Risto Hietala

Packet Synchronization Test Automation

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 3.10.2016

Thesis supervisor:

Prof. Riku Jäntti

Thesis instructor:

M.Sc. (Tech.) Jonas Lundqvist

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Risto Hietala

Title: Packet Synchronization Test Automation

Date: 3.10.2016 Language: English Number of pages:8+66

Department of Communications and Networking

Professorship: Communications Engineering Code: S-72

Supervisor: Prof. Riku Jäntti

Instructor: M.Sc. (Tech.) Jonas Lundqvist

Telecommunications network operators are shifting from circuit switched backhaul technologies into packet switched networks to save costs with increasing traffic loads. Frequency synchronization was inherently provided by the circuit switched network, but has to be provided by other means in packet switched networks. One solution is Precision Time Protocol (PTP), defined in IEEE standard 1588, which can be used to create a master-slave synchronization strategy to a network. Synchronization quality is an essential factor when using any synchronization technology. Packet synchronization quality requirements in different situations are defined in ITU-T recommendation G.8261.

The objective of this thesis is to create test automation for ITU-T recommendation G.8261 Appendix VI performance test cases 12 through 17 for Precision Time Protocol. Hypothesis is that this automation will make testing more effective than if testing was done manually, allowing testing of more products in a smaller time frame.

Automated test system was planned and implemented with various measurement and impairment devices, and testing software to utilize them and to generate results.

As a result, PTP synchronization quality testing efficiency was increased by over 20 % while reducing the possibility for human errors.

Keywords: Packet synchronization, Precision Time Protocol, Maximum Time Interval Error, Software Test Automation

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Risto Hietala

Työn nimi: Pakettisynkronointitestauksen automaatio

Päivämäärä: 3.10.2016 Kieli: Englanti Sivumäärä:8+66

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoliikennetekniikka Koodi: S-72

Valvoja: Prof. Riku Jäntti

Ohjaaja: DI Jonas Lundqvist

Verkko-operaattorit vaihtavat matkapuhelinverkoissa käyttämiään tekniikoita piirikytkentäisistä pakettikytkentäisiin säästääkseen kustannuksia kasvavien liikennemäärien kanssa. Piirikytkentäisissä verkoissa taajuussynkronointi leviää verkkoteknologian myötä automaattisesti koko verkkoon, mutta pakettikytkentäisissä verkoissa se täytyy tuottaa muilla tavoin. Yksi ratkaisu ongelmaan on Precision Time Protocol (PTP), joka on määritelty IEEE standardissa 1588, ja jolla voidaan luoda verkkoon isäntä–renki -synkronointistrategia. Synkronoinnin laatu on keskeinen tekijä kaikissa synkronointiteknologioissa. Pakettisynkronoinnin laatuvaatimuksia eri tapauksissa on määritelty ITU-T suosituksessa G.8261.

Tämän diplomityön tavoitteena on luoda testausautomaatio ITU-T suosituksen G.8261 liitteen VI suorituskykytesteille 12–17 käyttäen PTP:tä. Hypoteesina on, että automaation avulla testauksesta tulee tehokkaampaa, kuin jos samat testit suoritettaisiin manuaalisesti. Näin entistä useammat tuotteet saataisiin testattua entistä lyhyemmässä ajassa.

Automatisoitu testausjärjestelmä suunniteltiin ja toteutettiin käyttäen valikoimaa erilaisia mittauslaitteita ja verkkoemulaattoreita, sekä näiden laitteiden hallintaan kehitettyä testausohjelmistoa.

Lopputuloksena PTP-synkronointitestauksen nopeus parani yli 20 prosenttia ja inhimillisten virheiden mahdollisuus väheni.

Avainsanat: Pakettisynkronointi, Precision Time Protocol, Ohjelmistotestauksen automaatio

Preface

This thesis has been carried out at Tellabs Oy (currently Coriant Oy) at Espoo, Finland.

I would like to thank my instructor Jonas Lundqvist for the idea and also for all the help during the years, especially related to this thesis, but also otherwise. I would also like to thank this thesis' supervisor, professor Riku Jäntti.

I am grateful for the help of everyone at Tellabs synchronization and testing teams, especially my superior Hannu Tuomisto and Heikki Laamanen who, along with Jonas Lundqvist, gave this opportunity for me.

Finally I'd like to thank my wife Niina, family and friends for their support and belief that I would one day finish this.

For Väinö.

Tampere, 28.8.2016

Risto Hietala

Contents

Symbols and abbreviations

Abbreviations

3GPP	3rd Generation Partnership Project
3GPP2	3rd Generation Partnership Project 2
ADEV	Allan Deviation
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
BDS	BeiDou Navigation Satellite System
CBR	Constant Bitrate
CDMA	Code Division Multiple Access
CES	Circuit Emulation Service
CESoPSN	Structure-Aware Time Division Multiplexed (TDM) Circuit Emulation Service over Packet Switched Network
CSV	Comma-separated Values
DS1	Digital Signal 1
DUT	Device Under Test
ETSI	European Telecommunications Standards Institute
FDD	Frequency-Division Duplexing
GCC	GNU Compiler Collection
GLONASS	Global Navigation Satellite System (brand)
GNSS	Global navigation satellite system (general term)
GPIB	General Purpose Interface Bus
GPS	Global Positioning System
GSM	Global System for Mobile communications
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITU	International Telecommunications Union
JPEG	Joint Photographic Experts Group
MAC	Media Access Control
MEF	Metro Ethernet Forum
MPLS	Multi-Protocol Label Switching
MRTIE	Maximum Relative Time Interval Error
MTIE	Maximum Time Interval Error
NTP	Network Time Protocol

PCM	Pulse Code Modulation
PDH	Plesiochronous Digital Hierarchy
PDV	Packet Delay Variation
ppb	parts per billion
ppm	parts per million
PNG	Portable Network Graphics
PRC	Primary Reference Clock
PSN	Packet Switched Network
PTP	Precision Time Protocol
RFC	Request For Comments
SAToP	Structure-Agnostic Time Division Multiplexing (TDM) over Packet
SCPI	Standard Commands for Programmable Instrumentation
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical Networking
STM-1	Synchronous Transport Module level-1
SVG	Scalable Vector Graphics
Tcl	Tool Command Language
TC12 ... TC16	ITU-T recommendation G.8261 Appendix VI performance test case 12 ... 16
TCP	Transmission Control Protocol
TD-SCDMA	Time-Division Synchronous Code Division Multiple Access
TDD	Time-Division Duplexing
TDEV	Time Deviation
TDM	Time-Division Multiplexing
TE	Time Error
TIE	Time Interval Error
TMM	Testing Maturity Model
UMTS	Universal Mobile Telecommunications System
VS	Visual Studio
WiMAX	Worldwide Interoperability for Microwave Access
XG-PON	10 Gbit/s Passive Optical Networking

1 Introduction

1.1 Background

Telecommunications networks have used circuit switched technologies for voice transmission, and after the success of the internet, packet switched technologies for data. Supporting both would lead to network operators having two different networks covering the same area: namely Synchronous Optical Networking (SONET) and Synchronous Digital Hierarchy (SDH) designed for circuit traffic and applying time-division multiplexing, and Internet Protocol (IP) and/or Ethernet for packet traffic having statistical multiplexing.

Operators are unifying their core networks to use only one type of technology, and packet switched solutions are favored. As voice transmission uses still heavily circuit switched protocols, they must be emulated in a packet switched world. One feature built in circuit switched protocols and missing (to same extent) in packet networks is the support for physical layer synchronization. The most common solutions for providing synchronization in packet switched networks are Synchronous Ethernet, Precision Time Protocol (PTP) and proprietary adaptive clock recovery solutions for Structure-Agnostic Time Division Multiplexing over Packet (SAToP) and Structure-Aware Time Division Multiplexed Circuit Emulation Service over Packet Switched Network (CESoPSN).

When such technologies are taken into use in communication networks, operators must be confident that synchronization quality is good enough in all possible situations the network might experience in normal operation. Behavior in special situations, as well as in normal operation, must be tested beforehand. From a vendor perspective, testing should be well specified, unchanged between different software releases, quick to start and take as little time as possible. From operator perspective, testing should be comprehensive and simulate real world scenarios.

1.2 Objectives

The objective of this master's thesis is to study test automation regarding packet synchronization testing. Selected technology for synchronization is Precision Time Protocol. Thesis has been done for Tellabs Oy and implementation is using Tellabs' test tools and processes, as well as Tellabs 8600 mobile backhaul routers as devices under test.

The scope for implemented test automation is International Telecommunications Union ITU-T recommendation G.8261 [1] Appendix VI performance test cases 12 through 17 for Precision Time Protocol and calculate Maximum Time Interval Error (MTIE) graphs for results. Testing is done for Tellabs 8600 including setup, measurement, packet delay variation emulation and result analysis of the tests.

1.3 Structure of the thesis

This thesis is divided into three general parts: literature study, design and implementation of an automated packet synchronization test system. Section 2 introduces synchronization in communication networks along with terminology and concepts. Section 3 discusses packet switched network synchronization with emphasis on Precision Time Protocol and related issues. Section 4 is an overview of software testing, how testing is seen in different software development process models, test case design and testing metrics.

Section 5 deals with packet synchronization testing requirements and planning of an automated system for this use. Options for possible system components are compared and the best ones selected. Section 6 discusses the implementation details, and solutions to problems encountered during implementation.

Section 7 has evaluation of the implemented system with previously shown metrics. Section 8 discusses conclusions from the whole thesis.

2 Communication network synchronization

Whenever there is communication between two separate parties, both of them must know at least vaguely what the other might be doing. Whether the question is about who has to listen and who to speak or when to expect for words or sentences to start, both parties have to be *synchronized* in order for the communication to work. This of course becomes more demanding with increasing number of parties taking part in the same conversation, and when the parties want to communicate at a faster pace. It is not enough that synchronization has been achieved once: it must be actively kept up, as the communicating parties' track of time can drift apart. Practically this can be attained by synchronizing again after some interval.

This section discusses the terminology used in (digital) communication synchronization, how the quality of synchronization can be measured and what kinds of different strategies there are for synchronizing communication networks. More in-depth analysis of packet network synchronization is presented in Section 3.

2.1 Terminology

International Telecommunication Union's standardization sector (ITU-T) has defined general glossary for synchronization in ITU-T recommendation G.701, *Vocabulary of digital transmission and multiplexing, and pulse code modulation (PCM) terms* [2]:

- *Significant instant* is "the instant at which a signal element commences in a discretely-timed signal". This can be for example the rising edge of a square wave signal, as highlighted in Figure 1a.
- *Synchronous (mesochronous)* is "the essential characteristic of time-scales or signals such that their corresponding significant instants occur at precisely the same average rate". Figure 1b has two synchronous signals when $\Delta t_1 = \Delta t_2$.
- *Plesiochronous* is "the essential characteristic of time-scales or signals such that their corresponding significant instants occur at nominally the same rate, any variation in rate being constrained within specified limits".
- *Non-synchronous* or *asynchronous* is "the essential characteristic of time-scales or signals such that their corresponding significant instants do not necessarily occur at the same average rate". Figure 1c has two asynchronous signals with $\Delta t_3 \neq \Delta t_4$.
- *Isochronous* is "the essential characteristic of a time-scale or a signal such that the time intervals between consecutive significant instants either have the same duration or durations that are integral multiples of the shortest duration".

Some more application specific terms are defined in ITU-T recommendation G.8260, *Definitions and terminology for synchronization in packet networks* [3]:

- *Phase synchronization* "implies that all associated nodes have access to reference timing signals whose significant events occur at the same instant (within the relevant phase accuracy requirement). In other words, the term phase synchronization refers to the process of aligning clocks with respect to phase (phase alignment)". Phase synchronization is illustrated in Figure 1d, where the two signals' significant instants occur at the same time.

- *Time synchronization* is "the distribution of a time reference to the real-time clocks of a telecommunication network. All the associated nodes have access to information about time (in other words, each period of the reference timing signal is marked and dated) and share a common time-scale and related epoch (within the relevant time accuracy requirement)". Time synchronization is illustrated in Figure 1e as having phase synchronous signals with significant instants

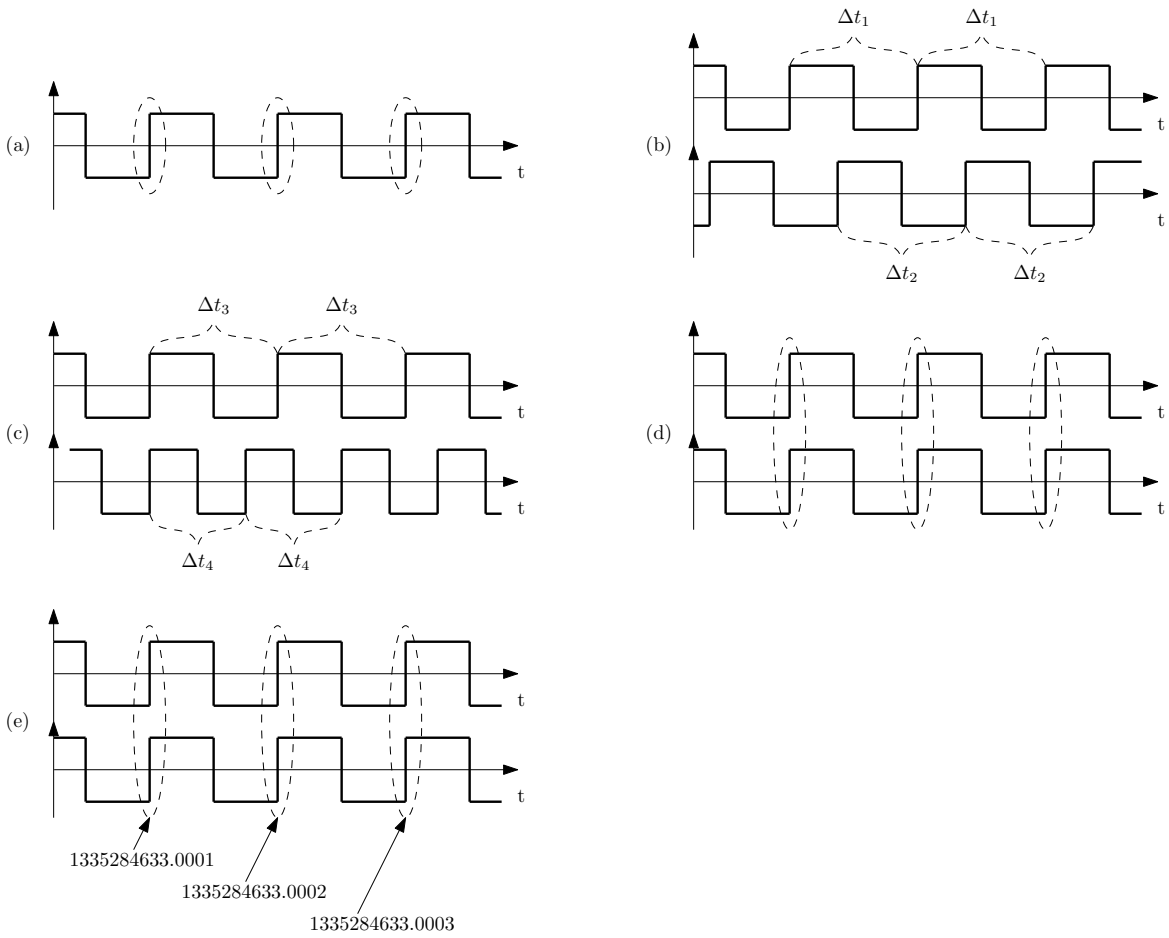

Figure 1: (a) Significant instants, (b) Synchronous signals, (c) Asynchronous signals, (d) Phase synchronous signals, (e) Time synchronous signals

Bregni defines some more specific terms for digital communications synchronization in his 2002 book *Synchronization of digital telecommunications networks* [4], from which the following are relevant to this thesis:

- *Symbol synchronization* or *clock recovery* means recovering symbol sequence timing from an analogue signal. This sequence timing information is needed for reading digital symbols at the right times from the analogue signal.

- *Word* or *frame synchronization* is used to distinguish code words from a symbol flow. Alignment words have to be searched from the bit stream first. These are special bit patterns which denote a certain place in the stream, usually the beginning of a frame.

- *Packet synchronization* is a general term for the means to compensate packet delay variation caused by a packet switched network. Different methods for timing recovery in packet networks are introduced in Section 3.

- *Network synchronization* defines the way to distribute frequency and/or phase over a communication network. Different strategies of network synchronization are discussed in Section 2.3.

2.2 Measurement definitions

Synchronization measurement guidelines are specified in ITU-T recommendations O.171 through O.175 ([5],[6], [7], [8], [9]), where each recommendation is for different transport technology: Plesiochronous Digital Hierarchy (PDH), Synchronous Digital Hierarchy (SDH), optical, Synchronous Ethernet and 10 Gbit/s Passive Optical Networking (XG-PON). This thesis will follow the ITU-T recommendation O.171, *Timing jitter and wander measuring equipment for digital systems which are based on the plesiochronous digital hierarchy (PDH)* [5].

Measured and derived quantities are used for verifying that a used clock's quality is good enough for the application it's used in. Technologies like the Global System for Mobile Communications (GSM) ([10]), Universal Mobile Telecommunications System (UMTS) ([11], [12]) and 3rd Generation Partnership Project 2 (3GPP2) CDMA2000 ([13], [14]) require different clock performance from the network devices. The requirements are discussed more thoroughly in Sections 3.3 and 3.4.

The following general terms, and others in the following sections, are from ITU-T recommendation G.810, *Definitions and Terminology for Synchronization Networks*[15]:

- *Jitter* is "short-term variation of the significant instants of a timing signal from their ideal positions in time". This "short-term" is defined as variation with frequency greater than 10 Hz. If the variation is slower, i.e. $f_j < 10\,\text{Hz}$, the phase noise is called *wander*.

- *Reference timing signal* is "a timing signal of specified performance that can be used as a timing source for a slave clock". Basically all synchronization measurements require some kind of reference signal to which compare the measured signal. The standard notes also, that "[a measurement reference timing signal's] performance must be significantly better than the clock under

test with respect to the parameter being tested, in order to prevent the test results being compromised".

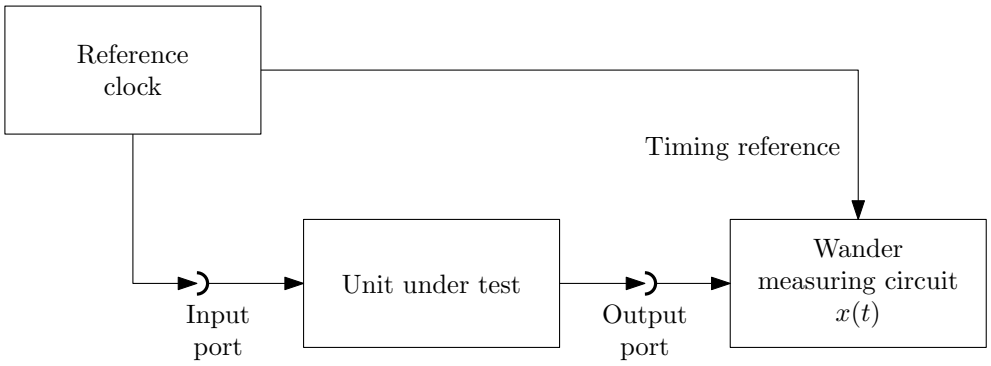

Figure 2: Synchronized wander measurement configuration [5]

A simplified block diagram from ITU-T recommendation O.171 [5] for wander measurements is shown in Figure 2. The standard specifies wander measurement guidelines in Appendix II, and lists the followind quantities for wander measurement:

- Time deviation (TDEV), according to G.810 [15]
- Maximum time interval error (MTIE), according to G.810 [15]
- Allan deviation (ADEV), according to G.810 [15]

The listed three quantities can be derived from time error (TE) and time interval error (TIE), which are thus introduced first in the following sections.

2.2.1 Time Error

Time error (TE) of a clock is the "the difference between the time of that clock and the frequency standard one" [15]. Time error is the basic function from where numerous different stability parameters such as Maximum time interval error (MTIE) and Time deviation (TDEV) are derived from. Mathematically, time error function can be expressed as the difference of two clocks' time functions:

$$\mathrm{TE}(t) = x(t) = \mathrm{T}(t) - \mathrm{T}_{\mathrm{ref}}(t) \tag{1}$$

However, this function in its continuous form is not practically attainable, and therefore sequences of time error samples can be used to denote actual measurement results:

$$x_i = x(t_0 + i\tau_0), \tag{2}$$

where:

i is the sequence number;
x_i is the i-th time error sample;
τ_0 is the sampling interval.

