physical hops.



## 5.5 Synchronization for TDD FH Experiments

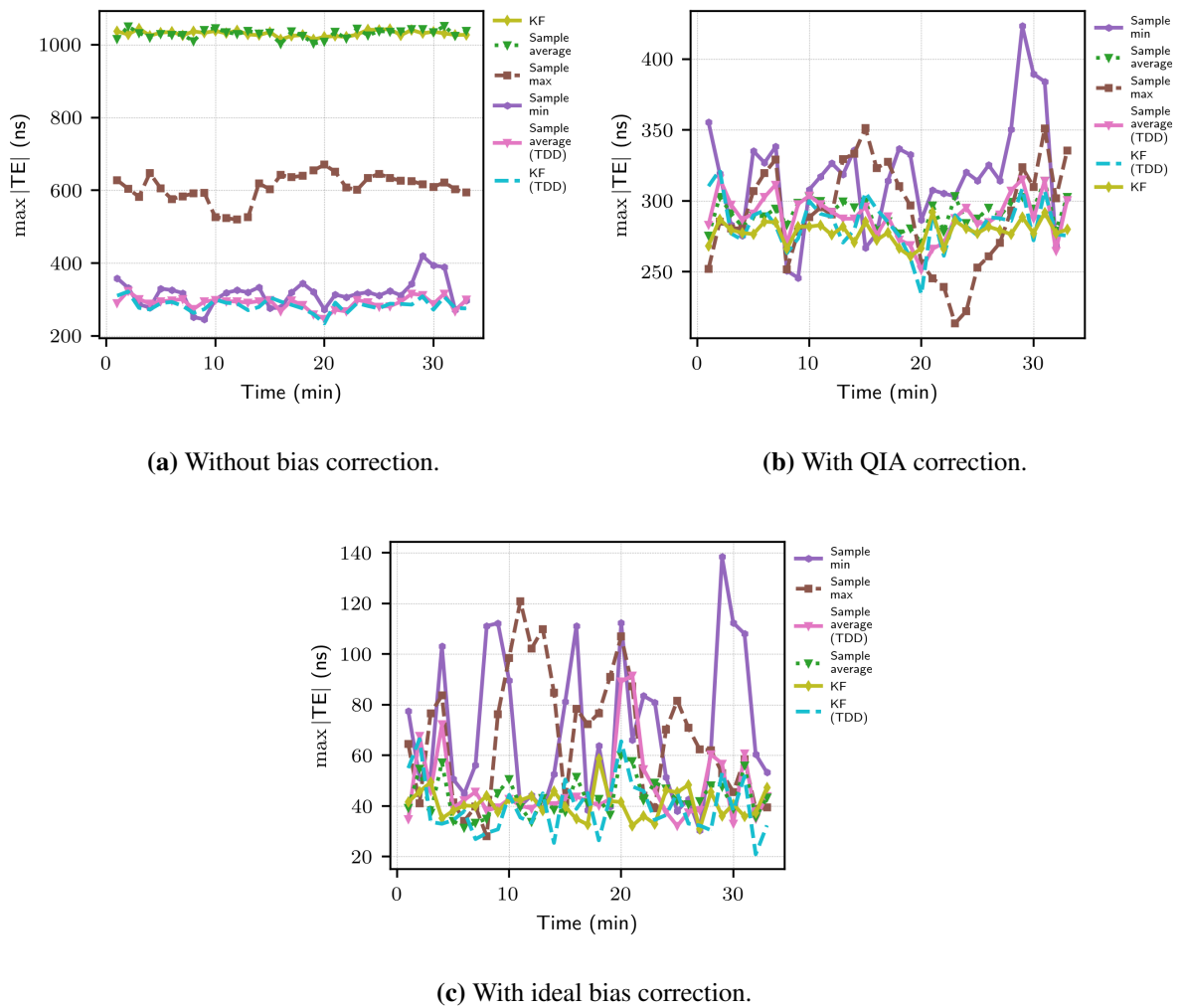
The objective of this section is to analyze the  $\max|\text{TE}|$  performance of the algorithms presented in Section 3.3 when PTP traffic is sharing the network with the TDD FH traffic, as discussed in Section 5.2.2. More specifically, this section presents an analysis in terms of the  $\max|\text{TE}|$  performance with and without bias correction, using the PTP-DAL correction as discussed in Section 4.6. Besides, the performance using the QIA corrections from Section 5.4 is also presented. Furthermore, this section presents the results from the experiments with XO and OCXO oscillators.

Fig. 5.22 shows the  $\max|\text{TE}|$  results from the experiment using the XO oscillator and four physical hops. Fig. 5.22a shows the  $\max|\text{TE}|$  over time without bias correction, Fig. 5.22b shows the  $\max|\text{TE}|$  after applying the QIA corrections presented in Fig. 5.16, and Fig. 5.22c shows the results with the ideal bias correction. To help with the analysis, Fig. 5.23 shows the violin plot version from the results shown in Fig. 5.22, which shows the minimum, maximum, average and shape of the  $\max|\text{TE}|$  results. The raw PTP measurements were omitted to improve the visualization of all figures. However, as shown in Fig. 5.21, the  $\max|\text{TE}|$  from the raw PTP measurements, calculated with (2.16), were 3364.23 ns and 835.95 ns for the experiments without bias correction and with QIA corrections respectively.

In Fig. 5.22a, note that the algorithms converged to the delay asymmetry as summarized in Table 5.5. The ones that use only the good TDD packets converged to the delay asymmetry affecting the good packets as summarized in Table 5.6. Because of that, the standard KF and the sample average are the two algorithms with the biggest  $\max|\text{TE}|$  results, close to 1030 ns, which is close to the average asymmetry summarized in Table 5.5. On the other hand, the TDD version of both algorithms, which only uses the packets classified as good to estimate the time offset, "KF (TDD)" and "sample average (TDD)", presented the best  $\max|\text{TE}|$  results, close to 290 ns, which is close to the average asymmetry summarized in Table 5.6. These results show that the estimated values of all algorithms tend to the inherent bias due to delay asymmetry, which indicates that the algorithms have a low variance.

The bias introduced by the delay asymmetry in a PTP-unaware network is one of the biggest sources of error and each algorithm will have a specific bias depending on the operation it performs on the packets. Therefore, in this scenario, the performance of the algorithm will heavily depend on the associated asymmetry.

Fig. 5.22b shows the  $\max|\text{TE}|$  results using the proposed QIA correction mechanism. By



