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

DEEP LEARNING SOFTWARE-BASED HOLDOVER FOR IEEE 1588 PTP SYNCHRONIZATION IN 5G NETWORKS

Rodrigo Gomes Dutra

DM: 11/23

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

Rodrigo Gomes Dutra

DEEP LEARNING SOFTWARE-BASED HOLDOVER FOR IEEE 1588 PTP SYNCHRONIZATION IN 5G NETWORKS

DM: 11/23

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

Rodrigo Gomes Dutra

DEEP LEARNING SOFTWARE-BASED HOLDOVER FOR IEEE 1588 PTP SYNCHRONIZATION IN 5G 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)

G633d Gomes Dutra, Rodrigo.
DEEP LEARNING SOFTWARE-BASED HOLDOVER FOR IEEE 1588 PTP SYNCHRONIZATION IN 5G NETWORKS / Rodrigo Gomes Dutra. — 2023.
130 f. : il. color.

Orientador(a): Prof. Dr. Aldebaro Barreto da Rocha Klautau Junior
Dissertação (Mestrado) - Universidade Federal do Pará, , 1, Belém, 2023.

1. Holdover. 2. Deep learning. 3. Transformer networks. 4. Precision Time Protocol. 5. Sincronização. I. Título.

CDD 621.3822

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

“DEEP LEARNING SOFTWARE-BASED HOLDOVER FOR PTP IEEE 1588 SYNCHRONIZATION IN 5G NETWORKS”

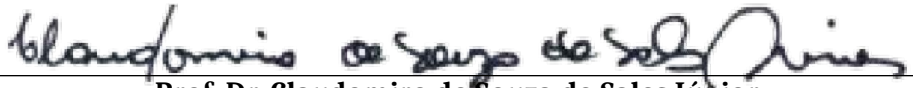
AUTOR: **RODRIGO GOMES DUTRA**

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 MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM: 28/03/2023

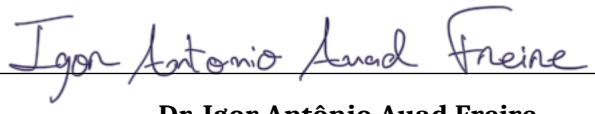
BANCA EXAMINADORA:

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



Prof. Dr. Claudomiro de Souza de Sales Júnior
(Avaliador Interno – PPGEE/UFPA)

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



Dr. Igor Antônio Auad Freire
(Avaliador Externo – IFCOMM TELECOMUNICAÇŒES)

VISTO:

Prof. Dr. Diego Lisboa Cardoso

(Coordenador do PPGEE/ITEC/UFPA)

Acknowledgments

I humbly take this opportunity to express my profound gratitude to the distinguished individuals whose contributions facilitated the successful completion of my Master's thesis.

Primarily, I extend my gratitude to the members of the Synchronization team in LASSE, formerly led by Igor Freire, for their guidance and support throughout my research. My esteemed colleagues Pedro Bemerguy, Camila Novaes, Juan Lopes, Douglas Santos, and José Neto, offered invaluable insights that enriched my ideas and elevated the quality of my research.

Furthermore, I am deeply indebted to my Professor Aldebaro Klautau, who has been a beacon of inspiration, guidance, and mentorship. His exceptional knowledge and expertise motivated me to achieve academic excellence. His constructive criticism and encouragement played an instrumental role in shaping my research work and realizing my academic goals.

I am also grateful to my former colleagues from LASSE, who work with me at SiDi Luan Gonçalves and Thiago Sarmento. Their friendship, support, and encouragement were invaluable sources of strength and motivation that contributed to professional development.

I thank the Innovation Center, Ericsson Telecomunicações S.A., Brazil for sponsoring this work.

Lastly, I would like to express my heartfelt appreciation to my family, particularly my wife Carolinne Fanjas, for their unwavering love, support, and understanding. Their encouragement, patience, and understanding have been pivotal to my success, and I cannot overstate the significance of their support.

In conclusion, I am deeply thankful to all who have played a part in my academic journey, and I attribute my professional achievements to the exceptional training and guidance provided by the LASSE team. With profound humility, I aspire to continue my academic growth and make meaningful contributions to the research field in the future.

Rodrigo Dutra

03/2023

Glossary

AIC	Akaike Information Criterion. 50, 51, 82
ANFIS	Adaptive Neuro-fuzzy Inference System. 34
ANN	Artificial Neural Network. 51, 52, 55
APTS	Assisted Partial-timing Support. 15, 26, 28, 30, 67, 92, 99
AR	Autoregressive. 45, 48, 50
ARIMA	Autoregressive Integrated Moving Average. 5–7, 34, 45, 46, 48–51, 65, 68, 72, 76, 81, 83, 85, 88, 97, 98
ARIMAX	Autoregressive Integrated Moving Average With Exogenous Variables. 35, 46
ARMA	Autoregressive Moving Average. 45, 49
ARMAX	Autoregressive Moving Average. 33
BBU	Baseband Unit. 1–3, 68
c-RAN	Centralized Radio Access Networks. 1, 2
CPRI	Common Public Radio Interface. 2
DOCXO	Double Oven-controlled Crystal Oscillator. 31
eCPRI	Enhanced Common Public Radio Interface. 2
EMAC	Ethernet Media Access Control. 68
FDD	Frequency-division Duplex. 24
FH	Fronthaul. 1, 68

FPGA Field-programmable Gate Array. 68, 69

FTS Full-timing Support. 15, 26, 28, 29, 36, 86

GNSS Global Navigation Satellite Systems. 3, 4, 6, 15, 26, 28, 30, 33, 36, 67, 92, 99

I Differencing Process. 48–50

IoT Internet Of Things. 1

LSTM Long Short-term Memory. 5, 6, 34, 35, 48, 57–60, 65, 68, 73, 75, 77, 85, 88, 89, 96–98

LTE Long Term Evolution. 22, 23

MA Moving Average. 45, 48–50

MIC Maximal Information Coefficient. 47

ML Machine Learning. 6, 34, 35, 73, 76, 77, 81, 88, 97, 99

MLP Multiple Layer Perceptron. xx, 52, 55

MO Multiple Output. 72, 77, 85, 88, 90

MTIE Maximum Time Interval Error. xx, 19, 21, 22, 31, 32, 36, 37, 80, 82, 86–89, 97

NLP Natural Language Processing. 58, 62, 64, 65

OCXO Oven-controlled Crystal Oscillator. xvi, 6, 9, 31–33, 35, 69, 80, 81, 87, 88, 90–92, 95, 97

PDV Packet Delay Variation. 35

PI Proportional-integral. 89, 98

PLL Phase Locked Loop. 28, 29, 33

PRTC Primary Reference Time Clock. 3, 4, 10, 23, 28, 36

PTP Precision Time Protocol. xx, xxi, 2–4, 6–8, 14–17, 19, 23, 26, 28–32, 35, 36, 67–70, 86, 87, 89, 91, 95–99

PTP-DAL PTP Dataset Analysis Library. 70

PTS Partial-timing Support. 4–6, 15, 16, 26, 28–30, 36, 37, 67, 71, 77, 86, 98

QoE Quality Of Experience. 2

QoS Quality Of Service. 2

RAN Radio Access Networks. 1

RBIS Reference Broadcast Infrastructure Synchronization. 6

RBS Reference Broadcast Synchronization. 6, 35

RF Radio Frequency. 1, 2

RNN Recurrent Neural Network. 55–58

RRU Remote Radio Unit. 1–3, 68

RTC Real-time Clock. 13

SARIMA Seasonal Autoregressive Integrated Moving Average. 50, 83

SARIMAX Seasonal Autoregressive Integrated Moving Average With Exogenous Variables. 50

SI System Of Units. 13

SVM-FIS Support Vector Machine Inference System. 34

SyncE Synchronous Ethernet. 3–5, 15, 16, 26, 28, 29, 31, 36, 86

T-SC Time Slave Clock. 3, 15, 31, 33, 36

TCXO Temperature Compensated Crystal Oscillator. 9, 31–34, 88, 91, 97

TDD Time-division Duplex. 3, 22–24, 91, 92, 97

TE Time Error. 19, 22, 23, 32, 33, 36, 37, 80, 86, 91, 92, 97

TIE Time Interval Error. 22

TLL Time-locked Loop. 68, 70, 71, 74, 98

ToD Time Of Day. 13, 17

VLAN Virtual Local Area Network. 69, 80

WSS Wide-sense Stationary. 41–43, 45–49, 73, 74

XO Crystal Oscillator. xvi, xvii, 6, 9, 31, 34, 35, 68, 69, 80, 81, 88–92, 95–97, 99

Symbols

$\mathbf{Attn}$	Scaled dot product attention. 61, 64
$\mathbf{Attn_{mask}}$	Transformer attention mask. 63
$\mathbf{Attn_{Scores}}$	Scaled dot product attention scores. 61, 63
$\mathbf{Attn}_{\mathbf{Scores}}^{\mathbf{Masked}}$	Scaled dot product attention masked scores. 63
$\mathbf{Attn_{weights}}$	Transformer attention weights. 61, 63
ρ	Autocorrelation function. 43
$\hat{\rho}$	Sample autocorrelation function. 44
$\hat{\gamma}$	Sample autocovariance function. 44
$\mathbf{B_{sz}}$	Batch size. 60, 61
θ	Bias factor. 52
Cov	Covariation function. 42, 43
d	ARIMA diferencing order. 49–51, 85
$\mathbf{d_k}$	Transformer's key dimension size. 61, 63
$\mathbf{d_{model}}$	Transformer main dimension size. 60, 61, 85
D	SARIMA diferencing order. 50, 51, 85
$\mathbf{Dec_{cros}}$	Decoder cross attention. 64
$\mathbf{Dec_{self}}$	Decoder self attention. 63, 64
ΔT	Time period between two consecutive measurements. 13
σ	Stardard deviation. 43
$\mathbf{dH}$	Transformer dimension per attention head. 61
Δ	Differenciation operator. 42, 47, 73, 74
$\hat{\Delta}_w^x$	Time offset estimation first differentiation. 73, 74, 76
x^d	Discrete time offset. 13, 17, 18, 20
y_f^d	Discrete fractional frequency offset. 13, 20
E	Moment function. 42, 43

$f(t)$	True instantaneous frequency. 12, 57, 58
f_{nom}	Nominal oscillators frequency. 12, 13
$f_{ref}(t)$	Reference oscillators frequency. 12, 13
$\mathbf{F_t}$	Feature size. 60
F_w	Feature window. 76, 77
K	Feature window size. 72, 75–77, 83, 84
M	Forecast horizon. 72, 74, 76, 77
$\hat{x}[n]$	Time offset estimation. 70, 71
$\mathbf{K_w}$	Key weight matrix. 61
$\mathbf{K}$	Transformer's Key projection. 61, 63
κ_i	PI loop integral constant. 70
κ_p	PI loop proportional constant. 70
L_w	Label window. 77
Y	Neural network's label. 54
L	Label window size. 72, 75, 77, 84
$\hat{E}$	Sample mean function. 44, 46
MSE	Mean square error function. 54, 81, 82
xh	Neuron's input. 52, 54
$\mathbf{H}$	Number of transformer attention heads. 61, 85
C_{state}	LSTM's cell state. 58
c	LSTM's cell state candidates. 57, 58
Y	LSTM autoregressive entries. 77
X	LSTM exogenous entries. 77
h	RNN's hidden state. 56–58
hl	Number of hidden layers. 85
η	Learning rate. 54
$\hat{Y}$	Neural network's output. 53, 54, 56, 77
p_j	Activation potential. 52–54
un	Weights that connect the input layer to the hidden layer. 55, 57, 58
vn	Weights that connect the hidden layer to the output layer. 55, 56

wn	Weight from past to current hidden state. 55, 57, 58
xn	Set of entries of a neural network. 55, 57
p	ARIMA proportinonal order. 48–51, 85
p	Probability density function. 42
P	SARIMA proportinonal order. 50, 51, 85
P_w	Prediction window. 77
$\phi(t)$	Random phase noise. 13
T	PTP timestamp. 17, 18, 70
$T_{21}[n]$	PTP master to slave timestamp difference. 17, 18
$T_{43}[n]$	PTP slave to master timestamp difference. 18
$\mathbf{Q}$	Transformer's Query projection. 61, 63
q	ARIMA moving average order. 49–51, 85
Q	SARIMA moving average order. 50, 51, 85
$\mathbf{Q_w}$	Query weight matrix. 61
$Y_{noise}[n]$	Random component of a time series. 43
s	SARIMA season size. 50, 82, 85
$s[n]$	Seasonal component of a time series. 43, 45
Σ	Sigmoid function. 57, 58
$\mathbf{Src}$	Transformer encoder input. 61, 63, 64
$T(t)$	Local clock's ToD. 13
$T_{ref}(t)$	Reference clock's ToD. 13
$\mathbf{T_s}$	Time step. 60–64
$tanh$	Hiperbolic tangent function. 57, 58
τ	Observation period. 22
τ_p	Temperature processed samples. 74, 76
$\mathbf{Tgt_{in}}$	Transformer decoder input. 63–65
$\mathbf{Tgt_{out}}$	Transformer decoder output. 64, 65
$\tilde{x}[n]$	PTP time offset measurement. 18, 70, 71
T_m	Total length of the measurement period. 22
$m[n]$	Trend component of a time series. 43, 45
$\mathbf{V_w}$	Value weight matrix. 61
$\mathbf{V}$	Transformer's Value projection. 61, 63

W	Set of weights. 52, 54
w	First difference window size. 73, 74, 83, 84, 88, 90
p	Second difference window size. 74
$x(t)$	Time offset. 13, 19, 57, 58
$y(t)$	Frequency offset. 12, 13
y_f	Fractional frequency offset. 12, 13
z	Time series realization. 44, 46, 47
Z	Time series stochastic process. 41–44

List of Figures

List of Tables

Contents

Abstract

This work proposes evaluates software-based algorithm mechanisms for maintaining the synchronization of a real-time clock in holdover operation when the timing reference input is unavailable. Three algorithms, Autoregressive Integrated Moving Average (ARIMA), long short term memory (LSTM), and Transformer networks, are implemented and trained using timestamps and temperature data acquired while the slave clock is locked to a master clock. When the slave clock loses its reference, the algorithm-based models take over and control the clock. The proposed method is evaluated on a testbed of IEEE 1588 Precision Time Protocol (PTP) clocks based on field-programmable gate arrays, where nanosecond-accurate timestamps are collected for offline analysis. The models are evaluated using two clocks, one cost-effective, cristal oscillator (XO), and one robust, oven controlled cristal oscillator (OCXO), in both constant and variable temperature scenarios. The results show that all algorithms can sustain clock synchronization accuracy within reasonable Time division duplex (TDD) synchronization limits over intervals of 1000 seconds in all temperature and clock scenarios, with the transformer-based holdover mechanism outperforming the statistical approach and LSTM network. This cost-effective software-based approach proves to be feasible for increasing clock accuracy during holdover operation and can be generalized to other holdover contexts, such as in a Global Navigation Satellite System (GNSS) scenario.

Resumo

Este trabalho propõe a avaliação de mecanismos baseados em algoritmos de software para manter a sincronização de um relógio em tempo real em operação de holdover, quando a entrada de referência de sincronização não está disponível. Três algoritmos, ARIMA, LSTM e redes de Transformer, foram implementados e treinados usando dados de carimbo de tempo e temperatura adquiridos enquanto o relógio escravo está sincronizado com um relógio mestre. Quando o relógio escravo perde sua referência, os modelos baseados em algoritmos assumem o controle e o mantém sincronizado. O método proposto foi avaliado em um teste de relógios de protocolo de tempo de precisão IEEE 1588 PTP baseados em uma bancada de testes baseada em FPGAs, onde *timestamps* precisos em nanossegundos foram coletados para análise offline. Os modelos foram avaliados usando dois tipos de relógios, um econômico, XO, e um robusto, OCXO, em cenários de temperatura constante e variável. Os resultados mostram que todos os algoritmos podem manter a precisão de sincronização do relógio dentro de limites dos requisitos de sincronização para comunicação TDD por intervalos de 1000 segundos em todos os cenários de temperatura e oscilador, sendo que o mecanismo de holdover baseado em redes Transformer superou a abordagem estatística e a rede LSTM. Esta abordagem de software de baixo custo é viável para aumentar a precisão do relógio durante a operação de holdover e pode ser generalizada para outros contextos de holdover, como em um cenário GNSS.

Chapter 1

Introduction

The need for synchronization among network elements and devices is essential in a growing range of modern applications, such as telecommunications [1], smart grids, data centers, industrial automation [2], the internet of things (IoT), financial applications, time-sensitive networks, distributed computing, and more. In fifth-generation (5G) mobile communications, synchronization capabilities are necessary not only on the radio interface but also on the backhaul and fronthaul (FH) networks, which transport data to, from, and within the radio base stations.

In this context, fronthaul refers to the transport network that connects the baseband unit (BBU) and the remote radio unit (RRU) in a cellular network. The BBU is responsible for processing the digital signals, while the RRU handles the analog radio frequency (RF) signals. In a traditional telecom network, the BBU and RRU are typically located close to each other, within the same equipment room or building, and are traditionally connected by a short coaxial or fiber-optic cable.

However, as network demands have grown, this traditional architecture has become increasingly challenging to manage and scale. Over the past few years, there has been a significant shift in the telecommunications industry towards deploying radio access networks (RAN) and centralized radio access networks (c-RAN) . This shift has been driven by a desire to increase network capacity and performance while reducing costs and improving efficiency.

RRUs and BBUs are essential components of modern telecommunications networks, especially in the context of c-RAN architectures. The centralization of BBU processing functions in a c-RAN architecture enables network operators to significantly reduce the number of BBUs required to support their networks while improving scalability and performance. However, this architecture increases the bandwidth utilization in the FH network. Since a significant amount

of processing is offloaded from the base stations, the c-RAN architecture poses stringent delay and jitter requirements for transporting FH data.

The common public radio interface (CPRI) [3] has been the most popular RRU-BBU interface to date. Nevertheless, it necessitates extremely high-capacity and ultra-low latency links for the digitized RF signal. Therefore, more efficient schemes based on other functional splits of the radio processing chain are required to support 5G. Additionally, the demand for a packet-switching-based FH network has resulted in an improved version of CPRI the enhanced common public radio interface (eCPRI) [4], which is designed for packet networks such as Ethernet and IP.

Packet-based networks, such as Ethernet, do not natively provide a way to distribute synchronization throughout the network. However, technologies are available for packet-based time distribution, including the IEE 1588 PTP, which is utilized in this study.

However, as the fronthaul system transitions to a packet-based architecture, new challenges must be addressed to ensure the quality of service (QoS) and quality of experience (QoE) for end users. One such challenge is the need for holdover mechanisms to maintain synchronization between the RRUs and the BBUs in the event of a loss of synchronization signal.

Holdover is a mechanism that enables the RRU to maintain synchronization with the BBU during periods of temporary loss of synchronization signal, such as during a network outage or when a signal is degraded. This mechanism ensures that the RRU continues to transmit data with accurate timing, which is critical for maintaining QoS and QoE.

Although holdover operation has been defined in several ITU-T recommendations and requirements, the potential benefits of software-based holdover mechanisms have not been thoroughly explored. Such mechanisms can offer a cost-effective solution without requiring further expenses in hardware, such as robust slave oscillators. This work aims to investigate this scenario, which will be further detailed in the subsequent chapters and sections.

As a reminder, Section 1.1 provides an overview of the main motivation for this work. It first introduces the concept of synchronization and then addresses the holdover operation problem under subsections 1.1.1 and 1.1.2. Section 1.2 outlines the chapters of this work, and Section 1.3 discusses the contributions of this work and publications resulting from this dissertation.

1.1 Motivation

1.1.1 Synchronization overview

Accurate clock synchronization is a critical aspect of time-sensitive applications employed in various sectors, as discussed in the preceding section. In many cases, particularly indoor ones, the Primary reference time clock (PRTC) obtains its time reference from the global navigation satellite systems (GNSS) and subsequently distributes this timing information to Time slave clock (T-SC) equipment through a packet-based network. Typically, timing distribution is accomplished using IEEE 1588 PTP and Synchronous Ethernet (SyncE), which have different use cases. PTP can deliver both time and frequency by exchanging timestamps over a packet-based network, while SyncE distributes frequency directly over the Ethernet physical layer. GNSS-disciplined clocks synchronize with the time scale disseminated collectively by multiple satellites.

The synchronization paradigm typically entails disseminating time and frequency from a robust PRTC to several T-SC equipment, which often possess less accurate clock oscillators. This is frequently due to the expense of high-accuracy oscillators, making it impractical to distribute them to all equipment, including user equipment. As a result, synchronization is required, necessitating the presence of at least one robust and highly accurate clock oscillator in the PRTC, which distributes its timing and frequency information across the entire network.

In telecommunications networks, particularly in the fronthaul, the need for synchronization is directly related to the network's utilization requirements. For instance, the fronthaul must maintain a maximum time error of 1.5 ms between the BBU and RRU to maintain functional the time-division duplex (TDD) operation in a 5G new radio network.

The synchronization distribution can be achieved in various ways, depending on the network's location and budget. For instance, in an outdoor scenario where the network owner is not concerned with costs, all clocks in the network may have an expensive GNSS source. However, in more realistic scenarios, only one or a few PRTC will have a GNSS source, and its timing and frequency information will be distributed to T-SCs using a combination of PTP and SyncE. To achieve a cost-effective solution, the network owner may use legacy Ethernet switches to distribute the PRTC's timing information to T-SCs equipment. Although this particular choice can save significant expenses by utilizing existing legacy Ethernet switches infrastructure, there is a trade-off, as legacy switches do not provide direct support to PTP, leading to problems such

as packet delay variation and bias.

Our work focuses on this cost-effective scenario, defined as partial-timing support (PTS), where the PRTC has access to a GNSS and distributes its timing notion through the use of PTP and may use or not SyncE. Our synchronization testbed features a legacy Ethernet switch capable of emulating up to four hops using VLANs. This work does not aim to mitigate the issues introduced using a non-PTP-aware infrastructure. Instead, we focus on the holdover problem, which will be discussed in the following subsection.

1.1.2 Holdover overview

The holdover operation can be broadly defined as the condition where a device cannot receive valid timing and frequency information from the synchronization source but instead relies on its internal oscillator to maintain its timing, and frequency accuracy [5]. The holdover state occurs when the synchronization source, such as a GNSS receiver, is unavailable or temporarily disrupted, and the receiving device has no other reliable synchronization sources.

In GNSS-based synchronization, holdover can occur due to various reasons such as loss of signal from satellites, local interference, or obstruction of the satellite signal due to environmental factors like tall buildings or natural obstacles. When a GNSS receiver loses its signal or experiences an interruption in its signal, it enters the holdover state and continues to generate timing and frequency signals based on the last known good measurement.

During the holdover state, the GNSS receiver's internal oscillator is used to maintain the timing, and frequency accuracy, which can result in a gradual drift in timing and frequency accuracy as the oscillator's stability is usually lower than the GNSS source's stability. The holdover state can last for a short period of time, typically ranging from a few seconds to several hours, depending on the device's holdover capabilities and the availability of other synchronization sources.

GNSS-based synchronization is perhaps the most common use case of holdover operation. Nevertheless, there are other cases of holdover use. In PTP networks, the holdover operation can occur by the degradation of the packets coming from the master clock, which can occur by high background traffic and even direct attacks in the network by hackers. Besides that, a PTP slave clock equipment may also enter in holdover by problems in the infrastructure itself, which includes the network and the master clock-related equipment. Alongside those cases, the holdover operation may start for maintenance reasons. At the same time, the slave clock

will try to maintain its synchronization notion at acceptable levels while qualified personnel is providing maintenance to the network.

Monitoring the holdover state's duration and performance is essential to ensure that the device's timing and frequency accuracy remain within acceptable limits. However, providing accurate timing and frequency correction signals during the holdover mode is not a trivial task due to the oscillator stability, which is highly influenced by its quality and environmental noise sources, such as temperature [6]. The duration and accuracy of the holdover state can be improved by using a more stable oscillator, increasing the oscillator's stability by temperature control. Although, as previously commented, robust and stable oscillators are costly. Thus, this standard way to improve holdover performance is unfeasible in cost-effective scenarios. A cost-effective way to improve holdover performance is to rely on a software-based mechanism to aid the slave clock during a holdover operation.

In the context of PTS deployments, ITU-T recommendation (G.8273.2) [7] defines two scenarios of holdover. The first is when the time reference (i.e., PTP) is lost, but SyncE still provides the frequency reference over the physical layer. The second is when both time (PTP) and frequency (SyncE) references are lost. This work focuses on the latter, which is not specified in [7] and is still subject to further investigations.

1.2 Dissertation outline

This work is organized as follows:

- Chapter 2, the fundamental concepts of synchronization are detailed, providing the basis for this work. This includes a description of the function of oscillators, synchronization protocols, the requirements considered for this work, and the metrics used.

- Chapter 3 thoroughly describes the holdover operation, outlining the related works, application scenarios, and the considered baselines for holdover operation.

- Chapter 4, the base time series theory used in the time series pipeline of this work is defined.

- Chapter 5 then defines the models used in this work, namely the autoregressive integrated moving average (ARIMA), long short-term memory (LSTM), and Transformer network models, as a way to provide a software-based holdover mechanism.With these concepts

established, Chapter 6 describes the testbed, the time processing pipeline, and the forecast strategies used in this work.

- Chapter 7 describes the experiments used to produce results with the proposed software-based holdover application with each model, and Chapter 8 concludes this work by outlining the main conclusions extracted from the results and future works.

1.3 Contributions

Our work contributes to the field of network synchronization by providing a holdover solution based on software that employs three distinct algorithms: ARIMA, LSTM, and Transformer networks. The ARIMA model is a classical model in the literature of time series focasting. As such, the ARIMA model serves as a baseline benchmark for testing the models based on machine learning (ML). The LSTM model is a deep learning model that been known to perform well for time series applications, while the Transformer networks are currently achieving even greater performance in the same applications. Thus, our work provides an evaluation of state-of-the-art models in the time series field, as well as classical statistical methods, in order to benchmark the deep learning models. This cost-effective approach offers an efficient means of maintaining acceptable levels of synchronization when external references are lost, even with the use of a low-cost oscillator (XO).

To evaluate the effectiveness of our holdover solution, we conducted experiments using actual data obtained from a PTP 1588 testbed, which featured PTS infrastructure and two different types of clocks: a cost-effective XO and a robust OCXO. Our results indicate that our approach can maintain acceptable levels of synchronization for a holdover period of 1000 seconds under varying clock and temperature conditions.

Furthermore, we validated our software-based holdover solution, which is based on a time-series approach and can be easily adapted to other use cases beyond PTP holdover. Since our solution is reliant only on data present in the slave clock, it has potential applications in other cases, such as GNSS holdover or broader over-the-air synchronization methods like reference broadcast infrastructure synchronization (RBIS), reference broadcast synchronization (RBS) and pilot based synchronization.

Our recent findings, which focused exclusively on the LSTM model, were published in a peer-reviewed article in Globecom titled "An LSTM-based Approach for Holdover Clock Dis-

ciplining in IEEE 1588 PTP Applications" [8]. This paper contributed to the holdover literature by conducting experiments in a 1588 PTP scenario using a deep learning model that utilized synchronization and temperature data. Additionally, we investigated current standards, such as ITU-T 8273.2 [9], which have cited the current evaluation scenario of this work as a topic requiring further investigation. Thus, our paper contributes to the development of innovative and efficient solutions for network synchronization, with potential applications in various fields. This dissertation builds upon the evaluations presented in our previous paper, providing comparisons with both a classical model in the literature (ARIMA) and the current state-of-the-art model (Transformer).

Chapter 2

Fundamentals of Time Synchronization in Packet-based Networks

This chapter provides an overview of the fundamental concepts necessary to understand this work comprehensively. To begin, Section 2.1 provides a detailed explanation of clocks and oscillators and their significance in synchronization. Subsequently, Section 2.2 examines the various types of synchronization, while Section 2.3 outlines the means by which synchronization is disseminated via synchronization protocols and the different types of timing and frequency distribution topologies employed. Section 2.4 elaborates on the use of PTP for time synchronization and the intricacies of this synchronization process. Following the synchronization sections, Section 2.5 delves into the use of metrics for measuring the quality of synchronization and clock stability. Finally, Section 2.6 comprehensively describes the synchronization requirements necessary for a fronthaul application in normal operating mode.

2.1 Clocks and Oscillators

Clocks are made up of two main components: a pulse counter and an oscillator. The oscillator, which can be a quartz crystal or an atomic resonator, generates regular pulses that are counted by the pulse counter. The frequency of the oscillator, or the number of times it pulsates in a given time period, is inversely related to the time interval between pulses, with a higher frequency corresponding to a shorter time interval.

Clock oscillators are essential for maintaining synchronization in many types of systems. They provide a consistent, stable reference frequency that can be used to coordinate the oper-

ation of different components. For example, in telecommunications and computer networks, clock oscillators are used to generate timing signals that synchronize the transmission and reception of data. This ensures that data is sent and received at the appropriate times, allowing the system to operate efficiently and effectively.

2.1.1 Types of oscillators

Many different types of oscillators can be used for synchronization, and the cost-effectiveness of each type can vary depending on the specific application and the system's requirements. Some common types of oscillators that are used for synchronization include:

- Quartz crystal oscillators (XO): These oscillators are popular due to their low cost, small size, and low power consumption. However, they can be impacted by various factors such as the cut and quality of the crystal, temperature, humidity, and even radiation. In addition, shocks can permanently change their frequency, and they may drift over time due to aging.

- Temperature compensated crystal oscillator (TCXO): To mitigate the impact caused by environmental factors such as temperature variation on the XO, the TCXO has temperature compensation circuits that can be used to limit the impact of temperature variations on the output frequency.

- Oven-controlled Oscillator (OCXO): This type of oscillator generates heat to maintain a constant, stable temperature at which it exhibits the greatest frequency stability. While these oscillators are more stable, they also use more energy, have longer startup times, and are more expensive.

- Atomic oscillators are based on the atomic properties of elements such as cesium or rubidium. These oscillators use a Voltage-Controlled Crystal Oscillator (VCXO) that is locked to a highly stable frequency reference generated by a microwave transition in the chosen atomic element. This reference frequency is very stable and resistant to environmental influences, and its stability is transferred to the VCXO. Atoms can absorb and emit electromagnetic energy at many different frequencies. However, hyperfine transitions are often used in atomic oscillators because they are highly stable, relatively insensitive to environmental effects, and occur in a convenient part of the spectrum.

The overall comparison between these types of oscillators, according to [10], can be visualized in Fig. 2.1. Rb and Cs stand for rubidium and cesium, respectively.

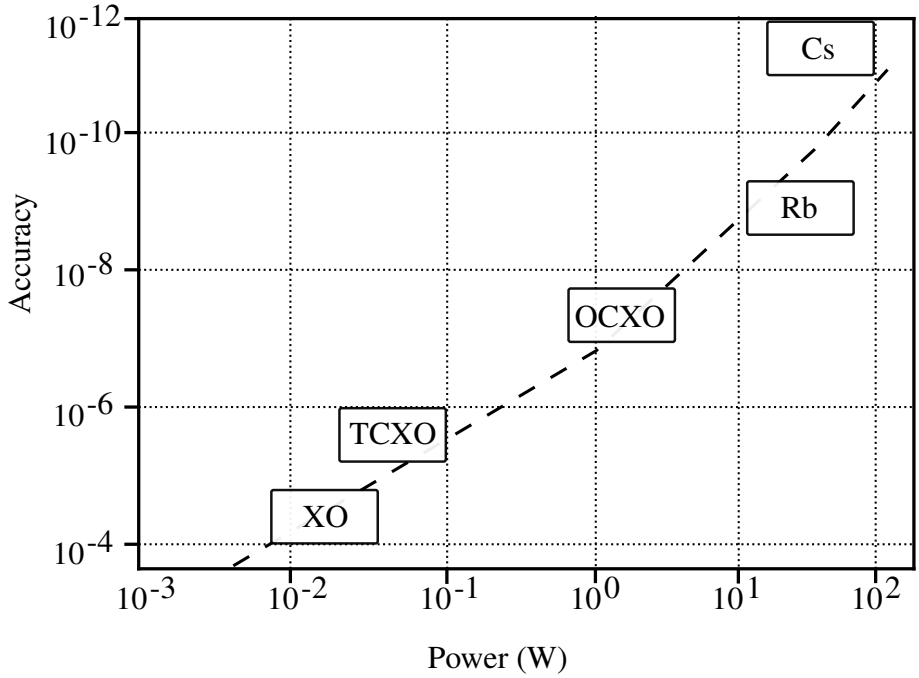


Figure 2.1: Comparison of accuracy versus power consumption of quartz and atomic oscillators [1].

Besides the apparent relation between accuracy and power consumption, [10] also points out that accuracy versus oscillator price and accuracy versus oscillator size has a similar relationship. Thus, to choose a clock, one should always consider a balance between those factors and the final application, e.g., for PRTC generally, a clock with great accuracy and precision is a better choice to assure the timing distribution in a synchronization network. Therefore the price should not be a primary concern in this case, but accuracy should.

2.2 Frequency and phase synchronization

Synchronization broadly refers to coordinating events within a system comprising two or more elements. This concept is commonly applied in scenarios where a single clock coordinates multiple clocks within a system. Thereby, distributing both frequency and phase synchronization throughout all elements.