2.2.2 Time Interval Error

Time interval error (TIE) is "the difference between the measure of a time interval as provided by a clock and the measure of that same time interval as provided by a reference clock" [15]. Time interval error function (TIE) is therefore defined as:

$$\begin{aligned} \mathrm{TIE}(t;\tau) &= [\mathrm{T}(t+\tau) - \mathrm{T}(t)] - [\mathrm{T}_{\mathrm{ref}}(t+\tau) - \mathrm{T}_{\mathrm{ref}}(t)] \\ &= x(t+\tau) - x(t), \end{aligned} \tag{3}$$

where τ is the observation time.

The same can be expressed using discrete values as:

$$\mathrm{TIE}(i\tau_0) = x_i - x_0. \tag{4}$$

2.2.3 Maximum Time Interval Error

Maximum time interval error (MTIE) measures "the maximum peak-to-peak variation of time errors within an observation time ($\tau = n\tau_0$) for all observation times of that length within the measurement period (T)" [15]:

$$\mathrm{MTIE} = \max_{1 \le t_0 \le T-\tau} \left(\max_{t_0 \le t \le t_0+\tau} [x(t)] - \min_{t_0 \le t \le t_0+\tau} [x(t)] \right). \tag{5}$$

Using discrete values over a single measurement period, $\mathrm{MTIE}(n\tau_0)$ can be estimated with the following formula, illustrated in Figure 3:

$$\mathrm{MTIE}(n\tau_0) \approxeq \max_{1 \le k \le N-n} \left[\max_{k \le i \le k+n} x_i - \min_{k \le i \le k+n} x_i \right], n = 1, 2, ..., N-1 \tag{6}$$

where:

n	is the number of samples in an observation time;
N	is the total number of samples;
x_{ppk}	is the peak-to-peak x_i within k-th observation;
$\mathrm{MTIE}(\tau)$	is the maximum x_{ppk} for all observations of length τ within T.

2.2.4 Time Deviation

Time deviation (TDEV) is "a measure of the expected time variation of a signal as a function of integration time" [15]:

$$\mathrm{TDEV}(n\tau_0) = \sqrt{\frac{1}{6n^2} \left\langle \left[\sum_{i=1}^{n} (x_{i+2n} - 2x_{i+n} + x_i \right]^2 \right\rangle}, \tag{7}$$

where angle brackets denote an ensemble average.

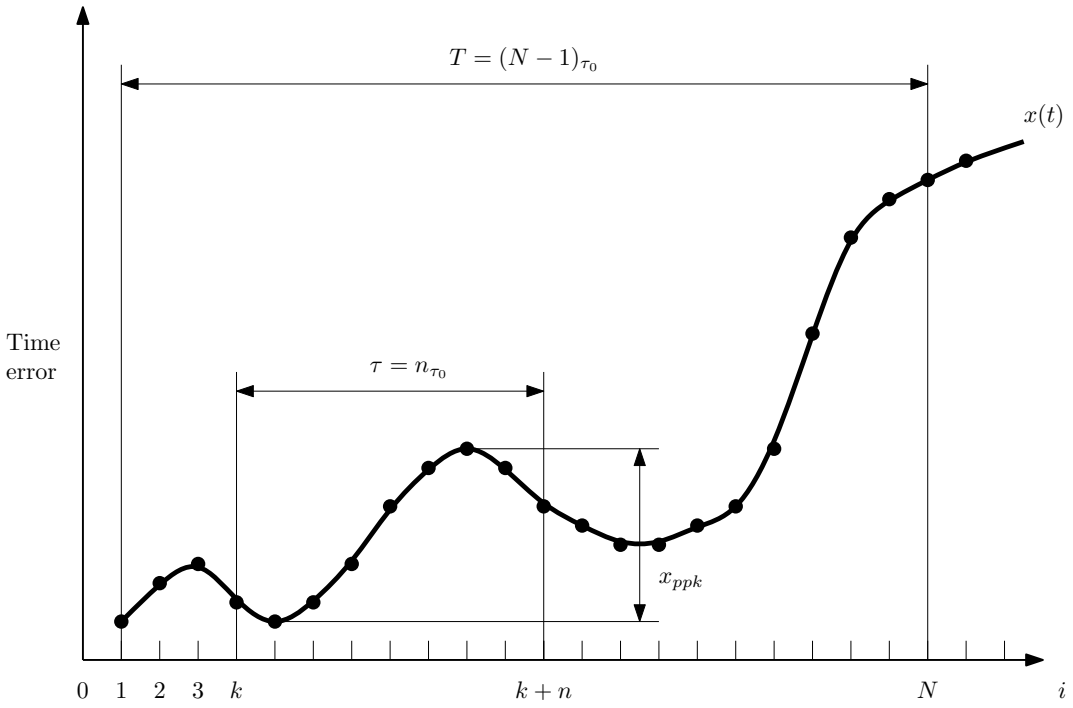

Figure 3: Maximum Time Interval Error from Time Error samples [15]

TDEV can be estimated based on a sequence of time error samples with the following formula:

$$\mathrm{TDEV}(n\tau_0) \cong \sqrt{\frac{1}{6n^2(N-3n+1)} \sum_{j=1}^{N-3n+1} \left[\sum_{i=j}^{n+j-1} (x_{i+2n} - 2x_{i+n} + x_i) \right]^2}, \quad (8)$$

where $n = 1, 2, ..., \left\lfloor \frac{N}{3} \right\rfloor$.

2.2.5 Allan Deviation

Allan deviation (ADEV) is "a non-classical statistic used to estimate stability" [16]. It is defined as [15]:

$$\mathrm{ADEV}(\tau) = \sqrt{\frac{1}{2\tau^2} \left\langle [x(t+2\tau) - 2x(t+\tau) + x(t)]^2 \right\rangle}, \quad (9)$$

where angle brackets denote an ensemble average.

ADEV can also be estimated with the following formula:

$$\mathrm{ADEV}(n\tau_0) \cong \sqrt{\frac{1}{2n^2\tau_0^2(N-2n)} \sum_{i=1}^{N-2n} (x_{i+2n} - 2x_{i+n} + x_i)^2}, n = 1, 2, ..., \left\lfloor \frac{N-1}{2} \right\rfloor. \quad (10)$$

2.3 Network synchronization

Bregni has categorized different network synchronization strategies in his 2002 book *Synchronization of digital telecommunications networks* [4]. Those are introduced in the following chapter, along with definitions from ITU-T recommendation G.701, *Vocabulary of digital transmission and multiplexing, and pulse code modulation (PCM) terms* [2] where applicable.

Full plesiochrony, or *anarchy*, is a no-synchronization strategy where all the network nodes rely on their own local clock for timing (Figure 4). This strategy was considered feasible in the 1960s because of decreasing cost of oscillators and relaxed synchronization requirements of transmission techniques. After digitalization and the introduction of mobile communication networks, the demands for synchronization have become more demanding, and networks can not be plesiochronous.

Master-slave synchronization, or *despotism*, is the idea of having one master reference clock in the network and synchronizing all other slave nodes to this clock (Figure 5). Slaves can be synchronized also indirectly via each other, making the topology tree-shaped. ITU-T defines monarchic synchronized network as: "a synchronized network in which a single clock exerts control over all the other clocks" [2].

Mutual synchronization is defined as "a synchronized network in which each clock exerts a degree of control on all others" [2]. A subtype of this is a *democratic*

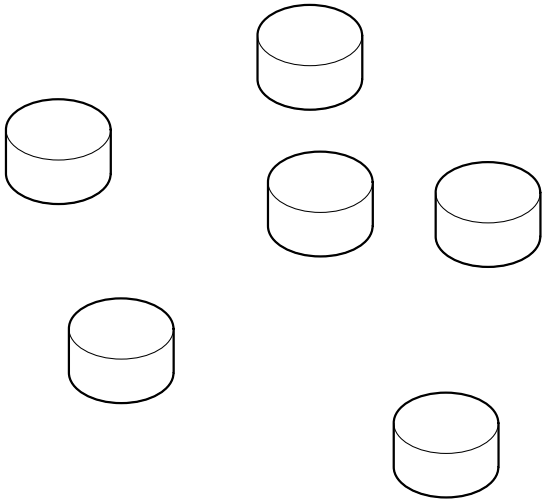

Figure 4: Full plesiochrony

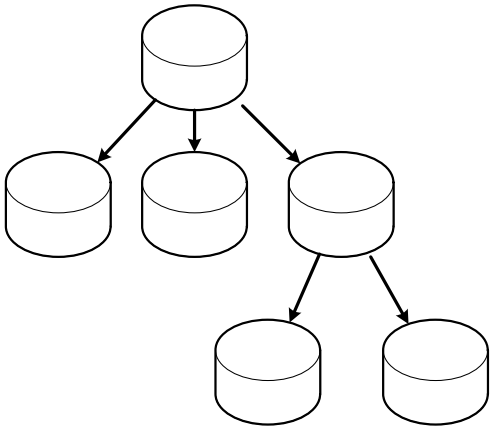

Figure 5: Master-slave synchronization

mutually synchronized network, where "all clocks are of equal status and exert equal amounts of control on the others; the network operating frequency (digit rate) being the mean of the natural (uncontrolled) frequencies of all the clocks" [2] (Figure 6). Controlling this kind of full mesh network requires complex algorithms but at the same time the strategy can have very good performance and reliability. Due to these attributes mutual synchronization is used only in special applications, e.g. in military networks.

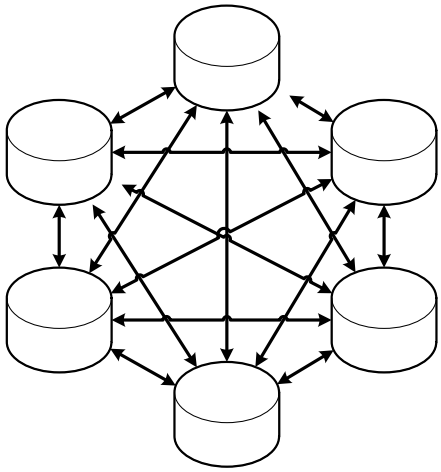

Figure 6: Mutual synchronization

Mixed mutual / master-slave synchronization, or *oligarchy*, has "a synchronized network in which a few selected clocks are mutually synchronized and exert control over all the other clocks" [2] (Figure 7). Oligarchy is a compromise between the two previous strategies: its reliability is better than pure despotism's while controlling this kind of network is simpler than one synchronized in pure democratic way.

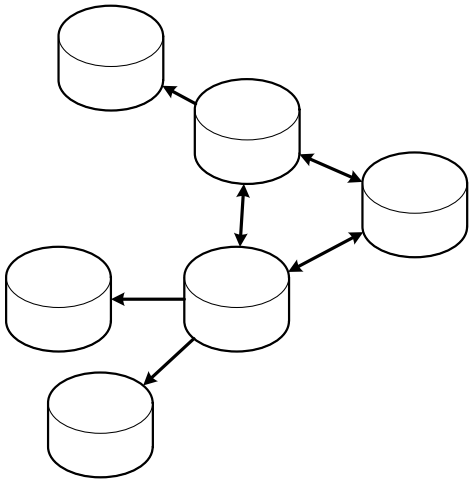

Figure 7: Mixed mutual/master-slave synchronization

Hierarchical mutual synchronization, or *hierarchical democracy*, is "a mutually synchronized network in which each clock is assigned a particular status which de-

termines the degree of control it exerts over other clocks; the network operating frequency being a weighted mean of the natural frequencies of all the clocks" [2] (Figure 8). Relative weights can be denoted as w_i ($0 \leq w_i \leq 1, \sum_i^N w_i = 1$). If $w_i = 1$ for one node, this strategy reduces to master-slave synchronization. Also, if w_i is equal for all the nodes this strategy is reduced to mutual synchronization.

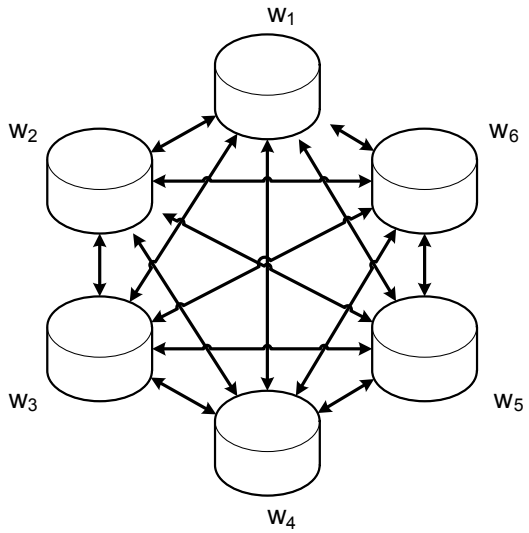

Figure 8: Hierarchical mutual synchronization

Hierarchical master-slave synchronization, or *hierarchical despotism*, is a variant of pure despotism: while nodes are normally synchronized from master to slave, the network is also hierarchical (Figure 9). In case of failure nodes can be also synchronized to or through their "sibling" nodes where as in pure despotism the synchronization path from a slave to master always goes through slave's parents and their parents. Hierarchical despotism is the most widely used method to synchronize modern telecommunication networks because it performs well while having limited cost.

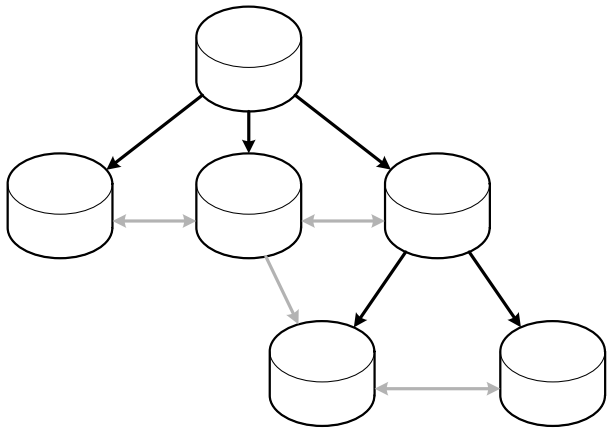

Figure 9: Hierarchical master-slave synchronization

Mixed plesiochronous / synchronous networks, or *independent despotic states*,

describe the way the world is currently synchronized (Figure 10). While a global navigation satellite system such as Global Positioning System (GPS), Global Navigation Satellite System (GLONASS), Galileo or BeiDou Navigation Satellite System (BDS) removes the technical barrier to synchronize all telecommunication networks with each other, national authorities do not want their communication networks to be reliant on US, Russian, European Union or Chinese authorities, respectively.

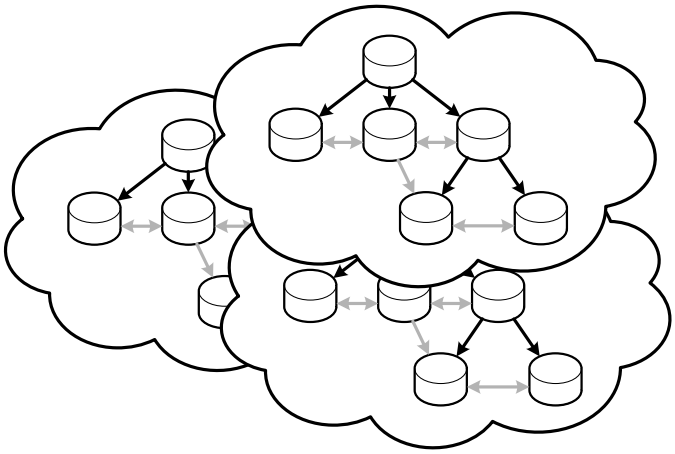

Figure 10: Mixed plesiochronous/synchronous networks

3 Packet switched network synchronization

Mobile operators have been shifting their backhaul networks from circuit-based Time-Division Multiplexing (TDM) technologies such as SDH/SONET towards packet-based technologies such as Metro Ethernet and Internet Protocol / Multi-Protocol Label Switching (IP/MPLS) in order to introduce cost savings and to harmonize the backhaul network. [17] As their installed base stations might support only TDM interfaces, there is a need to carry TDM traffic over packet-switched network. This problem is addressed by Internet Engineering Task Force (IETF) in its Request For Comments (RFC) 4553, *Structure-Agnostic Time Division Multiplexing (TDM) over Packet (SAToP)* [18] and RFC 5086, *Structure-Aware Time Division Multiplexed (TDM) Circuit Emulation Service over Packet Switched Network (CESoPSN)* [19].

These RFCs do not address the problem of synchronization with a packet switched network between TDM connections. There are two different approaches to this problem: plesiochronous and network synchronization methods, and packet-based methods. [1]

Having plesiochronous synchronization means that a Primary Reference Clock (PRC) is available wherever synchronization is needed independently from the rest of the network, for example through a global navigation satellite system (GNSS), such as Global Positioning System (GPS), Global Navigation Satellite System (GLONASS), Galileo or BeiDou Navigation Satellite System (BDS). Network synchronization methods then again refer to having master-slave synchronization using physical layer of the network, which is widely used in TDM networks. Packet Switched Networks' (PSN) network layer protocol Ethernet does not support network synchronization: Institute of Electrical and Electronics Engineers (IEEE) standard 802.3 [20] has a frequency accuracy requirement of ±100 parts per million (ppm). The protocol is however extended in ITU-T recommendation G.8262, *Timing characteristics of synchronous Ethernet equipment slave clock (EEC)* [21], where frequency accuracy requirement is much tighter ±4.6 ppm. This extended version is called Synchronous Ethernet and can be used for network synchronization.

Packet-based methods use a different approach: timing information is stored and transmitted in packets and recovered at the receiving end. Timing recovery is adaptive by nature, and is affected by impairments on packet flows. Examples of protocols providing packet-based synchronization are Network Time Protocol (NTP) defined in RFC 5905 *Network Time Protocol Version 4: Protocol and Algorithms Specification* [22], and Precision Time Protocol (PTP) defined for telecommunication use in IEEE 1588-2008 *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems* [23], and protocols using some method of recovering timing information from SAToP and CESoPSN pseudowires.

The following sections introduce the principles of PTP, what are the different impairments affecting packet-based synchronization, what is required from synchronization in different mobile technologies, and lastly how packet-based synchronization quality can be measured.

3.1 Precision Time Protocol

Precision Time Protocol (PTP) is an alias to the *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems* [23]. The protocol was first designed to be used in automation industry in a 2002 standard, and was enhanced in 2008 to better suit the needs of telecom industry. This section follows the revised 2008 version, especially from a telecom industry aspect.

Precision Time Protocol system consists of five kinds of nodes: ordinary clocks, boundary clocks, end-to-end transparent clocks, peer-to-peer transparent clocks and administrative nodes. There are two different ways of measuring the propagation delay of different PTP ports: delay request-response and peer delay mechanisms. The standard also defines ten different message types for handling these two synchronization methods and related clock management.

This section, as well as later sections, will focus on master-slave synchronization between ordinary and boundary clocks using delay request-response method. PTP synchronization messaging is shown as a protocol sequence diagram in Figure 11. Firstly, master sends a Sync message to slave and notes the send time as t_1. Slave receives Sync message and timestamps this event as t_2. t_1 is transmitted to slave either in the original Sync message or in a Follow_Up message sent after the Sync (easier to implement but requires more bandwidth). Then follows the acquisition of t_3 and t_4 with Delay_Req and Delay_Resp messages. Slave is able to calculate the delay offset with this information.