**Figure 5.22:**  $\max|TE|$  results from the experiment with four physical hops and XO oscillator.

**Table 5.5:** Ideal delay asymmetry from the experiment with XO oscillator calculated by PTP-DAL.

Operator	Minimum	Average	Maximum
Asymmetry (ns)	281	995.17	-563.50

**Table 5.6:** Static delay asymmetry from good packets from the experiment with the XO oscillator.

Operator	Minimum	Average	Maximum
Asymmetry (ns)	315	255.31	328.5

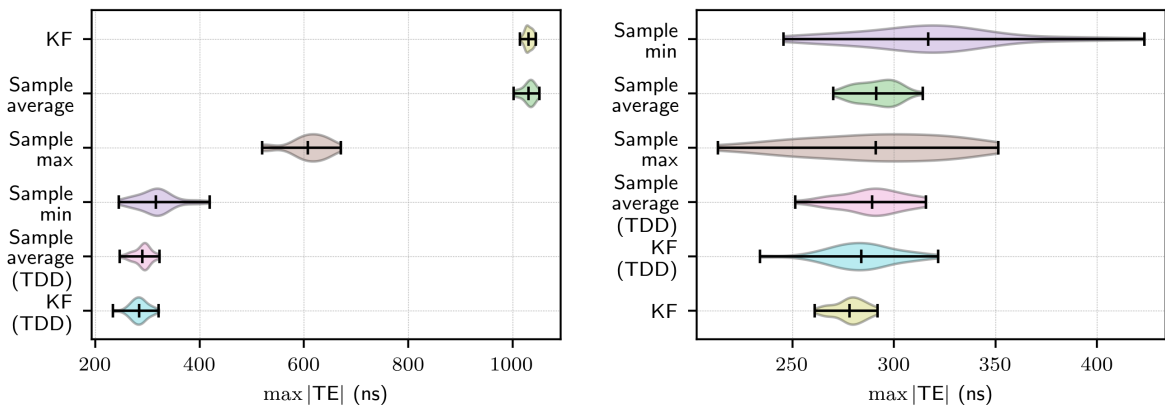
correcting the QIA bias, the performance of the algorithms that use the packets affected by queuing delay improved substantially. In this scenario, the algorithms use the corrected version from the PTP measurements from (3.32) or the compensated timestamp difference from (3.33). For example, KF, sample average and sample maximum are the algorithms most affected by bad packets. That is because, KF and sample average uses all the available data to estimate the time offset, and the sample maximum selects the packets that experienced the highest delays. Thus, these algorithms showed improved performance after applying the QIA corrections. For instance, KF and sample average presented a similar performance close to 1030 ns without any bias correction, and after correcting the QIA bias, the  $\max|TE|$  from KF and sample average decreased to approximately 278.1 ns and 291.3 ns, respectively. These results correspond to an improvement of approximately 70%. Besides, the sample maximum presented an improvement of 50%, going from an average  $\max|TE|$  of 607.3 ns to 291 ns.

On the other hand, the performance from the sample minimum and the algorithms that use only good packets (marked with the suffix "(TDD)") remained with the same performance in both scenarios. The reason is that the good packets are not affected by queuing delay. Thus, the algorithms that only use good packets do not have any bias associated with queueing delay. Besides, the sample minimum also does not benefit from the QIA correction. That is because the sample minimum naturally tries to select the good packets by using the packets with the minimum delay within a window to estimate the time offset. Therefore, comparing Fig. 5.22a and Fig. 5.22b, after applying the QIA corrections, all the algorithms present closer results.

Note that the two versions of KF and the sample average are the ones with better performance after the QIA correction. This can be related to the distribution from the error left on the raw PTP measurements after correcting the QIA, which, as shown in Fig. 5.17b, has a more Gaussian shape. Thus, the error left can be better filtered using KF, which is an optimal estimator in the minimum mean-square error (MSE) sense, and the sample average, which tends to the mean of the distribution in both directions. The proposed QIA method is very effective to correct the delay introduced by queuing delay, as shown in Section 5.4. However, after correcting the queuing, the PTP are still corrupted by all the other component that contributes to the delay asymmetry such as PHY latencies, network processing delay, and others. That is why the  $\max|TE|$  from the algorithms converged to a value between 278 ns and 316 ns.

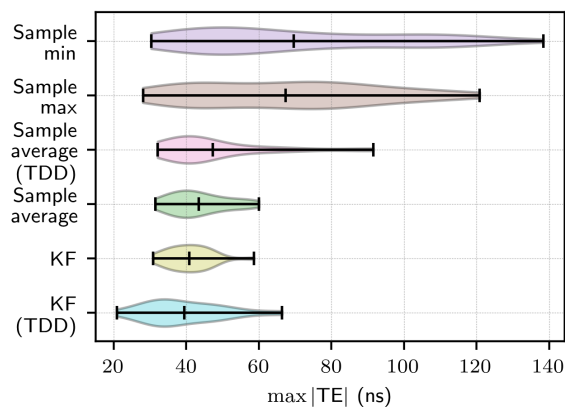
Fig. 5.22c shows the results with bias correction calculated by PTP-DAL. The ideal bias is calculated as discussed in Section 4.6. This type of correction is feasible in APTS scenarios,

where the PTP is used as a backup. Thus, the slave clock can use the primary method used for synchronization (e.g., GNSS) to calculate the bias present in the PTP measurements. Note that even in this scenario, the algorithm based on average (KF and sample mean) presents the best results, with a  $\max|TE|$  close to 40 ns, and the sample minimum and sample maximum present the two worst performances, with a  $\max|TE|$  approximately 70 ns. This can be explained by taking into consideration the delay distribution shown in Fig. 5.7. Note that, because the shape of the individual components resembles the Gaussian distribution, the minimum and maximum delays have fewer samples (and lower probability of being within a window) than the average ones. Thus, the consistency from the sample minimum and maximum is more significantly affected, which affects the time offset estimates.



(a) Without bias correction.

(b) With QIA correction.



(c) With ideal bias correction.

**Figure 5.23:**  $\max|TE|$  violin plots from the experiment with four physical hops and XO oscillator.

Fig. 5.24 presents the  $\max|TE|$  results from the experiment using the OCXO oscillator.

Fig. 5.24a shows the  $\max|\text{TE}|$  without any bias correction. Fig. 5.24b shows the results using the proposed QIA correction and Fig. 5.24c shows the results using the ideal bias correction. Finally, Fig. 5.25 shows the violin plot version to help with the visualization.

Note that similar to the results from the XO, the algorithms converged to the delay asymmetry summarized in Table 5.7 and Table 5.8. In the scenario without bias correction, shown in Fig. 5.24a, all the algorithms achieved very similar results in terms of  $\max|\text{TE}|$  compared with Fig. 5.22a. In this scenario, the algorithms that use only use good packets ("KF (TDD)" and "sample average (TDD)") and sample minimum, achieved the best performance in terms of  $\max|\text{TE}|$ , approximately 277.5 ns, 280.4 ns and 295 ns, respectively. While the normal version from KF, sample average and sample maximum achieved an average  $\max|\text{TE}|$  of 1036.88 ns, 1035.38 ns and 598.56 ns, respectively.

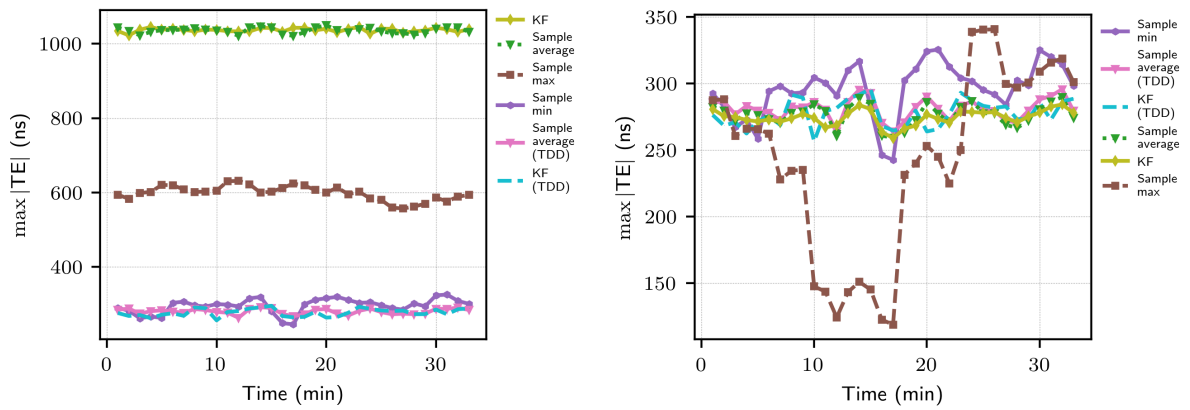
In the results with QIA correction, the algorithms converged to the asymmetry of the good packets (residual asymmetry after correction of the queuing delay) similar to the case with XO. The increase in performance is also similar, with approximately 70% for KF and sample average and 50% for sample maximum. Note that, in Fig. 5.25b, sample maximum is the algorithm with the minimum  $\max|\text{TE}|$  on average. However, it also presents a big variance. Thus, we can not consider sample maximum as the best algorithm in this case. Actually, sample maximum is sensitive to the outliers present in the difference of the timestamp after the correction of the QIA delay, using (3.33). Thus, the performance can be affected negatively or positively in an unpredictable way.