In this context, syntonization, or frequency synchronization, refers to coordinating events within a system that occur at the same rate. Fig 2.2 depicts this concept. This illustration shows the union of an oscillator and a counter, with the counter converting the analog signal output by

the oscillator into a digital signal. With each rising edge, the clock increments its time register. As such, the rate of repetition of these rising edges corresponds to the frequency of the clock. The system is considered to be syntonized when the slave clocks' oscillators are disciplined to compensate for any differences in counter frequencies.

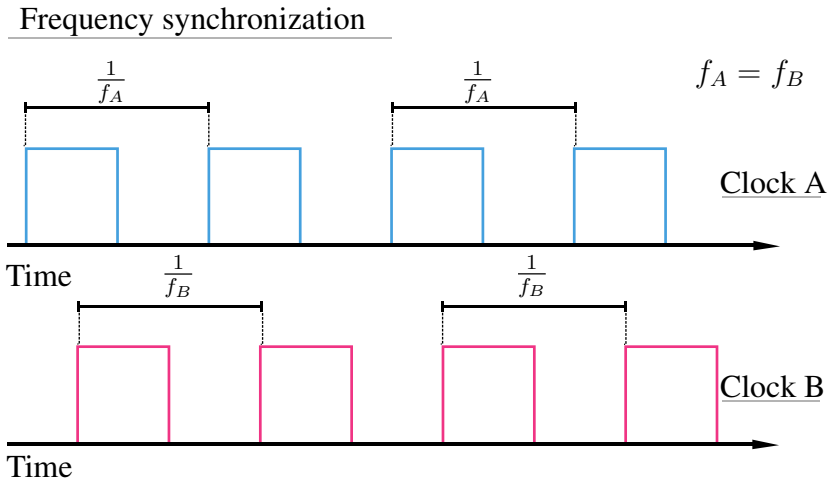


Figure 2.2: Frequency synchronization.

In addition to frequency synchronization, phase synchronization is also an essential concept in coordinating events within a system. Phase synchronization refers to aligning the rising edges between the two clocks' square waves, as illustrated in Fig 2.3. This type of synchronization is essential for scenarios in which various elements within a system must perform events at the same instant.

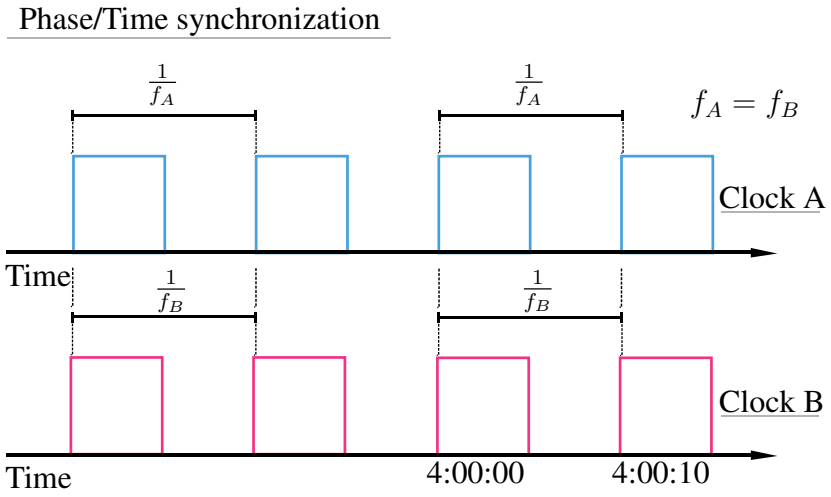


Figure 2.3: Time/phase synchronization.

Another essential concept within the coordination of events within a system is time syn-

chronization. Time synchronization refers to coordinating the clocks' time-of-day (ToD) in a system. Each device contains a real-time clock (RTC) module that keeps track of the ToD based on the device's oscillator frequency. It is worth noting that even if the oscillator frequencies are not the same between two devices, it is still possible to achieve time synchronization if the devices have the same ToD value in their respective RTCs. In specific cases, such as telecom networks, the ToD time series is also used to synthesize a phase-aligned local clock signal, so that time synchronization can be used to achieve phase synchronization [11].

2.2.1 Time and frequency offset

In this section, we present a formal definition of the frequency offset. The local oscillator frequency of a device's real-time clock (RTC) significantly impacts the device's time-of-day (ToD) notion. However, due to intrinsic limitations and external environmental factors, the oscillator frequency may deviate from the nominal frequency, compromising synchronism. The frequency offset, represented by $y(t)$, is defined as the discrepancy between the actual and nominal frequency values, as represented mathematically by the following equation:

$$y(t) = f(t) - f_{nom}, \tag{2.1}$$

where $f(t)$ and f_{nom} represent the actual frequency at instant t and the nominal oscillator frequency, respectively, in terms of hertz (Hz). This difference is commonly measured as a normalized value, resulting in the fractional frequency offset or fractional frequency deviation, as described in [5], as represented mathematically by as follows:

$$y_f(t) = \frac{f(t) - f_{nom}}{f_{nom}}. \tag{2.2}$$

The y_f is a normalized metric and therefore does not have a base unit. It is commonly given in parts per million (ppm) or parts per billion (ppb). To establish a more direct frequency difference metric with respect to a reference, the f_{nom} may be replaced with the reference frequency value, $f_{ref}(t)$. This calculation will consider the relation of the clock to its reference rather than its nominal value f_{nom}, as represented mathematically by the following equation:

$$y_f(t) = \frac{f(t) - f_{ref}(t)}{f_{ref}(t)}. \tag{2.3}$$

This equation represents more general scenarios where the reference does not have a static value, but rather its frequency behavior approximates a time-varying function represented by $f_{ref}(t)$.

Another metric that measures the difference between the local clock and a reference is the time offset. This metric is obtained by the difference between the time of day (ToD) values obtained from the local and reference real-time clock (RTC)s, as represented mathematically by the following equation:

$$x(t) = T(t) - T_{ref}(t), \tag{2.4}$$

where the difference $x(t)$ is given in terms of a time System of Units (SI) unit and the terms $T(t)$ and $T_{ref}(t)$ represent the local clock and reference ToD values, respectively. This time difference is the main variable of interest in this work, and more generally, it fits into a time series problem.

There exists a relationship between the frequency and time offset, as explained in Sections 2.4.2 to 2.4.4 in [12]. The final equation obtained in this relation is:

$$y_f(t) = \frac{dx}{dt}, \tag{2.5}$$

where $x(t)$ and y_f(t) represent the instantaneous time and fractional frequency offset, respectively. Thus, $y(t)$ can be seen as the amount of time offset accumulated in a given time period. For example, a 10 ppb fractional frequency offset value represents an accumulation of 10 ns of time offset error in one second.

The variable $y(t)$ can be calculated through the use of periodic measurements of time offset, represented as a set of discrete samples x^d. This results in $y_f^d[n]$, a discrete representation of $y(t)$, as outlined in 2.6:

$$y_f^d[n] = \frac{x^d[n] - x^d[n-1]}{\Delta T}, \tag{2.6}$$

where ΔT is the time period between two consecutive measurements. This time period is assumed to be short enough to consider $y_f^d[n]$ as a linear variable within its limits.

In addition to the definition of time offset outlined in 2.4, time offset can be modeled as specified in ITU-T Recommendation G.810 [5]. This is represented in 2.7:

$$x(t) = x_0 + y_0 t + \frac{D}{2}t^2 + \frac{\phi(t)}{2\pi f_{nom}}, \tag{2.7}$$

where the parameter x_0 represents the initial time offset, y_0 represents the initial frequency offset, with $f_{nom[n]}$ as the reference value, D is the linear fractional frequency offset drift rate, and ϕ represents the random phase deviations of the clock signal driving the real-time clock (RTC).

Equation 2.7 highlights the influence of frequency offset and external factors on the time offset. It is important to note that the ϕ parameter represents the random deviations in phase, which can be influenced by factors such as temperature, humidity, supply voltage, and pressure changes that affect the slave clock's oscillator. On the other hand, the D parameter represents a linear frequency deviation, which is affected by the aging of the clock's oscillator. This means that the ϕ parameter affects the time offset in the short term, while the D parameter affects the time offset in the long term.

2.3 Synchronization distribution scenarios

2.3.1 Synchronization protocols

Having established the fundamental concepts of time synchronization and frequency synchronization in section 2.2, the question of how to distribute these notions across a network remains. One approach is to utilize a timing or frequency protocol in conjunction with deploying the necessary technologies to connect the network's devices using the protocol.

The ITU-T categorizes these protocols according to their timing flows, or the method by which the actual timing information is transmitted:

- MESSAGE timing flow: Synchronization is obtained through exchanging messages between nodes. These messages belong to the application layer. This means that a specific application must run between two or more nodes to obtain synchronization. The PTP is an example of a protocol that uses message timing flows. The system is modified at the physical layer from the information contained in these messages.

- SERVICE timing flow: Synchronization is related to a specific service, such as PDH. When a connection is established between two endpoints, a synchronization service is created independently from other connections. Intermediate nodes are transparent to the synchronization service.

- PHYSICAL timing flow: Synchronization is obtained from the actual signal used to transmit data, which is then directly used to synchronize the local clock. This timing flow is node-to-node, without intermediate nodes. SyncE is an example of a physical timing flow.

The actual deployment of these protocols and, therefore, the needed infrastructure may depend on several factors, such as price and synchronization requirements, to comply. This trade-off impact directly on the synchronization upper bound in a network and also in the holdover performance as well.

2.3.2 Packet-based network synchronization topologies

In this work scenario, i.e., packet-based networks, the ITU-T (International Telecommunication Union - Telecommunication Standardization Sector) G-8264 defines three topologies for achieving this distribution: full-timing support (FTS), PTS, and assisted partial-timing support (APTS), which are illustrated in Fig. 2.4.

In an FTS network, all nodes are equipped with PTP aware infrastructure, providing mechanisms such as time boundary clocks and transparent clocks to mitigate PTP inherent problems, such as packet delay variation [13]. Also, these nodes count with support for physical layer synchronization mechanisms such as SyncE.

Integrating PTP and SyncE provides an effective solution for achieving accurate and stable phase synchronization. PTP serves as the primary source for frequency, phase, and time synchronization, while SyncE offers additional frequency stabilization, particularly during holdover periods when PTP is temporarily unavailable.

On the other hand, in PTS networks, synchronization is distributed across nodes that do not have the PTP aware infrastructure PTP packets and may have the equipment to support SyncE. These networks tend to have shorter transmission paths with fewer nodes but can still offer satisfactory performance. Note that in the PTS scenario, generally the use or not of the SyncE is optional, while PTP is either not supported in any node or only partially supported in the network over some (but not all) nodes.

The APTS is a particular case of PTS networks that have APTS clocks at the network edge, as shown in Fig. 2.4. In this case, the primary reference to the T-SC is the GNSS, while PTP or SyncE may be used as a reference in a holdover scenario, i.e., loss of connection or degradation of the primary source.

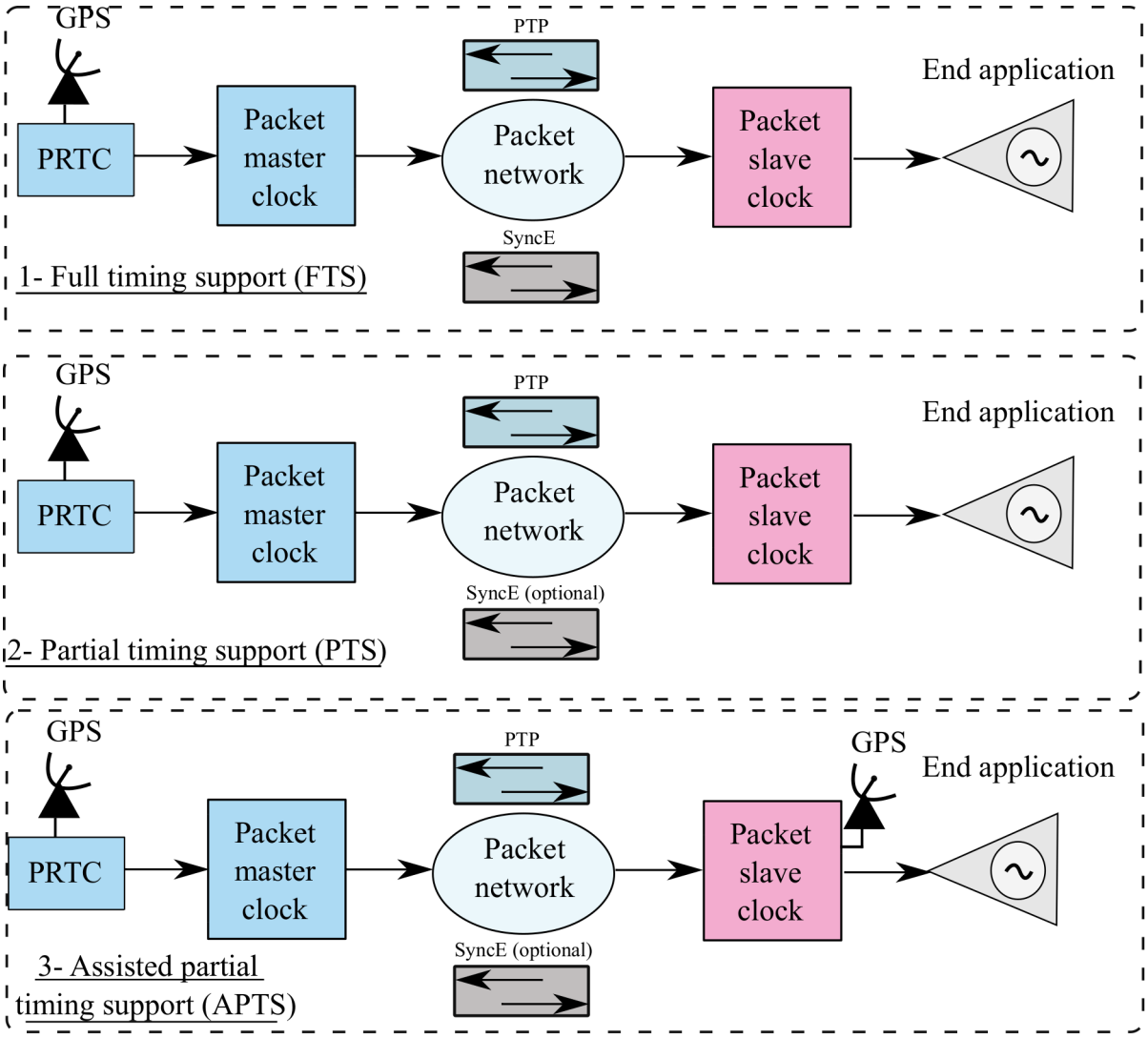


Figure 2.4: Synchronization distribution topologies.

Ultimately, having defined these concepts, it is essential to note that this work situates itself in a particular case of PTS scenario, where we use infrastructure that is PTP unaware and does not have access to SyncE, i.e, we use the PTP as the primary timing distribution mechanism. Thus, when the PTP networks suffer from disruption, the slave clock will not have access to any timing reference during the holdover operation.

The selection of these network topologies is crucial for the network operator, as it determines the protocol or combination of protocols to be used and ultimately affects the equipment used to compose the packet-based network, which in turn has a significant impact on the cost of deploying the network.

2.4 PTP synchronization

The IEEE 1588 protocol [14], commonly referred to as PTP, is a widely used method for providing phase and time synchronization. PTP was developed to synchronize time and phase over Ethernet by exchanging PTP messages. There are currently three different versions of the protocol, with the first version released in 2002, the second version in 2008, and the latest revision in 2019. The latest revision PTP version 2 (PTPv2) provides several enhancements over the previous versions, including support for multicast and unicast communication, improved security features, and the ability to handle more extensive networks with more devices while maintaining backward compatibility with the previous versions.

The PTP protocol utilizes a master and slave hierarchy for clock distribution. The slave clock synchronizes its time and frequency notion through PTP messages containing timestamps sent by the master clock. Within the protocol, there are two types of PTP messages: ordinary messages and event messages. The key difference between these two types of messages is that event messages require timestamps at the ingress and egress points, while ordinary messages do not require timestamps.

Timestamps for event messages are taken when the message passes through a reference point at the node's ingress and egress paths. This reference point can be determined through software or through the use of hardware, such as PTP infrastructure implemented in hardware. Even when using hardware support for timestamps, utilizing software to handle the synchronization process may still be necessary.

The PTP messaging scheme can be executed through either a one-step or two-step process. In a one-step scheme, the egress time is embedded in the message that caused the timestamp. In a two-step scheme, the egress time is sent in a follow-up message instead of being embedded in the event message.

As depicted in Fig. 2.5, an example of synchronization using the PTP message exchange is provided considering a two-step communication scheme, which is the scheme utilized in the current study's implementation. As the timestamps $T1$ and $T2$ are known by the slave clock, it is possible to estimate the time offset as follows:

$$T_{21}[n] = T_2[n] - T_1[n] = x^d + d_{ms}. \tag{2.8}$$

where in (2.8), the values $T_1[n]$ and $T_2[n]$ stand for the ToD timestamps taken respectively from the master and slave, and d_{sm} stands for the slave to master delay.

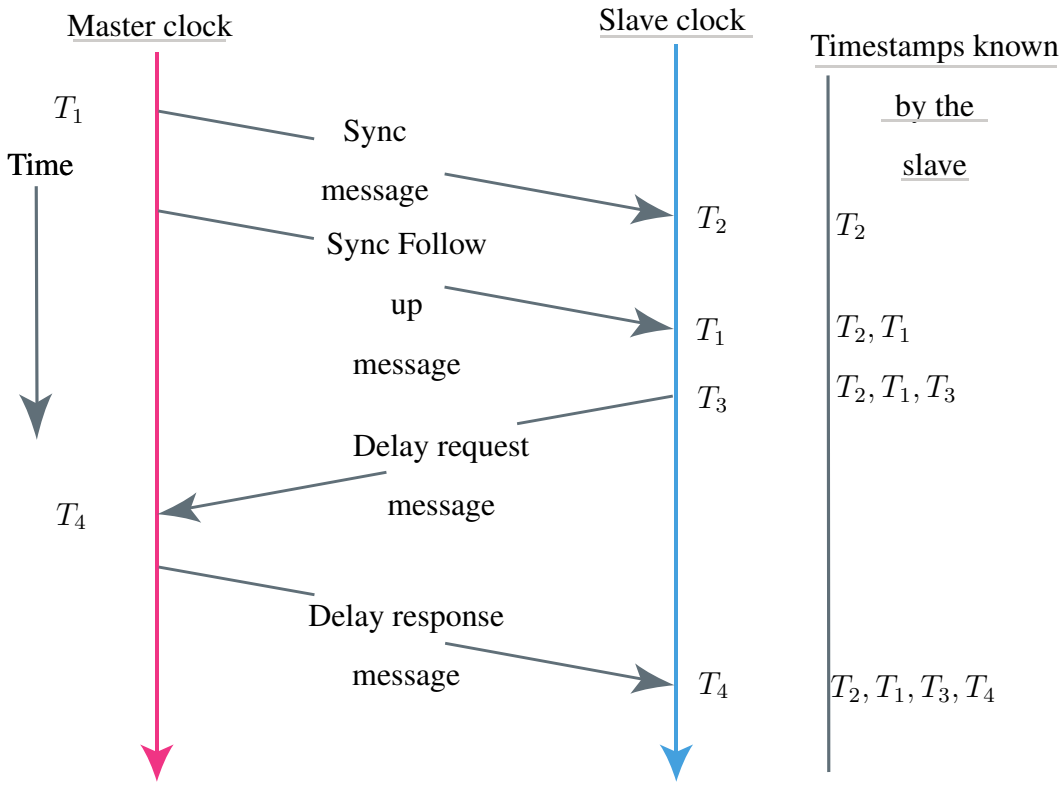


Figure 2.5: PTP delay request-response exchange.

Similarly, when the slave clock gains the knowledge of the timestamps T_3 and T_4, it is possible to calculate the time offset in the opposite direction as in (2.8):

$$-T_{43}[n] = T_4[n] - T_3[n] = x^d - d_{sm}. \tag{2.9}$$

However, both equations mentioned above rely on the knowledge of the slave-to-master and the master-to-slave delays, which the slave clock does not have. As a simplification, the slave can assume that the d_{sm} and d_{ms} are equal, and thus, the slave clock can estimate the time offset through the following the (2.10):

$$\begin{aligned}\tilde{x}[n] &= \frac{(T_{21}[n]) + (-T_{43}[n])}{2} \\ &= \frac{(T_2[n] - T_1[n]) - (T_4[n] - T_3[n])}{2} + \frac{d_{ms} - d_{sm}}{2} \\ &= \frac{(T_2[n] - T_1[n]) - (T_4[n] - T_3[n])}{2}.\end{aligned} \tag{2.10}$$

2.5 Synchronization and clock stability metrics

Clock stability and synchronization are closely related concepts. Clock stability refers to the ability of a clock to maintain accurate timekeeping over a period of time. Synchronization, on the other hand, refers to the process of coordinating the timekeeping of multiple clocks so that they all show the same time.

Clocks must maintain a stable and accurate time reference to achieve synchronization. For example, if an unstable clock loses or gains time significantly, it will quickly become out of sync with other clocks. On the other hand, a stable clock can maintain its time reference accurately, allowing it to remain in sync with other clocks.

In summary, clock stability is essential in achieving synchronization among multiple clocks. A stable clock can better maintain its time reference, keeping it in sync with other clocks.

A common way to measure the synchronization and clock stability is using time error (TE). In the literature, many authors may refer to TE and time offset in Eq (2.4) as the same metric. This work separates these metrics, in order to avoid confusion, and defines the TE as follows:

$$TE = x(t)_{slave} - x(t)_{true}, \tag{2.11}$$

where $x(t)_{slave}$ is the slave clock time offset notion, which came from a noisy measurement, such as the PTP raw measure in Eq (2.10), and $x(t)_{true}$ is the ground truth time offset.

On the other hand, one can achieve a better notion of clock stability using Allan deviation and MTIE metrics, as those take into account the system behavior over time, differently than TE that is a simple sample of difference between the slave and master clocks notions. As an example of that, a constant high TE value would not negatively impact Allan deviation and MTIE.

2.5.1 Allan deviation

The stability of crystal oscillators and atomic clocks has been observed to exhibit not only white noise but also flicker frequency noise. This challenges traditional statistical tools, such as the standard deviation, which cannot converge due to the divergent nature of the noise. Consequently, the standard deviation is not an adequate measure for evaluating the stability of

clock oscillators. To address this issue, David Allan introduced the Allan variance [15], which has now become a widely adopted method for assessing the performance of these devices.

To compute the Allan variance, the time series of the oscillator or clock readings are divided into overlapping intervals, and the standard deviation of the difference between the average of each interval and the average of the entire time series is determined. This variance, also referred to as the two-sample variance, is computed using the following equation:

$$\sigma_y^2(\tau) = \left\langle \sigma_y^2(2, \tau, \tau) \right\rangle \tag{2.12}$$

Where $\langle \cdots \rangle$ denotes the expectation operator, and τ is the observation period. Which is expressed as follows:

$$\sigma_y^2(\tau) = \frac{1}{2} \left\langle \left(y_f^d[n+1] - y_f^d[n] \right)^2 \right\rangle \tag{2.13}$$

$$= \frac{1}{2\tau^2} \left\langle \left(x^d[n+2] - 2x^d[n+1] + x^d[n] \right)^2 \right\rangle \tag{2.14}$$

The definition of Allan deviation, which is similar to the standard deviation and is just the square root of the Allan variance, is as follows:

$$\sigma_y(\tau) = \sqrt{\sigma_y^2(\tau)} \tag{2.15}$$

The preferred method for visualizing the stability of clock oscillators is the Allan deviation over the Allan variance due to several reasons. Firstly, the Allan deviation has the same units as the oscillator or clock being measured, which makes it more intuitive and easier to interpret than the Allan variance, which has units of time squared. Secondly, the Allan deviation is more resistant to non-stationary behavior or frequency changes of the clock, as it is a normalized version of the Allan variance and is less influenced by systematic errors and statistical outliers. Lastly, the Allan deviation provides a better signal-to-noise ratio than the Allan variance, allowing for easier identification of different types of clock noise and a more accurate estimation of the oscillator's stability. Hence, the Allan deviation is the preferred approach for plotting the stability of clock oscillators.

Furthermore, the Allan deviation is particularly useful for estimating stability due to noise processes such as White PM (Phase Modulator) or Flicker PM (related to the quantization noise), the White FM (Frequency Modulator) also known as Angle random walk, the Flicker FM also named BIAS instability and the Random Walk FM (Frequency Modulator). However,

it is not suitable for characterizing systematic errors like temperature effects, although temperature inherently affects the frequency stability and will have a noticeable effect on the Allan variance. The Fig 2.6 provides an example of this noise processes with the Allan deviation, adapted adapted from [16] and [17] as follows:

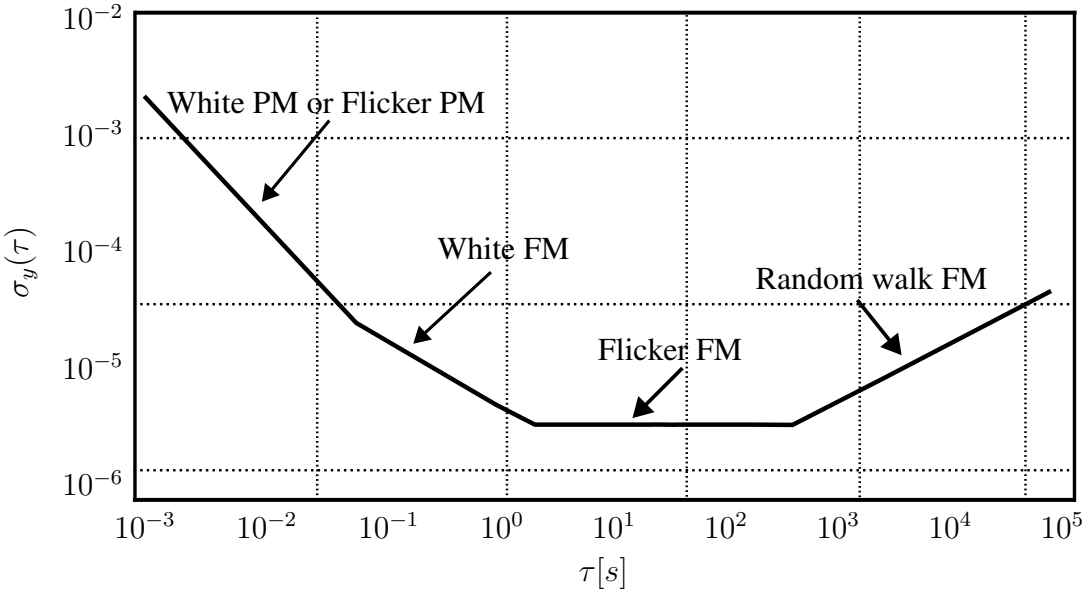


Figure 2.6: Allan deviation example plot.

One of the key advantages of the Allan deviation is that it considers both short-term and long-term stability. Other metrics, such as the standard deviation or the root mean square deviation, only measure short-term stability. By contrast, the Allan deviation provides a complete picture of the stability of a clock or oscillator over a wide range of time scales.

Overall, the Allan deviation is a valuable tool for characterizing the stability of clocks, oscillators, and other systems that exhibit random variations over time. It provides a more comprehensive view of stability than other metrics and has been widely adopted in various fields.

2.5.2 MTIE

The MTIE is an essential measure of clock performance, as it indicates the maximum amount of error that can be expected in the clock's timekeeping over a given period of time. A clock with a low MTIE is considered to be more accurate than a clock with a high MTIE. It is defined as the maximum difference between the clock's indicated time and the true time over a

given time interval. In other words, it is the maximum deviation of a clock's timekeeping from the true time its estimation is stated in the equation as follows:

$$MTIE(\tau) = \max_{n_0<n<T_m-\tau} \left(\max_{n<i<n+\tau} [x[i]] - \min_{n<i<n+\tau} [x[i]] \right), \tag{2.16}$$

where τ value represents the observation period, T_m is the total length of the measurement, and n_0 is the timestamp of the initial value.

One can interpret (2.16) as a sliding window of size τ that slides starting in the value n_0 and ends in the value $n_0 + T_m$. When the sliding window reaches the final value, the maximum and minimum values of time interval error (TIE) are computed. With this, the $MTIE(\tau)$ uses these values to produce its value. This process is repeated multiple times, and at each iteration, the τ window is incremented. In the final iteration, the τ window has the same length as the T_m window.

Unlike the TE metric, the MTIE has a monotonic behavior. This metric is related strongly to the concept of stability. As an example of that, the TE value in a determined time can be a high value, but if the TE have a low variation, the MTIE will contain low values.

2.6 Synchronization requirements in telecommunications

2.6.1 Timing requirements

The field of wireless telecommunications is constantly evolving, driven by the increasing demand for higher data rates, lower latency, and more reliable and stable connections. Emerging technologies are further pushing this trend, such as the advent of automated vehicles, remote surgery, and even higher-resolution video streaming. All of these applications require stable and fast communication to function effectively. As a result, the requirements for telecommunications networks must also evolve.

Ethernet is well-suited for radio access networks due to its wide availability and cost-efficiency. However, it does not inherently provide synchronization, as it is based on asynchronous packet transmissions.

In contrast, synchronous transmission schemes allow the receiving end to recover the sender's frequency. For example, Long term evolution (LTE) base stations that use the TDD scheme require accurate time synchronization to align TDD frames among adjacent TDD cells.

The Precision Time Protocol PTP is well-suited for this scenario, providing sub-microsecond time synchronization to the network. The ITU-T G.8271.1 [18] standard, as outlined in Table 2.1, has developed a series of phase and time error limits that different applications must adhere to in order to function correctly.

Table 2.1: Time error (TE) requirements.

Level of accuracy	TE requirement (error with respect to a common reference)	Type of application
1	500 ms	Billing, alarms
2	100 μs	IP delay monitoring, Asynchronous Dual Connectivity
3	5 μs	LTE TDD (large cell), Synchronous Dual Connectivity (for up to 7 km propagation difference between eNodeBs)
4	1.5 μs	UTRA-TDD, LTE-TDD (small cell), WiMAX-TDD (some configurations), Synchronous Dual Connectivity (for up to 9 km propagation difference between eNodeBs)
5	1 μs	WiMAX-TDD (some configurations)

This table exemplifies various use cases and the requirements associated with them. A newer version of the recommendation, G.8271.1 [19], focuses on the 1.5 μs TE budget, providing different scenarios and specific TE baselines. Fig. 2.7 describes a general end-to-end application of this TE budget, that is, the error between the end application time clock to the PRTC. This total end-to-end budget covers various types of errors caused by intrinsic behaviors of a packet-based network, like associated asymmetry errors and dynamic TE (jitter and wander components of the timing signal).

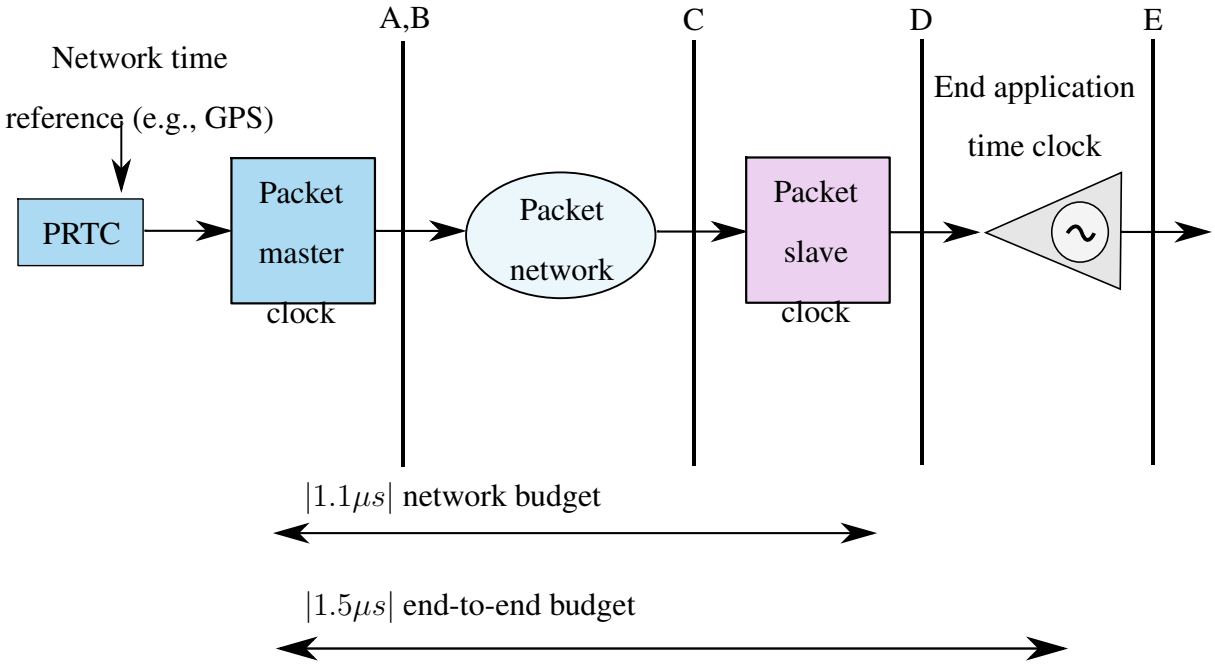


Figure 2.7: TE budget.