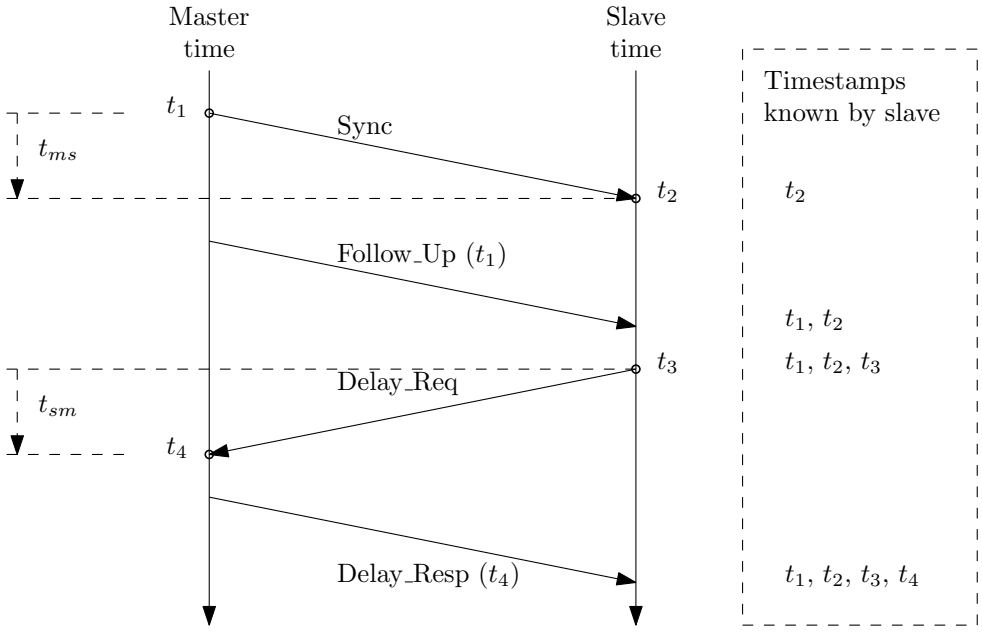

Figure 11: PTP timing message flowchart, two-message mode [23]

It is assumed that the path is symmetrical, i.e. that delay from master to slave is the same as the delay from slave to master, elaborated in equation (11).

$$t_{\text{delay\_ms}} = t_{\text{delay\_sm}} = t_{\text{delay\_1-way}} \tag{11}$$

The time difference between master and slave can be expressed as sum of the message propagation delay and the offset between master and slave clocks.

$$t_2 - t_1 = t_{\text{offset}} + t_{\text{delay\_1-way}} \tag{12}$$

Same as previous, time difference between slave and master is the sum of clock offset and message propagation delay. Here the offset has to be negative so that these two equations (12) and (13) are consistent.

$$t_4 - t_3 = -t_{\text{offset}} + t_{\text{delay\_1-way}} \tag{13}$$

From the equations 12 and 13 clock offset can be calculated at the slave end.

$$t_{\text{offset}} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{14}$$

While this way one can calculate the time offset between slave and master clock at one point of time, t_{offset} will change from one packet sequence to another. The next section will introduce how the packet delay variation and impairments affect packet-based synchronization.

3.2 Packet delay and impairments

Adaptive timing recovery from timestamp data is affected by impairments in the transport network. If delay through this packet network remains constant, timing recovery in destination node will experience only a constant phase shift in the recovered clock and there should not be frequency or phase wander.

Delay variations can be interpreted as changes in the source clock's phase or frequency, but there are lots of other reasons for delay variation in a packet switched network. These have to be taken into consideration when designing an adaptive timing recovery algorithm. Different delay factors are examined in the following sections based on a 2004 study *Analysis of Point-To-Point Packet Delay in an Operational Network* by Choi et al. [24] and ITU-T recommendation G.8261, *Timing and synchronization aspects in packet networks* [1].

Delay experienced by packets is a combination of the following factors:

- *Propagation delay* is "determined by physical characteristics of the path a packet traverses, such as the physical medium and its length". [24]
- *Transmission delay* is "a function of the link capacities along the path, as well as the packet size". [24]
- *Nodal processing delay* is "the time to examine the packet header and determine the route for the packet. It also includes checksum calculation time and the transfer time from an input to an output port." [24]
- *Queuing delay* is the time a packet has to wait after it has been processed and before it can be transmitted.

3.2.1 Equal-cost multi-path effect

Equal-cost multi-path routing is commonly used in core packet networks for load balancing and traffic engineering. Here equal-cost refers to different routing protocols' sum of link weights on a given path, which doesn't necessarily have anything to do with the physical properties along those paths. Therefore the propagation and transmission delay on might differ a lot between paths that are considered similar from packet routing perspective.

3.2.2 Minimum path transit time

Theoretically propagation and transmission delay set a minimum transit time that is linearly dependent on packet size. In practice, routers introduce also lots of other factors contributing to packet delay. However, these have also been proven to be linearly dependent on the packet size in a single hop [25]. Combining these factors, a minimum path transit time can be obtained as a function of packet size.

After equal-cost multi-path effect and minimum path transit time have been taken into account, the rest of the delay experienced by packet flows is variable by nature. Different types of variation are introduced in the following sections.

3.2.3 Random delay variation

The primary source for random delay variation is the behavior of switches and routers along the packet stream's path. Most of this is caused by queuing delays in egress interfaces as the packets have to wait for other packets to be transmitted. There is correlation with random delay variation and network load as the queues are longer with increased traffic.

3.2.4 Low frequency delay variation

Changing network load can be the reason for low frequency delay variation in packet switched networks. Most obvious example of these are the daily changes: networks tend to be more loaded during daytime than during night and cyclic with a 24 hour period. Low frequency delay variation can lead to phase wander in adaptively recovered timing and it has to be compensated as many clock specifications limit allowable phase wander over 24 hour or even longer periods.

3.2.5 Systematic delay variation

Using a transmission window or timeslot in egress interface can cause packets to experience a sawtooth-shaped delay. This results from the incoming constant bitrate (CBR) packet stream having different packet rate than what is the frequency of transmission window being open. Transmission windows are used for example in Worldwide Interoperability for Microwave Access (WiMAX) Media Access Control (MAC) [26].

Another type of systematic delay variation is *beating effect* against other CBR streams along the same route. If two streams are asynchronous and have slightly

different rates, the faster will end up having sawtooth-shaped delay variation with rising slopes and the slower with falling slopes.

3.2.6 Routing changes

Packet networks can experience routing changes due to network errors, protection switching, network reconfiguration, etc. The effect of this is a constant increase or decrease in delay along that path. If not dealt with, these changes can be interpreted as phase changes in the source clock. Routing changes with larger effect on the delay are easier to detect, while small delay change can be masked in general delay variation.

3.2.7 Congestion effects

Congestion is a temporary increase in network load which can cause severe delay variation or packet loss. Its duration is variable, lasting from seconds to a couple of minutes. However if a network has frequent and long-lasting congestion periods, it can be considered unusable for adaptive clock recovery.

3.2.8 Topology-dependent blocking mechanisms

Interaction with other flows can have varying effect on the delay experienced by the packet flow used for adaptive timing. Larger packets take longer to traverse the network and if there is a burst of larger packets sent before a flow of smaller ones, the smaller packets will catch up with larger ones at the same time being delayed in that node.

3.3 Mobile technologies' synchronization requirements

Mobile technologies have different requirements for synchronization depending on the standard. This section will gather some of generally used mobile standards' frequency and phase synchronization requirements.

Global System for Mobile Communications (GSM) base station radio interface timing requirements are defined in European Telecommunications Standards Institute (ETSI) Technical Specification (TS) 145010 *Digital cellular telecommunications system (Phase 2+); Radio subsystem synchronization* [10]. It requires the frequency accuracy of ±50 parts per billion (ppb) in a GSM base station and ±100 ppb in Pico base stations. The requirement comes from the base stations being able to do handovers of mobile phones between each other. This standard however does not specify the input timing signal requirements, but as the backhaul network has traditionally been PDH based [17], required input timing wander limits are in ITU-T recommendation G.823 [27] and G.824 [28].

UMTS Frequency Division Duplexing (FDD) base station timing requirements are defined in ETSI TS 125104 *Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (FDD)* [11] for the WCDMA

FDD radio interface. The frequency requirement for this kind of base stations is ±50 ppb and in FDD mode there is no phase requirement.

UMTS Time Division Duplexing (TDD) base stations require also phase synchronization, defined in ETSI TS 125105 *Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (TDD)* [12] as ±50 ppb frequency accuracy and 2.5 μs phase accuracy between neighboring base stations.

3rd Generation Partnership Project 2 (3GPP2) CDMA2000 base stations require frequency and time synchronization. The requirements are defined in standards 3GPP2 C.S0010-B, *Recommended Minimum Performance Standards for cdma2000 Spread Spectrum Base Stations* [13] and 3GPP2 C.S0002-C, *Physical Layer Standard for cdma2000 Spread Spectrum Systems* [14]. They state that the average frequency difference between the actual Code Division Multiple Access (CDMA) carrier frequency and the defined transmit frequency shall not be greater than ±50 ppb. It is also defined that all the 3GPP2 CDMA2000 base stations shall be time-aligned to a CDMA System Time. Time alignment error should be less than 3 μs and must be less than 10 μs.

3rd Generation Partnership Project (3GPP) Time-Division Synchronous Code Division Multiple Access (TD-SCDMA) base stations' radio interface timing requirements can be found in 3GPP TR 25.836, *Technical Specification Group Radio Access Network; NodeB Synchronization for TDD* [29]. The frequency accuracy must be ±50 ppb and phase accuracy between neighboring base stations must be 3 μs.

A common practice for base stations requiring phase or time synchronization is to equip them with GPS receivers. [1]

Synchronization requirements for the common mobile base station technologies are gathered in table 1. Frequency synchronization is usually required to be ±50 ppb and phase synchronization 2.5 μs.

Table 1: Synchronization requirements in common mobile base station technologies

Standard	Frequency (ppb)	Phase (μs)	Notes
GSM	±50 or ±100	-	
UMTS FDD	±50	-	
UMTS TDD	±50	2.5	
CDMA2000	±50	10	Common system time also
TD-SCDMA	±50	3	

3.4 Packet synchronization testing

Packet synchronization testing guidelines are defined in two standards: ITU-T recommendation G.8261, *Timing and synchronization aspects in packet networks* [1] and Metro Ethernet Forum technical specification MEF 18, *Abstract Test Suite for Circuit Emulation Services over Ethernet based on MEF 8* [30]. They describe the

test network, what must be measured, what kind of impairments to introduce, etc. This section will go through the packet synchronization testing sections of both standards.

3.4.1 ITU-T recommendation G.8261

International Telecommunication Union defines various synchronization measurement guidelines in ITU-T recommendation G.8261 Appendix VI [1], which are introduced in the following chapter. The recommendation defines 17 performance test cases in total, seven of which are defined for two-way synchronization (and are thus applicable to PTP).

The test network consists of 10 switches and is shown in Figure 12.

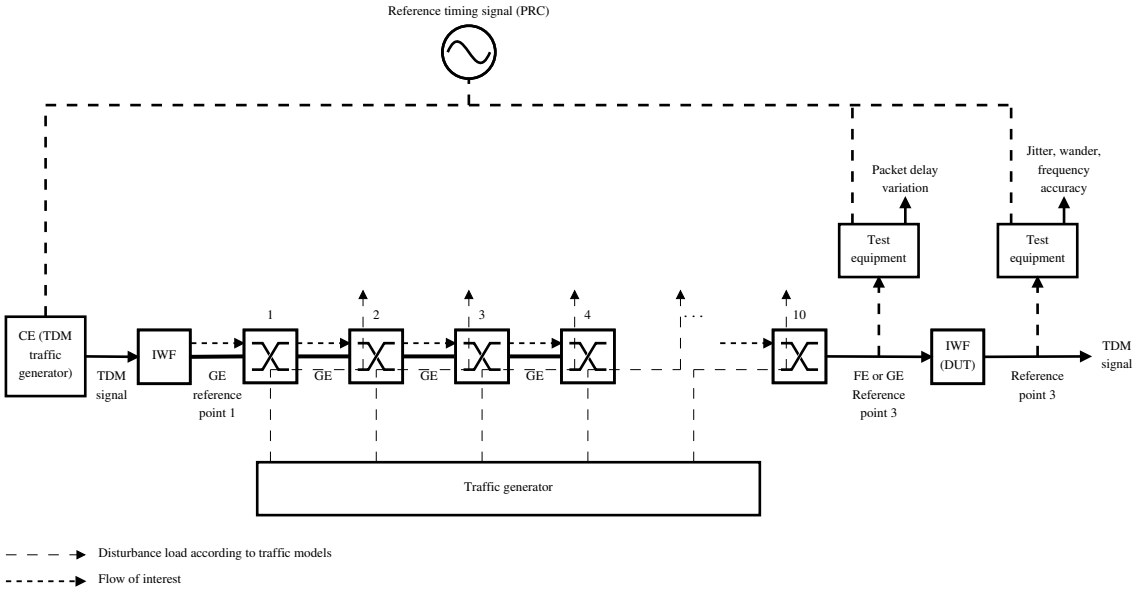

Figure 12: ITU-T recommendation G.8261 performance test topology [1]

The recommendation specifies that measurements shall start after a stabilization period of 900 seconds (15 minutes), and for each test case the following measurements shall be done:

- TIE, MTIE and MRTIE according to ITU-T recommendation G.823 [27] and G.824 [28]
- Frequency accuracy
- Packet delay variation

Figure 13 shows the delay variation profiles of test cases 12 through 16. Here, y-axis shows the relative background traffic load in test network (solid line in forward direction and dashed line in reverse direction), and x-axis is the time in hh:mm format. The test cases are following:

- Test Case 12 (TC12) models static packet load: the intermediate switches must have background load of 80 % for one hour in forward direction (server to client) and 20 % in reverse (client to server).

- Test Case 13 (TC13) models large and sudden changes in network load. Background load shall be 80 % for one hour, then drop to 20 % for one hour and repeating this three times in forward direction. Reverse direction should first have 50 % load for 1.5 hours, then repeating 10 % for one hour and 50 % for one hour until 6 hours has passed.

- Test Case 14 (TC14) models slow but large changes in network load. Background load shall be changed from 20 % to 80 % and back to 20 % in forward direction, and from 10 % to 55 % to 10 % in reverse direction within a 24-hour time period. This will result in increments and decrements of 1 % once every 12 minutes.

- Test Case 15 (TC15) models recovering from outages in the network. The test will have background load of 40 % in forward and 30 % in reverse direction. After stabilization period of 15 minutes, the network connection is removed for 10 seconds and then restored. After another 15 minutes stabilization, the network connection is removed for 100 seconds and restored. Test ends after third stabilization period.

- Test Case 16 (TC16) models short congestion periods in the network. This test case is much like number 15: instead of having network outages, network congestion is modeled by having background load increased to 100 % and then restored to 40 % / 30 %, first for 10 seconds and on second time for 100 seconds.

- Test Case 17 (TC17) models routing changes caused by failures in the network. A background load of 40 % in forward and 30 % in reverse direction is used. After stabilization period traffic is routed to skip one switch in the test network and introducing a constant delay of 10 μs in this link. Again after stabilization period, the cable is disconnected and traffic flow changed back to the original route through all the switches. The test is repeated with skipping three switches and adding a 200 μs delay on the link.

3.4.2 MEF Technical specification 18

Metro Ethernet Forum has defined synchronization related requirements in its technical specifications MEF 8, *Implementation Agreement for the Emulation of PDH Circuits over Metro Ethernet Networks* [31] and MEF 18, *Abstract Test Suite for Circuit Emulation Services over Ethernet based on MEF 8* [30]. MEF 8 has has two clauses related to synchronization that both refer to ITU-T recommendation G.823 [27] and G.824 [28] for actual requirements. MEF 18 Test Case 6 defines packet synchronization testing. There are six different cases out of which five are defined

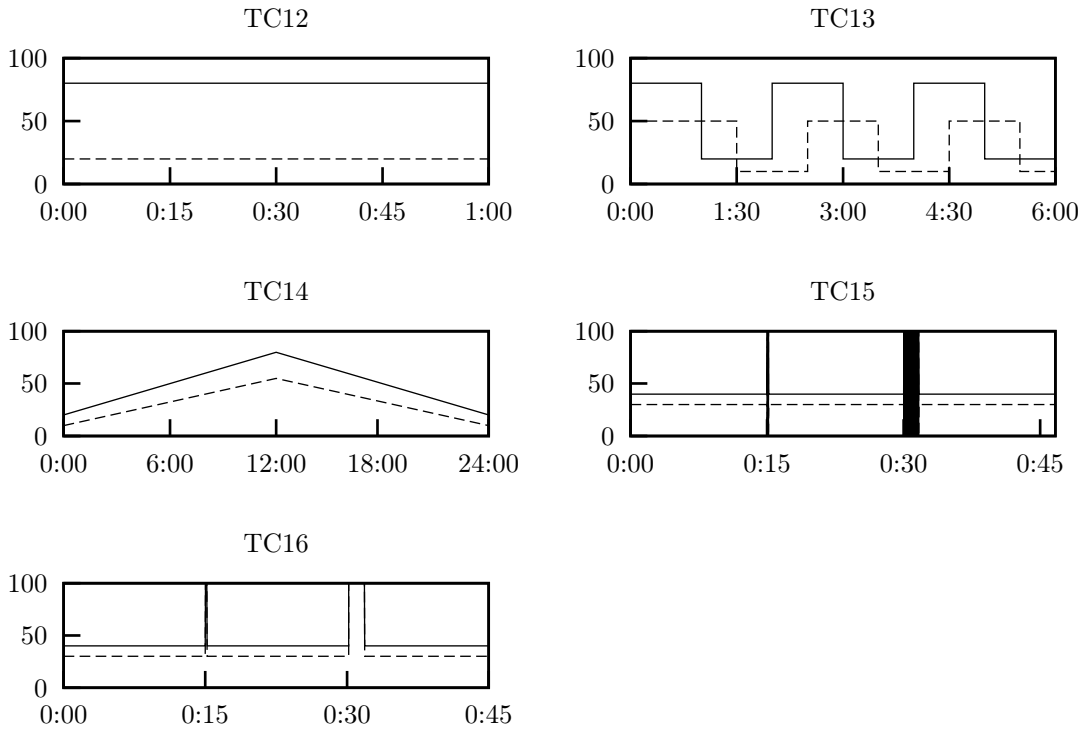

Figure 13: ITU-T recommendation G.8261 Appendix VI performance test cases 12 through 16 [1]

in ITU-T recommendation G.8261. The correlation between MEF 18 and G.8261 test cases is show in Table 2.

Table 2: MEF technical specification 18 synchronization test case correlation to ITU-T recommendation G.8261

Name [30]	MEF 18	G.8261
Static Load Test/Sudden Changes in Network Load	6a	1 & 2
Slow Variation of Network Load	6b	3
Temporary Network Outages	6c	4
Temporary Congestion	6d	5
Routing Changes	6e	6
Wander Tolerance	6f	n/a

All the test cases are to be done with ITU-T recommendation G.8261 defined traffic model 2 only. Test Cases 6c and 6d for temporary outages and congestion must be repeated 10 times in order to achieve consistent results. For each test, MRTIE (or MTIE) must be verified to be within ITU-T recommendation G.823/G.824 traffic interface wander limits over the duration of tests so that MEF technical specification 8 requirements can be satisfied.

4 Software testing

In his book, *The Art of Software Testing* [32], Myers defines software testing as follows: "testing is the process of executing a program with the intent of finding errors". IEEE 610.12, *IEEE Standard Glossary of Software Engineering Terminology* [33] defines testing as "the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component".

This section will introduce general software testing terminology, testing maturity model, strategies, test case design, how testing is included in different software development process models, what is regression testing and test automation, and finally how testing progress can be measured.

4.1 Terminology

It is essential to have a set of basic definitions for understanding the software testing process. The following list is collected from IEEE standard 610.12 [33]:

- *Error* The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
- *Fault* An incorrect step, process, or data definition.
- *Failure* An incorrect result.
- *Test* An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
- *Test Bed* An environment containing the hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.
- *Test Case* A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- *Quality* The degree to which a system, component, or process meets specified requirements.

4.2 Testing Maturity Model

Burnstein [34] introduces Testing Maturity Model (TMM) in her 2003 book, *Practical software testing*, as a reference where to compare different testing strategies and how testing is viewed in different software process models and organizations. The model has five levels of maturity, each consisting of a set of maturity goals, supporting maturity subgoals and activities, tasks and responsibilities. TMM is illustrated in Figure 14.