The main difference is in the results with ideal bias correction. After correcting the error due to the network delays, the achieved  $\max|\text{TE}|$  performance was better because the OCXO has better frequency stability than the XO. Thus, both versions from KF and sample average showed an average  $\max|\text{TE}|$  of approximately 20 ns. The sample minimum and sample maximum achieved a  $\max|\text{TE}|$  of 37.33 ns and 31.30 ns, respectively.

**Table 5.7:** Ideal delay asymmetry from the experiment with OCXO oscillator.

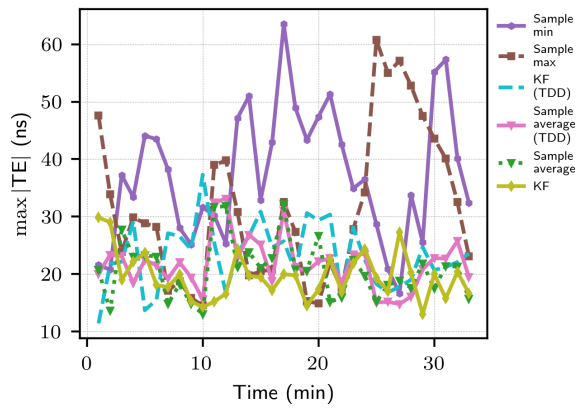
Operator	Minimum	Average	Maximum
Asymmetry (ns)	268.50	1022.93	-591.50

Recall from Section 3.3.3 that the formulation from packet filtering algorithms used in this work uses an adjusted version of the timestamp differences calculated with (3.3). The time



(a) Without bias correction.

(b) With QIA correction.

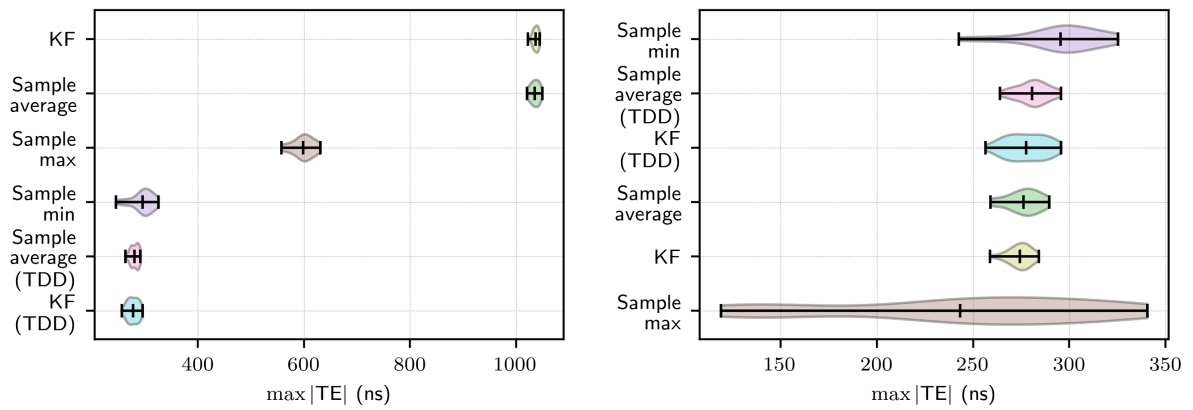


(c) With ideal bias correction.

**Figure 5.24:**  $\max|TE|$  results from the experiment with four physical hops and OCXO oscillator.

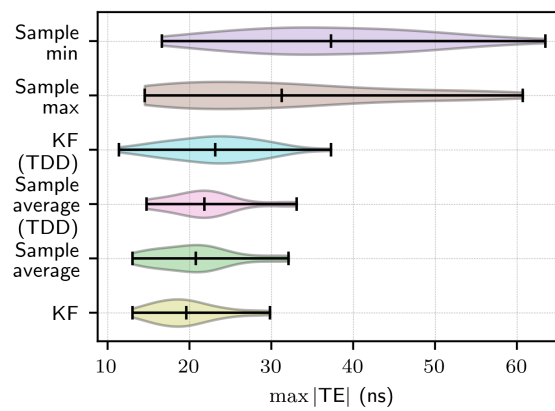
**Table 5.8:** Static delay asymmetry from good packets from the experiment with OCXO oscillator.

Operator	Minimum	Average	Maximum
Asymmetry (ns)	258.5	264.83	294



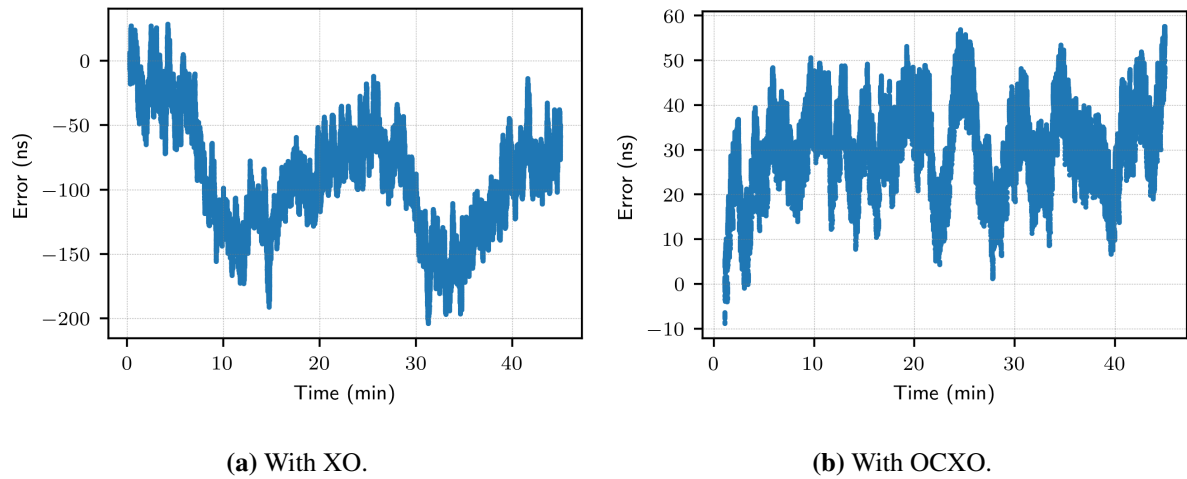
(a) Without bias correction.

(b) With QIA correction.



(c) With ideal bias correction.