2.6.2 Frequency requirements

Alongside time synchronization, frequency synchronization is a vital component in mobile networks [11] [20], particularly when it comes to handovers of user equipment (UE) between cells. This process necessitates the UE to precisely lock onto the new timing reference at the correct frequency. To ensure compliance, the timing recovery process of mobile equipment and the pull-in range must adhere to specific requirements for the frequency error that can be tolerated in the signal emitted by the base station (BS). Additionally, mobility can result in Doppler effects. Thus, a margin must be added to account for this potentiality, particularly in high-mobility environments. To set the limit on the total frequency error between a UE and the connected network, there must be a maximum tolerance for the frequency difference generated over the radio interface. Furthermore, compliance with frequency synchronization plays a significant role in fulfilling regulatory requirements and certifying that radio signals are generated with strict adherence to frequency accuracy standards.

Besides the macro necessity to provide frequency synchronization, requirements must be met depending on the transmission scheme adopted. In this context, two central transmission schemes separate uplink (UL) and downlink transmissions, the TDD, described in the previous section, and frequency-division duplex (FDD). Differently from TDD, FDD allows uplink and downlink transmission simultaneously and uses distinct frequency bands to do so. Both methods

must comply with frequency requirements described in Table 2.2.

Table 2.2: Frequency accuracy requirements.

Frequency accuracy	Technology
50 ppb	WCDMA
50 ppb	TD-SCDMA
50 ppb	LTE-FDD
50 ppb	LTE-TDD

Chapter 3

Holdover operation

Holdover operation in synchronization refers to the mode of operation of a synchronization system when it continues to maintain its synchronization without a primary reference source. It is an essential component of network timing systems, ensuring that the network continues to function even in the event of a failure or disruption of the primary reference source.

The importance of this particular mode or operation in synchronization lies in its ability to maintain the accuracy of time-sensitive processes and the continuity of operations without a primary reference source. This is critical in scenarios where reliable and accurate timing is essential, such as in financial transaction processing systems, telecommunications networks, and power generation and distribution systems.

In the context of time-sensitive network infrastructure, holdover is crucial in maintaining synchronization during primary source failure. This can occur for various reasons, including equipment malfunction and maintenance, network infrastructure rearrangement, and bad or corrupted synchronization signals from the primary reference. Holdover provides a temporary solution, allowing technicians sufficient time to repair or reconfigure the network and restore primary synchronization.

This chapter will provide a comprehensive definition of holdover and free-running modes, which Section 3.1 thoroughly describes. Subsequently, Section 3.2 will provide an overview of holdover scenarios in the context of FTS, PTS, and APTS scenarios. Following this, Section 3.3 will delve into the related works, their results in similar tests to this work, and how our work is situated within the related literature. Finally, Section 3.4 will discuss the requirements for successful holdover operation, including the role of technologies such as PTP, GNSS, and SyncE in ensuring reliable and accurate holdover operation.

3.1 Free running and holdover modes

This section introduces two central concepts crucial to this work: free-running and holdover modes. As detailed in the ITU-T Recommendation G.810 [5], the free-running mode commences when a clock loses communication with its external reference and can no longer control its phase or frequency. This mode persists until the clock reestablishes communication with the reference.

As illustrated in Fig. 3.1, consider two clocks, the reference clock A and the local clock B. At $t \leq 0$, the two clocks are synchronized in both time/phase and frequency. However, at $t > 0$, the local clock loses communication with the reference and experiences a change in its oscillator frequency, resulting in a constant frequency offset between the two clocks. Consequently, the time offset between the two clocks will increase steadily unless the local clock reestablishes communication with the reference. It is important to note that in real-world scenarios, the frequency offset between the two clocks is unlikely to remain constant. Thus the time offset will not increase at regular rates.

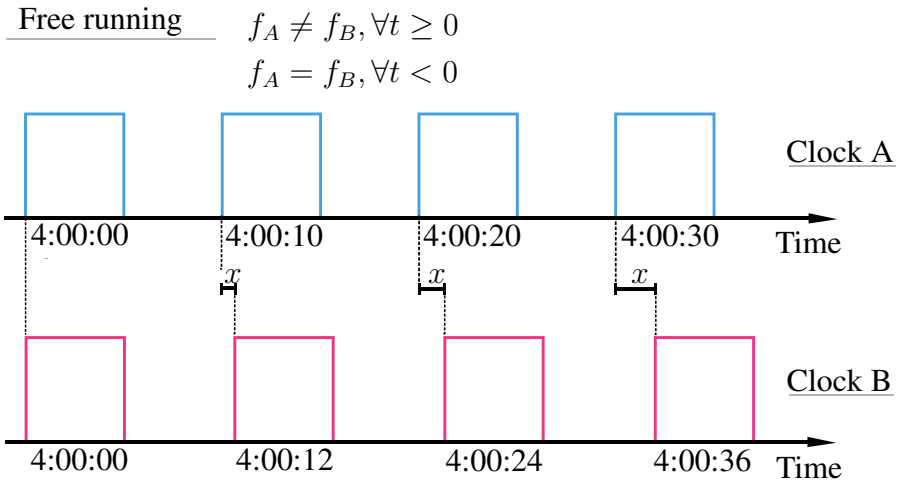


Figure 3.1: Free running slave clock

In contrast, the holdover mode involves the local clock accessing stored data from the external reference, allowing it to control its phase and frequency. With the use of hardware or software, the local clock can rely on this stored data to maintain an acceptable level of synchronization by controlling its output phase and frequency.

The holdover mode is formally defined by the ITU-T Recommendation G.810 [5] as an operating condition in which a clock has lost its controlling reference input and uses stored data

acquired during locked operation to control its output. The stored data is utilized to control phase and frequency variations, enabling the locked condition to be replicated within specifications. Holdover commences when the clock's output is no longer influenced by a connected external reference or in transition from it, and terminates when the clock reverts to the locked mode condition.

In the literature, some authors may define the state of the slave clock after losing its reference as a holdover, even without a mechanism purely focused on holdover operation. Nevertheless, it makes sense, as these authors use a disciplined scheme like a phase locked loop (PLL). As a causal mechanism, it stores information and takes time to fully stabilize (considering a scenario with an ideally stable reference). When the reference is lost, it has stored within the last known frequency. Thus, when the PLL operates in an open loop after it has a notion of the frequency, it can be considered a holdover mechanism. On the other hand, if this mechanism does not yet store data, the slave clock can be considered in a free-running state.

3.2 Holdover scenarios

Taking into account the synchronization scenarios expounded upon in the synchronization chapter (FTS, PTS, and APTS), the holdover operation may arise due to a myriad of reasons. One of the probable causes of holdover is the lack of signal from the GNSS in the PRTC. This scenario may come about owing to various factors, such as obstructions, electromagnetic interference, and malfunctioning of satellites.

In the event of a loss of GNSS signal in the PRTC, all synchronization scenarios enter a state where, despite the loss of GNSS signal, the network remains intact, and the synchronization between the slaves and the PRTC is still active. However, it is essential to note that the PRTC will slowly drift, and the network must maintain a certain level of synchronization to function correctly.

Although this particular holdover situation will not be addressed in our work, it is worth mentioning that it is perhaps the most common one, as GNSS synchronization and other over-the-air-based synchronization are more susceptible to disruption compared to protocols that rely on physical infrastructure, such as PTP and SyncE. Nevertheless, as our approach is generalistic, it could also be adapted to this situation.

3.2.1 FTS holdover

In FTS networks, there are several possibilities for providing a temporary source while in holdover, which depends on where the synchronization path was disrupted. In the event of a disruption in the PTP synchronization network, the SyncE must supply a temporary primary source to the slave clock. This specific situation has explicit requirements outlined in ITU-T G.8273.2 [7] and in ITU-T G.8172.1 [19]. These requirements are further elaborated in Section 3.4 of this chapter.

If the physical frequency input is lost, the FTS network will have to rely solely on the PTP network, similar to the PTS topology. Therefore, some degradation in the frequency control is expected. However, as an FTS has access to PTP-aware infrastructure, the synchronization quality will remain at high levels and may even be better than that of a PTP in normal operation.

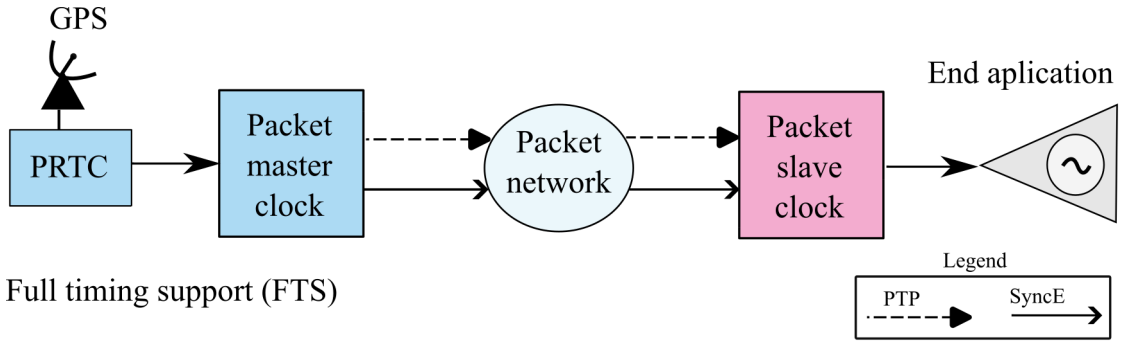


Figure 3.2: FTS synchronization paths

3.2.2 PTS holdover

As discussed in the previous subsection 3.2.1, the use of SyncE as a frequency reference during the holdover period is highly effective. However, in the case of PTS networks where SyncE may not be present, the slave clock must rely solely on its oscillator or another mechanism for synchronization. Typically, a PLL circuit is utilized as the mechanism in such networks, which is responsible for disciplining the local clock to the reference phase source. During holdover, this PLL operates in free-running mode, attempting to maintain the last known frequency.

This research is conducted in the scenario mentioned above, utilizing an IEEE 1588 PTP testbed as the primary and only reference for the slave clock, along with PTP-unaware switches. The main contribution of this work is the development of a software-based holdover mechanism

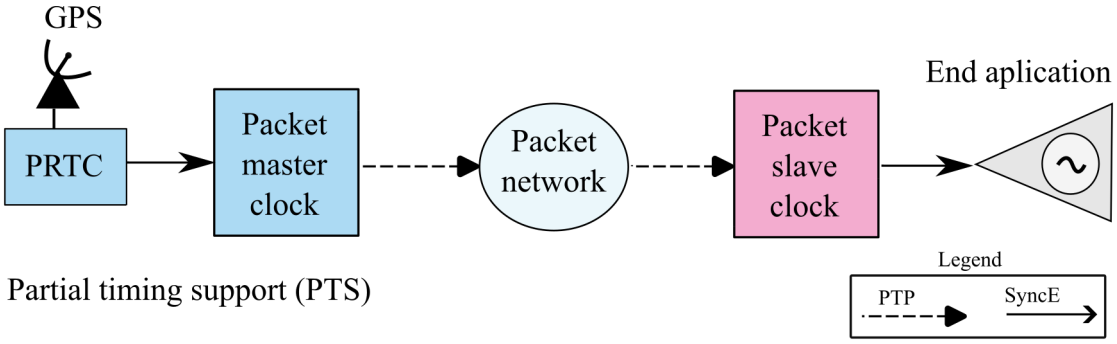


Figure 3.3: PTS synchronization paths

that can aid the slave clock during holdover operations without requiring the implementation of additional hardware.

3.2.3 APTS holdover

The APTS represents a specific instance of a PTS network, wherein the slave clock possesses access to a GNSS reference in addition to the PTP communication reference. Fig 3.3 illustrates this scenario. In this setup, the PTP communication operates as a backup reference in the absence of GNSS signals, rendering PTP as the primary holdover mechanism.

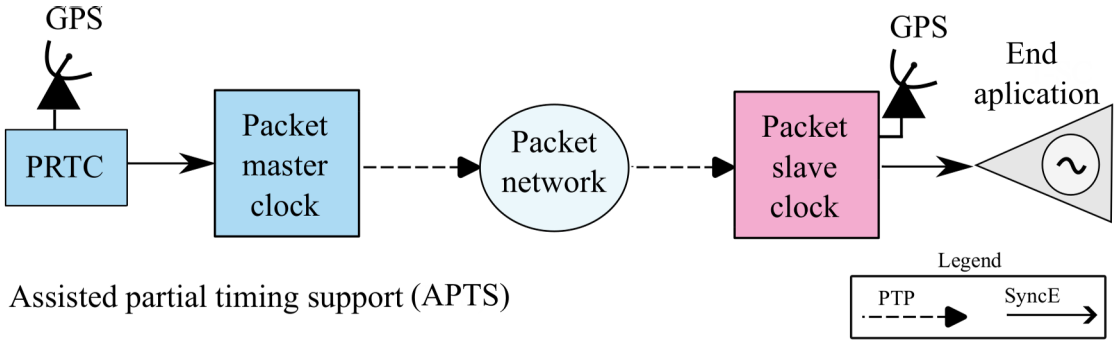


Figure 3.4: APTS synchronization paths

If both GNSS and PTP are disrupted, the slave clock depends solely on its oscillator discipline mechanism to maintain the last-known frequency, much like the PTS scenario. Consequently, our research could potentially support the slave clock as a final resort in all packet network synchronization schemes delineated in this work, and it can be adapted for GNSS holdover or other over-the-air-based synchronization schemes. The Chapter 6 elaborates on the specifics of our software-based holdover solution.

3.3 Related works

The literature about holdover operations can be broadly classified into three categories. The first category comprises studies examining the clock's robustness and its oscillator during holdover situations. The second category includes studies that utilize statistical software mechanisms to assist the local clock during holdover. The third category comprises studies that adopt a machine-learning approach to assist the slave clock when the reference is lost. This current work falls under the third category and also engages in comparison with statistical studies. Due to the nature of our datasets and their acquisition from the testbed, it is not feasible to evaluate the inherent robustness of the clock during simulated holdover situations. However, our results can still be compared with existing studies in this field.

3.3.1 Clock robustness tests in holdover mode

In clock robustness analysis in holdover scenarios, the literature plays a crucial role in providing insights into the performance of different types of clocks in controlled environments that emulate conventional deployments. This is accomplished by evaluating the accuracy of the clocks during holdover situations, with a specific focus on a fixed holdover horizon. In this work case, the horizon focused is of 1000 seconds. The studies cited in this context contribute to understanding the synchronization infrastructure industry by providing valuable information regarding the performance of various clock types under holdover conditions.

In that sense, [21] evaluate the T-SC holdover performance with different types of oscillators alongside the use or not of frequency physical layer input (SyncE). The authors conclude that their system using a double oven-controlled crystal oscillator (DOCXO), OCXO, or even a TCXO may comply with the MTIE masks from [7] if the system has access SyncE input during the holdover. However, the authors did not consider a case where the T-SC uses a low-cost oscillator such as XO, nor a case where the T-SC has access to some software-based holdover mechanism or a case where the temperature variation was above 1 Kelvin.

The results of [21] can serve as a reference for this work, with the caveat that a direct comparison of the results is not feasible due to architectural differences. These differences include the absence of perfect PTP synchronization prior to the holdover operation in this work, the availability of two clocks to provide a local slave clock (either an XO or OCXO), and note that our OCXO may not necessarily be equal to the OCXO employed in [21]. Despite these

differences, the results obtained by [21] are presented in Table 3.1.

Table 3.1: Results obtained by [21]

Type of holdover	MTIE (ns) at 1000 seconds		
	DOCXO	OCXO	TCXO
Time holdover without SyncE assist	31	65	3922
Time holdover with SyncE assist	23.56	12.0	6.4

Other studies in this field include [22]-[23], which carried out experiments in a smart grid scenario. These experiments required the establishment of standard metrics and testing methods for evaluating the performance of network systems and components in the next-generation substations. Both studies implemented an IEEE 1588 PTP Testbed for the power industry and focused on synchronization. To streamline the testing process, they employed a software-based testing dashboard, which facilitated the evaluation of the numerous requirements for accuracy, reliability, and interoperability.

The studies carried out various tests, including holdover operation robustness tests, using a network architecture that comprised multiple PTP-aware hops, each equipped with a slave clock. Each slave clock was tested in holdover operation for 1000 seconds, yielding TE metric results.

For a proper comparison between these prior studies and the present work, one should focus on the results of the clock in the first hop, or closer to the first hop as possible, of the network topology and pay attention to the type of clock oscillator used in each study. However, [23] does not clarify in which hop each clock was located in the holdover experiment. On the other hand, [22] explicitly informs the hop of the slave clock but intensionally hides the slave clock type to maintain the testbed's neutrality. This way, each clock may use either a TCXO or a OCXO. Nevertheless, considering that the slave clocks started the holdover operation from a TE close to 0-100 ns, we can consider their results a baseline for our work. The results of both studies are summarized in Table 3.2 using the Max $|\text{TE}|$ metric, as the authors did not provide MTIE results.

Table 3.2: Holdover results obtained by [22]-[23] considering a holdover duration of 1000 s

Author	Oscillator type	Hop	Max $\|TE\|$ in holdover (ns)	Max $\|TE\|$ before holdover (ns)
[23]	OCXO	-	2487	55
[23]	TCXO	-	4710	49
[22]	-	1	5936	96
[22]	-	3	10256	80
[22]	-	4	9400	96
[22]	-	4	3616	120
[22]	-	4	1736	80

3.3.2 Statistical-based holdover applications

Holdover solutions address two primary issues: compensating for temperature and aging-induced frequency deviations in the oscillator. These problems have different time frames, with the aging problem being a concern only in long-term holdovers (24 hours or longer) and temperature-induced deviations being a common problem in all holdover periods.

The authors in [24] tackle both issues using a Kalman filtering-based holdover synchronization approach. Both models use a digital PLL that provides the correction signal during the holdover mode. The authors tested these models using both simulated data and a real testbed.

The study in [6] evaluates the use of an autoregressive moving average (ARMAX) holdover-based approach to provide correction signals to the T-SC. During the controlled state, when locked to its reference, the algorithm running on the T-SC identifies the ARMAX parameters through a recursive error prediction approach. Once in holdover mode, the proposed algorithm provides correction signals to the local oscillator. However, unlike [24], the authors only tested the performance through simulation.

Equipment failure and maintenance are the typical use cases for the holdover mode. However, there are also other applications where this mode can benefit the T-SC's timing notion. For instance, [25] proposes an FIR-based holdover model to provide an alternate synchronization source in the presence of noisy signals. On the other hand, [26] proposes a Kalman-based holdover model to save energy by deliberately shutting down GNSS-based synchronization and relying on its holdover model as a temporary timing source.

Studies that focus on temperature compensation are not always directly applicable to the

holdover mode, but they have the potential to be useful within this area of research. One such study is presented in [27], which proposes a software-defined temperature compensated crystal oscillator TCXO.

This software-defined TCXO is an XO oscillator with a software-defined temperature compensation mechanism based on the Hammerstein-Wiener dynamic, non-linear model [28]. The authors utilize a self-learning approach to fit the parameters of the Hammerstein-Wiener model, they argue that this could compensate for frequency variations during holdover operation or in the presence of steep temperature variations.

However, the authors did not test the software-defined TCXO in an actual synchronization scenario. Instead, their experimental setup consisted of a computer that measured the oscillator frequency and a device to control the temperature to fit the non-linear model. Although a proper holdover test was not conducted, the authors' work still highlights the importance of incorporating mechanisms to improve clock performance, regardless of the quality of the oscillator, further advancing the field of study outlined in our work.

3.3.3 Machine learning based holdover applications

In this context of predicting the future value of the correction signal or the timing variable directly, the holdover applications fit well in the context of time series forecasting, for which the time series literature contains multiple approaches. After [29] introduced the transformer model in natural language processing, multiple authors proposed to use it in the time series forecasting scenario, representing the actual state of the art in this field. For example, [30] proposed an extended sequence time series forecasting model that outperformed LSTM, ARIMA, LogTrans, facebook's prophet, and many others testing in multiple benchmark datasets.

There are also ML investigations in the algorithm-based holdover field. For example, the works on [31], and [32] investigate a scenario where a ML-based model learns from a control module that disciplines the local clock and provides control signals during the controlled period. In holdover mode, the ML generates the control signal to the local oscillator. The main difference between both works is the ML algorithm chosen by the authors, [31] uses a adaptive neuro-fuzzy inference system (ANFIS) while [31] use support vector machine inference system (SVM-FIS) model. However, both authors do not provide any comparison of statistical models nor use the temperature as an input to the model, even though they consider the environmental effects (e.g., vibration, temperature, pressure, and humidity) as a noise source.

There are two major synchronization schemes in algorithm-assisted holdover studies. [6, 24, 31, 25], apply algorithms focusing on GPS disciplined oscillators. The other use case uses a RBS testbed [32]. Hence, none of these references proposes holdover algorithms for a PTP synchronization scheme, which suffers from unique problems, e.g., packet delay variation (PDV), link asymmetry, and packet loss.

The previous holdover studies that consider the effects of aging and the temperature effect, [6]-[24], adopt statistical methods, which can be surpassed by ML methods in many cases. The other two studies in [31]-[32], use ML methods but do not consider an environmental input, such as temperature.

This research proposes a holdover synchronization methodology utilizing a software approach. Specifically, we implemented three distinct models, namely autoregressive integrated moving average with exogenous variables (ARIMAX), a LSTM, and a transformer network, and assessed their efficacy using datasets acquired from an IEEE 1588 PTP testbed. The ARIMAX and LSTM models are regarded as robust models in the literature concerning time series, while the transformer network is a state-of-the-art model. For our investigation, we employed only the decoder portion of the transformer, similar to the approach employed in [33], and conducted long sequence prediction, much like [30]. Our proposed model aims to offer ML holdover that directly forecasts future time offset while accounting for temperature effects to enhance the performance of low-cost XO and even robust clocks, such as OCXO based ones. As such, we intend to expand upon our previous work [8] by utilizing a novel model and comparing its performance with that of our previous approach (LSTM) and ARIMAX. Also, we intend to add to the related works, providing novel experiments using datasets containing real data acquired from a PTP dataset.

3.4 Holdover requirements

Considering the holdover requirements, it is ideal for the slave clock to comply with the same specifications as during normal operation, as stated in subsection 2.6. However, specific requirements are necessary for the holdover operation, considering that this state is transient and the slave clock will eventually recover its reference. These requirements are designed to meet time horizons, with the two most commonly considered time horizons in holdover literature being 24 hours and 1000 s. Our work focuses solely on the requirements pertaining to the latter,

given the limitations of our testbed capture scheme, which can only accommodate a dataset of 6 hours. Therefore, it is unfeasible to concentrate on the longer holdover time horizon.

Considering possible failure scenarios in a FTS network, the recommendation ITU-T G.8271.1 [19] establishes holdover mode TE baselines for the FTS scheme, considering a short time horizon holdover, e.g., 1 minute, as described in Table 3.3:

Table 3.3: Holdover mode TE baselines.

TE requirement	Failure scenario
250 ns	a) Failures in the synchronization network that cause the end application clock to enter holdover for a short period (TE_{REA}).
400 ns	b) Failures in the synchronization network that do not cause the end application clock to enter holdover (TE_{HO}).

The failure scenario a) considered in Table 3.3 considers rearrangements that force the slave clock to enter a holdover state. For example, this might be triggered by a loss of PRTC traceability of one of the redundant master clocks in the network [19]. On the other hand, failure scenario b) considers a situation where the master clock temporally loses the GNSS signal, causing the master clock to enter a holdover state, but the slave clock operates normally. For this work, even though we consider a PTS network, we can also use the failure scenario a) as a baseline for our work.

The main application of this work focuses on a scenario where T-SC hypothetically loses the communication to its reference, the PRTC, and enters holdover mode. Therefore, the applicable baseline for this work is the TE_{REA}, where the end application enters in holdover for a short period, stated in [19] as 1 minute.

Now considering specifically the FTS network, ITU-T proposed two different requirements for holdover operation. The first is ITU-T 8273.2 [7], which proposes MTIE masks to express the frequency stability requirements when the slave clock enters in holdover state while maintaining access to a SyncE input, considering two temperature situations, variable and negligible (or constant), which is described in the Fig. 3.5.

The MTIE masks described in Fig. 3.5 denote two scenarios of holdover mode in a T-SC, where the slave clock loses the PTP communication and relies only upon the frequency layer physical input, using SyncE as the sole input. The first mask considers a constant temperature

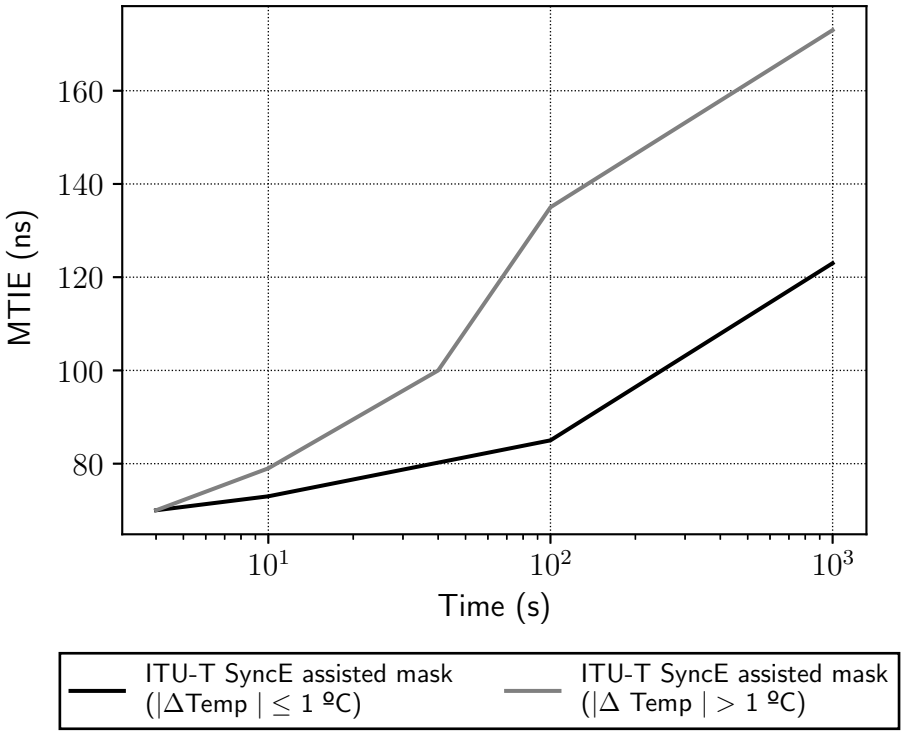


Figure 3.5: ITU-T G.8273.2 MTIE masks.

scenario, where the temperature should not exceed 1 Kelvin. The other mask considers a variable temperature scenario, where the temperature varies more than 1 Kelvin. This work does not use the SyncE protocol to deliver frequency synchronization. However, this work's results use these MTIE masks as baselines.

Also focusing on this particular holdover horizon, the ITU-T 8273.4 [7] states the requirements in terms of ΔTE, i.e, phase error, regarding PTS scenario, which Fig. 3.6 describes. The phase error is the amount of TE variation in the observation period, in this case 1000 seconds. Differently from the previous requirement, this case refers to the case where the slave clock does not have access to any reference. Thus, this requirement is entirely compliant with this work holdover operation tests. Thus, it will also be used in this work as a baseline.

In conclusion, the requirements directly applicable to this work results as baselines are the TE requirements stated in situation a) in the Table. 3.3 (TE_{REA}), the permissible TE stated in the Fig. 3.6, and the MTIE masks described in the Fig. 3.5.

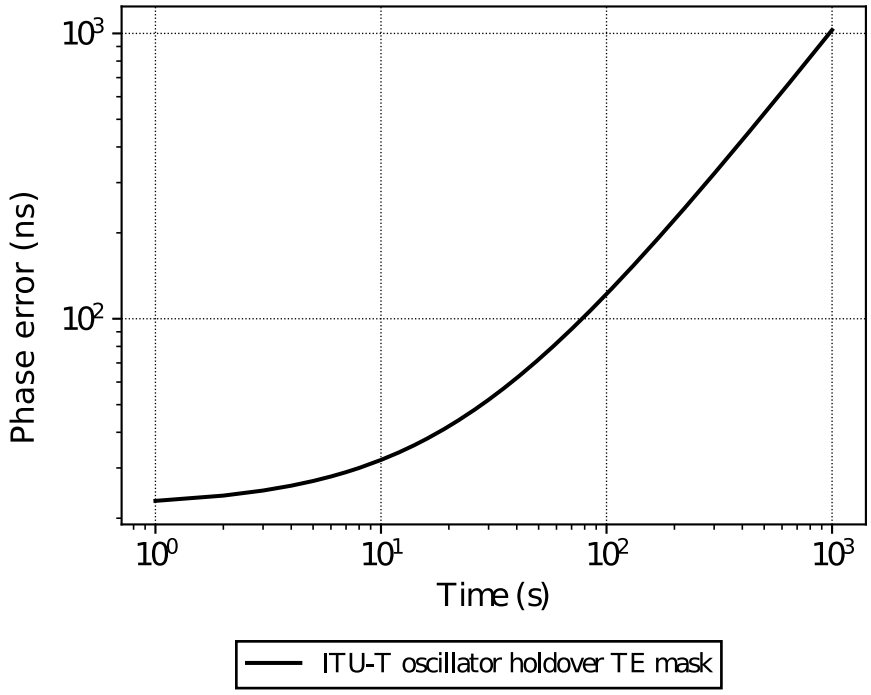


Figure 3.6: ITU-T G.8273.4 Permissible phase error under holdover operation at constant temperature

Chapter 4

Time series

This section will delve into the fundamental concepts of time series. As previously discussed in Section 2.2, the time offset is a problem that falls within the realm of time series. This work aims to address this problem and predict future time offsets. As such, these foundational concepts are crucial for understanding the proposed method outlined in this work.

According to [34], a time series is a set of observations recorded at specific times. On the other hand, a discrete-time series refers to observations recorded at specific intervals, resulting in a set of discrete observations of a particular variable.

One example of a time series is the classic dataset used by Box & Jenkins [35] in Fig. 4.1:

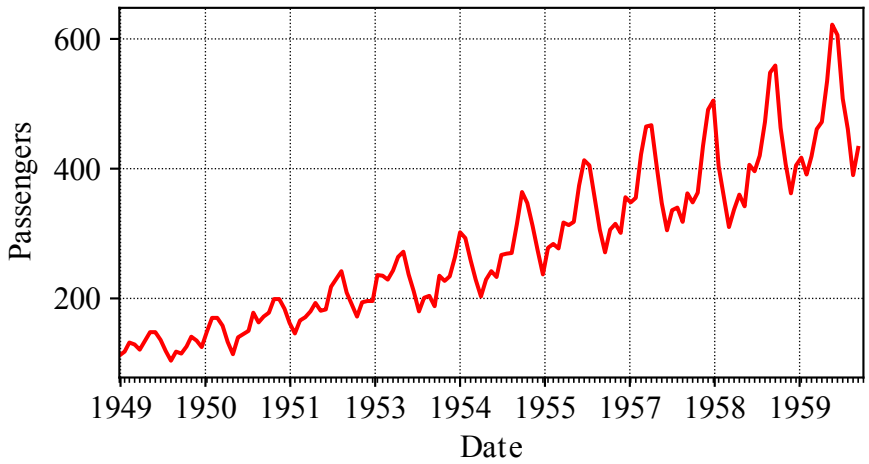


Figure 4.1: Airline passengers time series data.

In deterministic systems, two distinct categories of time series are present. The first category comprises deterministic time series, i.e., the series under this category are defined through

an analytical expression and are devoid of any random variables in their formulation. The second category comprises non-deterministic time series, which cannot be defined through an analytical expression as then encompasses random elements that hinder its description through analytical means.

There are several reasons why a time series may be non-deterministic. The first reason is insufficient information to describe the series entirely. The second reason is the presence of intrinsic random behaviors inherent in the nature of the time series. It is important to note that non-deterministic time series are stochastic processes and, therefore, this series can be analyzed based on probabilistic aspects instead of analytical ones.

In the time series literature, a non-deterministic time series can be defined as either a realization of a stochastic random process or a stochastic random process, depending on the author. This present work primarily focuses on non-deterministic time series and, for the sake of simplicity, denotes a time series process as a stochastic random process and the time series as a realization of this process.

In the field of time series analysis, two commonly referenced examples of stochastic processes are the white noise process and the random walk process, as outlined in [34]. These processes serve as fundamental models for understanding and analyzing non-deterministic time series. The white noise process, for instance, is a process that is characterized by zero temporal correlation between observations. In contrast, the random walk process is a sequence of independent and identically distributed random variables. These processes have been widely studied and used in time series analysis due to their simplicity and generality. Every realization of those processes yields a time series, shown in Fig. 4.3 and Fig. 4.2, respectively.

The main objective of time series analysis is to elaborate a forecast model that can estimate the future steps of this series. One way to achieve this is to model the time series process. However, this process could be more practical, as the series' nature may depend on multiple random variables and complex relationships between them. A way to achieve this is to create a model that uses past observations to estimate the series' future steps.

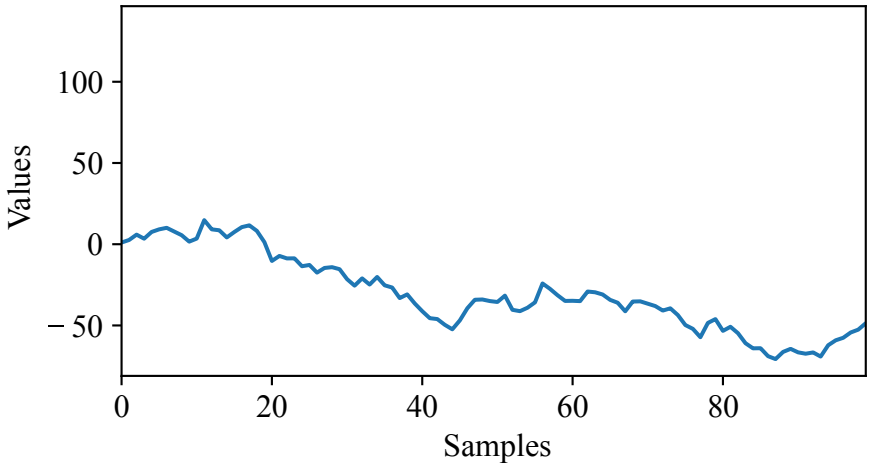


Figure 4.2: Random walk time series.