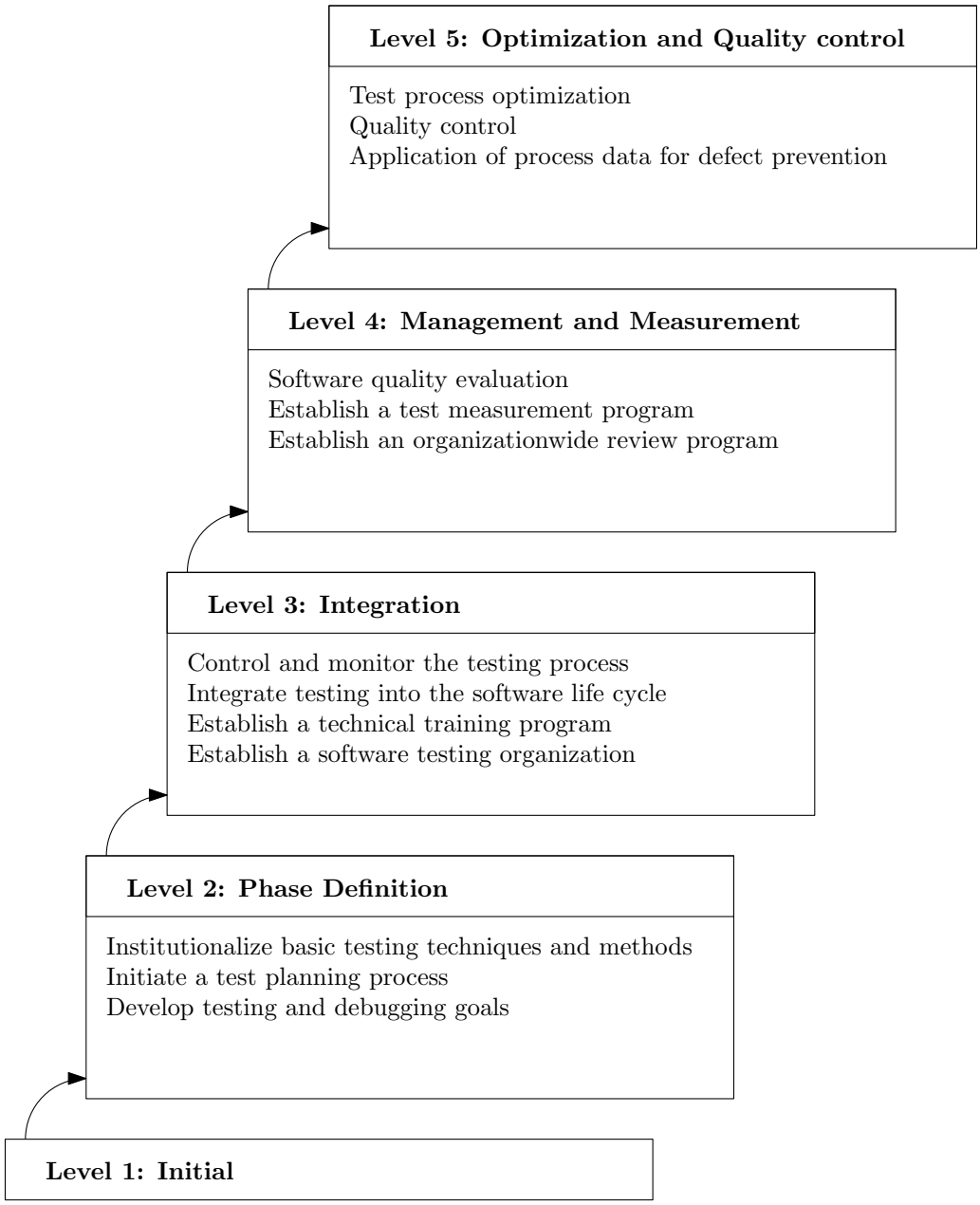

Figure 14: Testing Maturity Model [34]

Initial level has no maturity goals and objective of testing is to show that the software is minimally functional. On the Phase Definition level, testing and debugging goals are defined, testing planning process is initiated and basic testing techniques are used. Primary goal is to show that the software meets its stated specifications.

Integration level assumes software testing organization establishment, technical training program, testing integrated into the software life cycle and testing control and monitoring. Test objectives are based on specifications and user requirements, and are used in test case design.

Management and Measurement level focuses on the process aspects and its goals are establishing an organization-wide review program, a test measurement program and software quality evaluation. Reviews in all phases of software development are recognized as a testing activity complementing execution-based tests. Test cases from all projects are collected and recorded in a test database for reuse and regression testing.

Final Optimization and Quality control level aims at fault prevention, quality control and test process optimization. As all the testing mechanics and their efficiency measurement is in place in fourth level, fifth aims to optimize the mechanics. Automated tools support running and rerunning test cases as well as collecting and analyzing faults.

4.3 Strategies

There are many different approaches to test design, which can all be divided roughly into two groups: functional and structural testing. In order to build a comprehensive suite of tests, it is essential to know both strategies: what can be gained by using them and especially what restrictions do they, and thus testing in general, have.

This section follows Myers [32] including definitions from IEEE standard 610.12 [33] where applicable.

4.3.1 Functional testing

With functional, or black-box testing the tester is not concerned about the internal structure of the test subject. Instead, it is seen as a "black box" which will be given input, and a certain output is expected. For this reason, it is also called input/output driven testing. IEEE defines functional testing as "testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions" [33].

In order to test for all the possible errors, all possible inputs would have to be tested. This approach is called *exhaustive input testing*. It would mean a large amount of test cases even for the most simple practical software with inputs. For all different inputs, also the respective expected output must be defined. If the tested software has internal state, the problem becomes even more difficult: it is not enough to test only all possible input, but they have to be tested for all possible input sequences as well.

Even if there is a finite amount of correct inputs or input sequences, exhaustive testing would have to cover also infinite number of invalid inputs to verify that errors are detected correctly. As a conclusion it can be said that exhaustive input testing is impossible.

This implicates that in practice it is impossible to guarantee that a software is error-free by testing, and that economics is a fundamental issue in testing. The objective of test management is to maximize the number of errors found by a finite number of test cases.

4.3.2 Structural testing

In contrast to functional testing, structural, or white-box testing relies on knowing the internal structure of the software under test. IEEE defines structural testing as "testing that takes into account the internal mechanism of a system or component. Types include branch testing, path testing, statement testing" [33]. The goal is to fully test the software logic, executing all program lines at least once via all the different test cases and therefore testing all the software functionality. This approach is called *exhaustive path testing*.

Myers finds flaws in this thinking however. As with exhaustive input testing, the number of different paths in actual software is so huge that testing all of those would take an unpractically large amount of time. Moreover, even if all the different paths were tested the software could still be flawed. Exhaustive path testing does not guarantee in any way that the software would do what it is specified to. Secondly, the software may have missing paths, which are not detected with this type of testing. And thirdly, path testing would not detect data-sensitivity errors: when output would be correct with a certain input but incorrect with another.

4.4 Test-case design

IEEE defines a test case as "a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement" [33].

As testing for all possible errors is impossible, Myers states the key principle of test-case design as follows: "what subset of all possible test cases has the highest probability of detecting the most errors" [32]. While neither functional nor structural testing alone provide practically usable procedures for test-case design, parts of both can be used in developing test cases.

This section introduces three test-case design patterns: equivalence partitioning, boundary-value analysis and decision tables based on Myers [32], and others if noted.

4.4.1 Equivalence partitioning

Selecting the right subset of inputs for testing is crucial in increasing the probability of finding errors. A well-selected test case should have these two properties:

1. It reduces the number of test cases required for reasonable testing by more than one.

2. It covers a large set of other possible test cases and tells something about possible errors above the specific set on input values.

The first means that a test case should invoke as many inputs as possible to reduce the total number of test cases. Second, although seemingly similar, means that the input domain should be partitioned to a finite number of equivalence classes so that it can be assumed that any one input from one equivalence class would give equivalent results as any other inputs from that class. It is important to test both valid and invalid inputs.

Using equivalence partitioning, test case design should first identify the equivalence classes and then minimize the amount of test cases needed to cover them. Table 3 shows an example of input equivalence partitioning to reduce the number of relevant inputs from 12 shown (of course also inputs before -1 and after 10 would have to be tested) to 4. For example all inputs 0..3 fall into the same equivalence class B as they produce the same output 1, and therefore it is enough to test only with one of those inputs.

Table 3: Equivalence partitioning

Input	-1	0	1	2	3	4	5	6	7	8	9	10
Output	INV	1	1	1	1	2	2	2	2	3	3	INV
EQ class	A	B				C				D		A

4.4.2 Boundary-value analysis

Test cases that explore boundary conditions tend to have a higher rate of finding errors than those that do not. These boundaries can be found at directly on, above and beneath the edges of input and output equivalence classes. Boundary-value analysis can be distinguished from equivalence partitioning by these two clauses:

1. One or more elements from the edges of equivalence classes have to be selected for testing.

2. Test cases must take into account also the output equivalence classes along with input.

Boundary-value analysis is highly dependent on the test subject and its efficiency depends on the test designer's ability to identify equivalence class edges no matter how subtle they might be. On the other hand, using boundary-value analysis can drastically improve the error-finding rate of a test set.

Table 4 shows an example of boundary-value analysis. The selected boundaries are minimum and maximum input values for each equivalence class, and the first invalid input values before and after the valid value range.

Table 4: Boundary-Value Analysis

Input	-1	0	1	2	3	4	5	6	7	8	9	10
Output	INV	1	1	1	1	2	2	2	2	3	3	INV
EQ class	A	B				C				D		A
Boundary value	-1	0			3	4			7	8	9	10

4.4.3 Decision tables

A drawback in both equivalence partitioning and boundary-value analysis is that they do not take into account the different input combinations. Test cases might execute different inputs in an arbitrarily selected sequence, which probably isn't the optimal order.

Cause-effect graphing is a decision table method which analyses the software functionality so that optimal sequences of inputs can be found. First a specification has to be divided into workable pieces so that the graph doesn't become unusably large. Then the causes (inputs) and effects (outputs or alterations in the system) are identified and then linked with each other creating a cause-effect graph. A decision table can be then built based on this graph.

Jorgensen and Posey [35] show that a decision table can be created more easily by creating sets of conditions (inputs) which will cause certain actions (outputs). Table 5 shows an example of a decision table. For simplicity, all inputs can be only true (1), false (0) or indifferent (-). Outputs can be only true. Three different groups of output values can be identified, and the table can be reduced to three different cases, shown in Table 6.

Table 5: Decision Tables

Input 1	1	1	1	1	0	0	0	0
Input 2	1	1	0	0	1	1	0	0
Input 3	1	0	1	0	1	0	1	0
Output 1	1		1	1				
Output 2	1				1		1	

Table 6: Decision Tables - reduced version

Input 1	1	1	0
Input 2	1	0	-
Input 3	1	-	1
Output 1	1	1	
Output 2	1		1

4.5 Software development process models

Testing is an integral part of software development. Different software process models have different approaches to testing. The models have evolved generally towards a more flexible direction due to more demanding project schedules and requirement for better predictability on shorter time scales.

This section introduces Waterfall and V-models, and discusses testing in more recent agile software development models.

4.5.1 Waterfall model

Royce [36] introduced the waterfall model in his 1970 paper as an example of bad software process. The original model is shown in Figure 15. It was a de-facto industry standard in 1960s and allowed managers to plan software projects easily. IEEE has defined it as follows: "A model of the software development process in which the constituent activities, typically a concept phase, requirements phase, design phase, implementation phase, test phase, and installation and checkout phase, are performed in that order, possibly with overlap but with little or no iteration" [33].

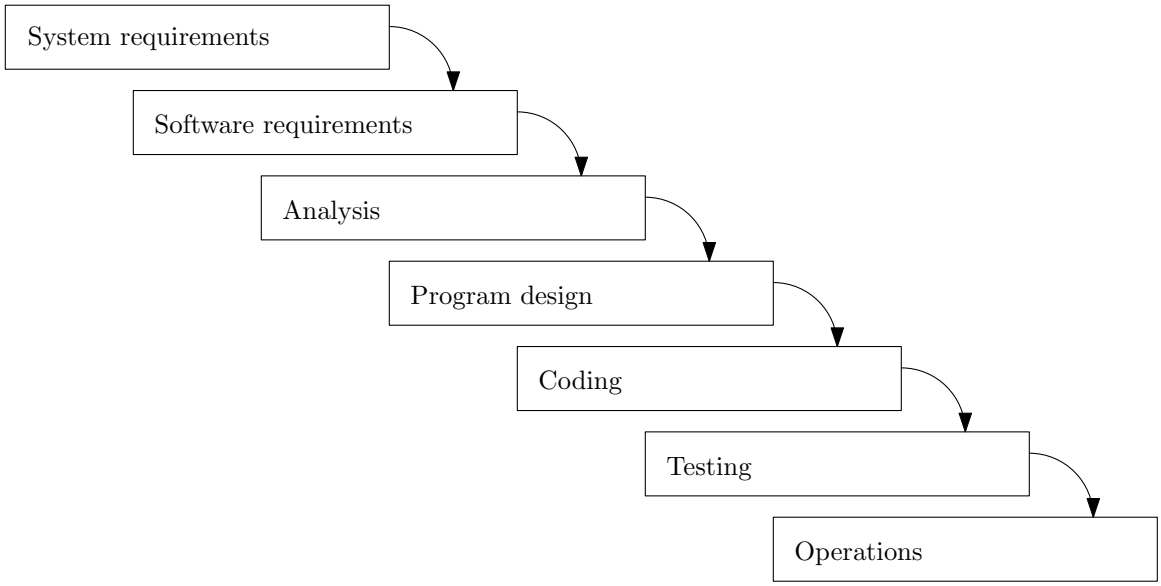

Figure 15: Waterfall model [36]

It has been criticized for the lack of feedback from one stage to another. Lots of specification, design and implementation problems are discovered after a system has been integrated, and following this model the process does not allow specifications to be changed easily with changing customer needs. [37]

Because testing is located as a separate step at the end of process model, it can easily become a "project buffer": exceeding the implementation deadline results in reduced testing time to meet the project deadline. [32]

Derivations from the waterfall model have been developed. One of these is the V-model, which has similar steps, but is designed to be more parallel. It is introduced in the following chapter.

4.5.2 V-Model

An improvement to the waterfall model is V-model which has been developed and is still used in software development processes in German public sector and military. [38] It answers to the waterfall model's problem of not having feedback from one

stage to another by introducing communication from different test levels back to implementation and design. V-model is illustrated in Figure 16. It distinguishes four different levels of testing: acceptance, system, integration and unit testing, which have become generally used terms in testing and are defined by IEEE:

- *Unit testing* is defined as "testing of individual hardware or software units or groups of related units" [33]. A unit is the smallest piece of software (for example a function) that can be tested and the tests are usually done by the person responsible for the source code under test. White box testing strategy is efficient in unit testing as the tested modules are not generally too large and as the test creator or test generating system is well aware of the internal structure of software under test. Due to the relatively simple nature of unit tests, test case generation can be automated.

- *Integration testing* is defined as "testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them" [33]. These components are verified for fulfilling functional, performance and reliability requirements. Black box testing strategy can be applied as integration testing is aimed to test that a system produces certain output with a given input.

- *System testing* is defined as "testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements" [33]. It is another black box testing level where the test cases assume to have no knowledge of system's internal functionality. System testing tries to mimic the use of a system in real-life scenarios: as a customer would want to use it.

- *Acceptance testing* is defined as "formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system" [33]. As defined, it is more of a formal procedure involving the supplier and customer and new faults in the product are not expected to be found any more at this stage.

Both of these introduced traditional models are quite cumbersome when developing software where time to market is critically important. A fairly recent solution to this problem is agile software development, an umbrella term for several different software process models. They are introduced in the following section.

4.5.3 Agile software development

Agile software development is a group of iterative software process models that are bound together by the Agile Manifesto [39].

Scrum is a process skeleton which includes a set of predefined practices and roles. There are three main aspects: each of the development stages are divided into fixed time intervals called *Sprints*. There is no predefined process for a Sprint but instead short daily *Scrum meetings* are held for the team to discuss about completed items, problems that have risen and task assignment. Third aspect is *Backlog* which stores

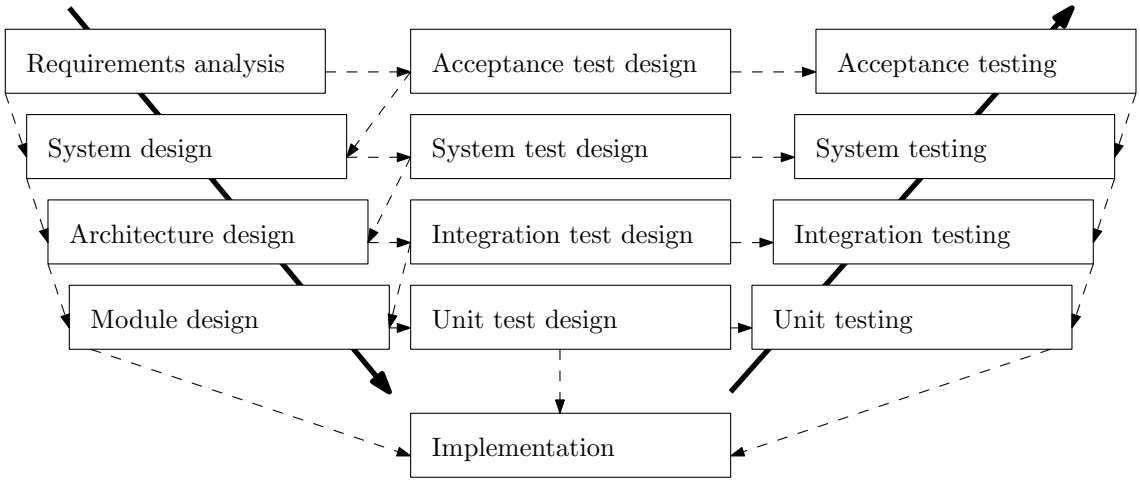

Figure 16: V-model [38]

all the work items for each Sprint. [40] Scrum does not provide any predefined model for testing, but then again testing can be done according to the Scrum process.

Extreme programming or *XP* attempts to reduce the cost of change in later stages of software development projects by having multiple short development cycles instead of one long one. The process has numerous defined practices such as: small releases, simple design, pair programming and continuous integration. Extreme programming emphasizes unit testing: programmers write their own unit tests and this is done before implementing the actual feature itself. Customers provide acceptance test cases at the beginning of each iteration. [41]

Dynamic Systems Development Method or *DSDM* is focused on projects that have tight schedules and budgets. It addresses the problems of exceeding budgets, missing deadlines and lack of user and top-management commitment by fixing the amount of functionality based on available time and resources rather than the other way around. It has nine practices which are imperative for the success of the method. One principle is *Testing is integrated throughout the life cycle*: system components should be tested by the developers and users as they are developed. Regression testing is also emphasized due to the incremental nature of development. [42]

Crystal Clear focuses more on people than processes. It has three required properties: frequent delivery, close communication and reflective improvement. Automated testing is one sub-property. Cockburn emphasizes the need for automated unit testing as it provides the developers more freedom of movement because they do not need to worry if their refactoring or other actions caused earlier implementation to stop working. [43]

In 2003, Vanhanen et al. [44] found that none of these three agile testing practices: *writing tests first*, *automated unit testing* and *customer writes acceptance tests* are actually used in the telecom industry. Automated unit testing was considered too difficult and time consuming, and having customer writing acceptance tests had communication problems. The projects included in this study used conventional testing approaches.

In a more recent study in 2006, Puleio [45] discusses problems faced when adopting agile process model in a software development team. He found out that testing practices were the most challenging part to get in place and function properly. Three points are emphasized: communication, automation and estimation. In the end his project was a success and agile testing is considered feasible and worthwhile.

4.6 Regression testing

Regression testing is defined by IEEE as: "Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements." [33]

Selecting the right tests for each new software version is economically the most crucial aspect of regression testing. All the features do not change from one version to another, and then again some features might have changes that have an impact on seemingly unrelated features. Cost per test might also change from test to another.

Rothermel and Harrold [46] have analysed different regression test selection tech-

niques in their 1996 paper. Most are based on information about the code of both previous and modified software versions while a few are based on software specifications. They have compared 13 different techniques by inclusiveness, precision, efficiency and generality. Only three of the techniques had been actually implemented: firewall, graph walk and modified entity.

4.6.1 Firewall test selection

Leung and White [47] described a regression test selection technique called firewall in 1990. It aims to retest only those software features that have either changed from the previous version or where errors could have been introduced by a change in some other feature. They observed a reduction of 65 % of test cases run when using this technique compared to retest-all method, while both found the same errors.