**Figure 5.25:**  $\max|TE|$  violin plots from the experiment with four physical hops and OCXO oscillator.



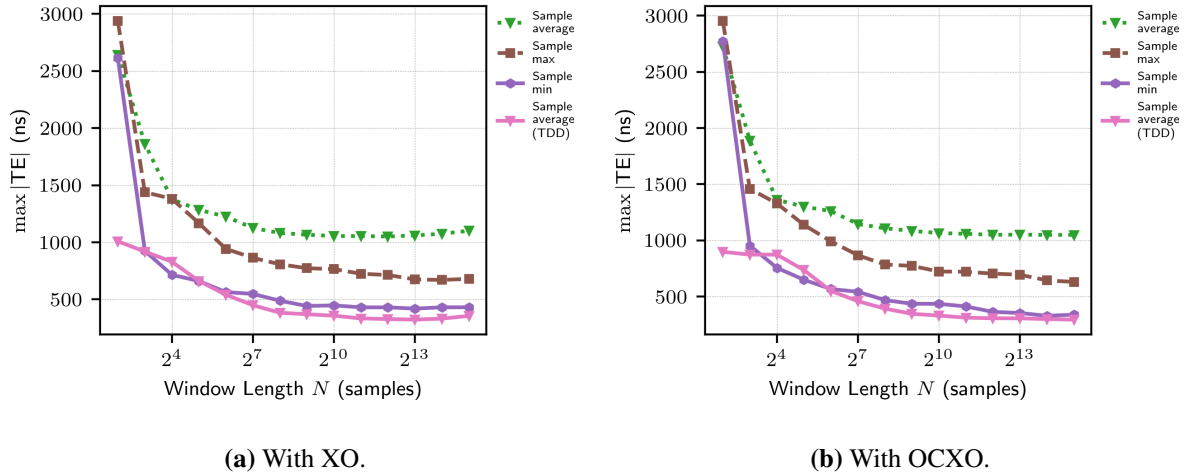
**Figure 5.26:** Cumulative drift estimation error on the experiment with four physical hops.

offset drift is used to enable large window lengths on the packet filtering algorithms. Fig. 5.26 shows the cumulative error between drift estimations and the true drift. Fig. 5.26a shows the results in the experiment with the XO, using  $W = 8$  and  $N = 1024$  in (3.21), which were the best parameters found by PTP-DAL's optimizer. Fig. 5.26b shows the results in the experiment with the OCXO with parameters  $W = 8$  and  $N = 4096$ . Note that, as expected, the observation window  $N$  with the OCXO is larger compared with the XO due to better frequency stability. Besides, the error in the experiment with the XO shows an oscillation, reaching approximately 200 ns of error, where the result with OCXO remains almost constant between 10 ns and 50 ns. These errors can be related to using the minimum operator for filtering the packets in (3.21), which has a high variation due to the distribution shape from Fig. 5.7a and Fig. 5.8a. However, even with such errors, this drift compensation strategy helps the performance of the packets algorithm by enabling larger window sizes.

Furthermore, all algorithms presented in this section use the optimal configuration found by PTP-DAL to generate the final results. For example, the packet selection algorithms use an observation window with length  $N$  and KF needs to initialize the covariance matrix  $\mathbf{Q}$ , which is unknown. For that, PTP-DAL implements optimizer methods that are used to find the best window length for the packet selection algorithms, and the best pair of covariances for the KF. Fig. 5.27 shows the optimization for the experiment with XO and OCXO. Both figures show the window lengths in terms of  $\max|\text{TE}|$  for the case without bias correction. The other optimization plots for the other two scenarios, with QIA correction and with ideal bias



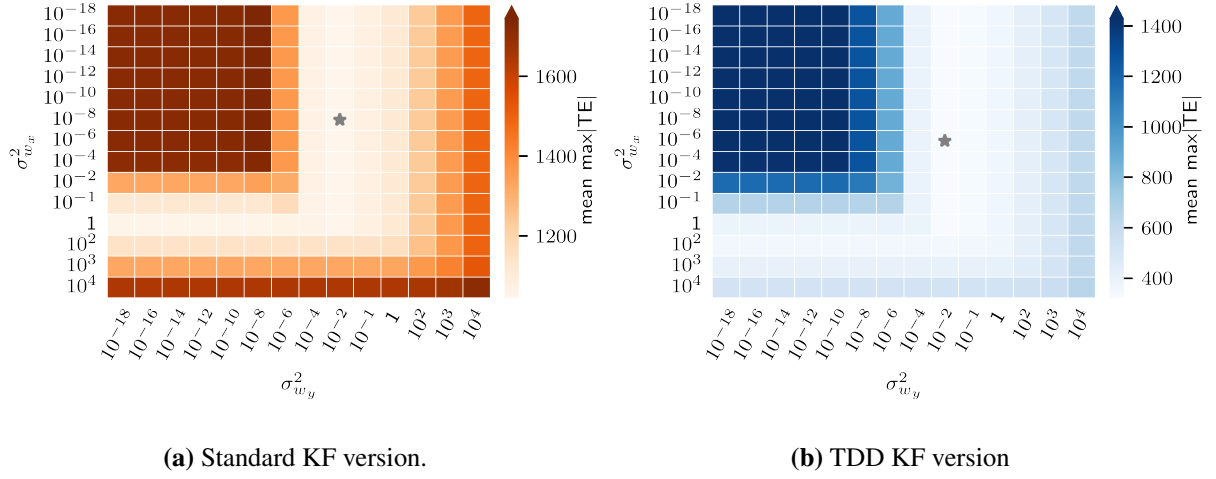
correction, were omitted since they present a similar idea. Fig. 5.28 shows the optimization process for the two implementations of the KF filter. The optimization plot only shows the result without bias correction because the KF optimal parameters do not change for the other scenarios, as the bias correction shifts the  $\max|\text{TE}|$  by a constant value, which does not change the optimal point.



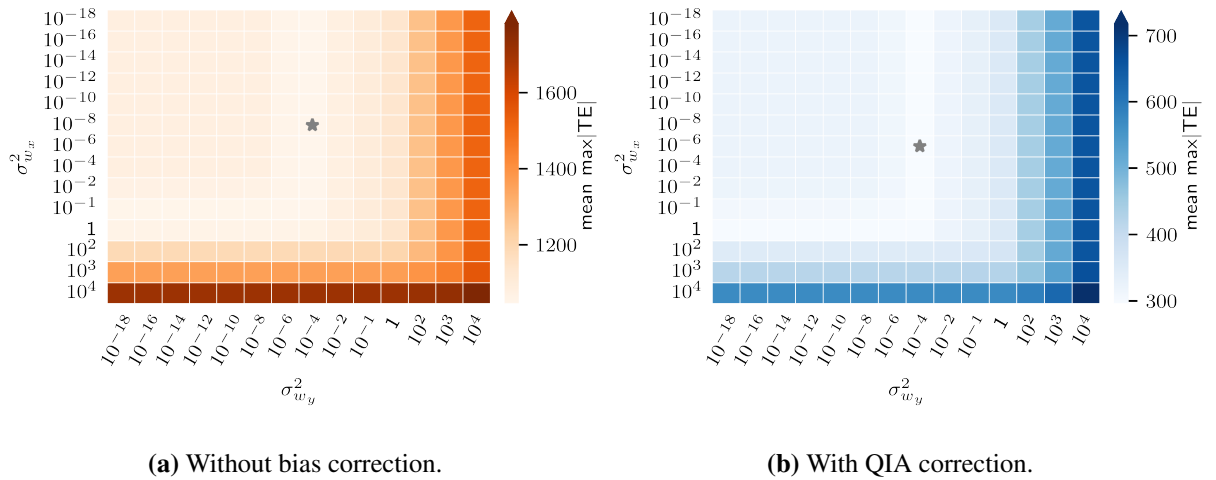
**Figure 5.27:**  $\max|\text{TE}|$  in terms of window length using four physical hops and without bias correction.