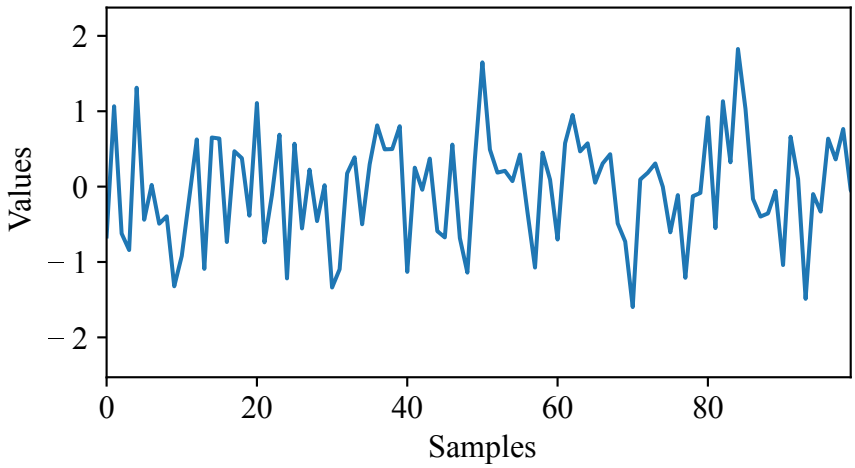


Figure 4.3: White noise time series.

4.1 Stationary models and time series

In order to properly analyze a time series problem, it is vital first to understand the concept of stationary processes. There are two main types of stationary processes: wide-sense stationary (wide-sense stationary (WSS)) and strict-sense stationary.

In simple terms, if $Z[n]$ is a time series process, it is WSS if it exhibits statistical properties that are similar to those of its time-shifted version $Z[n+h]$. Formally, a WSS time series process is characterized by its first and second-order moments and covariance function. Therefore, it is essential to understand the definitions of these functions to formulate the WSS conditions.

A moment of j-th order is defined by [36] as follows:

$$E(Z[n]^j) = \sum_{a=-\infty}^{\infty} a^j p_Z(a; n), \tag{4.1}$$

where Z is a random variable representing the time series process, $p_Z(a; n)$ is the probability density function of the random variable at time n, and a is a set of integers. The first-order moment corresponds to the mean function of the random variable.

The covariance function, as defined by [36], is given by:

$$Cov(A, B) = E\Big(E(A - E(A))E(B - E(B))\Big). \tag{4.2}$$

The covariation in (4.2) can be interpreted as a measure of the degree of joint variability between the random variables A and B. In the context of time series analysis, this equation represents the autocovariance function of a time series $Z[n]$ and its time-shifted version $Z[n+h]$.

Based on the concepts as mentioned earlier, the conditions that define a WSS time series process $Z[n]$ can be expressed as:

- $E(Z[n]^2) < \infty$,
- $E(Z[n])$ should be independent of n,
- $Cov(Z[n], Z[n + h])$ should be independent of n for any h.

Understanding the statistical properties of a time-series process that possesses the WSS conditions is crucial for effectively dealing with such a time series before fitting it into any model. This is because a WSS process is more predictable and less variable compared to non-WSS processes. Accordingly, by employing these conditions, one can efficiently classify a time series process as either WSS or non-WSS. For instance, a white-noise process can be classified as WSS, while a random walk process is non-stationary in terms of WSS conditions.

Furthermore, some operators can transform a non-stationary time series into a WSS one by altering the series' statistical properties. The differentiation operator is an example of such an operator, and the following Eq (4.3) defines it as:

$$\Delta Z[n] = Z[n] - Z[n - 1]. \tag{4.3}$$

this differentiation can also be applied using lags greater than 1, namely being a seasonal differentiation :

$$\Delta_d Z[n] = Z[n] - Z[n - d] \tag{4.4}$$

where the parameter d represents the size of seasonal differentiation. Typically, d has the same value as the period of the time series process $Z[n]$.

Understanding the concept of time series components is essential for comprehending how these operators can modify the time series' inherent statistical properties to attain the WSS conditions. The subsequent subsection introduces this concept.

4.2 Time series components

A time series can be expressed as the sum of different components, as shown by the equation below:

$$Z[n] = m[n] + s[n] + Y_{noise}[n]. \tag{4.5}$$

Here, $m[n]$ is a slowly changing function known as the trend component, $s[n]$ is a function with a known period d referred to as the seasonal component, and $Y_{noise}[n]$ is the random noise component that satisfies the conditions of a WSS process. These components can be observed in Fig 4.1. The trend component is noticeable as the Airline passengers time series has a clear upward trend, and the presence of the seasonal component can be observed in the periodic peaks in the data.

In addition to visual inspection, various functions can extract information about a time series and its components. One such function is the autocorrelation coefficient function, which was first defined by Pearson [37]. This function can also provide information about the statistical properties of a time series and is given by:

$$\rho_h(Z[n]) = \frac{Cov(Z[n], Z[n+h])}{\sqrt{(E(Z[n]^2) - E(Z[n])^2)}\sqrt{(E(Z[n+h]^2) - E(Z[n+h])^2)}}, \tag{4.6}$$

for simplicity the term $\sqrt{(E(Z[n]^2) - E(Z[n])^2}$ can be substituted with the term $\sigma_{Z[n]}$, that represents the standard deviation function, resulting in:

$$\rho_h(Z[n]) = \frac{Cov(Z[n], Z[n+h])}{\sigma_{Z[n]}\sigma_{Z[n+h]}}. \tag{4.7}$$

It should be noted that this section refers specifically to the autocorrelation function as used in statistics and that different fields may use different definitions. For the purposes of this work, all references to the autocorrelation function or coefficient will refer to the autocorrelation coefficient function as defined by Pearson [37] in (4.7).

The autocorrelation, described in (4.7), applies to a time series process $Z[n]$ defined by its stochastic process equation but cannot be directly applied to a realization of this process, denoted by $z[n]$. In such scenarios, there exist methods that can estimate (4.1), (4.2), and (4.7) for realizations of time series processes. Specifically, the estimation of the mean function is expressed as follows:

$$\hat{E}(z[n]) = \frac{1}{N}\sum_{n=0}^{N-1} z[n], \tag{4.8}$$

where the time series $z[n]$ starts with the index 0 and has the maximum value index of $N-1$, which is the maximum number of samples. Following the same line of thought, the estimation of the autocovariance of a time series z [n] is as follows:

$$\hat{\gamma}_h(z[n]) = \frac{1}{N}\sum_{n=0}^{N-|h|} (z[n+|h|] - \hat{E}(z[n]))(z[n] - \hat{E}(z[n])), \tag{4.9}$$

Finally, the estimation of the autocorrelation coefficient for a time series is given by [37]:

$$\hat{\rho}_h(z[n]) = \frac{\hat{\gamma}_h(z[n])}{\hat{\gamma}_0(z[n])}. \tag{4.10}$$

Eq (4.10) represents the sample autocorrelation coefficient, commonly used in statistical analysis [37]. The effectiveness of this coefficient is illustrated in Fig.4.4 and Fig.4.5, which respectively display the results of applying (4.10) to a white noise time series and a random walk time series.

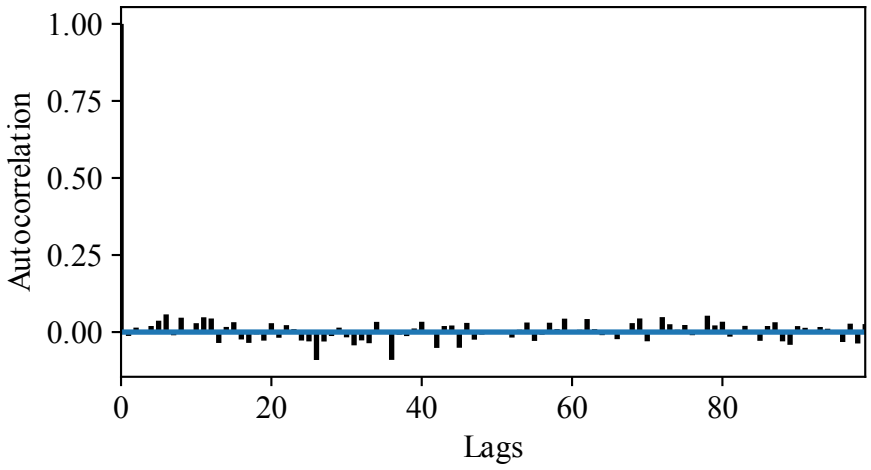


Figure 4.4: Sample autocorrelation coefficient of a white noise process realization.

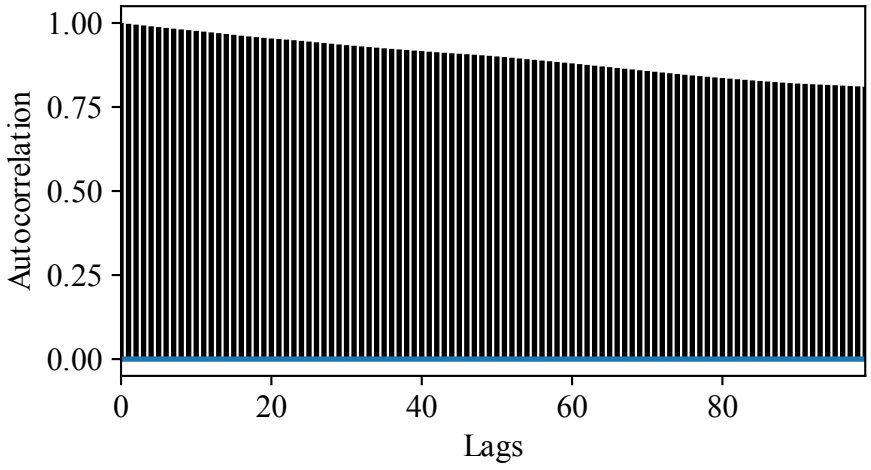


Figure 4.5: Sample autocorrelation coefficient of a random walk process realization.

In the case of the white noise time series, the autocorrelation coefficient drops rapidly from a maximum value of 1 at lag 0 to the subsequent lag and then oscillates around the same value. On the other hand, the random walk time series exhibits the opposite behavior: the autocorrelation coefficient gradually decreases and does not remain at a steady value for long.

Fig.4.6 displays the application of (4.10) to the airline passengers time series (Fig.4.1). The behavior of the autocorrelation coefficient is similar to that of the random walk time series, gradually decreasing and failing to reach a low value (0.00) as the lags increase. This indicates the non-stationarity of the series [38], which can be attributed to the presence of the trend component $m[n]$. Additionally, the periodic peaks that occur every 12 lags indicate the presence of the seasonal component $s[n]$.

Various techniques are available to transform a non-stationary time series into a series that is either stationary or close to stationary in terms of WSS. One such method involves using the differentiation operator, as expressed in (4.3). This operator can be applied to a time series several times until the trend component vanishes. Similarly, seasonal differentiation, as shown in Eq (4.3), can be used to remove the seasonal component by setting the d parameter equal to the period of the seasonal component.

Once the transformation process is complete, the resulting time series can be utilized as input to various prediction models. One classic example is the combination of the autoregressive (AR) and moving average (MA) models to form the autoregressive moving average (ARMA) model. This model is often used in conjunction with differentiation to yield the ARIMA model, which can be used to forecast non-stationary time series [35].

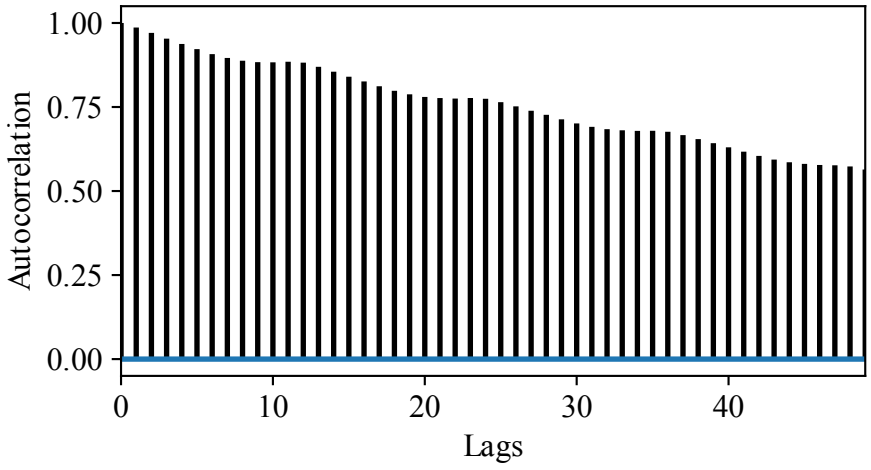


Figure 4.6: Autocorrelation of the airline passengers time series data.

4.3 Multivariate time series analysis

The preceding subsections examined univariate time series and their preprocessing for use in forecast models. In contrast, multivariate time series are strongly correlated with variables other than their past lags. A forecast model for multivariate time series can take advantage of other variables, improving accuracy.

Multivariate time series forecast models can concentrate on one series and use others as potentially valuable inputs. The endogenous variable, which represents the autoregressive series whose future value is correlated with its past values, is the primary focus of this model. In contrast, the external series are the exogenous variables.

This subsection and this work concentrate on bivariate time series models to predict the future state of a series using other series as external inputs. A classic example of this application is the ARIMAX model, which, like the ARIMA model, relies on differentiation through the integration term to render the target series WSS.

The initial step in bivariate time series analysis involves examining the correlation between the two series. Let z represent the target series and $e_x[n]$ the exogenous input series. The Pearson correlation coefficient [37] can be used to measure the linear correlation between these two as follows:

$$r(z[n], e_x[n]) = \frac{\sum_{n=0}^{N-1}(z[n] - \hat{E}(z[n]))(e_x[n] - \hat{E}(e_x[n]))}{\sqrt{\sum_{n=0}^{N-1}(z[n] - \hat{E}(z[n]))^2}\sqrt{\sum_{n=0}^{N-1}(e_x[n] - \hat{E}(e_x[n]))^2}} \tag{4.11}$$

The output of (4.11) ranges from -1 to 1, with values near -1 indicating a strong negative correlation, values near 1 indicating a strong positive correlation, and values near 0 indicating a weak or no linear correlation between the two series. A scatter plot of the two series can also provide evidence of their correlation.

In the context of time series analysis, it is vital to consider linear correlations between variables and the possibility of complex non-linear relationships. While the Pearson correlation coefficient helps measure linear correlations, it may not detect non-linear relationships. The maximal information coefficient (MIC) is a measure that can detect such non-linear relationships [39]. If the MIC suggests a significant mutual correlation coefficient while the Pearson correlation coefficient is low, a robust forecast model should consider non-linear relationships between the variables.

In addition to correlation analysis, it is necessary to understand the concepts of integration and cointegration order in bivariate time series regression. In time series analysis, the integration order refers to the number of differentiations required to transform a non-stationary time series into a weakly stationary series (WSS), as shown below:

$$u[n] = {\Delta_d}^c z[n], \tag{4.12}$$

where $u[n]$ is WSS and c denotes the number of difference operators Δ of window size d. For simplicity, this section notates the c order as a superscript of the difference operator.

The cointegration order, on the other hand, denotes the necessary order c of the difference operator applied to the linear combination of two series (endogenous and exogenous) to result in a WSS variable.

$$u[n] = {\Delta_d}^c z[n] + \beta({\Delta_d}^c e_x[n]), \tag{4.13}$$

where the term $u[n]$ denotes the result variable of the sum, which has WSS conditions. The concept of cointegration order conflicts with the concept of individual integration order, so two series are cointegrated only if the cointegration order is less than each series' integration order.

In scenarios where the variables are cointegrated, those may be differentiated using this mutual cointegration order. If not, the variables should be differentiated using the individual integration order before inserting them into a forecast model.

Chapter 5

Time series prediction algorithms

In accordance with the definitions presented in Chapter 4, this chapter provides a detailed explanation of the models employed in this study to provide a software-based holdover. Firstly, in Section 5.1, the statistical model ARIMA is defined. Following this, Section 5.2 elaborates on the fundamentals of the neural network models utilized in this study. Lastly, Sections 5.3 and 5.4 respectively provide a detailed explanation of the LSTM and Transformer networks employed in this study.

5.1 ARIMA

The ARIMA model [40], also known as the autoregressive integrated moving average model, is a statistical method used to forecast time series data. It is a popular method in econometrics and finance for modeling and predicting trends in financial data such as stock prices, interest rates, and economic indicators.

This model combines three components: AR, differencing process (I), and MA. The autoregression component involves using past values of the time series to predict future values, which the (5.1) describes below:

$$y[n] = \phi_0 + \phi_1 y[n-1] + \phi_2 y[n-2] + ... + \phi_p y[n-p] + e[n]. \tag{5.1}$$

where the ϕ_0 up to ϕ_p are the constant terms up to the order p, which is the same order as the AR process, and the term $e[n]$ represents the error term, which is assumed to have a constant variance and a mean of zero, i.e, a WSS process.

A moving average model can be expressed similarly to an autoregressive model, except that the terms included in the linear equation refer to present and past error terms rather than

present and past values of the process itself. (5.2) describes the MA model of order q is expressed as follows:

$$y[n] = \mu + \theta_1 e[n-1] + \theta_2 e[n-2] + ... + \theta_q e[n-q] + e[n]. \tag{5.2}$$

With these two components, one can potentially generalize a stationary time series, namely with the ARMA model, as follows:

$$y[n] - \phi_0 - \phi_1 y[n-1] - \phi_2 y[n-2] - ... - \phi_p y[n-p] = \tag{5.3}$$

$$\mu + \theta_1 e[n-1] + \theta_2 e[n-2] + ... + \theta_q e[n-q] + e[n]. \tag{5.4}$$

Prior to incorporating the differencing component of the ARIMA model, it is necessary to introduce the back-shift operator B, which plays a key role in the organization of (5.4). The back-shift operator facilitates the introduction of the differencing component into the ARMA model, ultimately generating the ARIMA model. The back-shift operator, denoted as B, is defined in (5.5) as follows:

$$B^i y[n] = y[n-i], \tag{5.5}$$

now making use of the back shift operator the (5.4), can be rewritten as:

$$(1 - \sum_{i=0}^{p} \phi_i B^i) y[n] = (1 + \sum_{i=0}^{q} \theta_i B^i) e[n]. \tag{5.6}$$

Ultimately, the back-shift operator can be introduced in the (5.4) to introduce the differencing component of ARIMA, making the ARIMA model capable of processing non-stationary time series, turning then into a WSS series, as described in (5.6), as follows:

$$(1 - \sum_{i=0}^{p} \phi_i B^i)(1-B)^d y[n] = (1 + \sum_{i=0}^{q} \theta_i B^i) e[n], \tag{5.7}$$

where the d component represents the order of the differencing term.

Ideally, when fitting the components of the ARIMA model to a particular time series, the I of order d must correspond to the integration order of the time series. This concept is elaborated in Section 4.3, where the integration order is defined as the number of times the time series must be differentiated in order to become a WSS series.

It should be noted that the differentiation and differentiation inversion processes are typically performed by the ARIMA model. However, in our study, these processes are carried out by the data pipeline, which will be further elaborated on in Chapter 6.

In addition to the conventional ARIMA model, the concept of seasonal periods in the ARIMA model, thereby forming the seasonal autoregressive integrated moving average (SARIMA) model. Like the ARIMA model, the SARIMA model comprises three key components (P,D, Q). However, in the SARIMA model, these components are also applied to the seasonal differences of the data, in addition to the regular differences. The equation presented in Eq. (5.8) characterizes the SARIMA model.

$$(1-\sum_{i=0}^{p}\phi_i B^i)(1-\sum_{i=0}^{P}\Phi_i B^{s\cdot i})(1-B)^d(1-B^s)^D y[n] = (1+\sum_{i=0}^{q}\theta_i B^i)(1+\sum_{i=0}^{Q}\Theta_i B^{s\cdot i})e[n]. \tag{5.8}$$

Where in the (5.8), s represents the seasonal period, with $s = 12$ signifying an annual period if monthly data are being utilized. The terms P, Q, and D correspond, respectively, to the seasonal version of the Autoregressive (AR), Moving Average (MA), and Integrated (I) components, while Φ and Θ indicate the constant terms for the seasonal AR and MA, respectively.

In addition to the standard SARIMA model, it is possible to incorporate an exogenous time series $x[n]$ to fit the endogenous time series $y[n]$ better by introducing exogenous terms in the equation. This results in the seasonal autoregressive integrated moving average with exogenous variables (SARIMAX) model, given by:

$$(1-\sum_{i=0}^{p}\phi_i B^i)(1-\sum_{i=0}^{P}\Phi_i B^{s\cdot i})(1-B)^d(1-B^s)^D y[n] = \tag{5.9}$$

$$(1+\sum_{i=0}^{q}\theta_i B^i)(1+\sum_{i=0}^{Q}\Theta_i B^{s\cdot i})e[n] + (1+\sum_{i=0}^{r}\zeta_i B^i)x[n], \tag{5.10}$$

where $x[n]$ represents the exogenous time series, ζ is the coefficient applied to each time step of $x[n]$, and r represents the number of time steps considered for the exogenous time series.

For the sake of simplicity, this study shall refer to all ARIMA variations discussed herein as ARIMA. The parameters (P, D, Q) order shall determine whether the model is, in fact, an ARIMA model or a SARIMA model, e.g, if $(P, D, Q) = (0, 0, 0)$, the model is the standard ARIMA, and SARIMA otherwise.

In order to determine the optimal combination of parameters for the ARIMA model, this study will employ the use of the Akaike information criterion (AIC). The AIC is a statistical measure used for model selection that aids analysts in comparing the goodness of fit of different

models and selecting the one that fits the data best.

In the context of ARIMA modeling, the AIC is utilized to assess the quality of various models with different parameter values, namely (p, d, q, P, D, Q). The principle of parsimony forms the basis of AIC and states that simpler models that explain the data well are preferred over more complex models that fit the data more closely but have more parameters.

The AIC is calculated as shown in Eq. (5.11), where k represents the number of parameters in the model, and L represents the likelihood function of the model. The likelihood function evaluates how well the model fits the data, while the AIC penalizes models with more parameters, favoring simpler models. Therefore, the optimal value of AIC should correspond to the model that best fits the data without overfitting.

$$AIC = 2k - 2\ln(L). \tag{5.11}$$

5.2 Neural Networks

As described briefly in Section 3.3, artificial neural network (ANN) are widely used in various scenarios to achieve different objectives. This is possible due to their ability to learn and adapt their parameters based on the situation and their capacity to incorporate non-linear behaviors. ANN models consist of a parallel distributed system composed of computational units called neurons, which can be allocated in one or more layers and make various interconnections, usually unidirectional.

In most ANN models, each connection possesses a weight that stores acquired knowledge, serving as a weighting factor for each neuron's input in the network. The concept of ANN was first introduced in [41], where the authors presented details of how a computational system can represent a neuron's behavior and memory. However, this first work did not introduce the idea of the learning process of a neural network. On the other hand, the authors of [42] introduced this concept by adding weights to each neuron's connections.

Another significant milestone in the history of neural networks was when Frank Rosenblatt [43] demonstrated that the network could be trained to classify patterns by adjusting the weights' values. This model is known as the perceptron and has three layers in its simplest form, consisting of an input layer, a hidden layer, and an output layer.

5.2.1 MLP networks

The multilayer perceptron (MLP) is a type of neural network that bears resemblance to the perceptron originally proposed by [43]. Unlike the perceptron, the MLP contains multiple hidden layers, each comprising numerous interconnected neurons as illustrated in Fig. 5.1. During the training process, the weights linking these neurons are adjusted to optimize the network's accuracy with respect to a specific expected output in a supervised learning scenario.

In supervised learning, the anticipated output is often referred to as a label. Notably, this terminology is consistent across a variety of tasks including regression and classification. In this work, which addresses the task of time-series regression, the label represents the predicted future time offset.

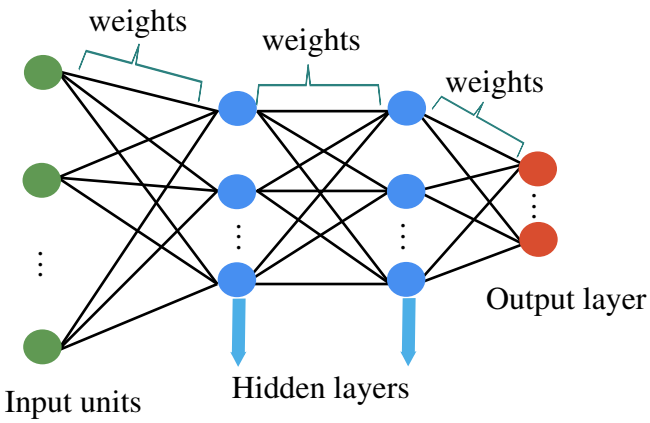


Figure 5.1: MLP network.

The structure of a neuron in the MLP model and its information processing mechanism are illustrated in Fig.5.2. The input information of a neuron is represented by a set xh, and it is associated with a set of weights, W, as dictated by Eq.(5.12) for a given neuron j. This equation combines the input vector xh_i and the weights W_i through a product. Subsequently, a weighted sum of all the products is computed, and the bias term θ is subtracted.

$$p_j = \sum_{i=1}^{n} xh_i W_i - \theta_j. \tag{5.12}$$

The p_j value in (5.12) represents the activation potential, which is the input argument to the activation function. One of the most utilized activation functions in the ANN's theory is the sigmoid activation function, shown in Fig. 5.3. This activation function is described in

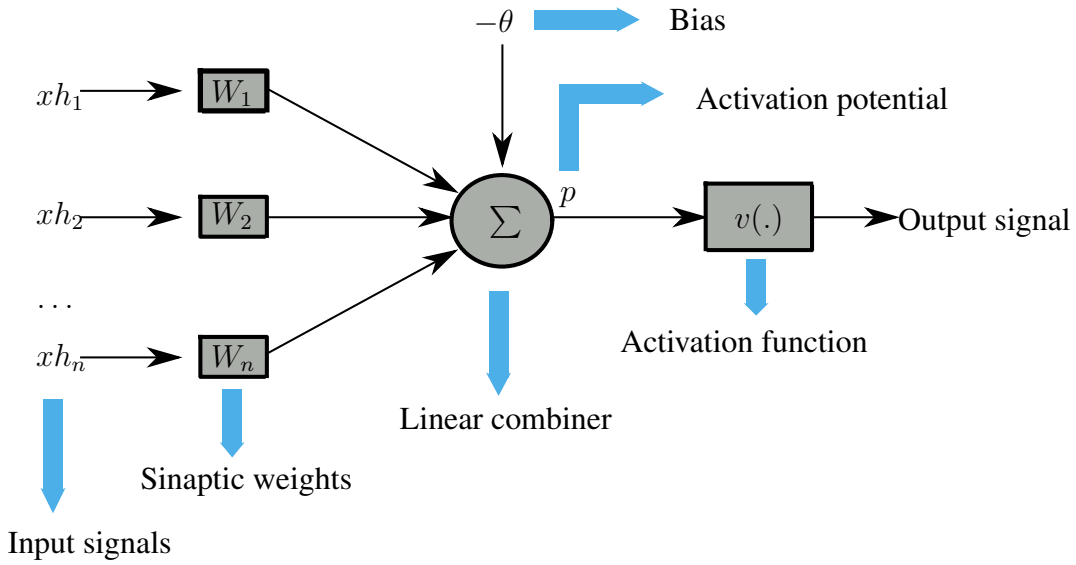


Figure 5.2: Artificial neuron model of an MLP network.

(5.13), where parameter e represents Euler's constant and $\hat{Y}$ is the output of the neuron after the activation uses the activation potential p_j as an argument.

$$\hat{Y} = \frac{1}{1 + e^{-p_j}}. \tag{5.13}$$

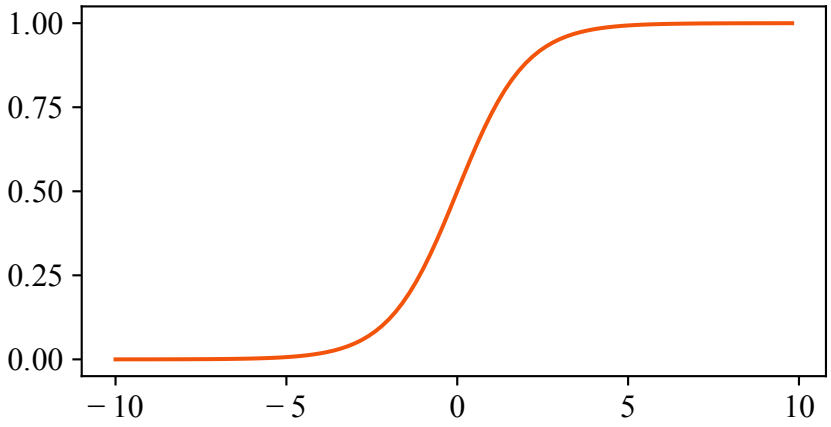


Figure 5.3: Sigmoid activation function.

5.2.2 Training process of a neural network

The present study employs a supervised learning approach, wherein each input data point is associated with a target output, referred to as the label, while the input data is commonly

referred to as features. Given a set of training data, the trainable parameters of the neural network, such as weights and biases, are adjusted to minimize the difference between the network's outputs and the expected output [44].

The backpropagation algorithm is the primary algorithm used to train the neural network, and it involves three steps: feedforward, error calculation, and backpropagation. During the feedforward step, the input data enters the first layer of the neural network and propagates through the hidden layers until it reaches the output layer. The output of each neuron in the network is calculated using (5.13). Following this, the training algorithm can evaluate the error using a mean squared error (MSE) function, as shown below:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} -1(Y_i - \hat{Y}_i)^2. \tag{5.14}$$

The preceding equation indicates the mean square error (MSE) function utilized to measure the neural network's error, where $\hat{Y}_i$ represents the output of the neural network, Y_i represents the actual label that the neural network is being trained to predict, and N denotes the total length of the neural network's output layer.

The final stage of the training algorithm is backpropagation, where the network updates the weights, considering the discrepancy between the network output and the label. To achieve this, the algorithm computes the gradient of the cost function, which corresponds to the partial derivative of the cost function concerning the weights:

$$\frac{\partial MSE(Y_j, \hat{Y}_i)}{\partial W_{ij}} = \frac{\partial MSE(Y_j, \hat{Y}_i)}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial p_j}\frac{\partial p_j}{\partial W_{ij}}, \tag{5.15}$$

where the value of this function indicates the direction in which a given weight W_{ij} associated with a neuron j must be altered by the algorithm to obtain the minimum value of the cost function.

The solution of the partial derivative, considering the sigmoid as the activation function, is as follows:

$$\frac{\partial MSE(Y_j, \hat{Y}_i)}{\partial W_{ij}} = 2(Y_j - \hat{Y}_i)[\hat{Y}(Y_j) - (1 - \hat{Y}(Y_j))]xh_j. \tag{5.16}$$

Finally, in the last step, the backpropagation algorithm can backpropagate the error to each weight W_{ij} using (5.17), where the η value represents the learning rate, which helps the network converge.

$$W_{ij} = W_{ij} - \eta\frac{\partial MSE(Yj, \hat{Y}_i)}{\partial W_{ij}}. \tag{5.17}$$

5.2.3 Recurrent networks

The neural network architectures discussed thus far, including the perceptron and MLP, treat inputs as independent entities, rendering them suboptimal for handling sequential data, such as time series data, as discussed in Section 4 [45]. In contrast, the recurrent neural network (RNN) model can effectively process sequential input data since it incorporates a memory mechanism to store information from past computations.

Fig 5.4a demonstrates a basic perceptron network architecture consisting of one neuron each in the input, hidden, and output layers. The un parameter denotes the weight linking the input layer to the hidden layer, while vn represents the weight connecting the output layer to the hidden layer. This architecture cannot consider previous outputs of the hidden layer states, as it does not incorporate a memory mechanism. Rather, it merely considers the current input's hidden layer output. By contrast, the RNN model has hidden layer neurons with connections pointing backward, as depicted in Fig 5.4b. The weight parameter wn corresponds to the weight from the past to the current hidden state. Thus, past outputs can be incorporated into the current computation.