Firewall technique requires knowledge of a call graph, which shows the hierarchy of how software units might call each other. A firewall is built around the units that are not modified, but directly interact with the modified units (direct parents and children of modified units in the call graph). Regression testing is done for units inside this firewall.

Figure 17 shows an example of a call graph with modified units A_i and unmodified units U_j. Inside the firewall are all modified units and those unmodified units which have direct connection to any of the modified ones: (A_1, A_2, A_3, A_4, U_2, U_4, U_5, U_6, U_7, U_8).

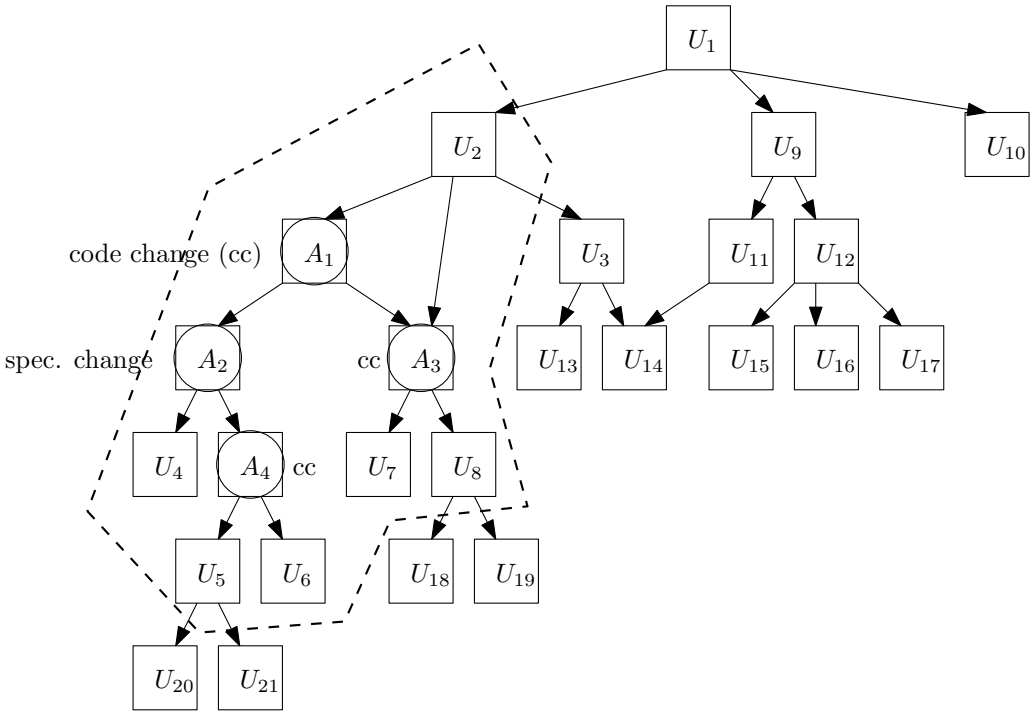

Figure 17: Firewall test selection [47]

4.6.2 Graph walk test selection

Rothermel and Harrold [48] introduced a regression test selection algorithm in 1993, which was later named as graph walk test selection. It assumes knowledge of unit call graphs for previously tested and modified software versions, and paths that existing test cases execute in the earlier call graph.

The algorithm compares nodes in current and previous call graph: if their children differ, then all tests which travel through that node in previous call graph are the ones that can detect errors in the new software. If the nodes' children are the same, this process is repeated with those children.

4.6.3 Modified entity test selection

Chen, Rosenblum and Vo [49] designed a test selection algorithm they called TestTube in 1994. Rothermel and Harrold [48] however call this technique modified entity test selection as it describes the algorithm's functionality better. This technique assumes knowledge of changed units in the software, but not how they are called.

The algorithm states that if there have been changes in functions or in non-functional entities such as variables that a test uses, that test must be rerun for the changed software. Otherwise that test will not discover any new faults in the code.

4.7 Test automation

In contrast to manual testing, automated testing uses software to control the setting up, executing and reporting tests. Motivation for test automation is that by automating tests they can be run faster (with less manual work) and thus increasing regression test coverage.

At first sight, test automation might seem very tempting but there are lots of examples of failed test automation projects due to false expectations or bad execution. This section discusses the drawbacks and benefits of test automation based on Kaner [50], Bach [51] and Oliveira et al. [52]

Kaner [50] in 1997 and Bach [51] in 1999 have listed drawbacks and false assumptions about test automation. Automating, documenting and running test cases once may take up to 10 times more time than running them manually. From software process perspective this is always unwanted as almost all the models emphasize the effect of finding defects as early as possible. Long duration of automation creates both direct (increased time spent on testing) and indirect (delayed first faults) costs. To combat this, only tests that are going to be run multiple times should be automated at all, and first run(s) of those tests should still be done manually.

Simple tests are easy to automate. This leads to low power and easy to run automated tests which do not find faults. Kaner estimates that automated tests are responsible only for 6–30 % of all the faults found. Automation can also lead to hard to detect false positives if the automated tests themselves have faults. These effects can be negated with correct test automation process with reviews and verifications of the automated tests.

The amount of required human interaction with automated test cases can be easily underestimated. Interpreting results and changing tests when specifications change requires always a person. Also running tests and reporting results can rarely be fully automated. This has to be taken into account in test automation resourcing.

With these pitfalls in mind it is possible to build sensible and effective test automation. Bach lists principles to help in automating test cases:

- There is a distinction between test automation and the test case that it automates. The latter should be in a form that is easy to review.
- Test automation should be considered as a baseline test suite which is used along manual testing rather than as a total replacement of manual testing.
- The product must be mature enough so that implementation changes that require changes to test cases do not increase test automation costs too high.

With a good test automation process and decisions about what to automate, test automation drawbacks can be minimized. Oliveira et al. [52] list benefits of test automation that might not be intuitive in their 2006 paper.

Testing especially towards the end of software projects is under a tight time pressure, and that situation human errors are likely. Automated test cases function the same way from one run to another and can reduce possible manual mistakes.

Automation increases testing efficiency as multiple test cases can be run simultaneously, during manual test run, a tester can only concentrate on one test at a time. Automated test cases can also run without interruptions and at times when there are no testers present.

Test automation can include some level of result analysis so that tester does not have to go through every step and verify each test case result separately. For reporting, automation can generate a higher level abstraction of test a run's status.

In order to verify the efficiency of testing in general, some metrics for testing must be used. The subject is introduced in the following section.

4.8 Testing metrics

Kan et al. [53] show different proven software testing metrics in their 2001 article. This section introduces three testing metrics from that article: test progress S curve, test defect arrivals over time and test defect backlog.

Test progress S curve is recommended for test progress tracking. It includes graphs of the amount of run and completed test cases relative to time along with planned progress. With this kind of graph, management can notice early if test actions start to fall behind, and take action. Figure 18 shows an example curve with run and successfully completed test cases as bars, and planned progress as a line. This graph can be further modified by adding weights on the test cases, so that the more important a test is, the more it will affect the bars. Weighing can be done based on experience, or automatically based on coverage for example. The

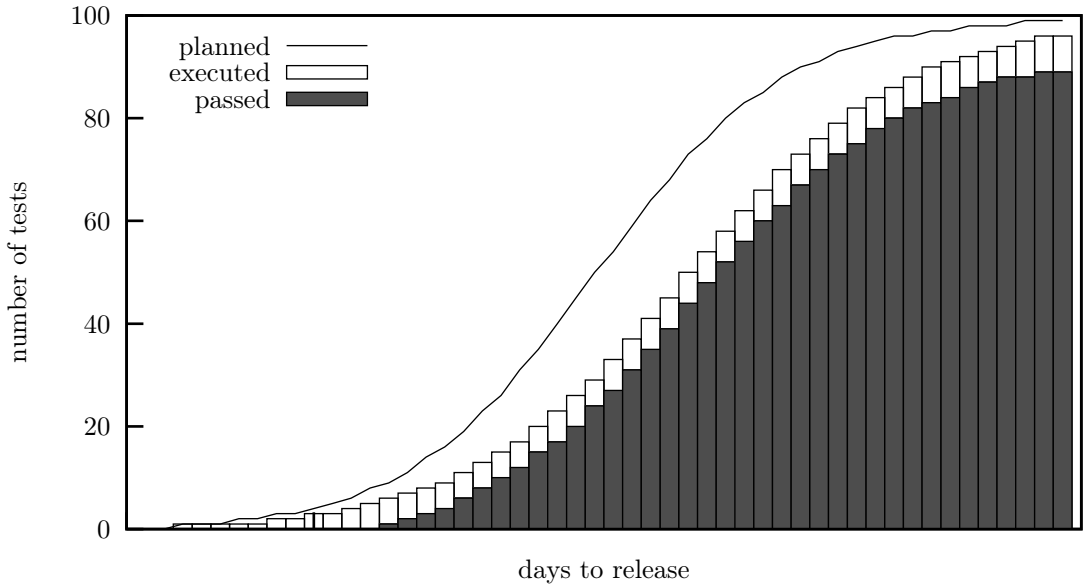

Figure 18: Test progress S-curve, following [53]

time scale is good to tie to the tested software's release date, so that S curves from different projects can be easily compared with each other.

Defect arrivals over time give some insight on when defects can be expected to be found during testing. For graphing this behavior, there should again be time relative to release date on x-scale, and the number of found defects per time unit on y-scale along with the same metric from an earlier project. Figure 19 is an example of this metric.

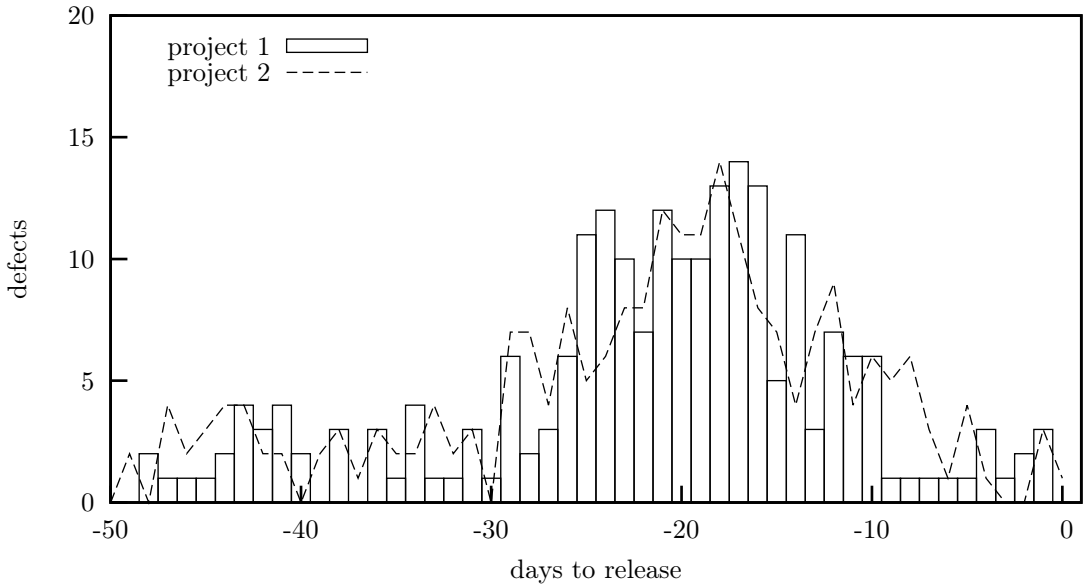

Figure 19: Defect arrivals over time, following [53]

Test defect backlog over time shows the amount of open (non-fixed) defects compared to time. Having this metric graphed can explain something about the two previous graphs: a large number of open defects can slow down the testing process (S curve) and also delay the finding of new defects (defect arrivals over time). The graph should be similar to defect arrivals over time, but instead of cumulated defects it should show the number of open defects. Again, comparison with previous projects will make this metric more valuable. Figure 20 is an example of this metric.

These three metrics can be easily used to measure testing process per projects. Moreover, the metrics can be used to confirm or deny the success of enhancing tests or testing processes. These metrics also support the intuition that having testing done earlier is better for the software project.

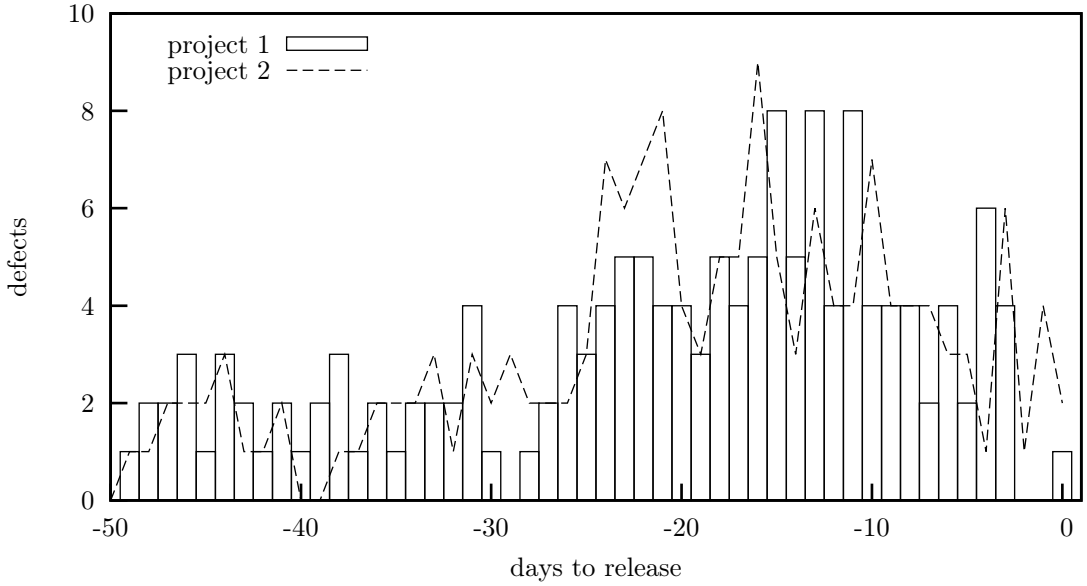

Figure 20: Test defect backlog, following [53]

5 Requirements and planning

Testing packet switched network synchronization is cumbersome if done manually. There are lots of changing variables and configurable devices in tests, and the test duration can vary from tens of minutes to days. Lots of post-processing is also required in order to represent the measured results in a suitable way, and finally all the results have to be archived for possible further use. If regression testing of packet synchronization can be automated, performance results can be obtained much more often and thus reacting to impacts on performance is faster.

Automated packet synchronization testing can also enable new test cases being done as the workload from manual testing can be shifted to test case design. For example running tests in special environments (e.g. interoperability with third-party devices, customer network packet delay variation, etc.) would be easier if the cost of setting up synchronization testing is reduced by automation.

The objective for this automated testing system is *to be able to test ITU-T recommendation G.8261 Appendix VI performance test cases 12 through 17 for Precision Time Protocol and calculate Maximum Time Interval Error (MTIE) graphs for results*. These test cases load both forward and reverse directions with two different background traffic profiles.

The hypothesis is that automating this testing allows testing wider range of network elements in a tight software release schedule than what could be achieved with manual testing. This section discusses the design of all the testing system parts, different MTIE algorithms and synchronization quality requirements.

5.1 Designing an automated test system

All the general packet synchronization test cases can be abstracted to these phases:

1. Configure master and slave devices, impairment generation and measurement device.
2. Wait for the slave device timing to lock.
3. Start impairments (background traffic or packet delay variation (PDV) profile on network emulator).
4. Wait for stabilization period.
5. Start TIE measurement.
6. Wait for measurement period.
7. Stop measurement.
8. Calculate MTIE based on the results.
9. Plot TIE and MTIE.

10. Compare MTIE to appropriate mask(s).

11. Generate a report and store the results.

This process is investigated in this section from the end to beginning: result reporting, graph plotting, packet delay variation (PDV) generation and finally the measurement itself. MTIE calculation is a larger subject, and is investigated more thoroughly in Section 5.2.

5.1.1 Reporting

As well as for all the other testing, also packet synchronization test results should be clearly presented and easy to read. For G.8261 testing this means that for each test case, there should be a TIE and an MTIE figure of the results with appropriate masks. As higher level abstraction there should also be a list of the test cases with pass/fail indication so that the reader does not have to interpret the graphs herself.

Being a regression test that is run for various different software versions, test results should be archived. Accessing earlier results should be straightforward and they should not take up too much disk space. For possible further investigation of test results, the measurement data should be saved along with the resulting report. Requirements with possible solutions are listed in Table 7.

Table 7: Requirements for reporting and comparison of solutions

Requirement	TestNET
Text-presentation of results	yes
Support for graphs	yes
MTIE calculation	no
Result archiving	no
Graph generation	no

In practice there is only one option for reporting. An in-house built test automation environment, *TestNET* is used. This environment contains interfaces to communicate with network elements and measurement equipment, framework for test logic and test result reporting. The final result is a HyperText Markup Language (HTML) file, or multiple files depending on test configuration. TestNET is controlled with *Tool Command Language (Tcl)* [54], which is an interpreted script language commonly used in testing. An object-oriented extension called *[incr Tcl]* [55] is also used.

Comparing with requirements, TestNET generates text-presentation of results automatically and has support for graphs as images via user-defineable HTML. MTIE calculation is not supported. Results are given as flat files and it is up to the test engineer to archive them correctly. Graph generation from text-formatted data is not supported.

MTIE calculation itself has various different options for algorithms and those are investigated further in Section 5.2. Graph generation is investigated in the next section.

5.1.2 Graph plotting

Graph generation from text-formatted data is essential for MTIE and TIE presentation. As TestNET does not support this, another solution must be found. Main required functionality is that graph plotting from text-based data can be automated and that output graphs are in a format that is understood by modern web browsers. For convenience, a graphing utility could resize graphs automatically based on the given data.

Table 8: Requirements for graph plotting

Requirement	Office Spreadsheet	gnuplot	GD
Automated plotting	yes	yes	no
Reasonable output format	no	yes	no
Automatic resize	yes	yes	no

Table 8 compares three options for graph plotting: Office spreadsheet programs, gnuplot [56] and GD Graphics library [57]. Office spreadsheet programs such as *Microsoft Excel* [58] or *Apache OpenOffice Calc* [59] are generally used in light-weight graph calculation and presentation. They are very efficient and easy to use when a graph has to be created for the first time from arbitrary source data. In this case the source data is always in the same format and multiple graphs have to be generated for each test run, so this option is not optimal. Spreadsheet programs have their own scripting capabilities with macros, but they are diverse and dependent on the used program.

gnuplot [56] is a command-line graphing utility available for all major operating systems. It was originally created as plotting software for the academic world, and has grown since to support various different uses and output formats. gnuplot is controlled with specific configuration files and plots can be drawn from arbitrary plot data or mathematical functions.

GD Graphics Library [57] is a graphics software library originated from American National Standards Institute (ANSI) C but has nowadays interfaces for multiple other programming languages, including Tcl. It includes various commands for creating and manipulating raster images. Increased flexibility has a downside for graph plotting use however: it is much more cumbersome to use than gnuplot or spreadsheet programs as all the plotting functions, axes, etc. would have to be created from scratch.

Because of its relative easy to setup, portability, flexibility and performance in repeated plotting with very little changes to setup, gnuplot is clearly the best option of these three.

5.1.3 Generating packet delay variation

ITU-T recommendation G.8261 test cases require varying impairment traffic to be present in each of the tests. Details about the type of background traffic are defined in G.8261 Appendix VI [1]. Two different traffic models consist of three different packet flows: two constant bit rate (CBR) flows with different packet sizes and one bursty flow with maximum packet size.

As defined in the standard, generating this background traffic requires a 10-switch network and a packet generator that can supply those switches with the correct type of packets. *Spirent AX/4000* [60] test system with ethernet modules is one candidate of this type.

As the tests are defined in an appendix and thus are not normative, background traffic can be emulated for easier and repeatable testing. Emulating the impairment caused by this background traffic can be done by delaying individual packets of the flow-of-interest in the same way as they would be delayed in a switch network with specified background traffic. *Calnex Paragon* [61] and *Ixia XGEM* [62] are network emulators with this capability.

Requirements for packet delay variation (PDV) generation are quite simple. The process must be possible to be automated, the PDV profile should be exactly the same from one test run to another and the profiles should be readily available. Table 9 shows the three options compared by these requirements.