In the packet selection optimization shown in Fig. 5.27, note that the curves start with a higher  $\max|\text{TE}|$  and reach a plateau. This behavior indicates that the algorithms benefit from increasing the window length until a certain point, after which there are no benefits, and the performance may start to degrade. Moreover, the best window lengths found for sample minimum, maximum, average and average (TDD) were  $2^{13}$ ,  $2^{14}$ ,  $2^{12}$  and  $2^{13}$ , respectively. For the experiment with the OCXO, the best window lengths found were  $2^{14}$ ,  $2^{15}$ ,  $2^{14}$  and  $2^{15}$ , respectively. Note that, due to superior frequency stability, the optimal window length in the experiment with the OCXO was bigger for all estimators.

In the KF optimization, PTP-DAL tries to minimize the average  $\max|\text{TE}|$  by varying the diagonal values of the  $\mathbf{Q}$  matrix in a range from  $10^{-18}$  to  $10^4$  and selecting the pair that yields the lowest mean  $\max|\text{TE}|$ . Recall from Section 3.3.2 that the matrix  $\mathbf{Q}$  is defined as the state noise covariance matrix. The variance of the process noise affecting the time offset is denoted as  $\sigma_{w_x}^2$  and the variance of the process noise affecting frequency offset is denoted as  $\sigma_{w_y}^2$ . Fig. 5.28 shows the mean  $\max|\text{TE}|$  of the system, represented with a range of colors when the filter is initialized with each pair of variances. The pair that yielded the lowest  $\max|\text{TE}|$  is marked with



**Figure 5.28:** Heatmap from KF Q matrix tuning using four physical hops and the XO oscillator.



**Figure 5.29:** Heatmap from KF Q matrix tuning using four physical hops and the OCXO oscillator.

a star. The heatmap plot can also be used to analyze the impact of the process noise on the filter performance. Note that, in both cases, the pair of variances selected for the regular KF (processing all PTP measurements) was very close to the pair selected for the TDD version of KF (processing good packets only). This is because, independent of the observation samples,  $Q$  represents the intrinsic stochastic uncertainty present in the system state model.

Fig. 5.29 shows the optimization process for the experiment with the OCXO. Note that Fig. 5.29 shows that the experiments with the OCXO perform better when the diagonal values of  $Q$  are initialized with smaller values than in the experiments with the XO. This is because the OCXO has better stability than the XO. Consequently, the variance of the noise affecting the time and frequency offsets are smaller. We can also see that the experiments with the XO are more strongly affected by the variance values.

# Chapter 6

## Conclusion and Future Work

This chapter provides a summary of the content presented in this work and emphasizes the results. From the results, some conclusions are presented and directions for future research are proposed.

### 6.1 Conclusion

This work investigated the PTP performance over a cost-effective PTP-unaware network that presents a periodic utilization pattern due to the TDD system. In PTP-unaware networks, the PDV and delay asymmetry are the two main sources of error that affect the PTP performance, which can be even worse in a TDD system due to the periodic and asymmetric traffic. In this scenario, we showed that using the information about TDD system can help to alleviate the noise that affects the PTP measurements.

In Chapter 3 we presented different techniques that can be used to classify the PTP exchanges as good or bad depending on the delay experienced by those packets or based on the instant the PTP packet is transmitted with respect to the TDD time slot configuration. More specifically, this work used the results from the proposed K-means-based classifier that is used to classify the packets into good and bad individually in each direction, m-t-s and s-t-m. Additionally, we discussed different algorithms used in the literature to estimate the time and frequency offset using the PTP measurements, such as KF and the packet filtering algorithms. Besides, we showed how the information about "good" and "bad" packets could be used to improve the performance of these algorithms.

In this scenario, we also proposed a real-time mechanism used to estimate QIA delay

affecting the PTP measurements. The proposed method uses the information about the TDD periodic behavior to estimate the queuing delay. More specifically, the method relies on the fact that with a high probability, the PTP packets will experience the maximum and minimum delays, which can be used to calculate the queuing delay in each direction. We have shown that the proposed method can effectively compensate for the queuing delay, in some cases leading to performance improvements as high as 70%.

In Chapter 5, we presented an in-depth analysis of the delay suffered by the PTP packets with and without FH traffic and with different network configurations. We showed that different switches introduce different processing delays. Moreover, we showed that one switch introduces a delay that can be modeled as uniformly distributed, and a chain of multiple physical hops introduces a processing delay that can be modeled with a Gaussian distribution. Additionally, we also analyzed the delay experienced by the PTP packets when PTP shares the network with FH traffic. In this scenario, we showed that the delay distribution is complex and contains multiple levels of delay.

Finally, we presented an evaluation of the performance of different synchronization algorithms in terms of  $\max|\text{TE}|$ . These algorithms are widely used in PTP-unaware network as they are helpful to filter the PDV and improve the time offset estimation. In all scenarios, the algorithms converged on average to the associated delay asymmetry which suggests that the algorithms have a very low variance. Our evaluation also showed that the average estimators, such as sample average and KF, yield better results compared to sample minimum and sample maximum in all the evaluated scenarios. The performance improvement from the average estimators is related to the shape of the delay distribution. Additionally, KF and sample average, in a scenario without any bias correction, presented a better result when using only the good packets, going from a  $\max|\text{TE}|$  of approximately 1030 ns to 290 ns. In the scenario with QIA correction, all the algorithms presented similar results close to 290 ns, which is close to the static asymmetry from the network. Finally, in the APTS scenario, the KF achieved the better result with approximately 40 ns with the XO and 20 ns with the OCXO.

## 6.2 Future Work

There are several future investigations to follow this work. First, the classification based on the packet location, presented in Section 3.2.1, can be further explored to better understand how the changes on the FH change the location of the PTP packets. Second, an in-deep analysis of the relation between the PTP packet location and the TDD cycle. For instance, a study to find the optimal combination between PTP periodicity and TDD cycle configuration. Third, a new method of classification based on machine learning that combines the information from the PTP cycle location with the classification based on the experienced delays could be explored. Fourth, an analysis with distinct FH traffic, i.e., with packets of different sizes and different types of network topology, would be interesting to understand the impact of such scenarios on the PTP performance. Fifth, an analysis regarding the impact of FH packet size on the queuing delay distribution that affects PTP performance. Lastly, new ways could be explored to improve the PTP performance by dynamically scheduling the transmission of *Sync* and *Delay-Req* based on the TDD time slot configuration.

# Appendix A

## Discrete Kalman Filter

The KF is an optimal estimation algorithm in the minimum MSE sense. It was first introduced by Rudolf E. Kálmán in [42] and is a well-known and popular solution to state estimation problems. The filter provides an efficient recursive way to estimate the state of a linear dynamic system by combining the mathematical system model with a series of noisy measurements observed over time. The idea is that, by combining these two pieces of information, we can estimate the system state better than the estimate produced by either information alone.

### A.1 State-space Representation

The KF formulation assumes a system state given by the process of the form:

$$\mathbf{s}_k = \mathbf{A}_k \mathbf{s}_{k-1} + \mathbf{w}_{k-1}, \quad (\text{A.1})$$

where  $\mathbf{s}_k$  is a  $(n \times 1)$  vector which contains the  $n$  state variables of the given dynamic system at the time step  $k$ ,  $\mathbf{A}_k$  is a  $(n \times n)$  matrix defined as the *state transition matrix*, and  $\mathbf{w}_{k-1}$  is the *state noise* defined as an  $(n \times 1)$  zero-mean Gaussian vector. The matrix  $\mathbf{A}_k$  is assumed to be known, and it transform the state  $\mathbf{s}_{k-1}$  at time step  $k - 1$  to the state at time step  $k$ . Also, the state noise usually does not refer to a specific source of noise. Instead, it is used to model a combination of uncertainty sources between the system model compared to the real-world system.