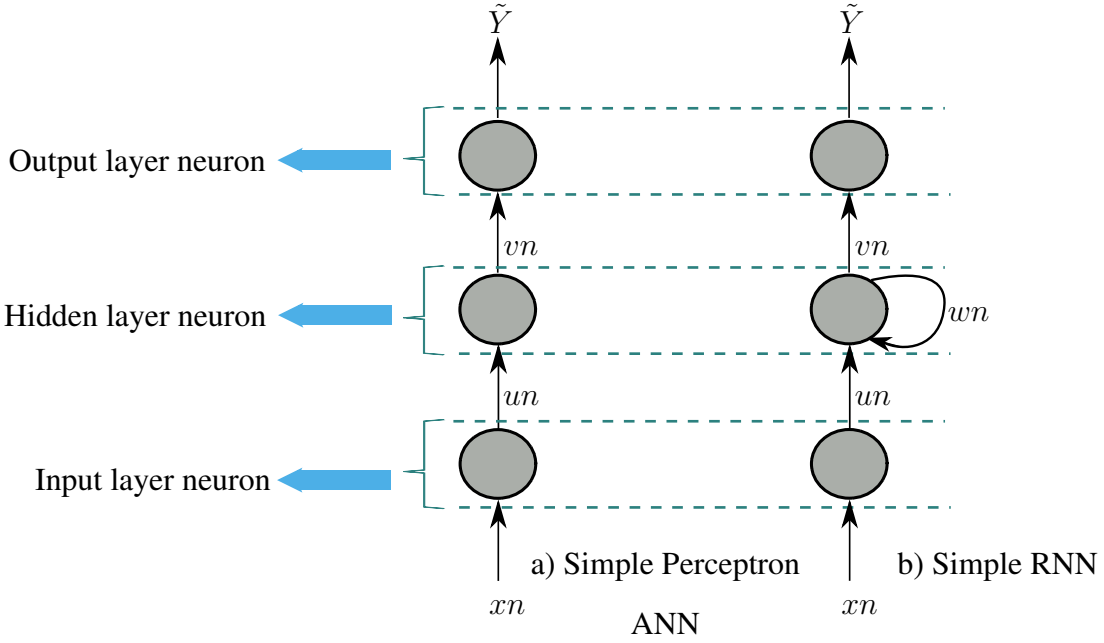


Figure 5.4: Simple RNN.

For a more thorough comprehension of the RNN's operation when presented with a set of inputs xn at specific time intervals t, the RNN can be unwrapped along the time axis, associating each of its states to a temporal position. The diagram depicted in Fig. 5.5 illustrates the temporal

dynamics of the basic RNN [45].

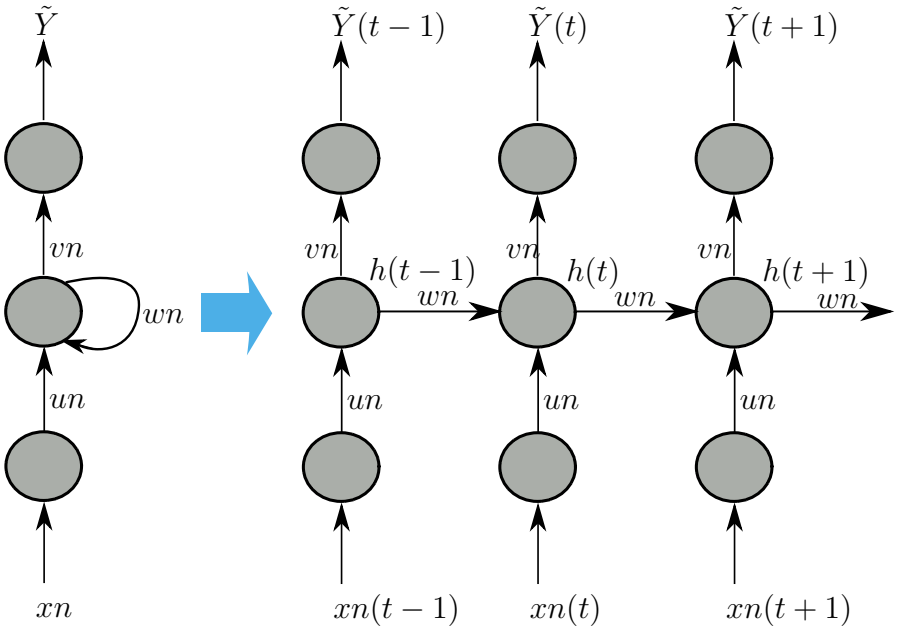


Figure 5.5: Unfolding of a simple RNN in time.

The output of the RNN at any time step t is as follows:

$$\hat{Y}(t) = vn.h(t). \tag{5.18}$$

In typical applications of the RNN model, multiple RNN units may exist in each layer, and each unit has its memory from the previous state.

Unlike conventional perceptron networks, the training algorithm for RNNs utilizes back-propagation through time [46], which involves unrolling the network along the time axis and applying the backpropagation algorithm as explained in Section 5.2.2.

5.3 LSTM

The architecture of the RNN could successively introduce the sense of sequence into neural networks. However, it has the drawback of storing all the previous information, even if it is irrelevant to the current task. This issue arises because not all the past samples may strongly correlate with the current step, as exemplified by the autocorrelation plot in Fig 4.6 discussed in Section 4. Storing this unnecessary information may mislead the RNN and lead to the vanishing or exploding gradient problem, as documented in [47].

The LSTM model was developed to address this challenge, generating a variation of the RNN that can discard the irrelevant information and avoid the problem mentioned earlier. The critical difference between the LSTM and the RNN is that the former can learn which information to retain and which to discard.

In particular, the LSTM network comprises a modified version of the RNN that features a repeat module inside the neuron (or unit) with four interactive layers, as illustrated in Fig. 5.6.

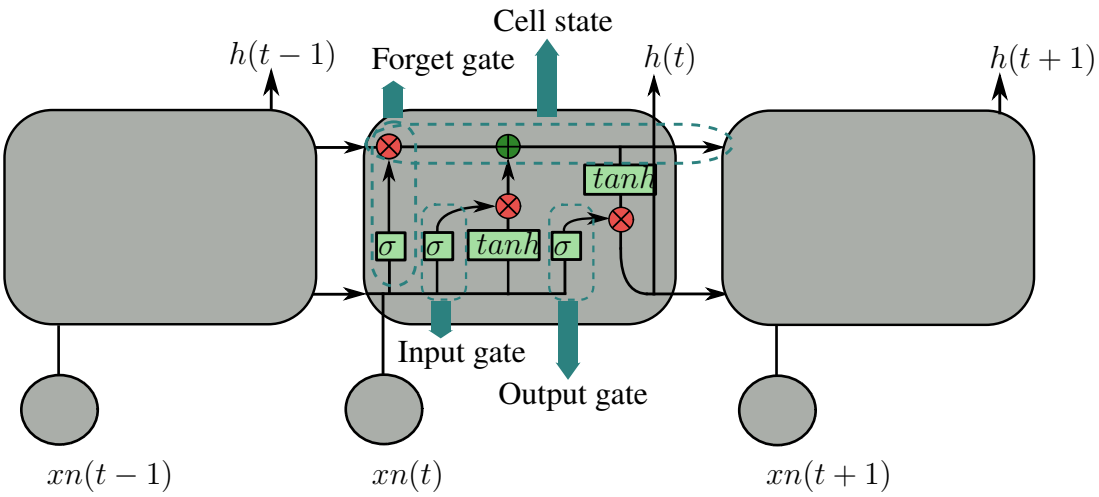


Figure 5.6: LSTM unit.

As Fig. 5.6 describes, the top part of the LSTM unit denotes the cell state. This state uses three different gates: forget, input, and output. The forget gate uses a Sigmoid function (denoted by the symbol Σ), and takes two inputs, the previous hidden state $h(t-1)$, and the current cell input $xn(t)$, and outputs a value between 0 and 1 as described in (5.19). As the name suggests, the forget gate decides which information to discard from the cell state.

$$f(t) = \Sigma(un_f x(t) + wn_f h(t-1)). \tag{5.19}$$

The input gate and a hyperbolic tangent function $tanh$ determine which new information to inject into the cell state. Firstly, the $tanh$ function generates potential candidates for the cell state as follows:

$$c(t) = tanh(un_c xn(t) + wn_c h(t)), \tag{5.20}$$

where, similarly to the forget gate's weights, the parameters un_c and wn_c are weights respectively of the input and the hidden of the cell state. After that, the input gate selects which

information gets added to the cell state as follows:

$$i(t) = \Sigma(un_i x(t) + wn_i h(t-1)). \tag{5.21}$$

The result of the two (5.20) and (5.21), passes through a point wise multiplication $\otimes$, which generates a element wise multiplication between the two functions [48], combining the result of the two equations as follows:

$$I(t) = c(t) \otimes i(t). \tag{5.22}$$

This way, the new cell state denoted as $C_{state}(t)$ can be calculated by (5.23), which combines the results of (5.19), (5.22) and the previous cell state $C_{state}(t-1)$.

$$C_{state}(t) = f(t) \otimes C_{state}(t-1) + I(t). \tag{5.23}$$

Finally, to calculate the output of the LSTM unit, a $tanh$ function receives the cell state $C_{state}(t)$ as input, then this result is multiplied with the output gate's result, this calculation decides what the LSTM unit's hidden state. This process is described in (5.24), where similarly to the previous equations, the parameters un_o and wn_o are weights respectively of the input and the hidden of the output gate.

$$h(t) = tanh(C_{state}(t)) \otimes \Sigma(un_o x(t) + wn_o h(t-1)). \tag{5.24}$$

With this final (5.24), all states and outputs of a simple LSTM can be understood. Similar to the traditional RNNs, the LSTM network utilizes the backpropagation through time algorithm.

5.4 Transformer networks

In 2017 the study [29] entitled as "attention is all you need" first introduced the transformer network, an entirely novel model to solve sequence-to-sequence tasks, focused at this moment in Natural language processing (NLP), more specifically text-to-text translations. Its architecture can be seen in Fig. 5.7. It counts with an encoder-decoder structure commonly found in recurrency models in sequence-to-sequence tasks. The main difference between transformers to the standard recurrence-based models is how the encoder and decoder are constructed. Transformers use self-attention mechanisms to build the encoder and decoder module and rely upon positional encodings to insert the notion of position in the input and output sequences.

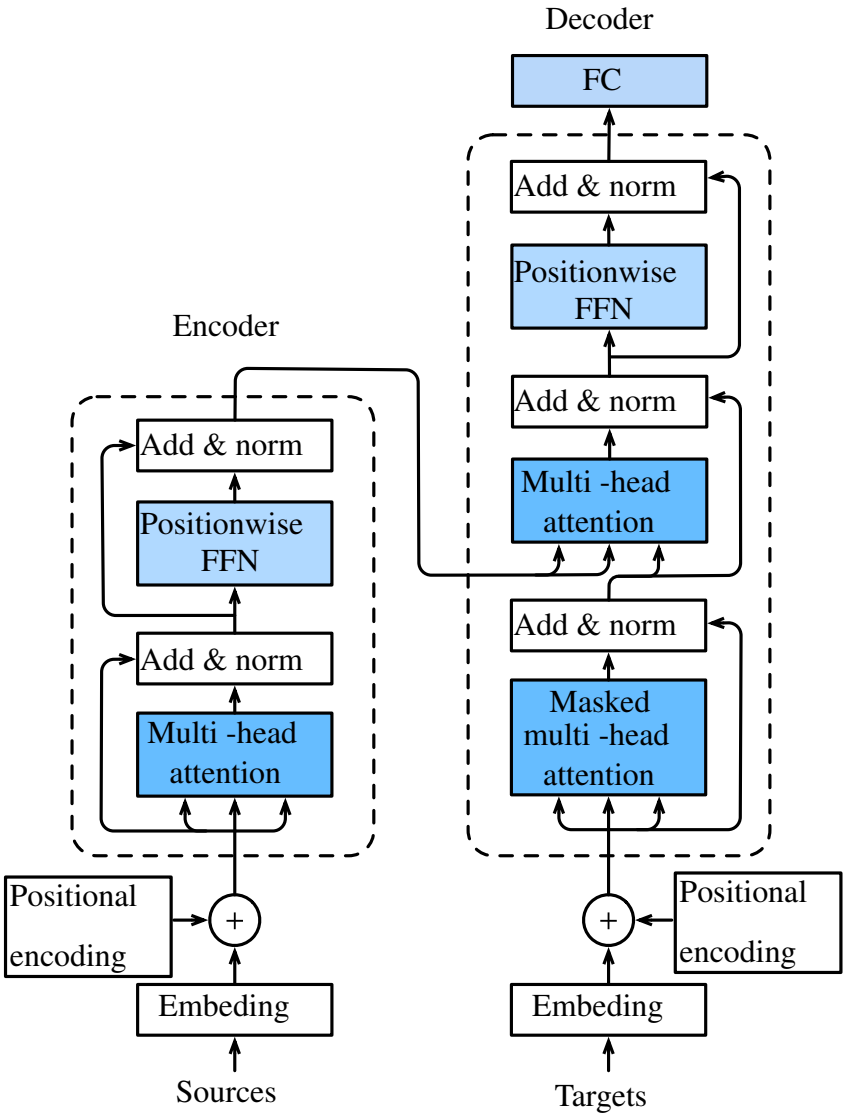


Figure 5.7: Transformer model architecture.

The change from recurrency layers to attention mechanisms significantly enhanced performance, logistics, and inference explicability. According to [49], transformers brought with them key advantages for time series tasks:

- The self-attention mechanism can catch even longer relationships of the input than the recurrence layers. The transformer can generate a rich representation of the input sequence, treating each sequence's positions equally. At the same time, models such as LSTM tend to prioritize a limited number of time steps stored in their hidden state.

- In contrast to recurrence-based models, transformer networks can be trained in a parallel distributed manner. As it does not depend on any recurrence mechanism, it does not

depend on its previous states of itself. Therefore it may be trained in a parallel and distributed manner, which is crucial for larger models with millions and even billions of hyper-parameters.

- Transformers produce potentially explicable inferences. Self-attention can output a scoring matrix of the input sequence, and depending on the application, this score can easily explain its inference.

5.4.1 Transformer Components

As shown in Fig. 5.7, the standard transformer introduced by [29] can be divided into five key components, those being the positional encoder, embedding layer, multi-head attention, encoder module, and decoder module.

Embedding layer This layer is responsible for transforming the entry features into a representation in the dimension $\mathbf{d_{model}}$, to be further processed in the other components. The embedding layer does that by passing its entries by a fully connected network, this way mapping the input of shape $\mathbf{B_{sz}} \times \mathbf{T_s} \times \mathbf{F_t}$ to a representation along $\mathbf{d_{model}}$, of shape $\mathbf{B_{sz}} \times \mathbf{T_s} \times \mathbf{d_{model}}$. This mapping process is originally used to transform integers word tokens to a float representation in $\mathbf{d_{model}}$ dimension. One can visualize this process in the time series context as generating a richer representation of the features, fusing them in a single dimension.

Positional encoder Differently than the LSTM that uses the recurrency mechanisms to process sequences, the transformer network does not natively possess the ability to perceive sequences. Thus, to solve this problem, the transformer architecture counts with the positional encoder. This mechanism is capable of inserting the notion of sequence in an array of features by using sine and cosine functions of different frequencies [29], inserting this sequential notion in a vector of dimension $\mathbf{d_{model}}$ as follows:

$$PE_{pos,2i} = sin(pos/1000^{2i/\mathbf{d_{model}}}), \tag{5.25}$$

$$PE_{pos,2i+1} = cos(pos/1000^{2i/\mathbf{d_{model}}}), \tag{5.26}$$

Multi head attention This particular attention mechanism can catch the relationships between each of its inputs via the scaled dot product attention, introduced by [29], which is as

follows:

$$\mathbf{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T_s}}}{\sqrt{\mathbf{d_k}}})\mathbf{V} \tag{5.27}$$

$$\mathbf{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\mathbf{Attn_{weights}})\mathbf{V} \tag{5.28}$$

$$\mathbf{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{Attn_{Scores}}\mathbf{V} \tag{5.29}$$

where $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ represents respectively the Query, Key and Values, and $\frac{1}{\sqrt{\mathbf{d_k}}}$, is the scaling factor based on the key dimension size($\mathbf{d_k}$). A set of three different trainable embedding layers are responsible for generating those from the transformer's input, respectively $\mathbf{Q_w}$, $\mathbf{K_w}$, $\mathbf{V_w}$, as follows:

$$\mathbf{Q} = \mathbf{Q_w} \cdot \mathbf{Src} \tag{5.30}$$

$$\mathbf{K} = \mathbf{K_w} \cdot \mathbf{Src} \tag{5.31}$$

$$\mathbf{V} = \mathbf{V_w} \cdot \mathbf{Src} \tag{5.32}$$

This process ultimately generates three different representations of the input ($\mathbf{Src}$) itself, each with shape $\mathbf{B_{sz}} \times \mathbf{T_s} \times \mathbf{dH} \times \mathbf{H}$, where $\mathbf{H}$ is the number of attention heads, and $\mathbf{dH}$ is given by $\frac{\mathbf{d_{model}}}{\mathbf{H}}$. This way, each attention head $\mathbf{H}$ will possess a part of $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ projections, and calculate the scaled dot product attention with it, outputting each a $\mathbf{Attn}$ and $\mathbf{Attn_{Scores}}$. Finally, the multi-head attention concatenates each $\mathbf{Attn}$ value and passes it through a linear layer, generating its final output. This entire process is described in Fig. 5.8. In the explicability context, one may visualize the attention scores from each attention head, defined in 5.29 as the softmax component in the scaled dot product attention. These scores matrices have the shape of $\mathbf{T_s} \times \mathbf{T_s} \times \mathbf{H}$, meaning that each attention head has its own perception of the transformer's inputs relationships. One may use these scores to extract explications to the transformer output and hypothesize on top of that.

Encoder After the process of the positional encoder, the encoder block receives the entries called $\mathbf{Src}$ in shape $\mathbf{B_{sz}} \times \mathbf{T_s} \times \mathbf{d_{model}}$ and processes it using it's of n identical encoder layers, each of which consists in a self-attention sub-layer using n_{head} attention heads and a fully-connected feedforward sub-layer, followed by a normalization layer.

The resulting vector has the same shape as $\mathbf{Src}$, representing this meaningful projection, incorporating the attention from the self-attention process. Thus, one can view this output as the encoder understanding extracted from the input. Therefore the encoder block can be seen

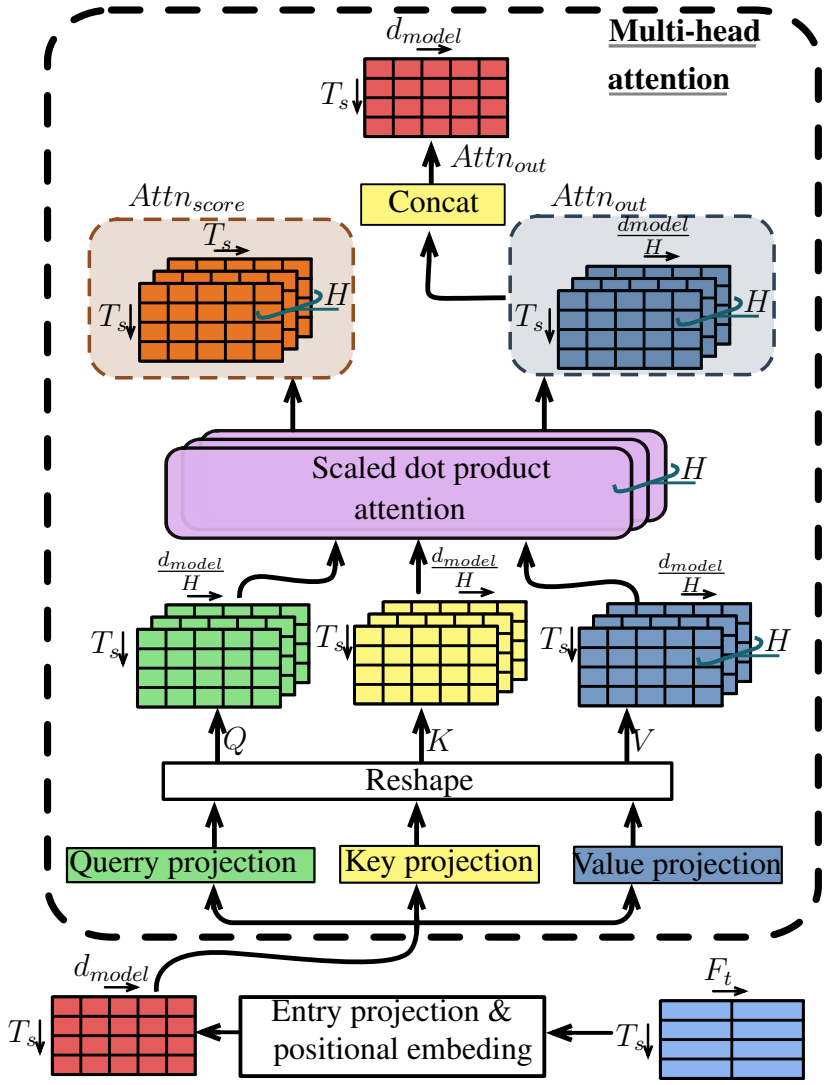


Figure 5.8: Multi head attention flux

as the block responsible for understanding the transformer input and generating a meaningful representation. This entire flux and each step outputs shape is described in Fig. 5.9.

Due to its high capacity to extract meaningful representations from sequences, multiple applications use the transformer encoder alone. These applications are mainly derived from the need to understand a sequence, thus, focused mainly on classification tasks, such as sentiment prediction, named entity recognition, and various others in the NLP field. In the time series scenario, one may use encoder-only transformer models to perform time series classification tasks, such as anomaly detection.

Decoder The decoder block process begins after the encoder process, where the decoder block receives the encoder's output, here named as memory $\mathbf{T_s}$ and an entry vector named

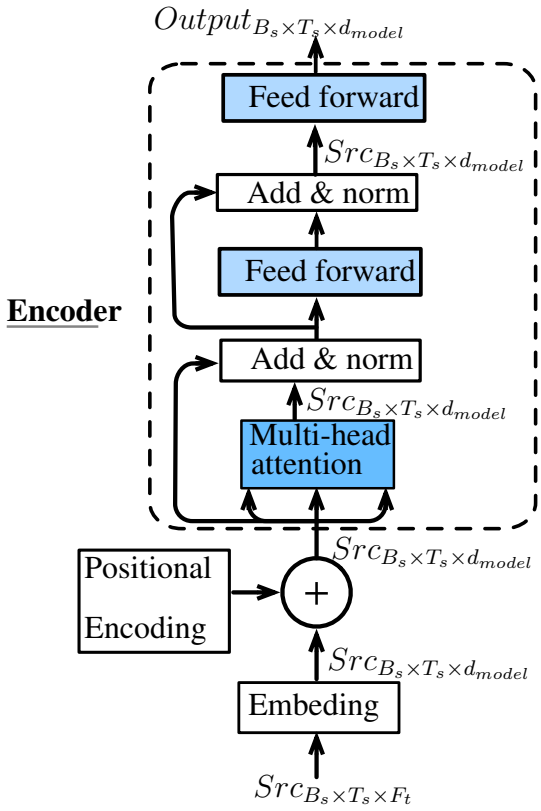


Figure 5.9: Transformer encoder information flux

target $\mathbf{Tgt_{in}}$. With the $\mathbf{Tgt_{in}}$, the decoder first generates a self-attention representation named $\mathbf{Dec_{self}}$. This self-attention follows the steps as the encoder process using the $\mathbf{Src}$ input, with one major change, the presence of an attention mask, which is introduced in the attention scores ($\mathbf{Attn_{Scores}}$) calculation, as follows:

$$\mathbf{Dec_{self}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{Attn}^{\mathbf{Masked}}_{\mathbf{Scores}} \cdot \mathbf{V} \tag{5.33}$$

$$\mathbf{Dec_{self}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\mathbf{Attn_{weights}} + \mathbf{Attn_{mask}}) \cdot \mathbf{V} \tag{5.34}$$

$$\mathbf{Dec_{self}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T_s}}}{\sqrt{\mathbf{d_k}}} + \mathbf{Attn_{mask}}) \cdot \mathbf{V} \tag{5.35}$$

The $\mathbf{Attn_{mask}}$ is a causal attention mask used to prevent the decoder from peeking into the future, i.e., in a timestamp n, the resulting attention scores are calculated without information of timesteps greater than n. This mask, described in Fig. 5.10, is composed of one (1) and minus infinity $(-\infty)$ values, which the attention mechanism uses to restrict the $\mathbf{Attn_{weights}}$ inside the softmax component in 5.35. The resulting $\mathbf{Attn_{Scores}}$ possesses only causal relationships between its timesteps, whereas non-causal relationships will possess a 0 value. After this initial step, the decoder block then uses the $\mathbf{Dec_{self}}$ and $\mathbf{T_s}$ representations to generate a

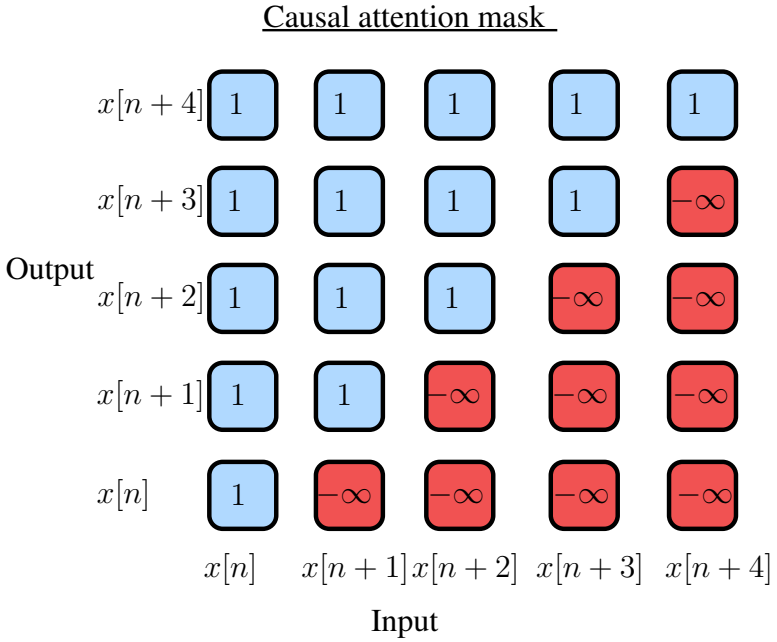


Figure 5.10: Decode causal attention

cross-attention $\mathbf{Dec_{cros}}$. This cross-attention uses the $\mathbf{T_s}$ to generate its Key and Query matrices, while it uses the $\mathbf{Dec_{self}}$ to produce its Value notion. While the self-attention output $\mathbf{Attn}$ may represent the Values embedded with the $\mathbf{Src}$ inner relationships, the cross-attention output $\mathbf{Dec_{cros}}$ represents the Values embedded with the $\mathbf{Dec_{self}}$ and $\mathbf{T_s}$ relationships. Ultimately, one can view this process output as a richer representation of the $\mathbf{Tgt_{in}}$ and $\mathbf{Src}$ relationships.

The decoder block represents the generative part of the transformer network, commonly used in NLP tasks involving text production as output, including summarization and text generation tasks. In a time series context, this block is responsible for producing predictions. Depending on the decoding process, it may produce recursive one-step-at-a-time or many predictions simultaneously.

With the $\mathbf{Dec_{cros}}$ latent representation, the decoder block passes it through dense, addition, and normalization layers, resulting in the transformer output value. This output value was originally designed to produce one-step-ahead predictions at each time step in an autoregressive decoding style. Thus, in a time series scenario, the decoder would receive a sequence named $\mathbf{Tgt_{in}}$, output a prediction $\mathbf{Tgt_{out}}$ as follows:

$$\mathbf{Tgt_{in}} = [x[n], x[n+1], x[n+2] \dots x[n+m]] \tag{5.36}$$

$$\mathbf{Tgt_{out}} = [x[n+1], x[n+2], x[n+3] \dots x[n+m+1]] \tag{5.37}$$

During the training phase, the $\mathbf{Tgt_{in}}$ and $\mathbf{Tgt_{out}}$ are composed of real values. Thus the

only difference is that $\mathbf{Tgt_{out}}$ is one step ahead of $\mathbf{Tgt_{in}}$. The literature often calls this process teacher forcing training, where the algorithm trains the model to converge in an auto-regressive style using only values from the aimed time series. In the inference phase, the $\mathbf{Tgt_{in}}$ entry vector only possesses the starting value, this being equal to the last known time step. At each inference, the $\mathbf{Tgt_{in}}$ receives the last prediction to produce the next value, and so on. This particular prediction scheme is the iterative auto-regressive scheme and is used in the standard transformer architecture.

5.4.2 State of the art transformer in time series

Following the introduction of the first transformer architecture, several works have investigated its applicability in addressing sequence problems beyond NLP, particularly in time-series tasks. In this regard, [50] conducted a study to evaluate the performance of standard transformer networks, specifically iterative inference, for flu prediction tasks. Their results indicated that the proposed approach outperformed commonly utilized ARIMA and LSTM models.

Moreover, [33], [49], [30], and other researchers have explored the potential of transformer networks for time-series tasks and proposed novel ways to utilize the components of the transformer and the self-attention mechanism.

[33] proposed a method to reduce memory usage in the attention mechanism of transformer networks by employing a LogSparse attention mechanism. This approach improved the forecasting accuracy of time-series data with fine granularity and strong long-term dependencies, even with a limited memory budget.

[49] introduced the concept of utilizing a single part of the transformer, specifically the encoder, for pre-training and downstream tasks such as classification and regression.

While capable of producing one-step ahead predictions at each timestep, this decoding style does not necessarily restrict the transformer architecture. One may change the attention mask 1 and ∞ values distribution to produce multi-step ahead predictions. The authors of [30] extensively research this approach in their study, where the authors aim for a multi-step ahead prediction executed in a single inference. The masking scheme used by the authors is described in Fig. 5.11.

An alternative approach is to utilize decoder-only architectures, similar to encoder-only models, with some notable differences. One such difference is that encoder-only architectures often have an output layer designed for classification tasks, such as a linear layer with many

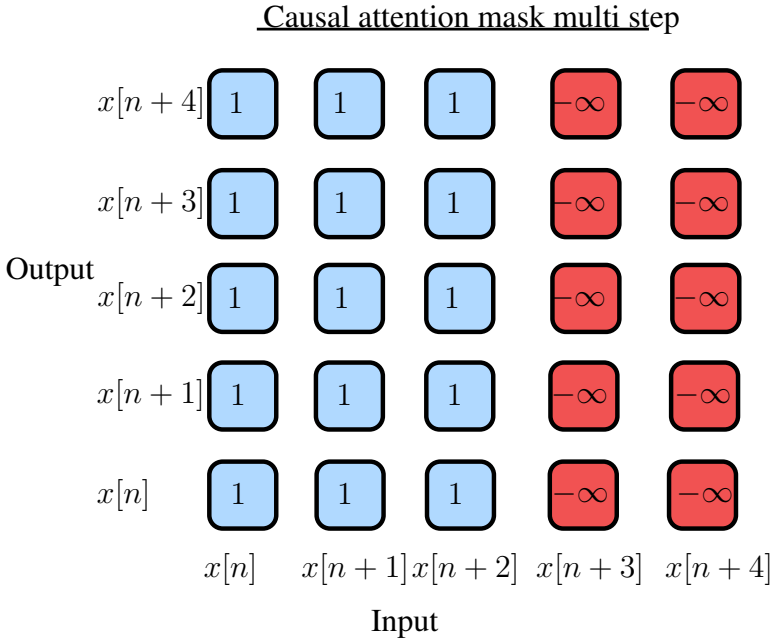


Figure 5.11: Decode causal attention in a multi-step scheme

classes for each time series or time step. In contrast, a decoder-only transformer should have a linear layer capable of expressing one or more values per time step. Another key difference is the use of attention masks, typically employed in a semi-supervised pre-training process to estimate a randomly masked token in a non-causal manner for encoder-only models. Decoder-only models should utilize attention masks for causal prediction tasks, whether one-step or multi-step oriented.

In this work, we utilize a transformer architecture that consists solely of a decoder designed to generate multiple output steps in a single interaction. To achieve this, we adopt the architecture proposed in [49] and employ the same masking technique as described in [30]. This structure allows us to balance model size, which is small enough to run on a local GPU and maintain a good performance.

Chapter 6

Proposed method

The primary objective of the proposed method is to address the issue of a free-running state in the slave clock during a reference loss in a PTP synchronization scenario, precisely, in a PTS deployment scenario [51] by proposing the use of a software-based holdover to provide a temporarily timing notion during the holdover state. This deployment scenario refers to a situation where all nodes in a network rely solely on PTP as a synchronization source, as presented in the subsections 2.3.2 and in 3.2.2. To this end, all proposed models utilize temperature and past time offset estimates to assist the slave clock during holdover operation.

In addition to the PTS deployment scenario, it is also possible to employ PTP as a secondary reference to a higher-level reference, such as GNSS, which is known as the APTS [51]. However, this work did not investigate an APTS deployment scenario, but it could be a means to extend the concepts and experiments of this study.

Fig 6.1 depicts a situation where it is assumed that the slave clock has maintained communication with its reference for several hours before a loss occurs and has access to temperature variation data through a temperature sensor. Utilizing the temperature and synchronization measurements time-series data, the slave clock can feed this data into a holdover algorithm to maintain adequate synchronization levels during the loss of reference.

Consequently, this work proposes to employ the proposed models alongside a time series processing pipeline to assist the slave clock in the scenario depicted in Fig 6.1. Furthermore, offline experiments using datasets captured from a PTP synchronization testbed were conducted to train, validate and test the proposed models.

This chapter is organized as follows: Section 6.1 presents the testbed setup and the process used to obtain datasets for this work. Section 6.2 provides a detailed explanation of the

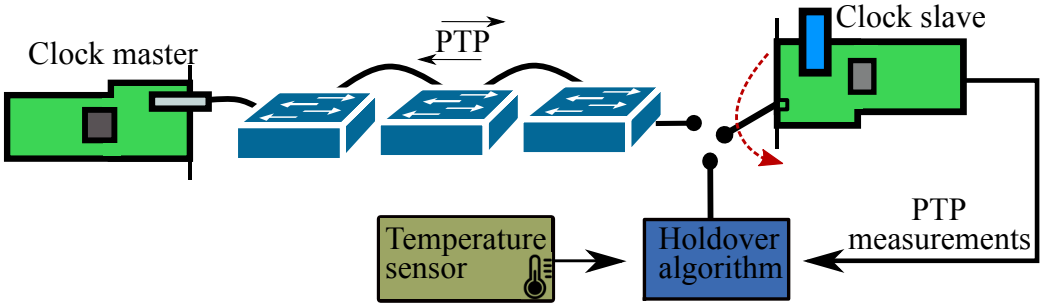


Figure 6.1: Algorithm assisted holdover in a PTP scenario

time-locked loop (TLL) algorithm in the context of PTP, and how it filters noisy time offset measurements for further use in the three selected models, namely ARIMA, LSTM, and Transformer network. Finally, Section 6.3 describes the proposed approach that incorporates the proposed models in combination with time-series analysis and transformations.