Table 9: Requirements for PDV generation

Requirement	Spirent AX/4000 with a switch network	Calnex Paragon	Anue XGEM
Automated usage	yes	yes	yes
Repeatability	no	yes	yes
PDV profile availability	no	yes	yes

AX/4000 with switch network is remote-controllable via Tcl but it lacks repeatability as the packets are randomized differently between runs. Also Spirent does not provide PDV profiles as it is: they would have to be scripted independently. Calnex Paragon and Ixia XGEM are similar devices in features and both support these requirements. Calnex Paragon was chosen due to schedule and other non-technical reasons over Ixia XGEM.

5.1.4 Measurement

The last important part in synchronization testing setup is the measurement device. It has to be able to compare the input synchronization signal to a reference clock signal and record the Time Interval Error (TIE). After this TIE data from a test is recorded, it can be further analyzed for MTIE, TDEV and other values.

Important features in measurement devices are automated usage and supported interfaces. Even if the test system is not concerned about the differences in synchronization quality between different interfaces, the clock signal must be transferred to

the measurement device somehow. Most common options in the devices are clock signals (e.g. 2.048MHz or 10MHz) and data signals (e.g. E1, Digital Signal 1 (DS1), Synchronous Transport Module level-1 (STM-1) or Synchronous Ethernet). Scalability is measured by the cost of adding another measurement to be run in parallel. Table 10 shows three possible measurement devices compared by these requirements.

Table 10: Requirements for TIE measurement

Requirement	JDSU ANT-20	Pendulum CNT-90	Calnex Paragon
Automated usage	yes	yes	yes
SDH/SONET supported	yes	no	no
2.048/1.544MHz square wave supported	yes	yes	no
Synchronous Ethernet supported	no	no	yes
Scalability	2nd	1st	3rd

JDSU ANT-20 [63] is a measurement device for SDH and PDH networks supporting both ANSI and ETSI defined carrier systems. Automating measurement with ANT-20 works via a telnet (Transmission Control Protocol / Internet Protocol, or TCP/IP) connection with a text-based communication protocol using Standard Commands for Programmable Instruments (SCPI).

Pendulum CNT-90 [64] is a frequency counter/analyzer used for measuring frequency, time interval or phase. Like ANT-20, Pendulum CNT-90 also uses SCPI protocol. It can be used remotely via the General Purpose Interface Bus (GPIB) for automated measurements. Personal computers (PC) do not generally have a GPIB interface, so an additional device is required for connecting to a GPIB device from a PC. One possibility is to use a GPIB to TCP/IP gateway, such as *Agilent E5810A* [65]. It will allow the GPIB devices to be used with a telnet (TCP/IP) connection, which is very flexible to program.

Calnex Paragon [61] is a newer device for testing packet-based synchronization. It supports also Synchronous Ethernet and its control software is remote controllable via Tcl scripting. Due to its support of only Synchronous Ethernet interfaces, it cannot be used for as wide range of testing as the other options.

In initial testing, ANT-20's remote control software was found to be unreliable, which made CNT-90 a better choice for measurement device. It is also the best to scale from these options (i.e. the most inexpensive device).

After TIE measurement, processing is required to present the results as MTIE graphs. This problem is addressed in the following section.

5.2 Maximum Time Interval Error algorithms

Theory for MTIE calculation was introduced in Section 2.2. Three algorithms for calculating MTIE are presented here: naive, Extreme Fix by Dobrogowski and Kasznia [66] and Binary Decomposition by Bregni [4]. They each differ in computation

speed and memory used. Implementations and efficiency comparison is discussed in Section 6.3.

5.2.1 Naive algorithm

Naive algorithm searches for the minimum and maximum sample value for all window positions. This will become very computationally intensive if there is lots of data (i.e. long measurement and/or high sampling frequency), or if MTIE is calculated for lots of window sizes.

Algorithm is described more thoroughly as flowchart in Figure 21, where

N	is total number of samples
$n(i)$	is TIE value at position i
k	is observation window size
x	is window starting position
i	is current sample position
p-p(k, x)	is a peak-to-peak value with k at x
MTIE(k)	is the maximum peak-to-peak value with k

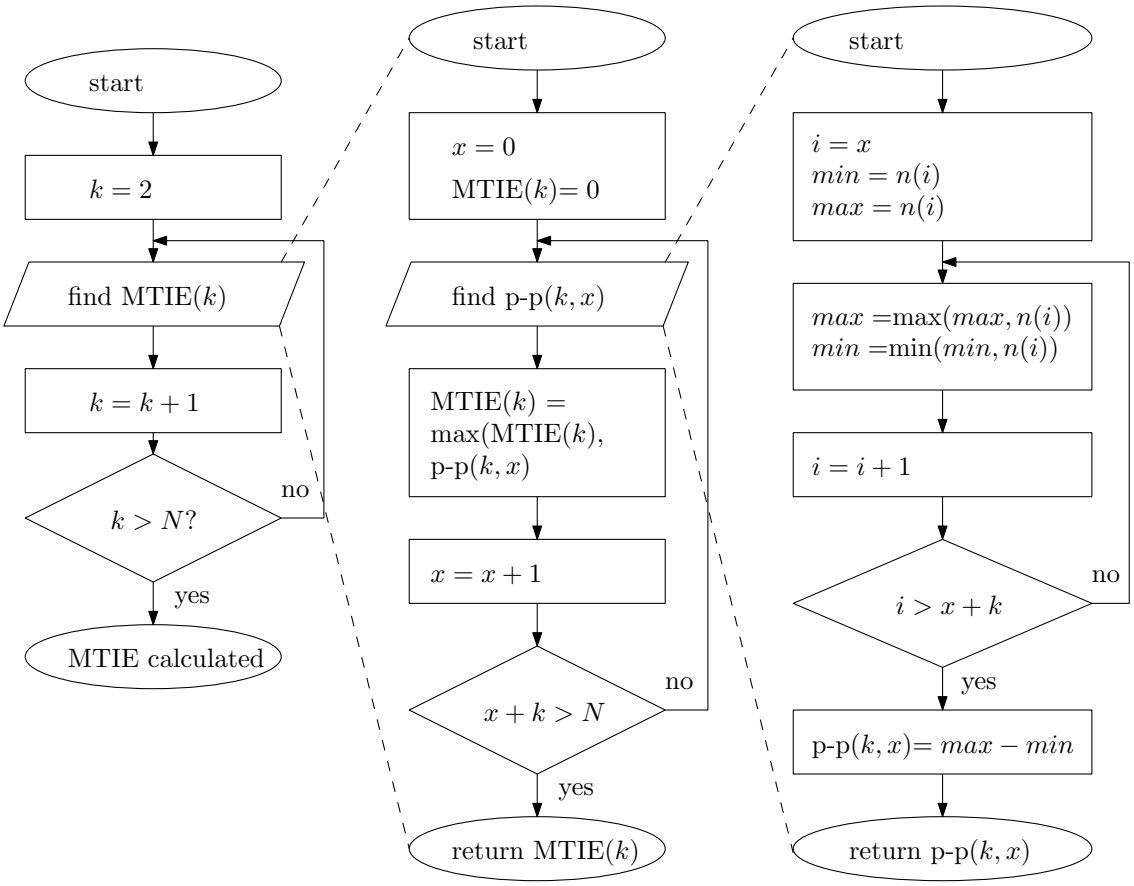

Figure 21: Naive algorithm flowchart

5.2.2 Extreme fix

Dobrogowski and Kasznia have presented more efficient versions for calculating MTIE [66]. Extreme Fix algorithm is based on the idea of moving the window all the way to the point when MTIE value can actually change.

For example, in Figure 22 at window position x the peaks are at p_1 (minimum) and p_2 (maximum). The samples between x and p_1 become uninteresting as they cannot effect the overall peak-to-peak value in any way. Window positions $x + 1$, $x+2$, ..., p_1-1 can have greater peak-to-peak value than position x, but they cannot be greater than at position p_1. Therefore it is enough to search for peak values at p_1 after x. In case either peak value is at the beginning of the window, for example at position p_1, the window is moved by one sample, here to $p_1 + 1$.

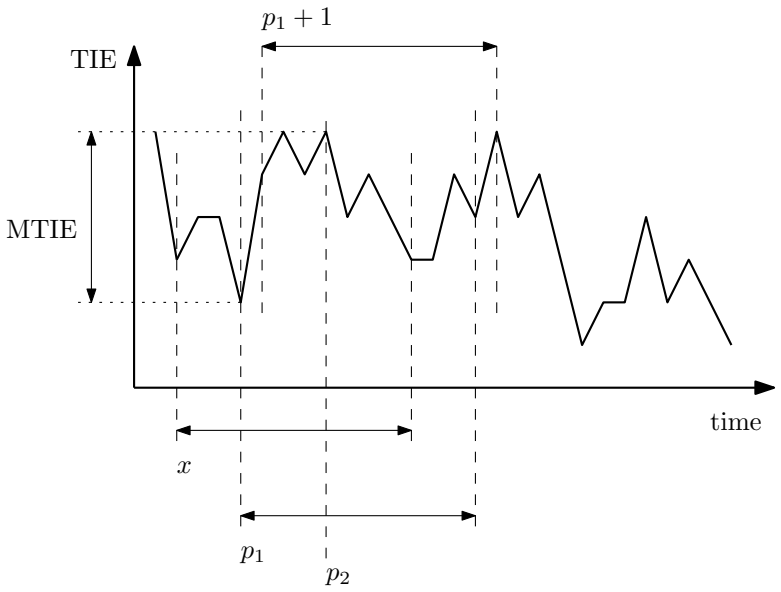

Figure 22: Extreme Fix algorithm

Comparing with naive algorithm, Extreme Fix is different in the Figure 23 flowchart's second and third column. The additions require only a small amount of more memory for two extra variables and a small amount of extra operations when deciding the next x. Saved number of calculations depends on input data. In worst case (the measured samples are increasing or decreasing through the whole data set), Extreme Fix's performance is reduced to that of naive algorithm.

5.2.3 Binary decomposition

The Binary Decomposition algorithm described by Bregni [4] has a different approach. It reduces the amount of calculations by using the minimum and maximum TIE values from previous rounds of iteration. A drawback with using this algorithm is that it can calculate MTIE for only $k = 2^j$ sample windows. The algorithm consumes also more memory than Extreme Fix or naive.

An example of this algorithm with 16 TIE samples is illustrated in Figure 24. On first round ($j = 1$, i.e. $k = 2$) this algorithm doesn't differ from naive as all

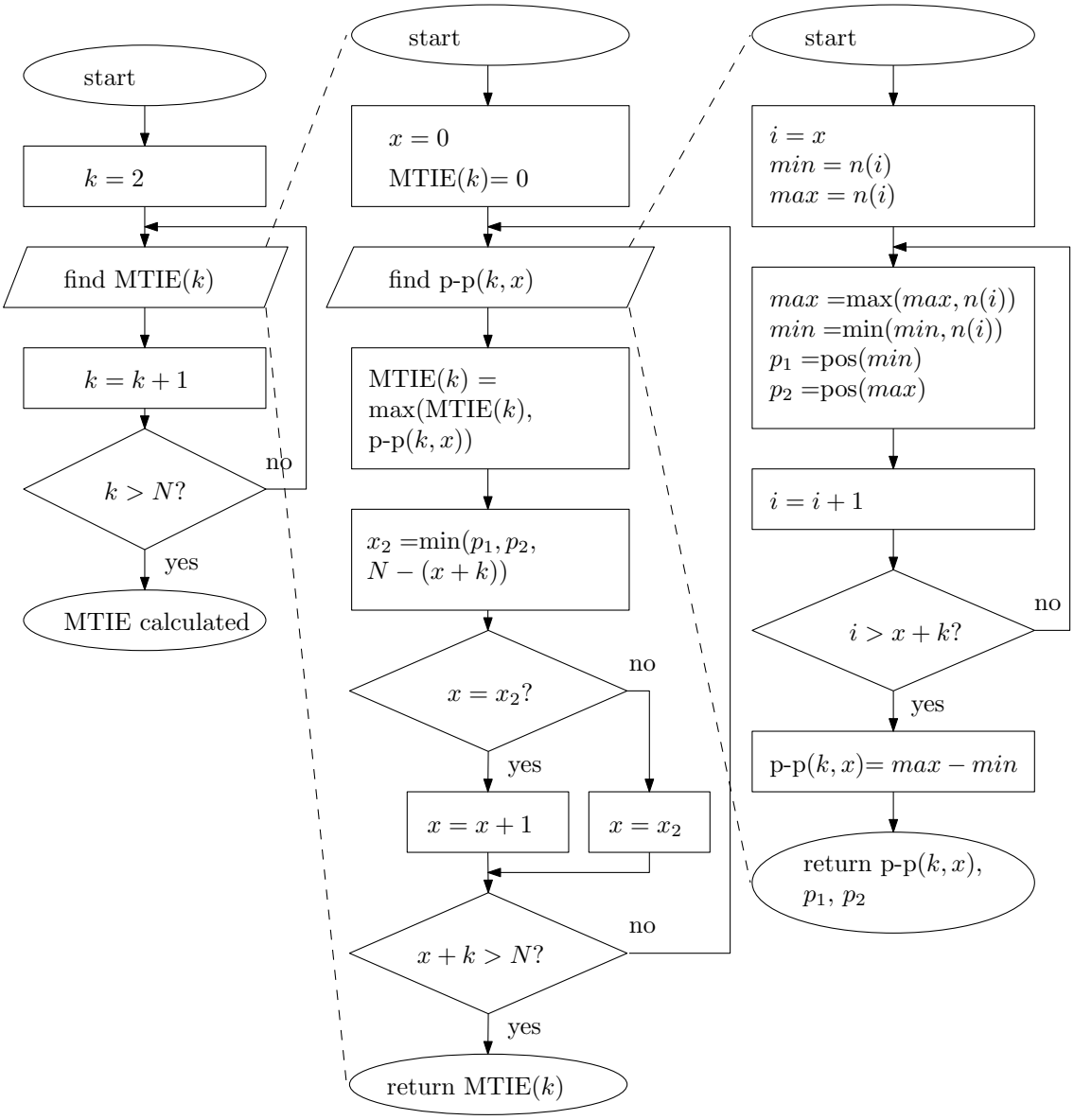

Figure 23: Extreme Fix algorithm flowchart

the samples have to be gone through. Peak values are saved in a 2 by N-1 matrix. From second round onwards ($j > 1$), the previous round's peak values are searched for minimum and maximum for each window position.

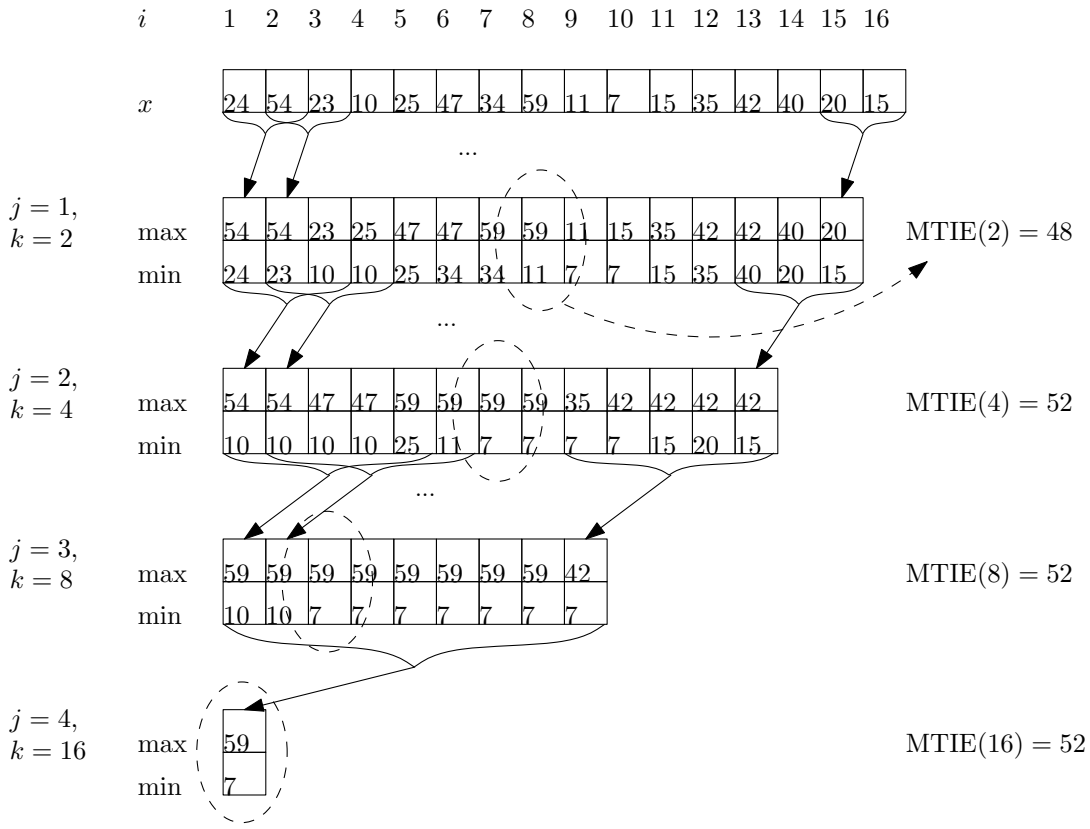

Figure 24: Binary Decomposition algorithm

5.2.4 Mask matching

Most standards define synchronization quality requirements as MTIE masks (for example ITU-T recommendation G.8261 [1]). Deciding whether an implementation meets the standards' requirements, a test system must be able to compare MTIE results with appropriate masks.

A test is passed if for all calculation points the MTIE value is smaller than allowed mask value. In practice, these masks are given as a coordinate tables or as a function depending on the observation window size. Measurement results' resolution then again depends on the MTIE calculation parameters.

Comparing measured MTIE points that have mask defined at the same window size is trivial, as is comparing measured MTIE with two mask points that have both either smaller or greater values. If a calculated MTIE is between the closest mask's values as in Figure 25: $(x_{m1} < x < x_{m2}) \ \wedge \ (y_{m1} < y < y_{m2})$, equation (15) can be used to calculate y_{m3}, which can be then compared to y.

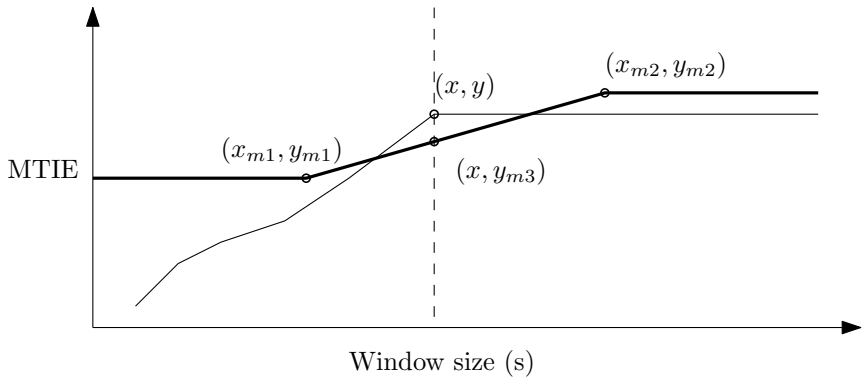

Figure 25: Mask matching

$$y_{m3} = \left(\frac{y_{m2} - y_{m1}}{x_{m2} - x_{m1}}\right) x + \left[y_{m1} - \left(\frac{y_{m2} - y_{m1}}{x_{m2} - x_{m1}}\right) x_{m1}\right] \tag{15}$$

Comparing results with appropriate masks this way, MTIE graphs can be abstracted to pass/fail status. This will simplify the interpretation of results and will also make the results applicable to TestNET reporting format.

5.3 Synchronization quality requirements

After MTIE has been calculated, it is of course very important to know what to compare it to. This section will introduce some relevant synchronization requirements.

ITU-T recommendation G.8261 [1] defines different deployment cases with specific synchronization budgets, which are shown in Figure 26. Deployment case 1 has the Circuit Emulation Service (CES) located as an island between two SDH segments, while deployment case 2 has CES on the edge of a network.