The measurements of the process are assumed to be linearly related to the system states. Hence, the observations can be modeled as follows:

$$\mathbf{z}_k = \mathbf{H}\mathbf{s}_k + \mathbf{v}_k, \quad (\text{A.2})$$

where  $\mathbf{z}_k$  is a  $(m \times 1)$  vector with the measurements of the system at time step  $k$ ,  $\mathbf{H}_k$  is a  $(m \times n)$  matrix assumed to be known and defined as the *observation transition matrix*, and  $\mathbf{v}_k$  is the *measurement noise*, which is assumed to be a  $(m \times 1)$  zero-mean Gaussian vector.

Furthermore, the state and measurement noises,  $\mathbf{w}_k$  and  $\mathbf{v}_k$ , are zero-mean, mutually uncorrelated, with covariance matrix  $\mathbf{Q}_k$  and  $\mathbf{R}_k$  respectively given by:

$$\mathbf{Q}_k = \begin{cases} \text{E} [\mathbf{w}_k \mathbf{w}_k^T], & i = k \\ 0, & i \neq k, \end{cases} \quad (\text{A.3})$$

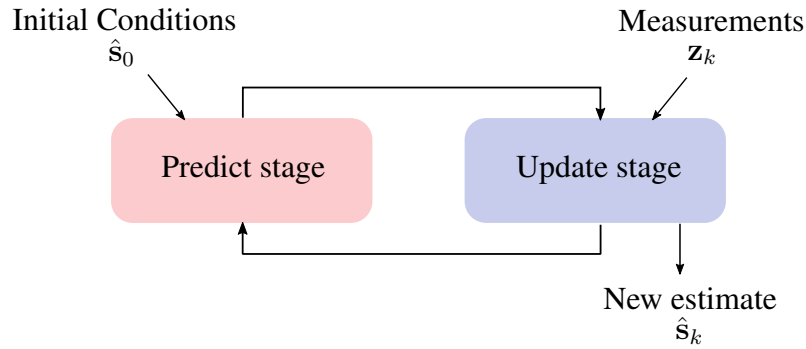
$$\mathbf{R}_k = \begin{cases} \text{E} [\mathbf{v}_k \mathbf{v}_k^T], & i = k \\ 0, & i \neq k, \end{cases} \quad (\text{A.4})$$

$$\text{E} [\mathbf{w}_k \mathbf{v}_i] = 0, \quad (\text{A.5})$$

where  $\mathbf{Q}_k$  is defined as the *state noise covariance matrix* and  $\mathbf{R}_k$  is the *measurement noise covariance matrix*. Both matrices are 0 when  $i \neq k$  since the noise vectors are not correlated with any of the previous samples.

The overall goal of KF is to find the best estimate of the system state at each time step using the value predicted by the system model and using the information provided by the noisy measurements. For that, it starts by predicting the behavior of the system at the next time step (a prior estimate), e.g., using (A.1). Then, it adjusts the value predicted using the available measurements (a posteriori estimate). Thus, the KF process can be described in terms of two stages, *predict* and *update*, as shown in Fig. A.1.





**Figure A.1:** Predict and update stages from Kalman filtering.

## A.2 Prediction Stage

The prediction stage is responsible for propagating the mean of the previous system state, denoted as  $\hat{s}_{k-1}^-$  and its associated *state estimation error covariance matrix*, denoted as  $\mathbf{P}_{k-1}^-$ , from time step  $k - 1$  to time step  $k$ . The minus sign in superscript indicates that it is an *a priori* state estimate. That is because, in the prediction stage, although it is an estimate from the current instant  $k$ , it does not include the information provided by the observed measurements, which will only be done in the *update* stage.

In the prediction stage, the system state at time step  $k - 1$  is projected ahead to time step  $k$  via the transition matrix  $\mathbf{A}$ , as follows:

$$\begin{aligned}
 \hat{s}_k^- &= \mathbb{E}[\mathbf{s}_k] \\
 &= \mathbb{E}[\mathbf{A}\mathbf{s}_{k-1}] + \mathbb{E}[\mathbf{w}_{k-1}] \\
 &= \mathbf{A}\hat{s}_{k-1},
 \end{aligned} \tag{A.6}$$

where the contribution of the state noise  $\mathbf{w}_{k-1}$  is zero since it is assumed to be a zero-mean ( $\mathbb{E}[\mathbf{w}_{k-1}] = 0$ ) random vector and uncorrelated with any previous  $\mathbf{w}$ 's as pointed out in (A.3).

Next, we derive a recursive expression for the covariance matrix associated with the *a priori* estimation error. First, we assume that we know the error covariance matrix associated with  $\hat{s}_{k-1}^-$ . The estimation error at time step  $k - 1$  is zero-mean and defined as:

$$\mathbf{e}_{k-1}^- = \mathbf{s}_{k-1} - \hat{s}_{k-1}^-, \tag{A.7}$$

and the error covariance matrix associated with  $\hat{\mathbf{s}}_{k-1}^-$  is:

$$\begin{aligned}\mathbf{P}_{k-1}^- &= \mathbb{E} [\mathbf{e}_{k-1}^- \mathbf{e}_{k-1}^{-T}] \\ &= \mathbb{E} [(\mathbf{s}_{k-1} - \hat{\mathbf{s}}_{k-1}^-) (\mathbf{s}_{k-1} - \hat{\mathbf{s}}_{k-1}^-)^T].\end{aligned}\quad (\text{A.8})$$

In a similar way, the error covariance matrix associated with the new *a priori* state (after projecting) is calculated using (A.1) and (A.6), as:

$$\begin{aligned}\mathbf{e}_k^- &= \mathbf{s}_k - \hat{\mathbf{s}}_k^- \\ &= (\mathbf{A}\mathbf{s}_{k-1} + \mathbf{w}_{k-1}) - \mathbf{A}\hat{\mathbf{s}}_{k-1}^- \\ &= \mathbf{A}\mathbf{e}_{k-1} + \mathbf{w}_{k-1},\end{aligned}\quad (\text{A.9})$$

and the corresponding covariance matrix is defined as:

$$\begin{aligned}\mathbf{P}_k^- &= \mathbb{E} [\mathbf{e}_k^- \mathbf{e}_k^{-T}] \\ &= \mathbb{E} [(\mathbf{A}\mathbf{e}_{k-1} + \mathbf{w}_{k-1}) (\mathbf{A}\mathbf{e}_{k-1} + \mathbf{w}_{k-1})^T] \\ &= \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}_k.\end{aligned}\quad (\text{A.10})$$

This way, using (A.10), the error covariance matrix can also be projected ahead. One can observe that the covariance of the state estimate error becomes larger at the prediction stage due to the summation with  $\mathbf{Q}_k$ , which means that the filter uncertainty always increases in the prediction stage.

Finally, the equations (A.6) and (A.10) comprise the prediction stage of the KF and are the ones needed in a practical implementation.