6.1 Testbed and dataset obtention setup

The present study introduces an algorithmic approach to holdover applications, which is evaluated on datasets obtained from a real PTP synchronization testbed. This testbed comprises two RRU devices and a BBU that implements an Ethernet-based FH. This study focuses on achieving master and slave time synchronization and simulating holdover situations. Thus the radio segment of the FH and the temporal alignment of frames sent by the RRU device are disregarded. Specifically, the study investigates the feasibility of implementing holdover operation aided by the proposed models: ARIMA, LSTM, and transformers.

6.1.1 Testbed

This study employs the identical testbed as [52], which presents a comprehensive account of the testbed. The testbed consists of two Xilinx Virtex 7 field-programmable gate array (FPGA)s that implement a PTP master and slave devices. A PTP-capable Ethernet media access control (EMAC) with hardware timestamping is initialized in the FPGA, along with a time counter that is powered by a free-running clock signal.

As in [21]-[22]-[23], different oscillators are employed on the slave clock side (RRU) of the testbed, including a low-cost oscillator and a robust oscillator. Specifically, the low-cost oscillator is the ICS844021I (XO) clock generator, with a frequency tolerance of ± 50

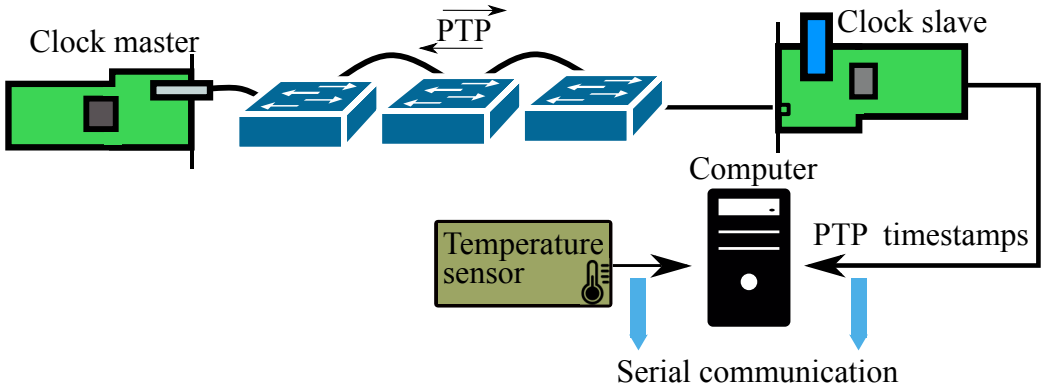


Figure 6.2: Testbed setup

ppm, which is embedded in the Virtex 7 FPGAs to test a low-cost implementation. For robust implementation testing, this study uses the AD9548 (OCXO) as the local reference clock in the slave clock side, with a frequency tolerance of ± 5 ppb.

Therefore, a substantial difference exists between low-cost and robust oscillator options. In deterministic terms, it can be stated that the selected OCXO is ten thousand times more stable than the chosen XO. Consequently, a significant performance difference between the robust and low-cost implementation scenarios is expected in this work's results. Due to this divergence, the proposed algorithm's potential to enhance the slave's accuracy in holdover operation, even in low-cost applications, can be examined.

The present work adopts a synchronization testbed architecture depicted in Fig. 6.2. In this architecture, the described switches are emulated using VLANs configured in a commercial switch, the Intelbras SG 2404 MR, which is not PTP-aware switch. This feature allows the testbed to employ several hops in the tests in an automated manner, making it possible to capture tests from one up to four hops. However, in order to conduct experiments that are closer to an ideal PTP connection scenario, as in G.8273.2/Y.1368.2 and [21], this work uses only one hop without additional virtual local area network (VLAN)s between the master and slave clocks.

The testbed is also equipped with two temperature sensors, an MCP9808, and an LM35, connected to an Arduino board for monitoring the ambient temperature effect on synchronization performance. As recommended by ITU-T G.8271.1 [7] and [21], the testbed is also provided with access to ground truth time offset, with a precisely synchronized 1PPS output measured on both the master and slave clocks. This is essential for registering the actual time offset, as well as the time offset of the proposed algorithm and the time offset in the controlled stage.

6.1.2 Datasets obtention setup

This study employs datasets generated by the testbed described in Section 6.1.1, which allows for post-processing evaluation of the proposed algorithm's impact on a PTP synchronization scenario, without the need to execute the algorithm in real-time in the testbed.

The datasets are acquired by a computer that communicates via a serial interface with the slave clock and an Arduino device, which is connected to the temperature sensors. A program running on the computer reads the serial communication, logs the temperature, and acquires timestamps (T_1, T_2, T_3 and T_4) from the slave, as well as ground truth timestamps (PPS).

The datasets comprise temperature data, PTP timestamp measurements, and ground truth (1PPS) timestamps. Using this information, the open-source PTP dataset analysis library (PTP-DAL) [53] post-processes the datasets, performing PTP calculations with timestamps from a free-running slave clock. However, it should be noted that the real free-running behavior after a locked period of several hours is not present in the datasets, as the free-running timestamps are post-processed during the analysis.

6.2 PTP time locked loop (TLL)

In numerous applications, particularly in scenarios where PTP is deployed over networks lacking timing awareness [54], the slave clock initially processes the estimates of (2.10) and subsequently applies corrections to the local clock. This process is warranted as the estimates of (2.10) are susceptible to noise originating from various factors, such as delay asymmetry between the master-to-slave and slave-to-master directions and network delay variations.

To perform this filtering, this work proposes an algorithm that initially feeds the time offset estimates from (2.10) into a proportional-integral (PI) controller, denoted as the TLL. The resulting post-processed time offset is represented as:

$$\hat{x}[n] = \hat{x}[n-1] + \kappa_p e[n-1] + \kappa_i \sum_{i=0}^{n-1} e[i], \tag{6.1}$$

where κ_p and κ_i are the proportional and integral constants, respectively, and $e[n] = \tilde{x}[n] - \hat{x}[n]$ denotes the loop error between the input measurement from (2.10) and the loop's prediction of the time offset $\hat{x}[n]$. Moreover, this work assumes $\hat{x}[0] = \tilde{x}[0]$ as the initial condition.

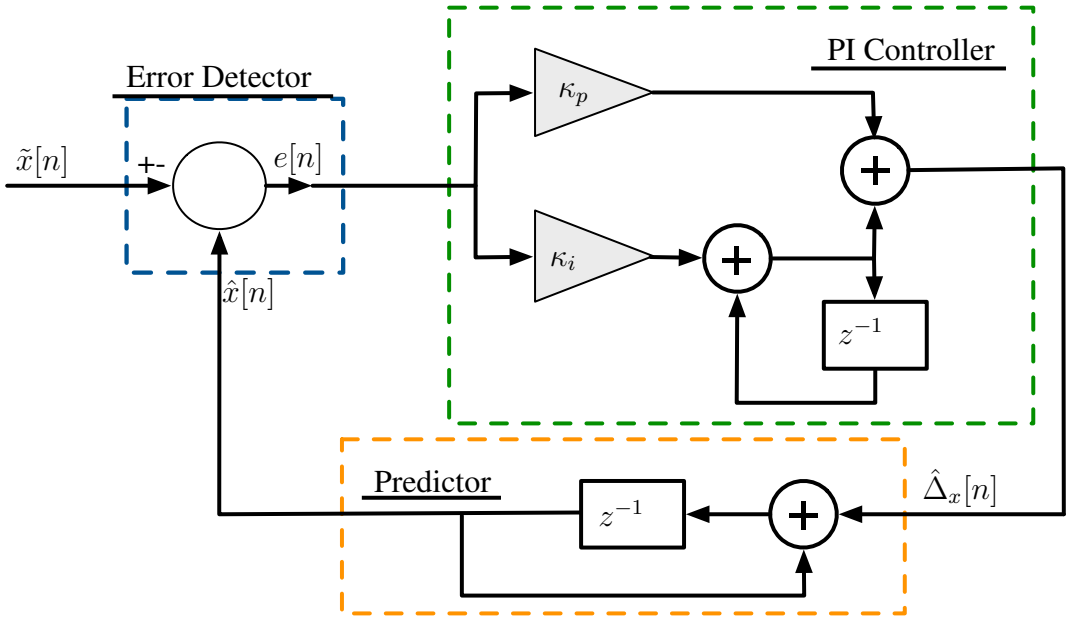


Figure 6.3: TLL structure

The purpose of the control loop is to minimize the error $e[n]$ between the input measurement from (2.10) and the loop's prediction of the time offset $\hat{x}[n]$. As the loop processes $\tilde{x}[n]$ samples over time, it learns the average time offset drift between consecutive time offset observations, leading to a reduction in the error between the measurements from (2.10) and the loop's prediction.

All raw estimation samples are processed by the TLL to produce the estimates $\hat{x}[n]$. These estimates are then utilized in the training process of the holdover model. As this work focuses on a PTS application scenario, the estimates from (6.1) are used as a feature and label in the holdover model's training, thereby simulating a real PTS scenario where the slave clock does not have access to the ground truth time offset.

6.3 Proposed architecture

To support holdover operation, the proposed model in this work utilizes past estimates of $\hat{x}$ obtained through the TLL process and past temperature values as inputs. Using these inputs, the short-term holdover models can learn a time series model to forecast the entire holdover period.

For multiple-step predictions, two common strategies are direct and iterative. The direct

approach involves training the model to predict only one step into the future, repeating the training process for each distinct prediction interval. This produces multiple models with the same parameters but trained for a specific forecast step. In contrast, the iterative approach predicts one step per iteration, utilizing previous predictions as inputs. However, this approach may lead to error accumulation in longer forecasts.

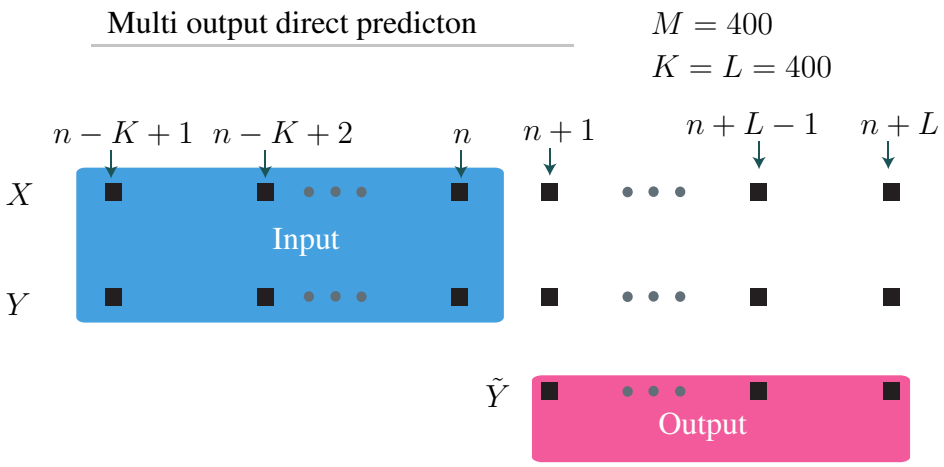


Figure 6.4: MO strategy

This dissertation uses a multiple output (MO) strategy [55], where the training algorithm fits the model to predict multiple *steps*, i.e., multiple time offset values ahead. Fig. 6.4 illustrates how this strategy works, where the parameters M, K, L respectively represent the forecast horizon, the feature window size, and label window size. Therefore, the model focuses on a pre-defined forecast horizon. In this work, the time horizon is the short holdover period, defined by [7] as 1000 seconds.

There is also the possibility of combining the MO strategy with an iterative strategy, forming the MO-iterative strategy. The main difference between that strategy is that it can update the feature window in addition to relying on past samples, as shown in Fig 6.5, it can update the feature window. Nevertheless, there are multiple trade-offs between the two strategies. The Section 8 highlights some trade-offs in utilizing the two strategies.

6.3.1 Time series forecasting method

In statistical time series models, such as ARIMA, traditionally, it is necessary to have prior knowledge of the time series' stationary state before parameter selection and fitting. Hence, the first step in these methods involves applying the autocorrelation function to analyze the statis-

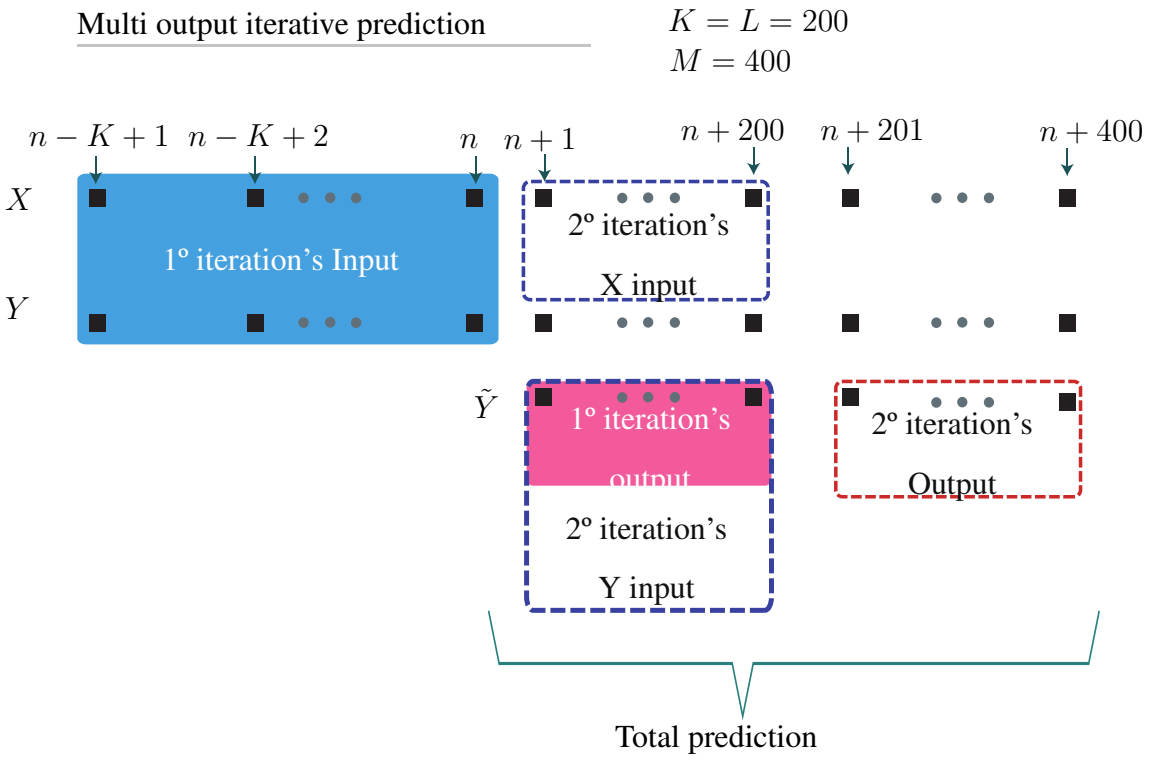


Figure 6.5: MO iterative strategy

tical properties of the time series. If the time series function does not satisfy WSS conditions, it is essential to transform the time function until it meets the WSS conditions, as described in Section 4.

However, the literature on ML approaches often does not discuss this process, as neural networks focused on time series processing such as LSTM and transformers can learn and predict short and long-term factors such as trend and seasonal patterns. Nevertheless, [56] suggests that the WSS form of the time series provides better performance as input than the original non-stationary form.

One way to obtain the WSS form of a time series is through a differentiation process, as stated in Section 4. The discrete first-difference form of the time offset is expressed using (4.4) as support:

$$\Delta_w x[n] = x[n] - x[n-w]. \tag{6.2}$$

Here, w is the window length of the discrete difference, and Δ denotes the difference operator. It is worth noting that this section focuses on estimating the time offset drift using the first differentiation of the time offset estimation. To simplify the notation, this section will refer to the first differentiation of the time offset estimation $\hat{x}$ as $\hat{\Delta}_w^x$. Thus, (6.2) can be represented

using estimations as follows:

$$\hat{\Delta}_w^x[n] = \hat{x}[n] - \hat{x}[n-w], \tag{6.3}$$

In order to satisfy the requirements of WSS, an additional differentiation is applied as follows since the time offset drifts can vary over time due to the oscillator's frequency offset fluctuations, and in some cases, the temperature can have a more significant impact on the oscillator than the frequency offset, as shown in (6.2):

$$\Delta_p \Delta_w x[n] = \Delta_w x[n] - \Delta_w x[n-p]. \tag{6.4}$$

It should be noted that the second differentiation process may not necessarily improve the performance of machine learning-based techniques. This is because it can harm the correlation between the time offset drift ($\hat{\Delta}_w^x$) and temperature samples ($\tau_p[n]$), even when the principles of co-integration are followed by applying the second differentiation to $\tau_p[n]$ as well. Moreover, the differentiation process requires a summation to reverse the differentiation, which can add noise to the result.

In our previous work [8], the second differentiation process was applied in only one case. In this present work, we have removed this process to simplify the time series processing pipeline. Therefore, it is assumed that the machine learning-based models applied in this work will learn the seasonal correlation between temperature samples and $\hat{\Delta}_w^x$, ultimately forecasting the $\hat{\Delta}_w^x$ by taking this correlation into account.

Using the samples obtained from (6.3), the model can learn and predict M steps by using previous estimations by the TLL in its training as features and labels in an auto-supervised manner. However, the pipeline must transform the time-offset drift samples into time-offset samples. Thus, the following equation expresses future time offset samples in terms of (6.3) and a past sample of $\hat{x}$:

$$\hat{x}[n+M] = \hat{x}[n-C_w] + \sum_{j=0}^{\lfloor \frac{M}{w} \rfloor} \hat{\Delta}_w^x[n - C_w + w(1+j)], \tag{6.5}$$

where, $C_w = w(\lfloor \frac{M}{w} \rfloor + 1) - M$ and $\lfloor \frac{M}{w} \rfloor$ denotes the floor operation in this division. This C_w term is an auxiliary operator in this discrete windowed integration.

Fig 6.6 illustrates the complete data pipeline, which includes decimation and interpolation steps. These steps reduce the amount of input data required to execute the model. The rationale

behind this is that if the model can perceive the overall behavior of the time offset evolution, then the sample rate can be reduced to facilitate the processing. Following the decimation and differentiation processes, the pipeline scales the input data, and finally, it reverts the scaling.

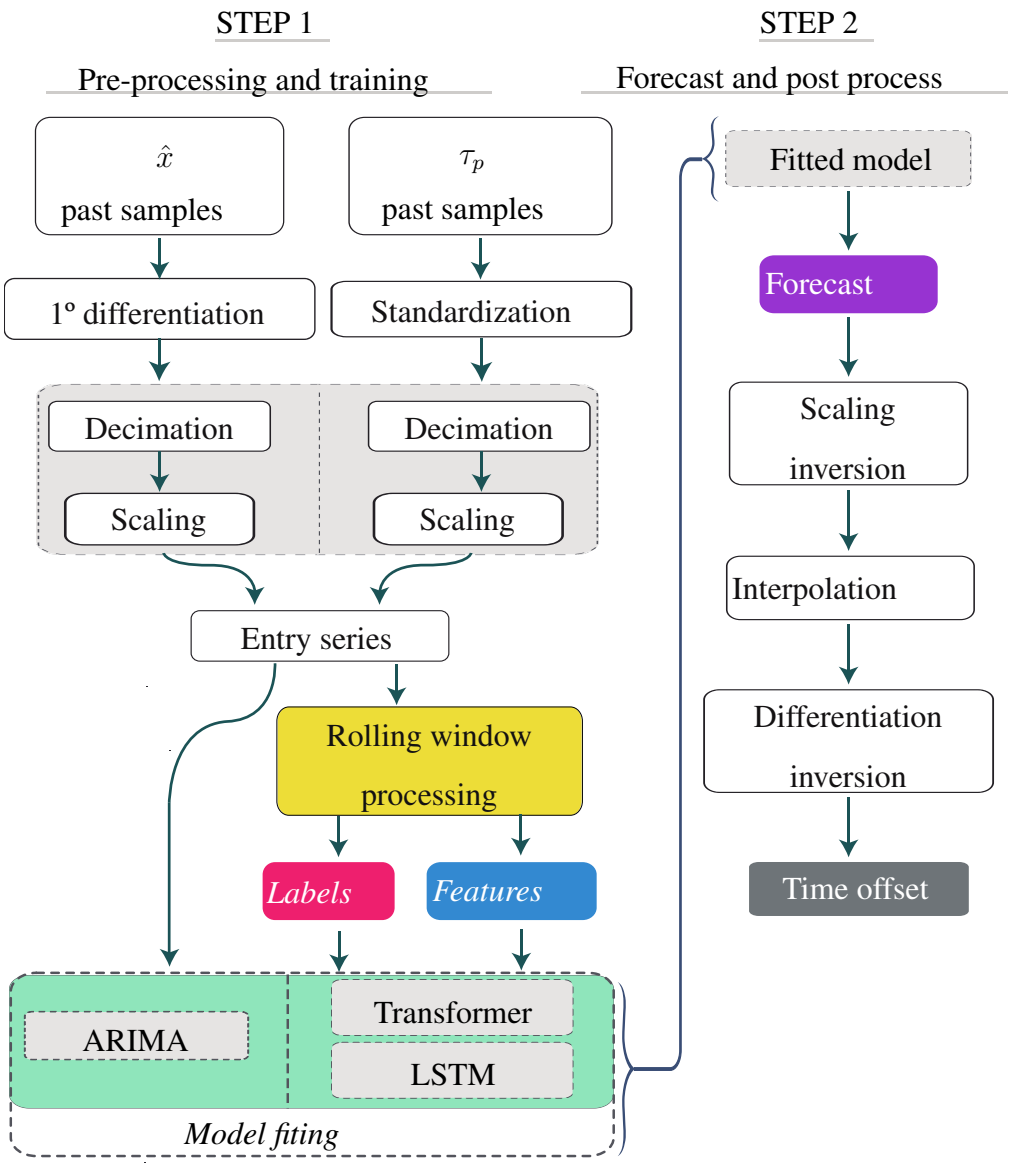


Figure 6.6: Model Data pipeline.

To prepare the samples obtained after the first step of data processing for the LSTM and Transformer networks, the pipeline formats them as features and labels, which the networks use for training and testing. The pipeline organizes the time series into windows, where each feature window consists of K differentiated time offset samples and K temperature samples. Each label window includes L differentiated time offset samples. The algorithm sequentially applies the rolling window process to the dataset, as depicted in Fig. 6.7. Before testing each

ML model, the algorithm inserts a gap of M samples between the training and test sets to ensure that the forecast does not contain overlapping information. This gap is unnecessary in the ARIMA model.

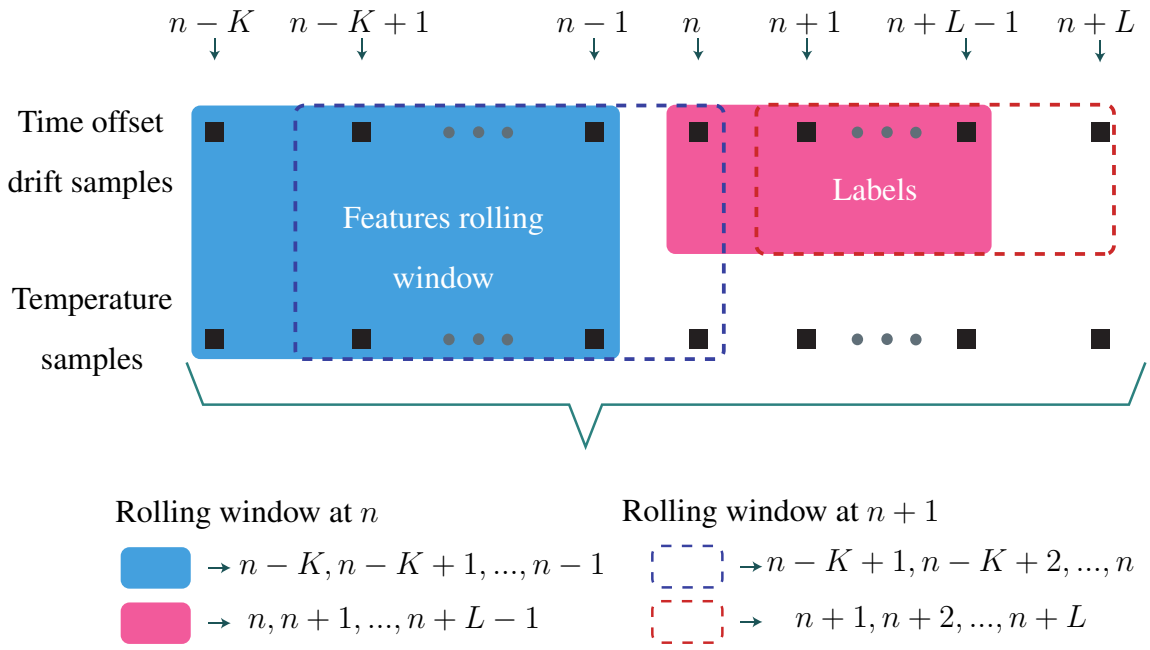


Figure 6.7: Rolling window processing.

It is important to note that this process is not required for statistical models such as the ARIMA model applied in this work. In the pipeline, the differentiated samples are directly inputted into the ARIMA model fitting process, eliminating the need for the windowing process. Fig. 6.6 shows this as a bypass to the ARIMA model.

Each feature window can be denoted as a matrix containing pre-processed temperature and time offset differentiated samples in this rolling window process. This way, a feature window starting at $n-K$, of size K that can be defined as follows:

$$F_w = \begin{vmatrix} \tau_p[n-K] & \tau_p[n-K+1] & ... & ,\tau_p[n-1] \\ \hat{\Delta}_w^x[n-K] & \hat{\Delta}_w^x[n-K+1] & ... & ,\hat{\Delta}_w^x[n-1] \end{vmatrix} \tag{6.6}$$

The rolling window process involves organizing the time series into windows, where each feature window is represented as a matrix containing pre-processed temperature and time offset differentiated samples. Alternatively, the feature window of size K starting at $n-K$ can also be denoted as:

$$F_w = \begin{vmatrix} X[n-K] & X[n-K+1] & ... & ,X[n-1] \\ Y[n-K] & Y[n-K+1] & ... & ,Y[n-1] \end{vmatrix}, \tag{6.7}$$

where Y denotes the differentiated time offset processed features, i.e, the endogenous time series, and X represents the post-processed temperature samples, the exogenous time series.

Alternatively, one can also represent the feature window using the time series notation discussed in Section 4.3. Here, the input consists of an endogenous time series Y representing the differentiated time offset processed features and an exogenous time series X representing the post-processed temperature samples. In this notation, the feature window can be defined as:

$$L_w = \begin{vmatrix} Y[n] & Y[n+1] & ... & ,Y[n+L-1] \end{vmatrix}, \tag{6.8}$$

It should be noted that both the feature and label windows utilize only the samples available on the slave clock side, which simulates a real-world PTS holdover application scenario. In such scenarios, the slave clock is unaware of the ground truth time offset.

Once the rolling window process of the input series is completed, the ML-based models are trained using the resulting feature and label windows. To test and validate these networks, they receive multi-input features and output a multi-step forecast. The information flow of the LSTM is depicted in Fig.6.8, while the transformer information flow is described in Fig.6.9. The resulting output of the networks is denoted as a prediction window, which can be defined as follows:

$$P_w = \begin{vmatrix} \hat{Y}[n] & \hat{Y}[n+1] & ... & ,\hat{Y}[n+L-1] \end{vmatrix}, \tag{6.9}$$

Finally, utilizing $L = M$, the ML-based models presented in this study are capable of forecasting all the steps of the time offset drift, which allows the data pipeline to estimate the time offset of the slave clock during the holdover operation.

It is important to note that the ML-based holdover algorithms can also integrate the MO strategy with an iterative strategy, known as the MO iterative prediction [57]. This approach enables the model to forecast the time offset using not only past samples but also data from past predictions and the most recent data. During each iteration, the model can predict L samples, and subsequently, the pipeline can update the time offset derived samples using the prediction and update L temperature samples using the most recent data. Therefore, the algorithm can employ the holdover model iteratively until reaching the total length of prediction M.

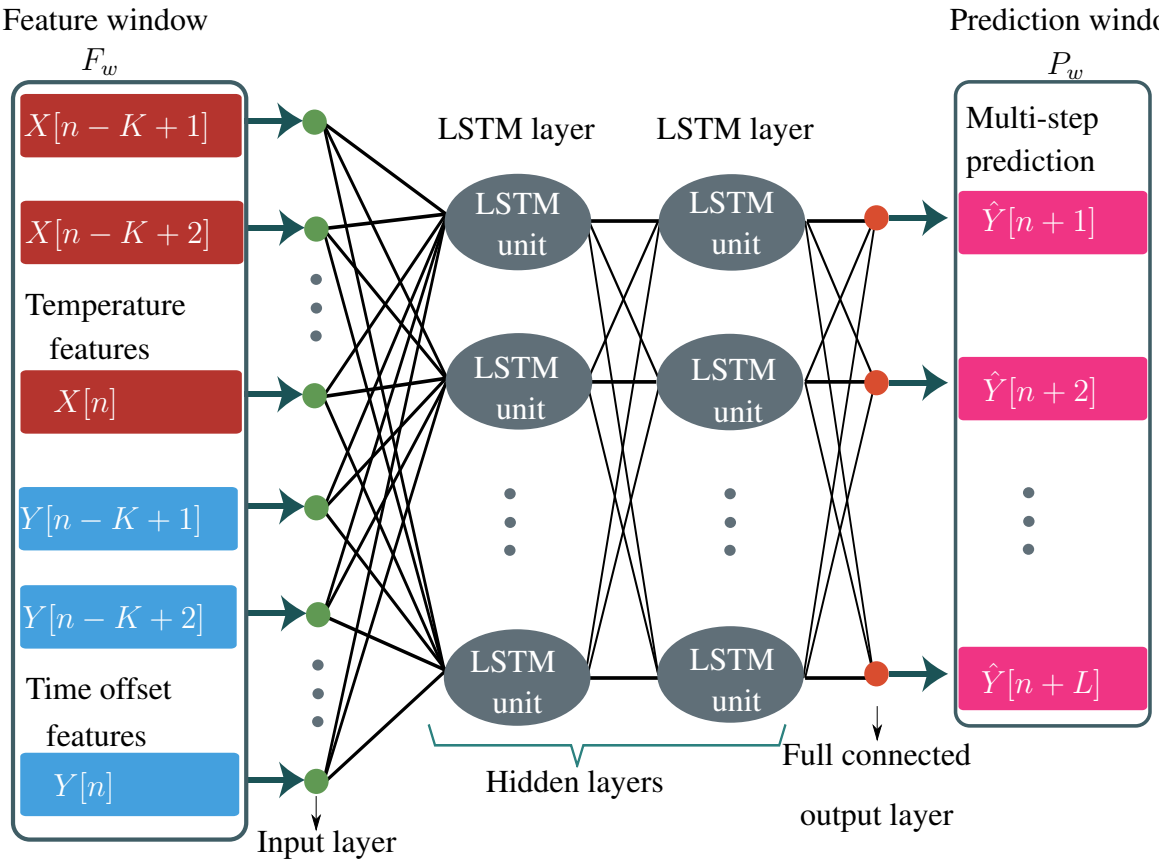


Figure 6.8: LSTM model architecture.

To determine the best combination of all model parameters, the algorithm conducts a random search, trains and validates the network multiple times through a walk-forward validation process [58], as depicted in Fig 6.10. This process utilizes samples only from the training set and creates several folds of training and validation sets, similar to a cross-validation process, but considers the time-series behavior of the input data. As a result, the algorithm does not randomize the training and validation sets, as is typical in standard k-fold cross-validation.