From this thesis' perspective, the main application of PTP is to synchronize base stations on the edge of operators' networks, so the applicable deployment case is the second one. Deployment case 2 has also two applications, A and B, where application B is synchronized from a TDM signal and is therefore not applicable here. ITU-T recommendation G.8261 Deployment case 2A wander budget is shown in Table 11, and is graphed in in Figure 27. Wander budged has been defined for observation window sizes from 0.05 s (exclusive) to 1000 s (inclusive). MRTIE requirement depends on observation window size on size range $0.05\,\text{s} < \tau \leq 0.2\,\text{s}$: $40\tau\,\mu$s, and $32\,\text{s} < \tau \leq 64$ s: $0.25\tau\,\mu$s, and is constant in size ranges $0.2\,\text{s} < \tau \leq 32\,\text{s}$: $8\,\mu$s, and $64\,\text{s} < \tau \leq 1000\,\text{s}$: $16\,\mu$s.

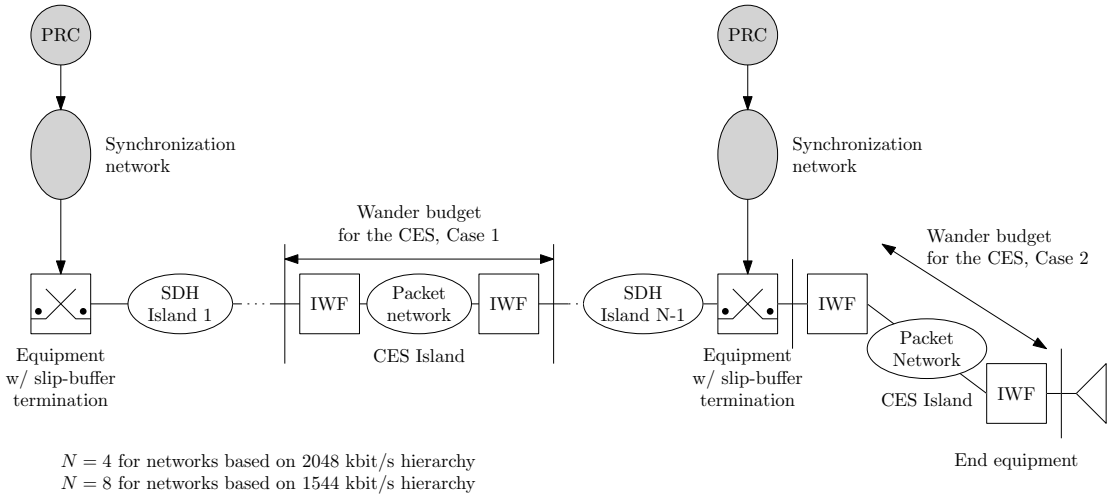

Figure 26: ITU-T recommendation G.8261 Network models for traffic and clock wander accumulation, Deployment Case 1 and Case 2 [1]

Table 11: ITU-T recommendation G.8261 Deployment case 2A: 2048 kbit/s interface wander budget [1]

Observation time (s)	MRTIE requirement (μs)
$0.05 < \tau \leq 0.2$	40τ
$0.2 < \tau \leq 32$	8
$32 < \tau \leq 64$	0.25τ
$64 < \tau \leq 1000$	16

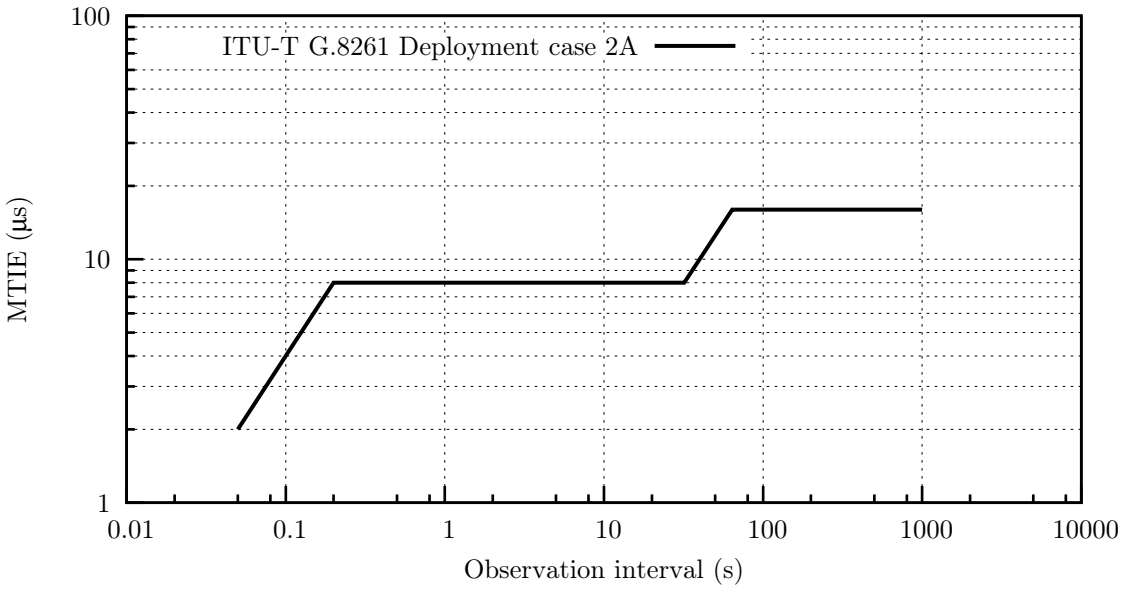

Figure 27: ITU-T recommendation G.8261 Deployment case 2A: 2048 kbit/s interface wander budget [1]

6 Implementation

Testing system was implemented with the components selected in Section 5. Test case design follows the principles enforced by TestNET: each test consists of test cases which include steps. Results from these are given in one report for each test.

This section will go through the implementation details of used test network and its components, control and reporting and different MTIE algorithm comparison.

6.1 Test network

Test network is illustrated in Figure 28. The used PTP master device, Symmetricom TP-5000 [67] is synchronized to a GPS timing reference and it distributes the reference clock signal to the rest of the network. PDV is generated with an emulator, Calnex Paragon [61], which is situated between master and slave devices. The device under test (DUT) is Tellabs 8605 [68], a mobile backhaul access router. TIE measurement is done with Pendulum CNT-90 [64] frequency counter/analyzer.

Slave and master devices, and PDV emulator are connected to each other with 1000Base-LX (gigabit single-mode optical) Ethernet. The network has two parallel reference clock signals: 2.048MHz and 10MHz due to the devices' clock input requirements. Measurement is done with a 2.048MHz clock signal from DUT clock output port.

Symmetricom TP-5000 [67] is used as a PTP master device. It is configured to use negotiation so that slave devices can request PTP streams with different parameters dynamically. Because of this, it doesn't require any configuration or commanding during the test execution.

Calnex Paragon [61] was selected as network emulator to introduce packet delays. Calnex provides PDV emulation playback profiles for G.8261 Appendix VI performance test cases 12 through 17 for both master to slave and slave to master directions. Paragon is controlled through a graphical user interface (GUI) software from a Windows PC. Calnex supplies a Tcl interface to control this GUI, which is used through the TestNET scripts.

Tellabs 8605 [68] mobile backhaul access router is configured with a Tellabs proprietary management protocol through TestNET. It will request a 128pps two-way PTP connection and use that to synchronize its internal clock. Output clock port is also enabled.

Pendulum CNT-90 [64] is controlled with SCPI commands through an Agilent E5810A [65] GPIB–TCP/IP gateway. It is configured to measure time interval error between its input ports (one for DUT, the other for reference). Results are fetched frequently during the test. Practically, controlling is done with a telnet connection from TestNET, and some minor library functions are created for this communication.

6.2 Test control and reporting

The device under test is controlled and test result reports are created with an in-house test automation system TestNET, which was introduced in Section 5.1.1.

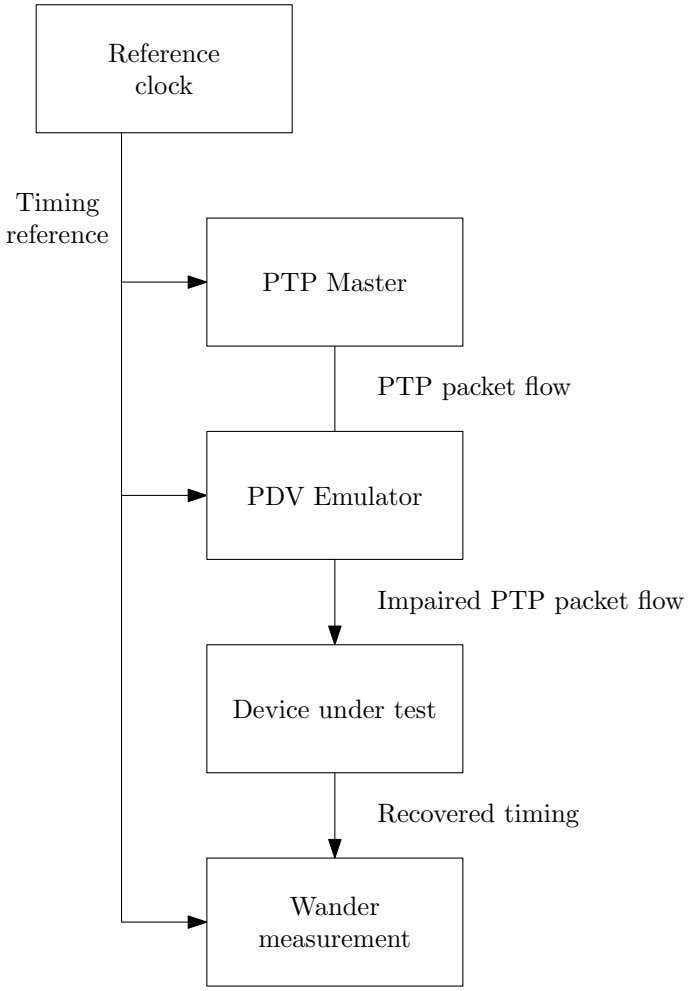

Figure 28: Test network

Tests are organized into three files with following categories: main script file, test parameters and test cases. Additionally there are libraries for test steps, which then again use libraries for communicating with the devices.

General architecture is is illustrated in Figure 29. Steps for each test case are similar to the principle introduced in Section 5.1.

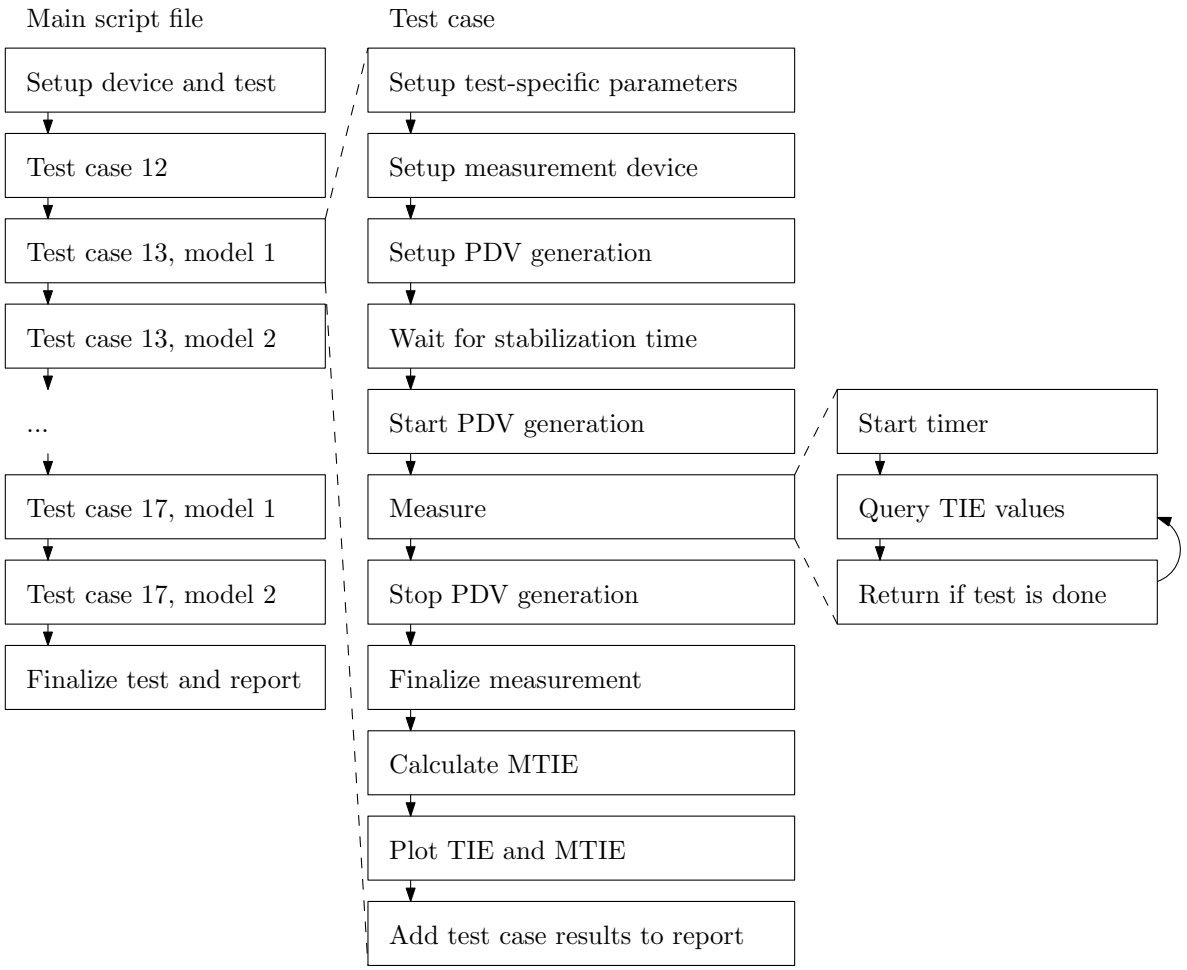

Figure 29: Test control architecture

6.2.1 Data processing

When raw TIE data is acquired from the measurement, it has to be processed to obtain TIE and MTIE graphs, and resulting MTIE has to be compared to appropriate masks. These results have to be presented in a clear way and both results and original measurement data have to be archived efficiently for possible further use.

TIE data from measurement contains two types of information: timestamps and corresponding TIE values. They are used in plotting TIE graph and calculating MTIE which are both unrelated to the measurement process itself and are run as separate programs.

For easy usage, TIE data is first saved as comma-separated values (CSV) plain-text and afterwards compressed. Data is gathered from one test set (G.8261 test cases 12..17) for each tested device type and software version, and stored along with graphs and MTIE CSV data. Due to the nature of raw TIE data (American Standard Code for Information Interchange (ASCII) formatted text), compressing with zip algorithm yields a disk space save of around 90 %, as can be seen from Table 12.

Table 12: Disk space saved with TIE data compression

	File size	Archived size	Reduction
Sample 1	5.9 MiB	553 KiB	90.7 %
Sample 2	26 MiB	2.9 MiB	89.1 %
Sample 3	92 MiB	8.3 MiB	91.0 %

6.2.2 Graph plotting

Both TIE and MTIE graph plotting is done using gnuplot, which is controlled with specific commands given to the program as a text-file. TIE graph presentation does not have any special configurations. Its header shows the DUT software version used, test case, date and data filename. MTIE graph has the same information and additionally it is shown in logarithmic scale both in x and y-axis, and has the appropriate masks plotted along with MTIE results. The masks are given as CSV files according to standards' definitions.

Output format was chosen to be a fixed-size Portable Network Graphics (PNG) file. For this kind of data, a vector graphics format would have been most suited because of portable scaling and file size. However the most common format, Scalable Vector Graphics (SVG) is not currently supported Internet Explorer version 6 or 7, which are both commonly used to view test results. In most common raster formats, Joint Photographic Experts Group (JPEG) and PNG file size difference (roughly 100KiB vs. 10KiB) was the main reason for choosing PNG as TIE and MTIE graph format.

gnuplot command files are generated with the main Tcl test script. Changing values are title, input and output file names and included masks. After the command file generation is done, gnuplot is executed from the test script.

Figure 30 shows an example output of gnuplot with TIE data from certain measurement. The graph's both axis are scaled automatically by gnuplot and header's missing fields are generated on per measurement basis from Tcl. Figure 31 has an example MTIE graph with previously introduced masks along with a calculated measurement result.

6.3 Maximum Time Interval Error calculation

The algorithms introduced in Chapter 5.2 were implemented one at a time. As the measurement times are long and calculating MTIE is not trivial, the most crucial aspect in implementation is the execution time.

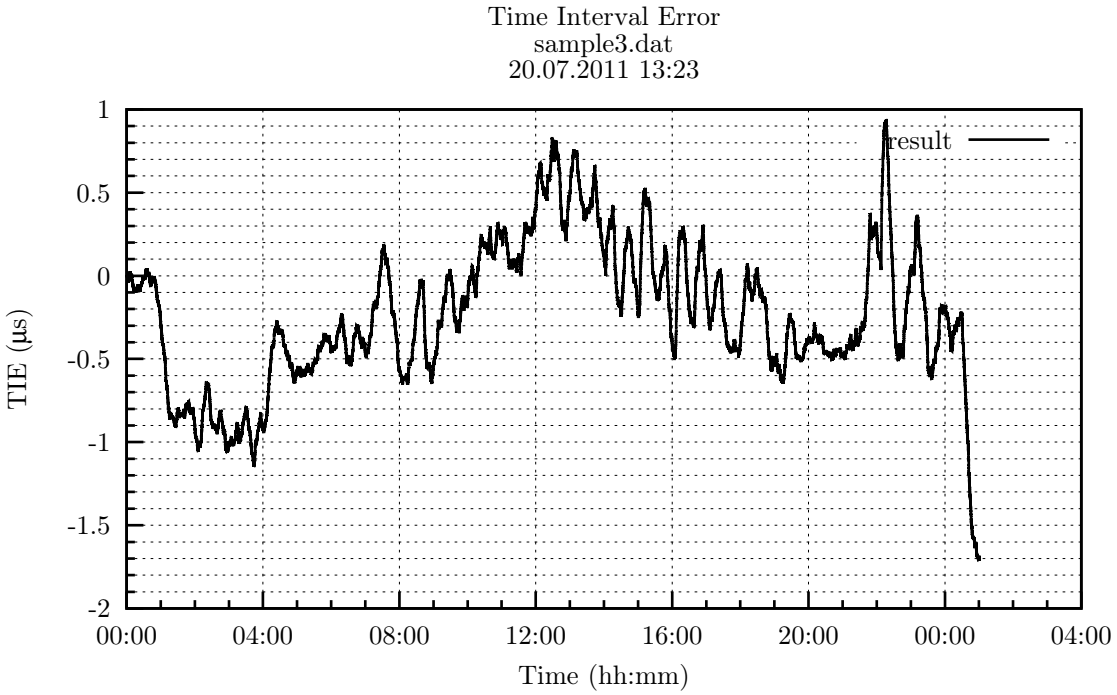

Figure 30: Example TIE figure

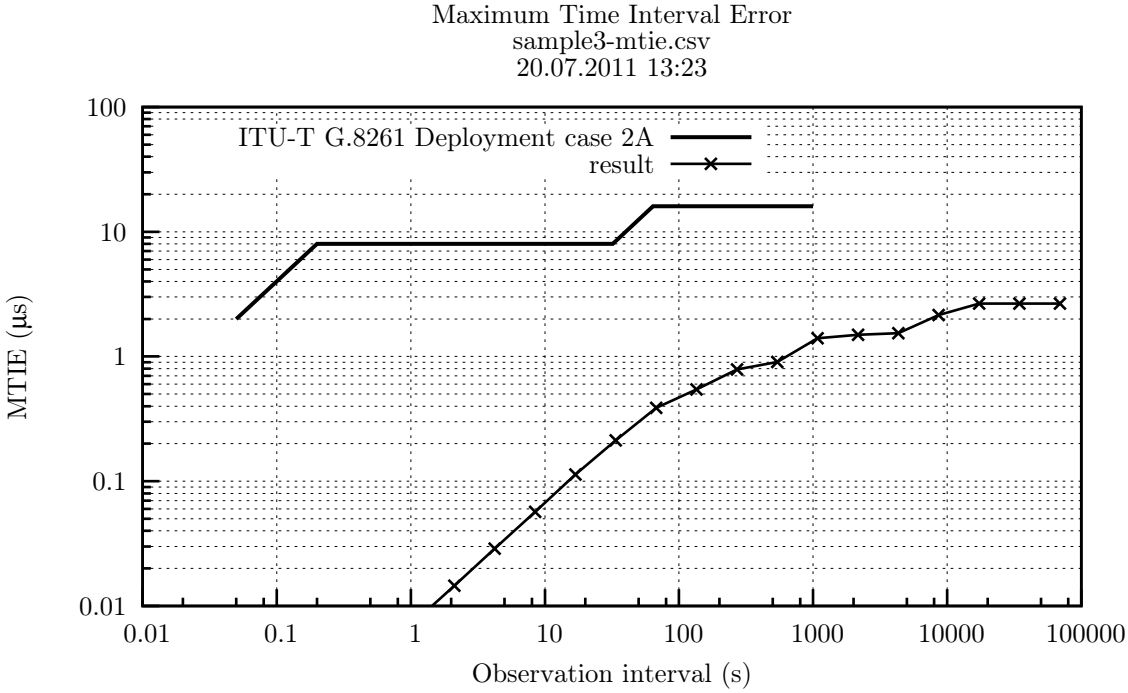

Figure 31: Example MTIE figure

Naive algorithm was quickly discovered to be unusable in real-world scenarios due to its poor performance. Extreme Fix algorithm was then implemented first in Tcl, and although it gave a huge boost in speed, it wasn't still fast enough. It was converted to C++ and compiled with two different compilers. Both were considerably faster than the Tcl implementation, and also a bit surprisingly Visual Studio (VS) [69] compiled version was notably faster than the one compiled with GNU Compiler Collection (GCC) [70]. Binary Decomposition algorithm was also implemented in C++ for further speedup, but it didn't have any benefit over Extreme Fix. These three different algorithms were compared with three different measurement sets. Results from these tests are show in Table 13.