### A.3 Update Stage

The update stage is responsible for combining the current *a priori* prediction  $\hat{\mathbf{s}}_k^-$  with the observed measurement  $\mathbf{z}_k$  in order to improve the state estimate. The output of this stage is referred to as the *a posteriori* state estimate  $\hat{\mathbf{s}}_k$ . For that, at each iteration, the KF produces a new estimate of the system's state by choosing a linear blending of the prior system state estimation and the observed measurement, according to the following equation:

$$\hat{\mathbf{s}}_k = \hat{\mathbf{s}}_k^- + \mathbf{K}_k \boldsymbol{\nu}_k, \quad (\text{A.11})$$

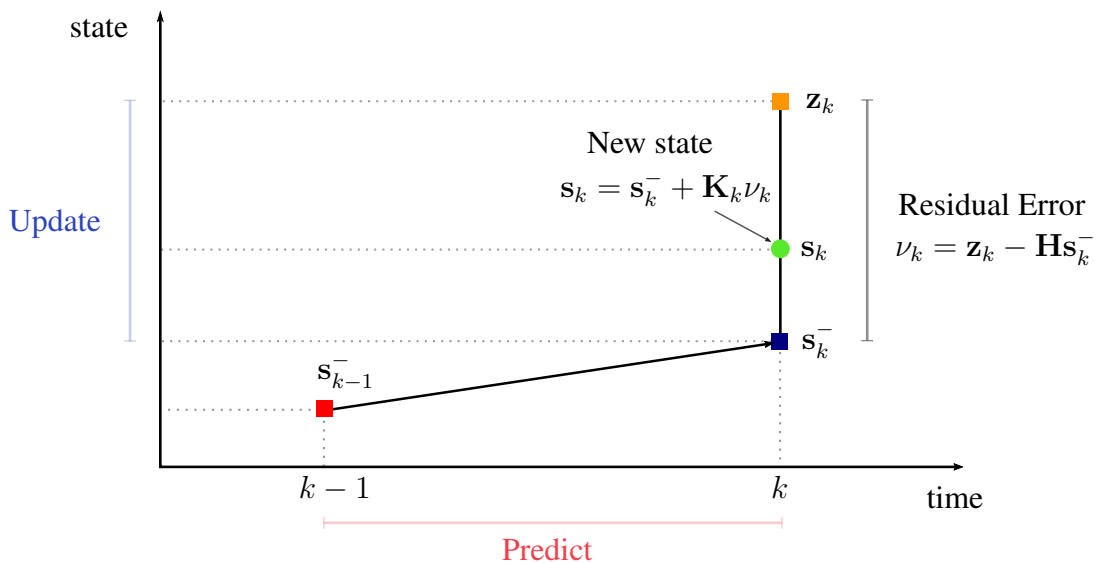
where  $\nu_k$  is the *innovation*<sup>1</sup> or *residual error*, which is the difference between the measurement  $\mathbf{z}_k$  and the transformed *a priori* estimation of the state, defined as:

$$\nu_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{s}}_k^-, \quad (\text{A.12})$$

the innovation can be seen as the "new information" that could be incorporated by the filter. Note that it is calculated in the so-called "measurement space", thus, the state  $\hat{\mathbf{s}}_k^-$  should be converted to the same unit that the measurements are being taken. For that, the matrix  $\mathbf{H}_k$ , called *transition matrix*, is used.

By substituting (A.12) in (A.11) one can obtain the well-known form to calculate the updated (*a posteriori*) state estimate:

$$\hat{\mathbf{s}}_k = \hat{\mathbf{s}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{s}}_k^-). \quad (\text{A.13})$$



**Figure A.2:** Pictorial representation of Kalman filtering process.

The "blending factor"  $\mathbf{K}_k$  will be chosen in a way that the value with a smaller uncertainty is considered more trustworthy. Hence, the new state estimate lies between the predicted and measured values, and also has a small uncertainty than either of them alone. Fig. A.2 exemplifies the process. More specifically, in the Kalman theory,  $\mathbf{K}_k$  is known as the *Kalman gain*, and

<sup>1</sup>Use the notation defined in [54], where the  $\nu$  represents the innovation, e.g., what is new in the measurement.

it aims to minimize the MSE of the state estimation. This minimization can be accomplished by minimizing the state covariance matrix that follows:

$$\mathbf{P}_k = \mathbb{E} \left[ (\mathbf{s}_k - \hat{\mathbf{s}}_k) (\mathbf{s}_k - \hat{\mathbf{s}}_k)^T \right]. \quad (\text{A.14})$$

By substituting (A.13) in (A.14) and performing the indicated expectations, it follows that:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + (\mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T), \quad (\text{A.15})$$

where (A.15) is a general expression to update the state covariance matrix and it is also known as Joseph form [55].

Next, the optimal  $\mathbf{K}_k$  can be calculated by taking the derivative of the trace of (A.15) with respect to  $\mathbf{K}_k$ , setting the result equal to zero and then solving for  $\mathbf{K}_k$ . For more details, see [56, Chapter 4]. Finally, the resulting  $\mathbf{K}_k$  that minimizes (A.15) is given by:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1}. \quad (\text{A.16})$$

By substituting the optimal  $\mathbf{K}_k$  from (A.16) in (A.15), one obtains the usual form to update the error covariance matrix:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^-. \quad (\text{A.17})$$

However, as shown in [56], (A.15) is valid for any gain, optimal or suboptimal, while the (A.17) is only valid for the optimal gain calculated by KF using (A.16). Besides that, (A.15) is shown in [56] to be more numerically stable than (A.17). Therefore, the filter implemented in this work uses (A.15).

Finally, the equations (A.6), (A.10), (A.13), (A.16) and (A.17) form the recursive Kalman filter algorithm.

# Bibliography

- [1] T. Cook, “Measuring time error,” PowerPoint presentation, 2014. [Online]. Available: [http://www.telecom-sync.com/files/pdfs/itsf/2014/Day3/1310-tommycook\\_calnex.pdf](http://www.telecom-sync.com/files/pdfs/itsf/2014/Day3/1310-tommycook_calnex.pdf)
- [2] E. Medeiros, M. Berg, and I. Almeida, “Network Entity For Synchronization Over a Packet-based Fronthaul Network,” US20220369251A1, 2022.
- [3] G. Gaderer, T. Sauter, F. Ring, and A. Nagy, “A novel, wireless sensor/actuator network for the factory floor,” in *SENSORS, 2010 IEEE*, 2010, pp. 940–945.
- [4] Z. Idrees, J. Granados, Y. Sun, S. Latif, L. Gong, Z. Zou, and L. Zheng, “IEEE 1588 for Clock Synchronization in Industrial IoT and Related Applications: A Review on Contributing Technologies, Protocols and Enhancement Methodologies,” *IEEE Access*, vol. 8, pp. 155 660–155 678, 2020.
- [5] J. Lopez-Jimenez, J. L. Gutierrez-Rivas, E. Marin-Lopez, M. Rodriguez-Alvarez, and J. Diaz, “Time as a Service Based on White Rabbit for Finance Applications,” *IEEE Communications Magazine*, vol. 58, no. 4, pp. 60–66, 2020.
- [6] M. Lévesque and D. Tipper, “A Survey of Clock Synchronization Over Packet-Switched Networks,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2926–2947, 2016.
- [7] H. Putnies, P. Danielis, A. R. Sharif, and D. Timmermann, “Estimators for Time Synchronization—Survey, Analysis, and Outlook,” *IoT*, vol. 1, pp. 398–435, 2020.
- [8] F. L. Walls and J. . Gagnepain, “Environmental sensitivities of quartz oscillators,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 39, no. 2, pp. 241–249, 1992.