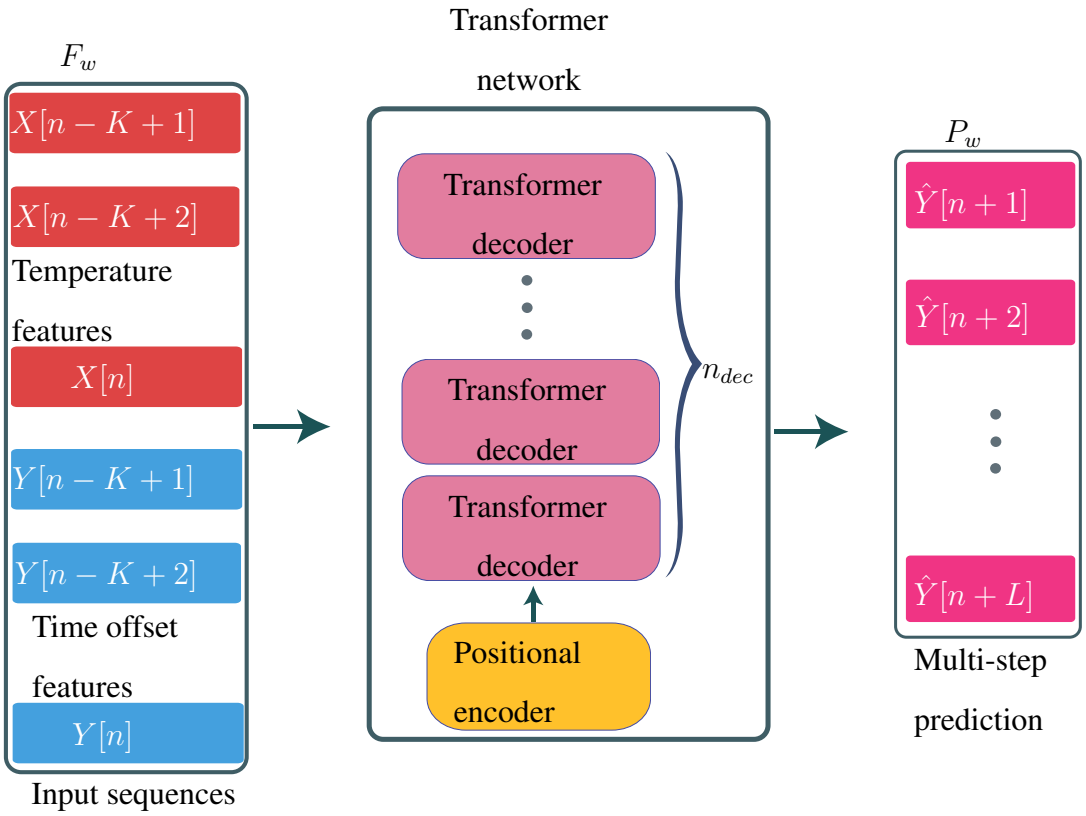


Figure 6.9: Transformer decoder architecture

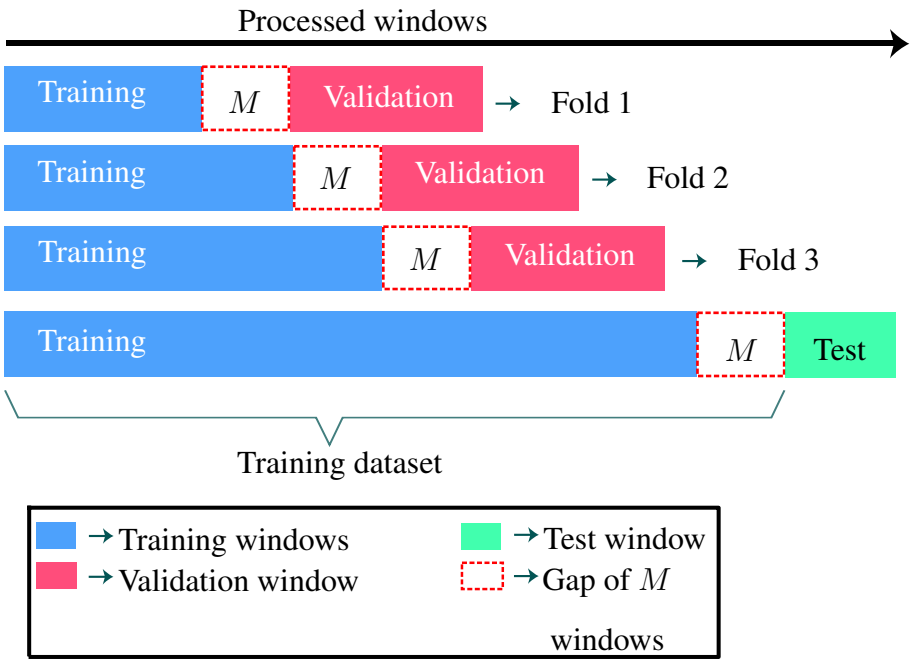


Figure 6.10: Walk forward validation.

Chapter 7

Results

This chapter details the results obtained with each model analyzed in this work. Firstly, it describes each experiment and its parameters in Section 7.1, and Section 7.2 details the baselines considered in each Experiment. After that, this chapter introduces and discusses the MTIE results obtained by the experiments in Section 7.3. Then, it discusses and introduces the TE results in Section 7.4.1. Lastly, this chapter discusses the results obtained in the sense of frequency stability, using the Allan deviation plot as the primary tool for this analysis.

7.1 Experiments

This work conducted experiments in an indoor environment characterized by distinct temperature patterns. Specifically, the experiments were divided into two categories: constant temperature, where temperature fluctuations were less than 1ºC, and variable temperature, where temperature fluctuations exceeded 1ºC. In experiments where the temperature varied, an air conditioner was used to control temperature, while it was turned off in experiments where the temperature was constant. This counter-intutitive behavior is because the air conditioner operates with cycles, introducing variability, whereas the scenario without air conditioning was recorded when the ambient temperature was naturally close to constant (no sudden environmental changes). The behavior of temperature with and without an air conditioner is depicted in Fig. 7.2, and the time offset drift behavior of the XO is shown in Fig. 7.3 for both cases.

All experiments used one hop between master and slave without further VLANs, as illustrated in Fig. 7.1. The synchronization message rate exchanged between the devices was four packets per second. Each dataset was obtained using either an XO or an OCXO as the local

oscillator. To adhere to the recommendations of ITU-T G.8271.1 [7] and [21], each dataset captured the ground truth time offset, with a precisely synchronized 1PPS output measured on the slave and master clock.

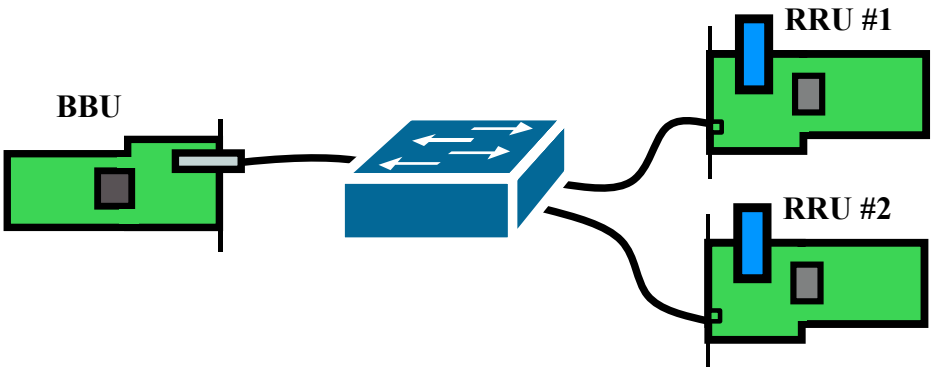


Figure 7.1: Testbed experiment setup.

Table 7.1 describes the four datasets used in the experiments. The temperature in this table corresponds to the average value obtained from two sensors, MCP9808 and LM35, using four measurements per second.

Table 7.1: Experiments description.

Experiment number	$\|\Delta$ Temp $\|$ during training (ºC)	$\|\Delta$ Temp $\|$ during holdover operation (ºC)	Oscillator
1	3.25	1.55	OCXO
2	0.375	0.34	OCXO
3	3.25	1.725	XO
4	0.75	0.3075	XO

Each dataset contains 6 hours of data, which the pipeline divides between training and test sets, according to the walk-forward validation scheme in Fig 6.10. Considering the ML-based models, the algorithm runs each model with random parameters until it encounters the best parameter combination for each model. After that, the run that obtained the best MSE metric using the walk-forward validation scheme is considered the best and therefore possesses the best parameters for the ML network and the pipeline parameters.

Now considering the ARIMA model, we perform a similar process. However, this time

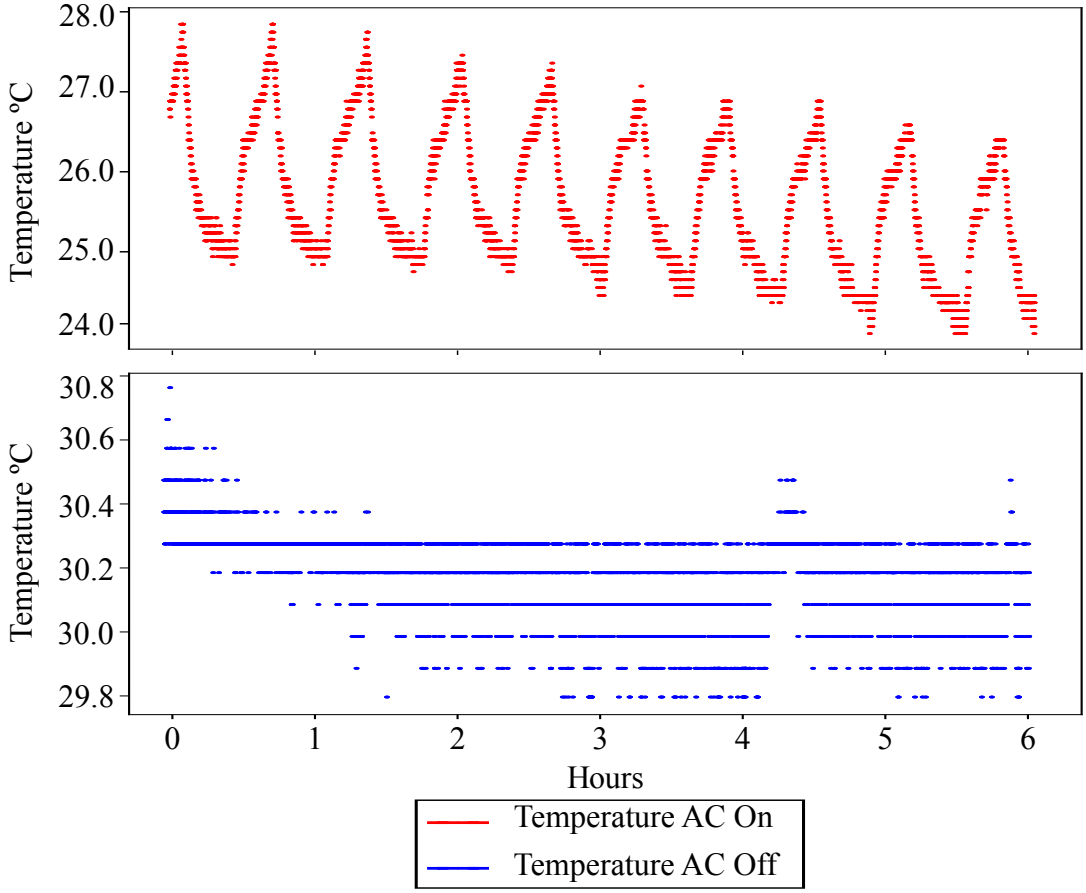


Figure 7.2: Temperature behavior depending on the air conditioner state.

the MTIE and the AIC are considered to select the best combination of parameters. After each fold, the algorithm generates the values of AIC and MTIE, the combination that produces the better values of MTIE and AIC at the same time is considered the better combination. The only parameter that is hardcoded in the algorithm is the s, as the seasonal period must be known and analyzed a priori.

Fig. 7.4 shows an example of the result of the walk-forward validation scheme for experiment 2. It is possible to see that each training and validation fold converges to similar error values as the number of epochs increases. Another strong indicator that the model did converge is the presence of similar MSE peaks in the training and validation for the same epoch and fold. This way, what the model learns in training relates directly to the same fold's corresponding validation error.

Table 7.2 presents each Experiment's time series data pipeline parameters. As all experiments employ datasets with four packets per second, the time horizon of 1000 s corresponds to 4000 samples per Experiment. Applying a decimation step order of 100 reduces the 4000

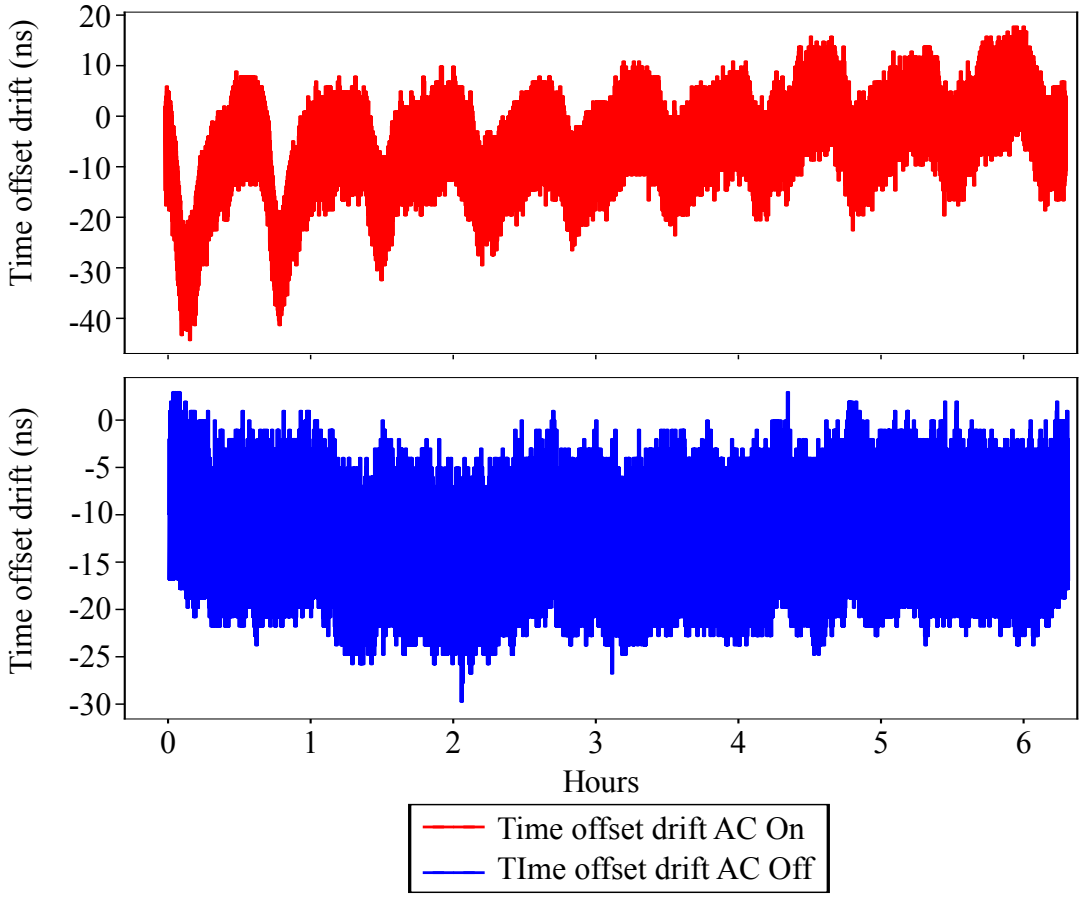


Figure 7.3: XO time offset drift behavior depending on the air conditioner state.

samples to 40. Therefore, each model generates a forecast for 40 samples, representing a time offset of 1000 s after completing all steps in the time series data pipeline. It should be noted that the parameters of the feature window (K) only apply to machine learning-based models. In contrast, the ARIMA model employs all available steps before conducting the forecasting.

In addition, Table 7.2 illustrates that this study employed matching values of drift window (w) and decimation step order. However, to execute the ARIMA model training process with seasonal components, the decimation step had to be increased as the computer utilized for the models' training process was incapable of handling the training process of SARIMA models. Therefore, different combinations of decimation steps and w were tested to find an appropriate match. After conducting numerous experiments, it was discovered that matched numbers produced the least amount of noise in the end-to-end process. As a result, we increased the decimation step to 100 while keeping the level of noise generated by the pipeline at low values.

Table 7.3 describes the best parameters encountered by each model. Each model takes 4 hours of data in the training process. The test set contains 1000 seconds of data taken imme-

Table 7.2: Experiments pipeline parameters.

Experiment number	Drift window w	Decimation step order	Feature window K	Label window L
1	100	100	40	40
2	100	100	40	40
3	100	100	40	40
4	100	100	40	40

diately after the gap of M samples. This division ensures the training algorithm fits the model with the most recent data before holdover.

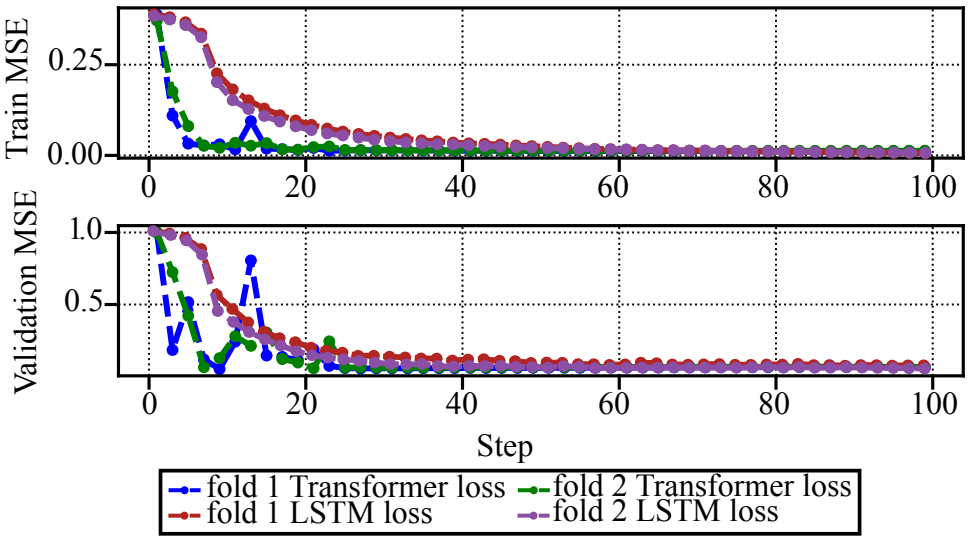


Figure 7.4: Experiment 2 walk forward validation MSE.

Table 7.3: Model parameters for each Experiment

Panel A: ARIMA parameters

Experiment number	p	q	d	P	Q	D	s
1	5	0	1	1	1	0	88
2	0	0	3	0	0	0	0
3	4	0	5	2	1	1	88
4	2	0	3	0	0	0	0

Panel B: LSTM parameters

Experiment number	LSTM units per layer	Hidden layers hl	Multi step strategy
1	120	2	MO iterative
2	80	2	MO iterative
3	100	1	MO iterative
4	190	1	MO

Panel C: Transformer parameters

Experiment number	$\mathbf{d_{model}}$	Number of decoder layers n_{dec}	Number of attention heads $\mathbf{H}$	Multi step strategy
1	128	3	2	MO iterative
2	128	3	8	MO
3	64	3	8	MO
4	64	2	4	MO

7.2 Baselines

Before presenting this study's results, it is pertinent to elaborate on the benchmarks that will be employed to evaluate our results. As elaborated in Sections 2.6, 3.4, and 3.3.1, this work incorporates a range of specifications and standards to assess the efficacy of a software-based holdover mechanism. These criteria encompass MTIE masks and TE thresholds and curves.

Regarding the MTIE metric, this work follows the recommendation provided in [7] as the primary requirement. This recommendation outlines two MTIE masks for situations where only frequency layer input is available and no time input is present. These masks consider an ideal PTP link that enters the holdover state within an observation interval of 1000 seconds, with one mask assuming a constant temperature (within ±1K) and the other assuming a variable temperature (above ±1K), while still connected to a SyncE frequency input. However, due to the utilization of a commercial PTP-unaware switch (Intelbras SG 2404 MR) in the hardware architecture of this study, an ideal PTP link was not employed. As a result, the tests conducted in this study may not be compatible with these masks. Nevertheless, the masks proposed in [7] can serve as a baseline for the MTIE results obtained in this study, as indicated in Section 3.4.

In addition to the MTIE masks, this study aims to examine the MTIE outcomes reported by [21]. However, it should be noted that our work cannot be directly compared to their results since we tested the holdover operation under a non-ideal PTP connection, whereas this was not the case in [21].

This study considers two scenarios regarding the TE requirements. In the first scenario, the slave clock must adhere to the overall TE budget, which considers the link asymmetry and other noise sources, as described in Section 2.6. In the second scenario, the slave clock must comply only with the holdover operation-specific requirements. However, this particular requirement may vary depending on the network topology. Although our experiments do not fit into a FTS scenario, we will use both FTS, and PTS TE baselines imposed respectively by the recommendations ITU-T 8271.1 [19], and 8273.4 [7].

Concerning the related work in the holdover literature, we can use the studies [23]-[22] as baselines for the TE metric, which Table 3.2 describes the values encountered by both references. Although, one should keep in mind that our work is not directly comparable to the tests made by [23]-[22], as discussed in Section 3.3.

Alongside the literature, we will also compare the current results with our previous work [8]. This way, we can better understand the benefits of the time series pipeline in this work

compared to the last work.

7.3 MTIE results discussion

Fig. 7.5 presents the MTIE results of the OCXO datasets. The results with constant temperature show that all models could learn the patterns of the PTP locked operation's time offset and predict a time offset within the baseline.

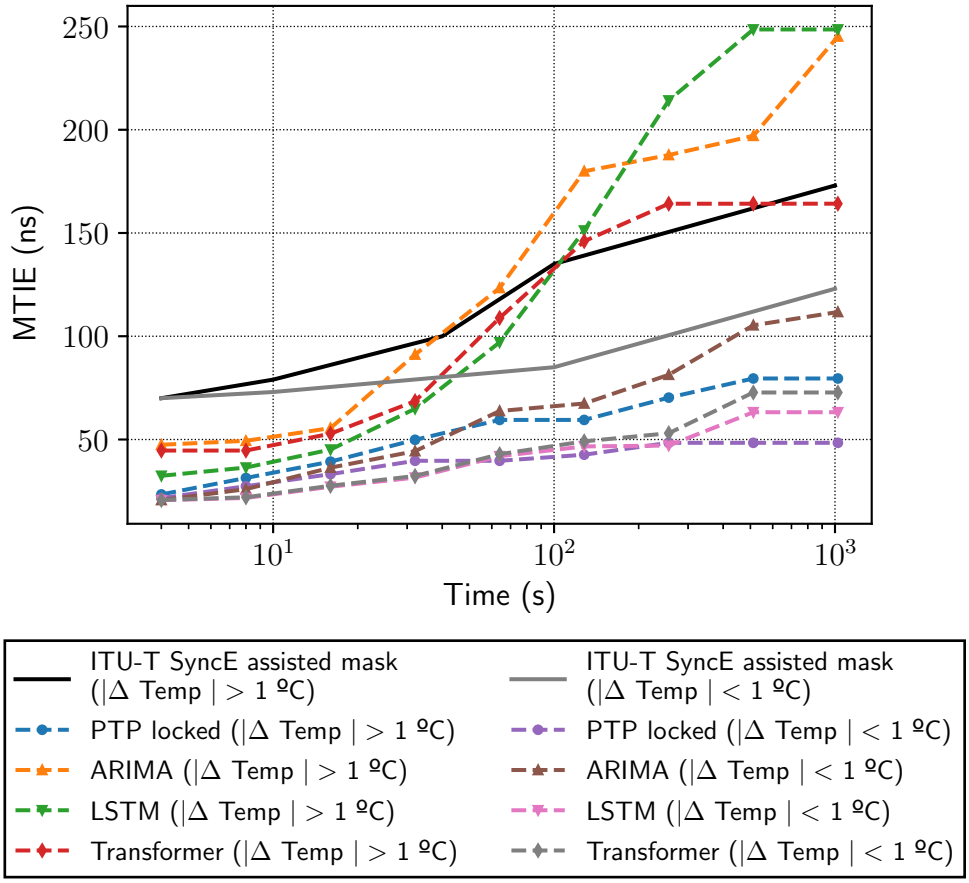


Figure 7.5: OCXO MTIE results.

In Experiment 1, the temperature exhibits a sinusoidal pattern. Similarly, the time offset drift also displays a sinusoidal pattern. Consequently, this temperature variation affects the frequency of the OCXO, as evidenced by the magnitude of the MTIE values of the PTP-locked OCXO in the variable temperature scenario compared to the constant temperature scenario. This significant difference is also visible in the performance of each model, where only the results for the transformer network are close to fitting in the ITU-T MTIE mask for a variable temperature.

Furthermore, it is worth considering that the frequency-temperature relationship in OCXOs and TCXOs may exhibit complex characteristics. This is because the frequency error incorporated in the oscillator frequency control mechanism is a residual error of the OCXO compensation scheme, as noted by [27]. However, this does not occur in the XO, as the entire temperature effect is passed to the frequency. Thus, Experiment 1 may exhibit the most complex frequency-temperature behavior of all experiments, and each model needed to be more complex than in the constant temperature scenario. For example, in this scenario, the transformer network required the use of the Multi-Objective (MO) iterative strategy instead of MO.

In our previous work [8], we employed a second differentiation process in this scenario using a decimation step of 10. With these pipeline parameters, the Long Short-Term Memory (LSTM) achieved an MTIE value of 175 ns in a observation period of 1000 seconds. However, in this work, the LSTM could not achieve this value, which can be attributed to the absence of the second differentiation process. Instead, we achieved a mark of 166 ns using a Tranformer network, while reducing the amount of data required to train and forecast by ten-fold by applying a decimation step of 100. This reduction in the amount of data needed for training and forecasting is an exciting step towards embedding ML-based solutions in the slave clock.

The constant temperature XO MTIE results in Fig. 7.6 shows that the ML based models could achieve sub-millisecond accuracy, achieving values close to 750 ns of error in both models. On the other hand, this was not the case for the ARIMA model, which achieved the mark of 1076 ns. Our last work [8] achieved the final MTIE value of 1208 ns, which is worse than every single model employed in this work. This difference can be explained by the optimizations of the time series pipeline used in this work by using matched values of w and the decimation step.

In a similar test in [21], a slave clock with a TCXO achieves 3922 ns of maximum MTIE considering a observation period of 1000 seconds in a free-running state for a constant temperature scenario. Thus, this shows the potential of software-based holdover applications.

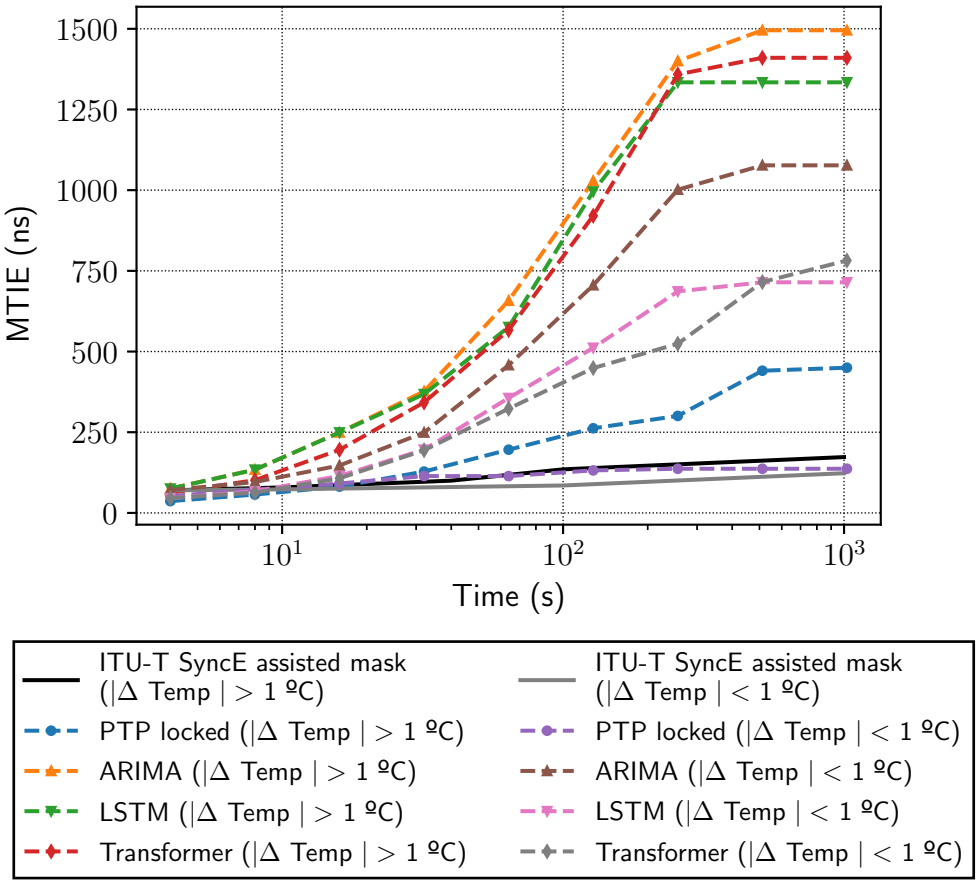


Figure 7.6: XO MTIE results.

Experiment 3 was also exposed to a variable temperature scenario with a seasonal pattern, similar to Experiment 1. However, the temperature had a more significant impact on the stability of the local oscillator, as the XO lacks any mechanism to provide temperature compensation in its frequency stability. This impact can be observed by comparing the values of PTP locked operation between the two temperature scenarios. In the variable temperature scenario, all models achieved similar results with MTIE values below 1500 ns, with the LSTM model achieving the lowest loss.

It is worth mentioning that our previous work [8] achieved an MTIE value of 1750 ns using the LSTM model. Hence, all the models used in this work have surpassed our previous results. This improvement can be attributed to the changes made in the time processing pipeline employed in this work compared to our previous work.

During each Experiment, the slave clock operates in a free-running mode. However, the actual free-running behavior of the oscillators after multiple hours of locked operation is not captured in the offline post-processing, where the proportional-integral (PI) controller calculates

the PTP locked operation. To estimate the free-running behavior, one can use (6.5) and consider the last drift of window w present for the entire holdover period as the only correction to the time offset value. Fig.7.7 depicts the estimated free-running behavior of all experiments.

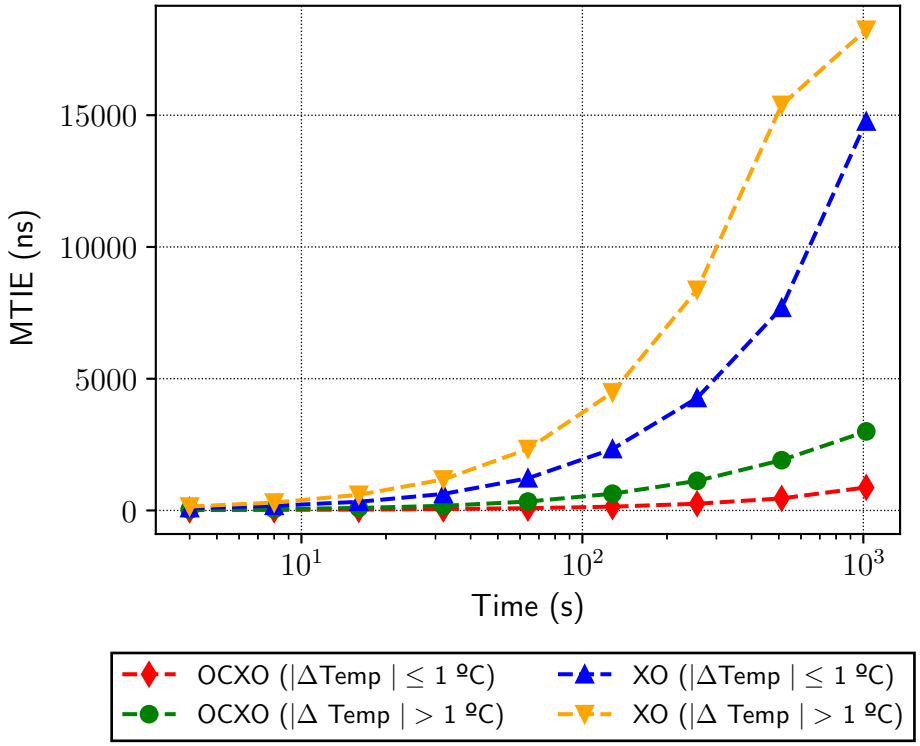


Figure 7.7: Free running MTIE of all experiments

As anticipated, the experiments that employed the OCXO as the reference clock yielded higher accuracy than the ones using XO. The temperature was a critical factor that affected the performance of the slave clocks. In Experiments 1 and 3, the temperature followed a period pattern resembling a sinusoidal wave, whereas, in the constant temperature datasets presented a behavior similar to a logarithmically variation within the range of 1 ºC. Both temperature patterns remained stable throughout the training and test sets. Therefore, when the temperature variation is regular and expected to fluctuate less than 1 Kelvin, using only past samples in the MO direct strategy enhances the model's performance. This way, the algorithm trains the model to counteract the effects of regular temperature variations on the slave clock's time offset drift, as was the case in Experiment 4 (constant temperature with the XO).

However, in cases where the temperature pattern is irregular or the oscillator is not tolerant to temperature variation, using the MO iterative strategy may yield better results than relying solely on past samples to forecast the entire holdover operation time offset.