First of all it can be noted that in the sample 1 data set, Extreme Fix algorithm reduces execution time by over 95 % compared to naive algorithm.

Comparing different implementations of Extreme Fix algorithm shows that Tcl is not designed to be used in calculation heavy programming. Where sample 2 data set took 15 hours with Tcl implementation, it took around one minute with both C++ implementations. While both GCC and Visual Studio compiled versions of the C++ Extreme Fix are usable, it is still notable that GCC version is roughly 50 % slower than Visual Studio version on large data sets (samples 2 and 3). This is probably due to heavier optimization parameters in the default configuration of Visual Studio.

Table 13: Comparison of different MTIE calculation algorithms and implementations

	Sample 1	Sample 2	Sample 3
Measurement duration	01:40:00	07:15:00	25:00:00
Number of samples	180 228	783 516	2 701 038
Algotithm execution time			
Naive (Tcl)	102:24:30	N/A	N/A
Extreme Fix (Tcl)	1:56:15	31:27:00	N/A
Extreme Fix (C++, GCC)	0:00:16	0:02:14	0:05:38
Extreme Fix (C++, VS)	0:00:06	0:01:04	0:02:30
Binary Decomposition (C++, GCC)	0:06:16	1:49:50	21:39:43

7 Evaluation

The most obvious benefit of this test automation is that nighttime can be used better when running the tests. Manually each test must be started during office hours, and depending on the duration of the tests, most will finish a lot before the next morning. This is shown in Table 14, taking into account the following schedule overheads along with PDV profile durations:

- Each test requires a stabilization period of 30 minutes before starting PDV profile.
- Both automatic and manual calculation of MTIE takes 5 minutes.
- Other test setup tasks take additional 5 minutes before each test.
- Testing can be done manually for 8 hours, and then it has to be paused until the next time divisible by 24 hours (the next morning).
- The fastest way to run this test set manually is to leave TC13s for the night as they take the longest time under 16 hours. "Manual optimized" in Table 14 is done like this.
- Test set completion has to happen at "daytime" because test analysis and continuing with different tests requires manual work.

From these results it can be seen that automated testing is 21 % faster than manual testing even when the use of nighttime is optimized. In practice, this result varies depending on when the test set is started, but automated testing will always be faster than manual because at least one nighttime can not be fully utilized when testing manually.

Previous result was calculated for one run of the G.8261 test set. Software projects however require this test set to be done for various different hardware platforms with different PTP settings. The speed gain effect will of course cumulate when the test set has to be run multiple times, and weekends introduce an even bigger delay than nighttime when test sets are executed manually.

Figure 32 shows a testing S-curve introduced in Section 4.8 when this test set has to be run ten times. Here the results for 10 sequential runs are ready 10 days earlier when using automated testing. The graphed data is rough and purely theoretical, but it can be seen that the delays caused by weekends on manual testing increase the time savings of automation from 20 % to over 40 %.

Along with getting the results earlier, using automated testing frees the test engineer to do something else while the tests are running. Also, running automated tests require less technical knowledge from test engineers as they no longer required to know all the test equipment specifics themselves. These two factors lead directly to cost savings in testing packet synchronization testing.

Table 14: Comparison of automated and manual runs of G.8261 test set

Automated		Manual		Manual optimized	
Start	0:00	Start	0:00	Start	0:00
TC12	1:40	TC12	1:40	TC14A	24:40
TC13A	8:20	TC13A	8:20	TC14B	49:20
TC13B	15:00	TC13B	15:00	TC15A	50:45
TC14A	39:40	Nighttime	24:00	TC15B	52:10
TC14B	64:20	TC14A	48:40	TC16A	53:35
TC15A	65:45	TC14B	73:20	TC16B	55:00
TC15B	67:10	TC15A	74:45	TC13A	61:40
TC16A	68:35	TC15B	76:10	Nighttime	72:00
TC16B	70:00	TC16A	77:35	TC17A10	73:25
TC17A10	71:25	TC16B	79:00	TC17A200	74:50
TC17A200	72:50	TC17A10	80:25	TC17B10	76:15
TC17B10	74:15	Nighttime	96:00	TC17B200	77:40
TC17B200	75:40	TC17A200	97:25	TC12	79:20
		TC17B10	98:50	TC13B	86:00
		TC17B200	100:15	Nighttime	96:00
Finished at	3 days, 3:40		4 days, 4:15		4 days, 0:00
Speed gain with automated testing			24:35		20:20
			25 %		21 %

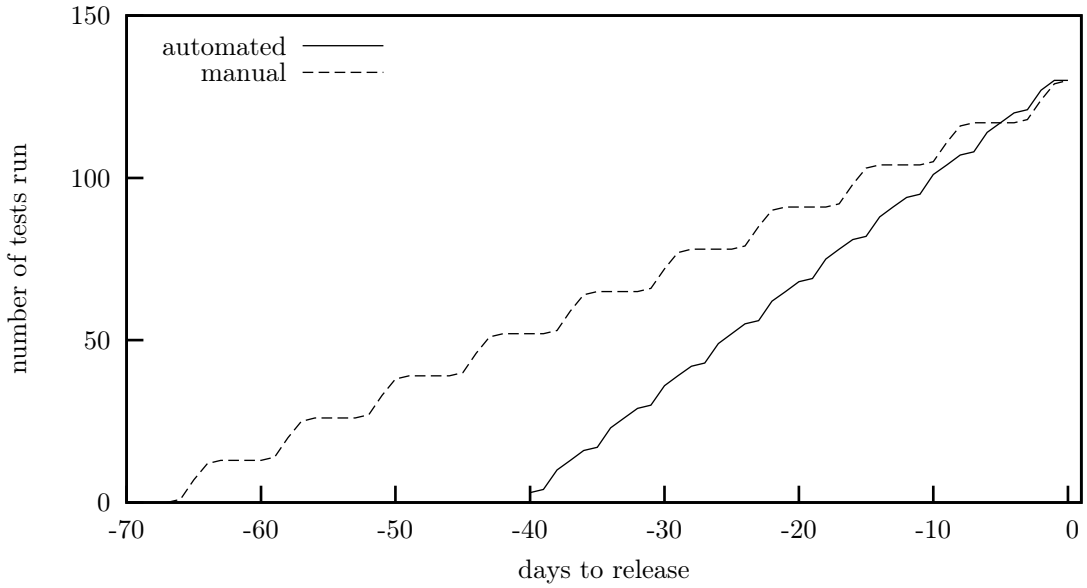

Figure 32: Test progress S-curve with 10 runs of G.8261 test set

8 Conclusions

The objective of this thesis was to create test automation for ITU-T recommendation G.8261 Appendix VI performance test cases 12 through 17 for Precision Time Protocol and calculate Maximum Time Interval Error (MTIE) graphs for results. First, a theoretical study for communications network synchronization in general, packet switched network synchronization and software testing was done. Then, the testing system was planned and different components for implementation were compared and after that the system was built. Finally the system's performance was compared with the situation where the same test suite was executed manually.

Evaluation showed that testing using the automated system required less human interaction and the test suite could be executed faster than if the tests were done manually. One test suite run has a speed gain of roughly 20 % when using the automated system, and with an example of ten test suite runs and taking weekends into account, this translates to 35 days faster time to market.

The next step in making the testing process even more faster would be to study if the test sets could be run in parallel for multiple devices under test. In embedded software design, it is common to have the same software release done for multiple hardware platforms at the same time. Synchronization with PTP is especially dependent on both the hardware and software, so all different hardware platforms will have to be tested with each software release. As the test sets are the same for different platforms, executing them in parallel could be beneficial, but is left outside the scope of this thesis.

While most implementation decisions were fairly straightforward, choosing the algorithm for MTIE calculation required some exploring. First of all, the most obvious algorithm implemented in Tcl was unusable even for the shortest tests as it took some 50 times more time to calculate the MTIE result than what was the duration of the test. A more clever algorithm reduced this ratio to around 1:1, which still seemed like a tremendous waste of time. Implementing the same algorithm in C++ reduced the calculation duration to a fraction of what it was. A third algorithm was also tried out, but it didn't yield any performance increase.

The best MTIE algorithm implementation could calculate a 25-hour measurement result in less than 3 minutes. This is of course still far from instantaneous, but as the automated system was not meant to be used interactively, it doesn't matter if the calculation doesn't give any results before a final result is calculated. However, if the MTIE result was to be shown interactively, it would be good to have an inaccurate result as soon as possible. This speedup at the cost of accuracy could be achieved with reducing the number of samples and different observation window sizes included in the calculation. Observation window count can be easily manipulated with the selected MTIE algorithm, but sample selection would require further study.

References

[1] *Timing and synchronization aspects in packet networks*, ITU-T recommendation G.8261/Y.1361, 2008.

[2] *Vocabulary of digital transmission and multiplexing, and pulse code modulation (PCM) terms*, ITU-T recommendation G.701, 1993.

[3] *Definitions and terminology for synchronization in packet networks*, ITU-T recommendation G.8260, 2010.

[4] S. Bregni, *Synchronization of digital telecommunications networks.* Chichester: John Wiley, 2002, ISBN: 0-471-61550-1.

[5] *Timing jitter and wander measuring equipment for digital systems which are based on the plesiochronous digital hierarchy (PDH)*, ITU-T recommendation O.171, 1997.

[6] *Jitter and wander measuring equipment for digital systems which are based on the synchronous digital hierarchy (SDH)*, ITU-T recommendation O.172, 2005.

[7] *Jitter measuring equipment for digital systems which are based on the optical transport network*, ITU-T recommendation O.173, 2012.

[8] *Jitter and wander measuring equipment for digital systems which are based on synchronous Ethernet technology*, ITU-T recommendation O.174, 2009.

[9] *Jitter measuring equipment for digital systems based on XG-PON*, ITU-T recommendation O.175, 2012.

[10] *Digital cellular telecommunications system (Phase 2+); Radio subsystem synchronization (3GPP TS 45.010 version 8.4.0 Release 8)*, ETSI technical specification 145 010, 2009.

[11] *Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (FDD) (3GPP TS 25.104 version 8.8.0 Release 8)*, ETSI technical specification 125 104, 2009.

[12] *Universal Mobile Telecommunications System (UMTS); Base Station (BS) radio transmission and reception (TDD) (3GPP TS 25.105 version 8.4.0 Release 8)*, ETSI technical specification 125 105, 2009.

[13] *Recommended Minimum Performance Standards for cdma2000 Spread Spectrum Base Stations*, 3GPP2 standard C.S0010-B, 2004.

[14] *Physical Layer Standard for cdma2000 Spread Spectrum Systems*, 3GPP2 standard C.S0002-C, 2004.

[15] *Definitions and terminology for synchronization networks*, ITU-T recommendation G.810, 1996.

[16] Time and frequency from A to Z: A to Al. National Institute of Standards and Technology. Accessed July 1, 2013. [Online]. Available: http://www.nist.gov/pml/div688/grp40/glossary.cfm

[17] O. Tipmongkolsilp, S. Zaghloul, and A. Jukan, "The evolution of cellular backhaul technologies: current issues and future trends," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 1, pp. 97–113, 2011.

[18] *Structure-Agnostic Time Division Multiplexing (TDM) over Packet (SAToP)*, IETF request for comments 4553, 2006.

[19] *Structure-Aware Time Division Multiplexed (TDM) Circuit Emulation Service over Packet Switched Network (CESoPSN)*, IETF request for comments 5086, 2007.

[20] *IEEE Std 802.3 - 2005 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, IEEE standard 802.3, 2005.

[21] *Timing characteristics of synchronous Ethernet equipment slave clock (EEC)*, ITU-T recommendation G.8262, 2007.

[22] *Network Time Protocol Version 4: Protocol and Algorithms Specification*, IETF request for comments 5905, 2010.

[23] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE standard 1588, 2008.

[24] B. Choi, S. Moon, Z. Zhang, K. Papagiannaki, and C. Diot, "Analysis of point-to-point packet delay in an operational network," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2004, pp. 1797–1807.

[25] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, "Measurement and analysis of single-hop delay on an ip backbone network," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 6, pp. 908–921, 2003.

[26] *IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Broadband Wireless Access Systems*, IEEE standard 802.16, 2009.

[27] *The control of jitter and wander within digital networks which are based on the 2048 kbit/s hierarchy*, ITU-T recommendation G.823, 2000.

[28] *The control of jitter and wander within digital networks which are based on the 1544 kbit/s hierarchy*, ITU-T recommendation G.824, 2000.

[29] *NodeB Synchronization for TDD*, 3GPP technical report 25.836, 2000.

[30] *Abstract Test Suite for Circuit Emulation Services over Ethernet based on MEF 8*, MEF technical specification 18, 2007.

[31] *Implementation Agreement for the Emulation of PDH Circuits over Metro Ethernet Networks*, MEF technical specification 8, 2004.

[32] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, *The art of software testing*, 2nd ed. Hoboken, N.J: John Wiley & Sons, 2004, ISBN: 0-47146-912-2.

[33] *IEEE Standard glossary of software engineering terminology*, IEEE standard 610.12, 1990.

[34] I. Burnstein, *Practical software testing : a process-oriented approach*. New York: Springer, 2003, ISBN: 0-387-95131-8 (printed), ISBN: 978-6-61-018845-1 (electronic).

[35] P. Jorgensen, *Software testing: a craftsman's approach*, 2nd ed. CRC Press, 2002, ISBN: 0-84930-809-7.

[36] W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, no. 8. Los Angeles, 1970.

[37] I. Sommerville, "Software process models," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 269–271, 1996.

[38] V-modell xt. Accessed February 2, 2013. [Online]. Available: http://v-modell.iabg.de/v-modell-xt-html-english/index.html

[39] M. Fowler and J. Highsmith, "The agile manifesto," *Software Development*, vol. 9, no. 8, pp. 28–35, 2001.

[40] M. Beedle, M. Devos, Y. Sharon, K. Schwaber, and J. Sutherland, "Scrum: An extension pattern language for hyperproductive software development," *Pattern Languages of Program Design*, vol. 4, pp. 637–651, 1999.

[41] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[42] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and analysis*. VTT Finland, 2002, ISBN: 951-38-6009-4 (printed), ISBN: 951-38-6010-8 (electronic).

[43] A. Cockburn, *Crystal clear: a human-powered methodology for small teams*. Boston (Mass.): Addison-Wesley, 2005, ISBN: 0-201-69947-8.

[44] J. Vanhanen, J. Jartti, and T. Kähkönen, "Practical experiences of agility in the telecom industry," *Extreme Programming and Agile Processes in Software Engineering*, pp. 1015–1015, 2003.

[45] M. Puleio, "How not to do agile testing," in *Agile Conference, 2006*. IEEE, 2006, pp. 7–pp.

[46] G. Rothermel and M. Harrold, "Analyzing regression test selection techniques," *Software Engineering, IEEE Transactions on*, vol. 22, no. 8, pp. 529–551, 1996.

[47] H. Leung and L. White, "A study of integration testing and software regression at the integration level," in *Software Maintenance, 1990., Proceedings., Conference on*. IEEE, 1990, pp. 290–301.

[48] G. Rothermel and M. Harrold, "A safe, efficient algorithm for regression test selection," in *Software Maintenance, 1993. CSM-93, Proceedings., Conference on*. IEEE, 1993, pp. 358–367.

[49] Y. Chen, D. Rosenblum, and K. Vo, "Testtube: A system for selective regression testing," in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*. IEEE, 1994, pp. 211–220.

[50] C. Kaner, "Pitfalls and strategies in automated testing," *Computer*, vol. 30, no. 4, pp. 114–116, 1997.

[51] J. Bach, "Test automation snake oil," in *14th International Conference and Exposition on Testing Computer Software, Washington, DC*, 1999.

[52] J. de Oliveira, C. Gouveia, and R. Quidute Filho, "Test automation viability analysis method," in *LATW2006: Proceedings of the 7th IEEE Latin American Test Workshop*, 2006, p. 6.

[53] S. Kan, J. Parrish, and D. Manlove, "In-process metrics for software testing," *IBM Systems Journal*, vol. 40, no. 1, pp. 220–241, 2001.

[54] Tcl Developer Xchange. Accessed November 20, 2013. [Online]. Available: http://www.tcl.tk/

[55] [incr Tcl] - Object-Oriented Programming in Tcl/Tk. Accessed November 20, 2013. [Online]. Available: http://incrtcl.sourceforge.net/itcl/

[56] gnuplot homepage. Accessed November 20, 2013. [Online]. Available: http://www.gnuplot.info/

[57] GD Graphics Library. Accessed November 20, 2013. [Online]. Available: http://www.libgd.org/

[58] Microsoft Excel spreadsheet software - Office.com. Accessed November 20, 2013. [Online]. Available: http://office.microsoft.com/en-001/excel/

[59] Apache OpenOffice Calc - OpenOffice.org. Accessed November 20, 2013. [Online]. Available: http://www.openoffice.org/product/calc.html

[60] Spirent AX/4000. Accessed November 20, 2013. [Online]. Available: http://www.spirentfederal.com/IP/Products/AX_4000/Overview/

[61] Calnex Paragon-X. Accessed November 20, 2013. [Online]. Available: http://www.calnexsol.com/products/paragon-x.html

[62] Ixia Ethernet Network Emulators GEM, XGEM. Accessed November 20, 2013. [Online]. Available: http://www.ixiacom.com/products/network_test/load_modules/network_emulators/gem_xgem/

[63] JDSU ANT Advanced Network Tester family. Accessed November 20, 2013. [Online]. Available: http://www.jdsu.com/productliterature/ant-consolidated_ds_opt_tm_ae.pdf

[64] Pendulum CNT-90 Frequency Timer/Counter/Analyzer. Accessed November 20, 2013. [Online]. Available: http://www.spectracomcorp.com/ProductsServices/SignalTest/FrequencyAnalyzersCounters/CNT-90TimerCounterAnalyzer/tabid/1280/Default.aspx

[65] Agilent Technologies E5810A LAN/GPIB Gateway. Accessed November 20, 2013. [Online]. Available: http://www.home.agilent.com/en/pd-1000004557%3Aepsg%3Apro-pn-E5810A/lan-gpib-gateway?&cc=FI&lc=fin

[66] A. Dobrogowski and M. Kasznia, "Time effective methods of calculation of maximum time interval error," *Instrumentation and Measurement, IEEE Transactions on*, vol. 50, no. 3, pp. 732–741, 2001.

[67] Symmetricom TimeProvider 5000. Accessed November 20, 2013. [Online]. Available: http://www.symmetricom.com/products/time-frequency-references/telecom-primary-reference-sources/timeprovider-5000/#.Uo0sN2QY1vY

[68] Tellabs 8605 Smart Router. Accessed November 20, 2013. [Online]. Available: http://www.tellabs.com/products/8000/tlab8605sr.pdf

[69] Visual Studio. Accessed November 20, 2013. [Online]. Available: http://www.visualstudio.com/

[70] GCC, the GNU Compiler Collection. Accessed November 20, 2013. [Online]. Available: http://gcc.gnu.org/