Evaluation and Standardizing of Phasor Data Concentrators

Hema A. Retty

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Virgilio A. Centeno, Chair

Jaime de la Ree Lopez

Arun G. Phadke

April 29, 2013

Blacksburg, Virginia

Keywords: IEEE C37.118, Phasor Data Concentrator, Phasor Measurement Unit, Synchrophasor, Wide Area Measurements

Evaluation and Standardizing of Phasor Data Concentrators

Hema A. Retty

Abstract

The power grid is interconnected in many ways; so that when disturbances occur in a small region, their effects can be seen across large areas causing major blackouts. In order to isolate the fault, measurements taken at different times throughout the blackout need to be collected and analyzed. With each measurement device having its own time source, time alignment can be a quite tedious and lengthy process. The need for a new time synchronized measurement device has arrived. The Phasor Measurement Units (PMU) is not only GPS time synchronized, but it also takes measurements as voltage and current phasors.

PMUs are becoming an integral part in many power system applications from load flow analysis and state estimation to analyzing blackout causes. Phasor Data Concentrators (PDC) collect and process PMU data. As such, it is important that PMU and PDC communication is seamless. PDCs are set up at multiple utilities and power authorities and also need to be able to communicate and send data to one another seamlessly to encompass analysis of large measurement systems. If these devices are not working similarly when processing and sending/receiving data, unnecessary problems may arise. Therefore it is important that there is an expectation as to how they should work. However, what is expected from these devices is not entirely clear. For this reason, standards such as IEEE C37.118.2-2011 [5] have been proposed to help make operation as uniform as possible. Unfortunately, the standards for PDCs are lacking and tend to only set up communication protocols. To help normalize PDCs, these standards need to be expanded to include all PDC operations and give little room for discrepancy as to what a PDC should do in any given situation. Tests have been performed on PDCs not only to see how they match up to current standards but on how they act outside of the standards.

To My Parents: Athirame Retty and Chandrika Retty

Acknowledgements

I am extremely thankful to my academic adviser, Dr. Virgilio Centeno, for all his guidance and support in pursuing this thesis. He was a mentor throughout my graduate studies and an invaluable source of information towards my research.

My appreciation and thanks goes out to Dr. Yaman Evrenosoglu for being a wealth of knowledge and continuously motivating me in my course work. I am grateful to my lab mates who have helped me on countless occasions throughout my graduate studies.

Lastly, I would like to thank my parents for their continuous support and encouragement, allowing me to achieve this goal.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

As the demand for electricity has been increasing, the power grids across the United States and around the world have been burdened. The increasing complexity of the power system has been a cause of the many major blackouts in recent years. The data measured during the disturbances must be collected over a large geographical area to determine the sequence of events and eventually the cause of the failure. Traditionally, the measurements are communicated by Remote Terminal Units (RTUs) and each RTU has its own local time source, therefore there may be large variations in the time stamp of the measurements from different RTUs. There are communication delays due to transmission or network setbacks while sending the data to its destination. These two factors create a large window for the time stamp of a state estimation scan. For the same reasons, in the past, it had taken months to time align data after a major disturbance had occurred. The need for a more reliable, time synchronized method for data capture had arisen.

1.2 Phasor Measurements

Phasors are complex numbers that represent the magnitude and phase angle of a sinusoidal waveform. In power systems, the phase angle difference between the voltages at two terminals of a line is directly linked to the active power flow across the line. The importance of phase angle measurements has been well known since the early 1980's. The significance of symmetrical component voltage and current magnitudes had come into light in the 1970's when efficient algorithms to determine faults on a three phase transmission line were being developed [1]. Finding the positive sequence phase measurements became necessary in power system

analysis and this initiated synchronized phasor measurement technology. During the same time period, the Global Positioning System (GPS) was being developed. This is a space-based satellite navigation system that connects to receivers on land to give accurate location and time information in all weather conditions as long as the receiver had an unobstructed line of sight to at least four of the satellites in orbit. This system has been fully operational since the mid 1990's when all 24 satellites owned by the U.S. Department of Defense were launched. The time synchronization component of GPS technology was incorporated into modern Intelligent Electronic Devices (IED) to create synchrophasors measurement devices such as Phasor Measurement Units (PMUs). The term synchrophasor refers to phasor measurements taken at the exact same time.