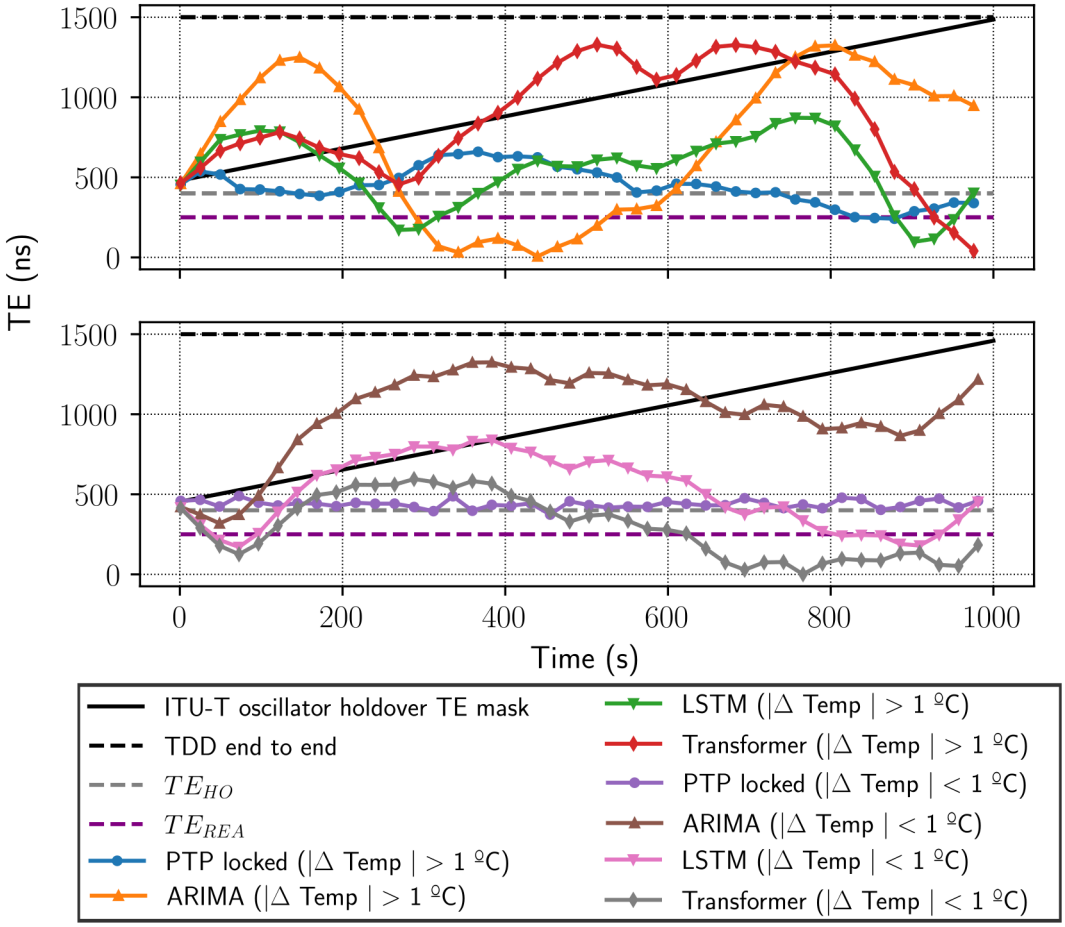


Figure 7.8: XO |TE| results

7.4 TE results discussion

Fig 7.8 displays the results of $|TE|$ for the XO in Experiments 3 and 4. In both experiments, all models were able to meet the TDD TE time budget of 1.5 ms, despite the strong asymmetry caused by the use of PTP-unaware infrastructure. However, none of the models could comply with the TE baselines recommended by the ITU-T.

In related works [22]-[23], similar holdover performance tests were conducted in a smart grid scenario with constant temperature, using PTP-aware infrastructure and OCXOs or TCXOs as slave clocks. The best and worst results achieved maximal TE of 1736 and 10256, respectively, as summarized in Table 3.2. Therefore, we can infer that the results of Experiments 3 and 4 compared to [22]-[23] demonstrate the capabilities of the software-based holdover. Moreover, in contrast to our previous work [8], all models in this study achieved results within the TDD TE threshold, whereas our last work results could not achieve the same mark.

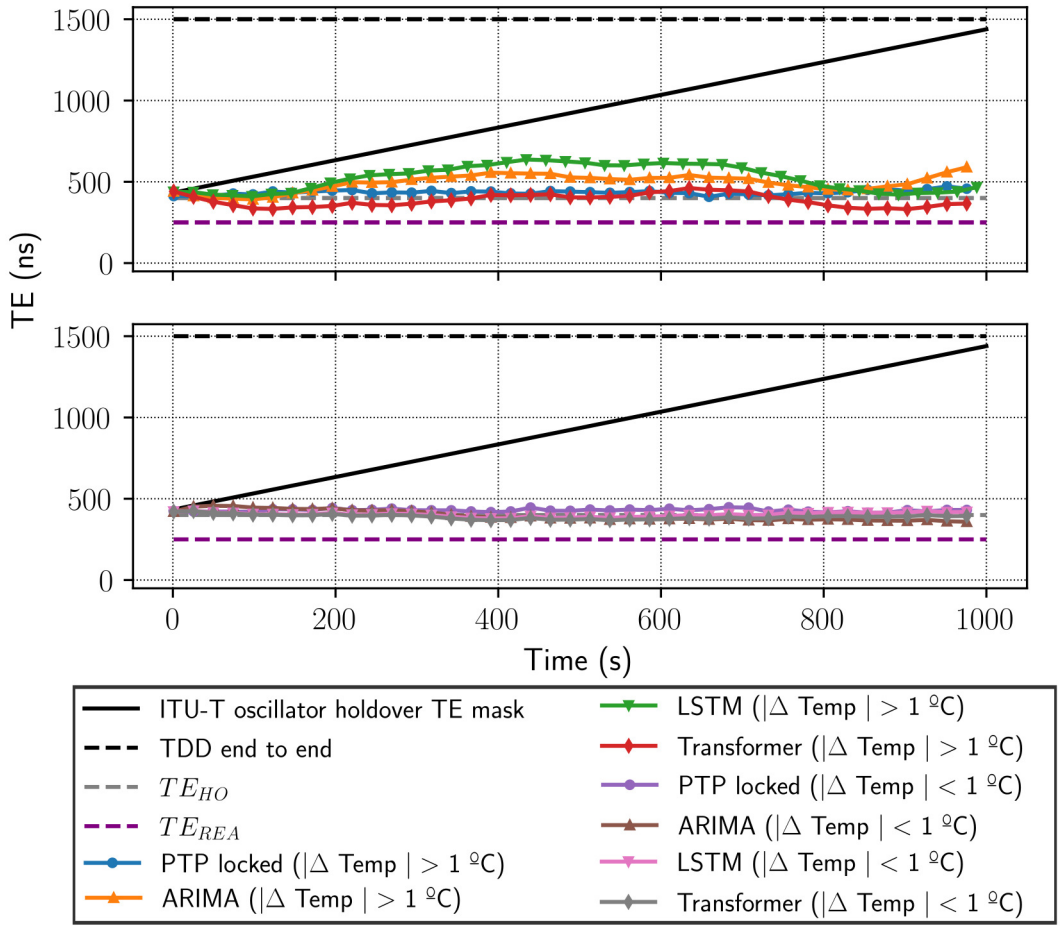


Figure 7.9: OCXO |TE| results.

Similarly to the XO results, the OCXO results in Fig. 7.9 managed to comply with the TE baseline budget even with the influence of the link asymmetry. Furthermore, even the enhanced clock stability provided by the use OCXO as the slave clock oscillator was enough to comply with the selected TE_{REA} and TE_{HO}. Besides these two baselines, the Experiments 1 and 2 had achieved the TDD and the phase error mask proposed by ITU-T G.8273.4.

Considering an APTS scenario where the slave clock has access to a GNSS reference, and it can use it to solve the link asymmetry, the XO results without the link asymmetry were greatly improved, however, even with the asymmetry correction the TE requirements besides the TDD were not able to be achieved. The OCXO results, in Fig. 7.12, also were significantly improved. Without the asymmetry, the OCXO experiments were able to maintain the TE error inside the margin of all TE requirements considered in this work.

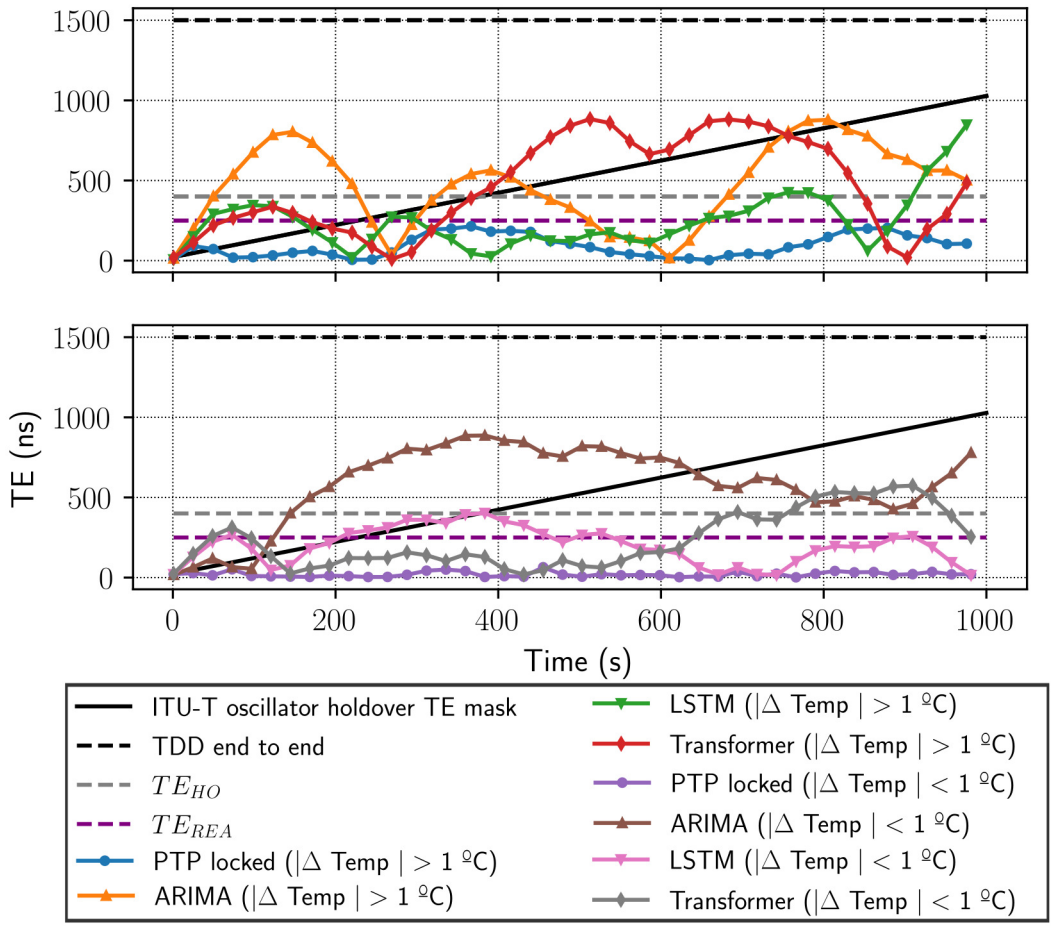


Figure 7.10: XO |TE| results results without bias.

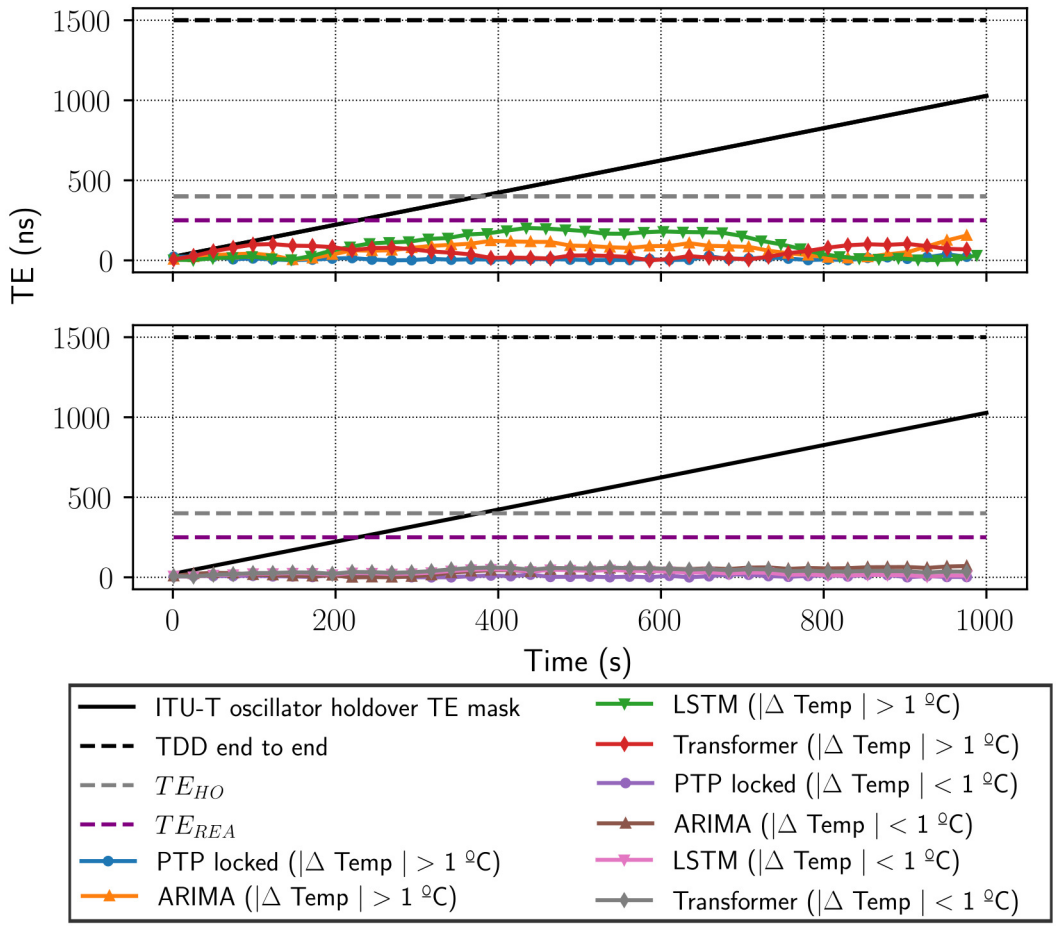


Figure 7.11: OCXO |TE| results without bias.

7.4.1 Allan deviation results discussion

Now considering the results in terms of Allan deviation, Fig 7.12 describes the results of Experiments 1 and 2 in terms of Allan deviation. In both scenarios, the free-running behavior resembles the classical free-running behavior, with a strong influence of random walk FM noise as the averaging time increases. Besides that, in Experiment 1, all the models analyzed reached similar levels, with the transformer network reaching slightly better results. For Experiment 2, all models reached the same level as the free running, reaching similar values to the PTP locked in many points.

Thus, for the OCXO, there is no significant influence of the proposed models to improve the frequency stability. The fact that the free running reached similar values to the algorithms contributes to this conclusion. This fact can be explained by the robust frequency stability mechanisms already present in the OCXO clock.

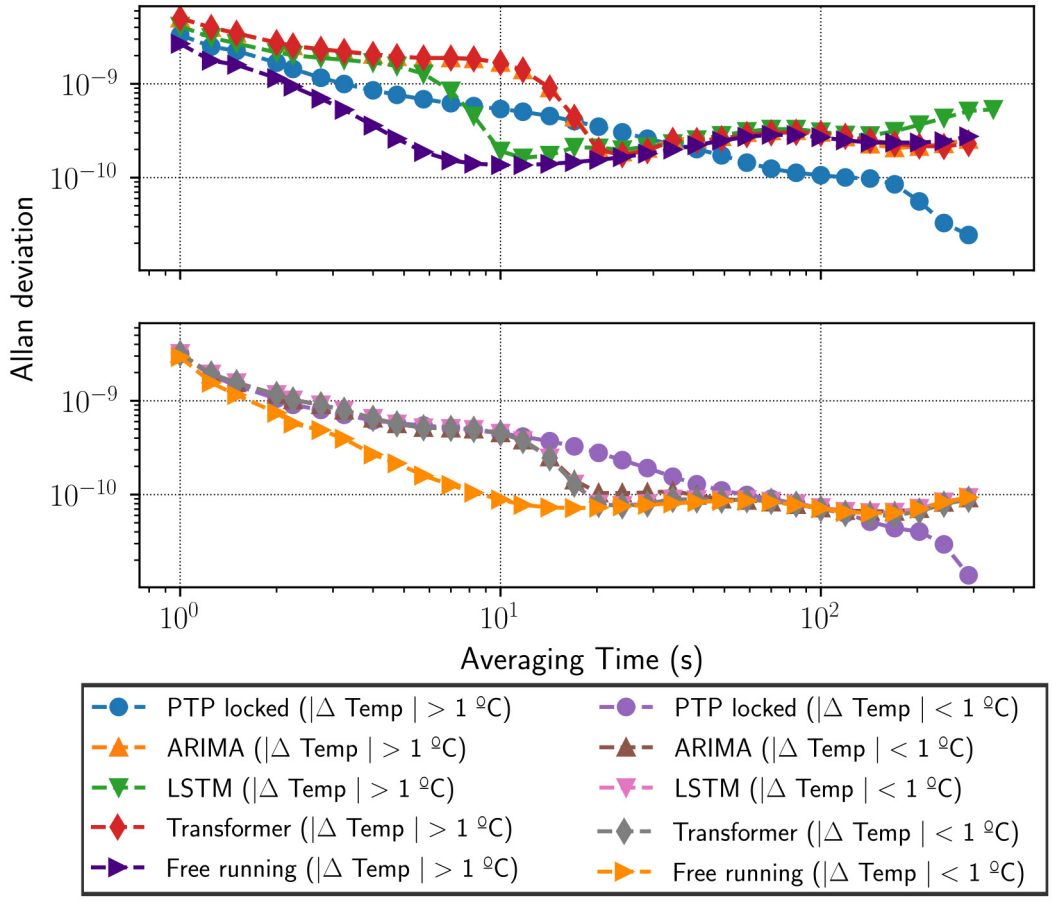


Figure 7.12: OCXO Allan deviation results

Considering the XO case, a similar conclusion can be made in the constant temperature

scenario, where each model comes close to reaching the same levels as the free-running result. On the other hand, the variable temperature scenario shows promising results regarding temperature compensation, as all models deviate from the free-running behavior and come close to the PTP locked values, with the slight advantage of the LSTM.

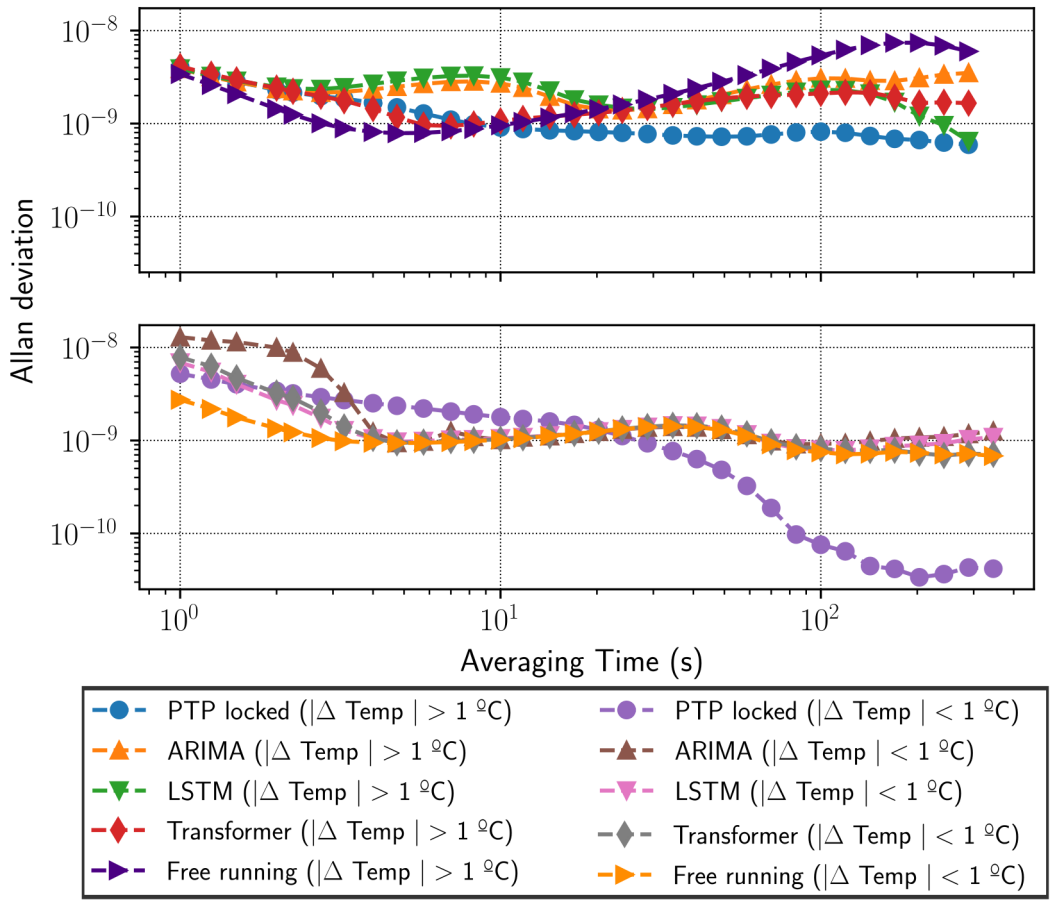


Figure 7.13: XO Allan deviation results

Chapter 8

Conclusion

This work presented a software-based holdover evaluation using three different models, a statistical-based model ARIMA, and two ML-based models, the LSTM and the transformer network. Alongside that, this software-based holdover experimented on various datasets containing different local oscillators and temperature variations. First, considering the OCXO results, the sole model that could almost comply with all MTIE masks proposed by [7] was the transformer network, even though during the PTP link is not ideal during the locked state. Considering the results obtained by each model with XO, it is safe to conclude that the models did not achieve the same accuracy as the OCXO. However, it could still improve the slave clock performance during the short-term holdover operation. In fact, it could achieve the limits for TDD operation, even considering the effect of link asymmetry.

Nevertheless, considering our last work [8] results, it is safe to assume that our optimized version of the data pipeline could provide better results in almost every case if compared to the last work. Besides that, optimizing the pipeline enabled us to reduce the amount of data used in each model tenfold, improving the possibility of embedding this solution in a slave clock.

Alongside that, the results of [21], [22], and [23] show the potential of our software-based approach, as the results obtained by these authors in free-running tests with TCXO oscillators in a similar test environment than us reached MTIE and Maximum |TE| worse than our software-based holdover approach using XO oscillator.

It is possible to conclude that in Experiment 3, software-based holdover using every model could compensate, in part, for the high drift variation caused by the temperature changes. This is demonstrated by the values of the Allan deviation analysis, where all three models produced values below the estimated free-running behavior.

For evaluating the best model for a software-based approach, it is crucial to consider that the transformer network and LSTM models reached very similar values in several results. However, the transformer model reached significantly better values in Experiment 1, which is the experiment that may present the most complex frequency-temperature behavior. This way, the transformer network may be a better choice for the software-based holdover application. It will always depend on the slave clock's hardware capabilities. The standard transformer network has several benefits compared to the LSTM, but embedding model sizes are not one of those. Thus it will always depend on the slave clock capabilities. The ARIMA model could provide a good performance tradeoff if a very lightweight software-based holdover is needed. On the other hand, depending on the slave clock hardware, the use of LSTM or Transformer networks could provide a great solution.

Another notable factor is that our work explored different situations than those presented in the related works. Our proposed software-based approach could also be expanded to other use cases besides the PTP slave case. As our approach is generalistic, it could be easily adapted to new topologies.

8.1 Future works

The current work is situated in a scenario of PTS, where the application solely relies on PTP for providing time and frequency distribution between the master and slave clocks. Nonetheless, the proposed method can be applied in the regular PTP operation, serving to filter the estimations done by the PI-TLL samples of equation (6.1), or even replace the PI-TLL, thus filtering the noisy PTP samples of equation (2.10).

Future research endeavors could further investigate additional holdover scenarios, such as an extended time horizon, where the model would target hours instead of minutes. In some cases, a short time horizon could enhance time synchronization, such as in mitigating packet loss. Here, the holdover model could estimate the time offset when packet loss is detected and revert to the locked control algorithm as soon as the application identifies the packets again.

Moreover, there is a possibility to explore the software-based holdover application in small periods, noisy signals, or intentionally relying on the holdover mechanism instead of the PTP locked signal depending on the situation. This could be useful in expanding the current work in scenarios where, for example, a significant increase in background packets is detected

in the networks, leading to a degradation of the PTP locked timing notion.

Another potential area of exploration is the use of massive training data in the Transformer network as a means of pre-training. As noted by [49], the pre-training of transformer networks for forecasting has not been extensively studied in the time series literature. As our data could be categorized based on the number of hops in the network, temperature variation, oscillator used in the slave clock, and background traffic, we could use the data specific to a situation to pre-train the transformer model and subsequently fine-tune it for the particular scenario.

The scenario of APTS also offers exciting possibilities for a ML model. In this scenario, PTP assists a more accurate synchronization source, such as GNSS. This approach enables an algorithm in the slave clock to learn the asymmetry when locked to the primary synchronization source. If this source becomes unavailable, PTP can provide time distribution using an ML algorithm for bias correction.

Another exciting avenue for investigation in the APTS scenario would be real-time training, i.e., the online learning training strategy of the model during the locked operation. The model could be trained during this operation, considering the primary source's time offset as a label and the PTP time offset measurement as a feature. When the primary source is lost, the PTP time offset estimation can be improved by the online learning model.

Outside the holdover operation, some concepts in this work could also be expanded. As seen in the results of the PTP locked synchronization in the XO experiment with variable temperature, it suffers from temperature degradation, even with the presence of PTP synchronization. Therefore, it would be interesting to test the ML-based algorithms proposed in this work to provide temperature compensation to the slave clock, similar to what [27] proposed.

Bibliography

[1] F. Girela-López *et al.*, "IEEE 1588 High Accuracy Default Profile: Applications and Challenges," *IEEE Access*, vol. 8, pp. 45 211–45 220, 2020.

[2] I. Godor, M. Luvisotto, S. Ruffini, K. Wang, D. Patel, J. Sachs, O. Dobrijevic, D. P. Venmani, O. Le Moult, J. Costa-Requena *et al.*, "A look inside 5g standards to support time synchronization for smart manufacturing," *IEEE Communications Standards Magazine*, vol. 4, no. 3, pp. 14–21, 2020.

[3] F. Cavaliere, P. Iovanna, J. Mangues-Bafalluy, J. Baranda, J. Núñez-Martínez, K.-Y. Lin, H.-W. Chang, P. Chanclou, P. Farkas, J. Gomes *et al.*, "Towards a unified fronthaul-backhaul data plane for 5g the 5g-crosshaul project approach," *Computer Standards & Interfaces*, vol. 51, pp. 56–62, 2017.

[4] " ECPRI Interface Specification. ," Common Public Radio Interface (CPRI), Available: http://www.cpri. info/spec.html v2.0, 2023.

[5] "G.810:definitions and terminology for synchronization networks," ITU-T, Standard, 1996.

[6] H. Zhou *et al.*, "Adaptive correction method for an OCXO and investigation of analytical cumulative time error upper bound," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control.*, vol. 58, no. 1, pp. 43–50, 2011.

[7] "G.8273.2/Y.1368.2:Timing characteristics of telecom boundary clocks and telecom time slave clocks for use with full timing support from the network," ITU-T, Standard, 2020.

[8] R. Dutra, I. Freire, P. Bemerguy, A. Klautau, I. Almeida, and E. Medeiros, "An lstm-based approach for holdover clock disciplining in ieee 1588 ptp applications," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[9] ITU-T, “Rec. G.8273.2:Timing characteristics of telecom boundary clocks and telecom time slave clocks,” Aug. 2019.

[10] J. R. Vig, “Introduction to quartz frequency standards. revision,” ARMY LAB COMMAND FORT MONMOUTH NJ ELECTRONICS TECHNOLOGY AND DEVICES LAB, Tech. Rep., 1992.

[11] D. Bladsjö *et al.*, “Synchronization aspects in LTE small cells,” *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 70–77, 2013.

[12] I. Freire, “FPGA Implementation and Evaluation of Synchronization Architectures for Ethernet-based Cloud-RAN Fronthaul,” Master’s thesis, Universidade Federal do Pará, 2016.

[13] 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.

[14] “Ieee standard for a precision clock synchronization protocol for networked measurement and control systems,” *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pp. 1–499, 2020.

[15] D. W. Allan *et al.*, “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.

[16] P. Skurowski and M. Pawlyta, “On the noise complexity in an optical motion capture facility,” *Sensors*, vol. 19, no. 20, p. 4435, 2019.

[17] E. Fernández, D. Calero, and M. E. Parés, “Csac characterization and its impact on gnss clock augmentation performance,” *Sensors*, vol. 17, no. 2, p. 370, 2017.

[18] *G.8271.1/Y.1366.1: Time and phase synchronization aspects of telecommunication networks,Amendment 1*, ITU-T, March 2018.

[19] *G.8271.1/Y.1366.1: Network limits for time synchronization in packet networks with full timing support from the network*, ITU-T, March 2020.

[20] 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.

[21] R. M. Kaminsky, "Test results of IEEE 1588v2 Network Synchronization Holdover Performance using Various Types of Reference Oscillators," in *2019 ISPCS*. IEEE, 2019, pp. 1–5.

[22] J. Amelot, J. Fletcher, D. Anand, C. Vasseur, Y.-S. Li-Baboud, and J. Moyne, "An ieee 1588 time synchronization testbed for assessing power distribution requirements," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2010, pp. 13–18.

[23] J. Amelot, Y.-S. Li-Baboud, C. Vasseur, J. Fletcher, D. Anand, and J. Moyne, "An ieee 1588 performance testing dashboard for power industry requirements," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2011, pp. 132–137.

[24] C. Nicholls and G. Carleton, "Adaptive OCXO drift correction algorithm," in *Proc. 2004 IFCS, 2004.* IEEE, 2004, pp. 509–517.

[25] Y. S. Shmaliy and L. Arceo-Miquel, "Efficient predictive estimator for holdover in GPS-based clock synchronization," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control.*, vol. 55, no. 10, pp. 2131–2139, 2008.

[26] D. Pallier, V. Le Cam, and S. Pillement, "Energy-efficient gps synchronization for wireless nodes," *IEEE Sensors Journal*, vol. 21, no. 4, pp. 5221–5229, 2020.

[27] V. Vozár and T. Kovácsházy, "Self-learnning of the dynamic, non-linear model of frequency-temperature characteristic of oscillators for improved clock synchronization," in *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, 2021, pp. 1–6.

[28] V. J. Mathews, G. Sicuranza *et al.*, *Polynomial signal processing*. John Wiley & Sons, Inc., 2000.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 11 106–11 115.

[31] W.-H. Hsu *et al.*, "Frequency calibration based on the adaptive neural–fuzzy inference system," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 4, pp. 1229–1233, 2009.

[32] ——, "SVM-based fuzzy inference system (SVM-FIS) for frequency calibration in wireless networks," in *Proc. 3rd ICCIT*. Citeseer, 2009, pp. 207–212.

[33] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," *Advances in neural information processing systems*, vol. 32, 2019.

[34] P. J. Brockwell, R. A. Davis, and M. V. Calder, *Introduction to time series and forecasting*. Springer, 2002, vol. 2.

[35] G. E. Box, G. M. Jenkins, and G. C. Reinsel, *Time series analysis: forecasting and control*. John Wiley & Sons, 2011, vol. 734.

[36] M. H. DeGroot, M. J. Schervish, X. Fang, L. Lu, and D. Li, *Probability and statistics*. Addison-Wesley Reading, MA, 1986, vol. 2.

[37] K. Pearson, "Vii. note on regression and inheritance in the case of two parents," *proceedings of the royal society of London*, vol. 58, no. 347-352, pp. 240–242, 1895.

[38] C. Chatfield and D. Prothero, "Box-jenkins seasonal forecasting: problems in a case-study," *Journal of the Royal Statistical Society: Series A (General)*, vol. 136, no. 3, pp. 295–315, 1973.

[39] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, "Detecting novel associations in large data sets," *science*, vol. 334, no. 6062, pp. 1518–1524, 2011.

[40] S. Makridakis and M. Hibon, "ARMA models and the Box–Jenkins methodology," *Journal of Forecasting*, vol. 16, no. 3, pp. 147–163, 1997.

[41] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[42] D. O. Hebb, *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[43] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[44] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[45] A. Géron, "Hands-on machine learning with scikit-learn and tensorflow: Concepts," *Tools, and Techniques to build intelligent systems*, 2017.

[46] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[47] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[48] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[49] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2114–2124.

[50] N. Wu, B. Green, X. Ben, and S. O'Banion, "Deep transformer models for time series forecasting: The influenza prevalence case," *arXiv preprint arXiv:2001.08317*, 2020.

[51] Calnex, ""what is partial timing support (pts)?"," *Whitepaper*.

[52] I. Freire *et al.*, "Testbed evaluation of distributed radio timing alignment over ethernet fronthaul networks," *IEEE Access*, pp. 1–1, 2020.

[53] I. Freire, "5G fronthaul synchronization via IEEE 1588 precision time protocol: Algorithms and use cases," Ph.D. dissertation, Federal University of Pará, Dec. 2020.

[54] *G.8275/Y.1369:Architecture and requirements for packet-based time and phase distribution, Amendment 2*, ITU-T, August 2017.

[55] Y. Zhou *et al.*, "Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts," *Journal of cleaner production*, vol. 209, pp. 134–145, 2019.

[56] D. T. Viedma, A. J. R. Rivas, F. C. Ojeda, and M. J. del Jesus Díaz, "A First Approximation to the Effects of Classical Time Series Preprocessing Methods on LSTM Accuracy," in *International Work-Conference on Artificial Neural Networks*. Springer, 2019, pp. 270–280.

[57] X. Wang and Y. Zhang, "Multi-step-ahead time series prediction method with stacking LSTM neural network," in *2020 3rd ICAIBD*. IEEE, 2020, pp. 51–55.

[58] D. Falessi, L. Narayana, J. F. Thai, and B. Turhan, "Preserving order of data when validating defect prediction models," *CoRR*, vol. abs/1809.01510, 2018. [Online]. Available: http://arxiv.org/abs/1809.01510