- [9] “NR Requirements for support of radio resource management,” 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.133, 2022, version 17.6.0.
- [10] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*. USA: Academic Press, Inc., 2020.
- [11] J.-C. Lin, “Synchronization Requirements for 5G: An Overview of Standards or Specifications for Cellular Networks,” *IEEE Vehicular Technology Magazine*, vol. PP, pp. 1–1, 06 2018.
- [12] *Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. 1588-2002, 2002.
- [13] “Common public radio interface (CPRI) specification V7.0,” 10 2015.
- [14] ITU-T, “Rec. G.8271: Time and phase synchronization aspects of telecommunication networks,” Mar. 2020.
- [15] L. Meng, L. Yang, W. Yang, and L. Zhang, “A Survey of GNSS Spoofing and Anti-Spoofing Technology,” *Remote Sensing*, vol. 14, no. 19, 2022.
- [16] ITU-T, “Rec. G.8275.1: Precision time protocol telecom profile for phase/time synchronization with full timing support from the network,” Mar. 2020.
- [17] *Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. 1588-2019, 2020.
- [18] ITU-T, “Rec. G.8275.2: Precision time protocol telecom profile for time/phase synchronization with partial timing support from the network,” Mar. 2020.
- [19] C. Novaes, I. Freire, A. Klautau, I. Almeida, and E. Medeiros, “Analysis of Kalman Filtering for Clock Synchronization in PTP-Unaware Networks,” *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1–6, 2021.
- [20] ITU-T, “Rec. G.8260: Definitions and terminology for synchronization in packet networks,” Mar. 2020.
- [21] ———, “Rec. G.810: Definitions and terminology for synchronization networks,” Aug. 1996.

- [22] D. W. Allan, "Time and Frequency (Time-Domain) Characterization, Estimation, and Prediction of Precision Clocks and Oscillators," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 34, no. 6, pp. 647–654, 1987.
- [23] J. A. Barnes *et al.*, "Characterization of Frequency Stability," *IEEE Transactions on Instrumentation and Measurement*, vol. IM-20, no. 2, pp. 105–120, 1971.
- [24] L. Galleani, L. Sacerdote, P. Tavella, and C. Zucca, "A mathematical model for the atomic clock error," *Metrologia*, vol. 40, no. 3, pp. S257–S264, Jun. 2003. [Online]. Available: <https://doi.org/10.1088%2F0026-1394%2F40%2F3%2F305>
- [25] D. A. Howe, D. U. Allan, and J. A. Barnes, "Properties of Signal Sources and Measurement Methods," in *Thirty Fifth Annual Frequency Control Symposium*, 1981, pp. 669–716.
- [26] I. Freire, "FPGA implementation and evaluation of synchronization architectures for Ethernet-based cloud-RAN fronthaul," Master's thesis, Federal University of Pará, Jan. 2016. [Online]. Available: <http://repositorio.ufpa.br:8080/jspui/handle/2011/8022>
- [27] J. Rutman and F. L. Walls, "Characterization of frequency stability in precision frequency sources," *Proceedings of the IEEE*, vol. 79, no. 7, pp. 952–960, 1991.
- [28] ITU-T, "Rec. G.8271.1: Network limits for time synchronization in packet networks," Mar. 2020.
- [29] *Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. 1588-2008 (Revision of IEEE Std. 1588-2002), 2008.
- [30] ITU-T, "Rec. G.8261: Timing and synchronization aspects in packet networks," Aug. 2019.
- [31] I. Hadžić and D. R. Morgan, "On packet selection criteria for clock recovery," *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 1–6, 2009.
- [32] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud RAN for Mobile Networks—A Technology Overview," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 405–426, 2015.

- [33] I. Hadžić and D. R. Morgan, “Adaptive packet selection for clock recovery,” *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pp. 42–47, 2010.
- [34] M. Anyaegbu, C. Wang, and W. Berrie, “A sample-mode packet delay variation filter for IEEE 1588 synchronization,” *2012 12th International Conference on ITS Telecommunications*, pp. 1–6, 2012.
- [35] M. Anyaegbu, C.-X. Wang, and W. Berrie, “Dealing with Packet Delay Variation in IEEE 1588 Synchronization Using a Sample-Mode Filter,” *Intelligent Transportation Systems Magazine, IEEE*, vol. 5, pp. 20–27, 12 2013.
- [36] I. Freire, C. Novaes, I. Almeida, E. Medeiros, M. Berg, and A. Klautau, “Clock Synchronization Algorithms Over PTP-Unaware Networks: Reproducible Comparison Using an FPGA Testbed,” *IEEE Access*, vol. 9, pp. 20 575–20 601, 2021.
- [37] A. Bletsas, “Evaluation of Kalman filtering for network time keeping,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 9, pp. 1452–1460, 2005.
- [38] G. Giorgi and C. Narduzzi, “Performance Analysis of Kalman-Filter-Based Clock Synchronization in IEEE 1588 Networks,” *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 8, pp. 2902–2909, 2011.
- [39] —, “A resilient Kalman filter based servo clock,” in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*, 2013, pp. 59–64.
- [40] Z. Yang, J. Pan, and L. Cai, “Adaptive Clock Skew Estimation with Interactive Multi-Model Kalman Filters for Sensor Networks,” in *2010 IEEE International Conference on Communications*, 2010, pp. 1–5.
- [41] G. Giorgi, “An Event-Based Kalman Filter for Clock Synchronization,” *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 2, pp. 449–457, 2015.
- [42] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *ASME Journal of Basic Engineering*, pp. 35–45, 1960.



- [43] C. Zucca and P. Tavella, “The clock model and its relationship with the Allan and related variances,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 2, pp. 289–296, 2005.
- [44] I. Freire, “5G fronthaul synchronization via IEEE 1588 precision time protocol: Algorithms and use cases,” Ph.D. dissertation, Federal University of Pará, Dec. 2020.
- [45] M. J. Hajikhani, T. Kunz, and H. Schwartz, “A Recursive Method for Clock Synchronization in Asymmetric Packet-Based Networks,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2332–2342, 2016.
- [46] Z. Chaloupka, N. Alsindi, and J. Aweya, “Clock Synchronization Over Communication Paths With Queue-Induced Delay Asymmetries,” *IEEE Communications Letters*, vol. 18, no. 9, pp. 1551–1554, 2014.
- [47] N. Alhashmi, N. Almoosa, and G. Gianini, “Path Asymmetry Reconstruction via Deep Learning,” *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, pp. 1171–1176, 2022.
- [48] J. Paulo, I. Freire, I. Sousa, C. Lu, M. Berg, I. Almeida, and A. Klautau, “FPGA-based testbed for synchronization on Ethernet fronthaul with phase noise measurements,” in *2016 1st International Symposium on Instrumentation Systems, Circuits and Transducers (INSCIT)*, 2016, pp. 132–136.
- [49] I. Freire, C. Lu, M. Berg, and A. Klautau, “An FPGA-based design of a packetized fronthaul testbed with IEEE 1588 clock synchronization,” in *European Wireless 2017; 23th European Wireless Conference*, 2017, pp. 1–6.
- [50] *Tri-Mode Ethernet MAC LogiCORE IP Product Guide*, Xilinx, Nov. 2015, v9.0.
- [51] *AD9547/PCBZ and AD9548/PCBZ User Guide*, Analog Devices, 2014, rev. 0.
- [52] *LogiCORE IP Ethernet AVB Endpoint User Guide*, Xilinx, Mar. 2011, v3.1.
- [53] “Common Public Radio Interface: eCPRI Interface Specification v2.0,” May 2019.
- [54] T. Kailath, “A general likelihood-ratio formula for random signals in gaussian noise,” *IEEE Transactions on Information Theory*, vol. 15, no. 3, pp. 350–361, 1969.

- [55] R. S. Bucy and P. D. Joseph, *Filtering for Stochastic Processes, with Applications to Guidance*. John Wiley and Sons, Inc., 1968.
- [56] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley and Sons, Inc., 1992.