1.3 Wide Area Measurement Systems

The devices called Phasor Measurement Units (PMUs) form the basis of Wide Area Measurement Systems (WAMS). A variety of WAMS based technologies have been proposed in the past few decades to monitor, control and protect power grids all over the world. As the name suggests, wide area literally refers to the wider geographical area that the measurements are time synchronized within. This term also implies all the modern resources used to measure, time tag and synchronize the measures taken in this area.

The measurements are required to be time tagged with Coordinated Universal Time (UTC) which is based on star alignment. In actuality, the devices are connected to a GPS receiver that acquires a time tag based on the atomic clock which is accurate to +/- 100 nanoseconds. The high precision 1 pulse per second (1 pps) of GPS receivers is used by PMUs to synchronize their sampling clocks to +/- 1 microsecond among PMUs. These devices provide precise data to monitor the grid at rates up to 60 samples per second which is invaluable information during the occurrence of a fault.

The WAMS architecture shown in Figure 1 consists of a network of Phasor Measurement Units and Phasor Data Concentrators (PDCs) that feed data into the Supervisory Control and Data Acquisition (SCADA) system [2]. The PMU takes phasor measurements of voltages and

currents that are time synchronized to Global Positioning System (GPS) clocks. These measurements are sent, at rates of 1 to 60 frames per second, to a PDC which aligns the time tags from all PMUs and combines them in a single data frame that is transmitted to the SCADA system and/or to any WAMS-based application [3].

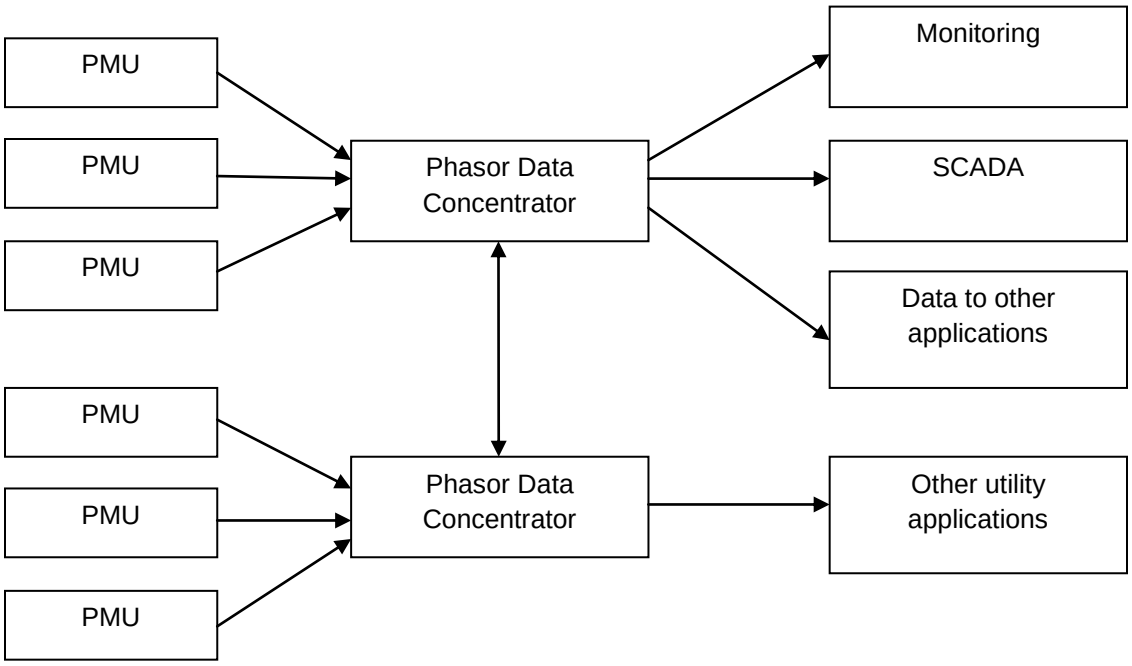

Figure 1.1: WAMS Architecture

This architecture had been refined over the years, but it was initially field tested by the Western Electricity Coordinating Council (WECC) as part of an Electric Power Research Institute (EPRI) project [4]. After the major Northeastern blackout of 2003 and the U. S. government Stimulus funding for smart grid modernization and development, power system authorities throughout the United States have begun to install PMUs and PDCs across their networks for incorporating phasor measurements into their SCADA system. The North American Electric SynchroPhasor Inititative (NASPI) supported by the North American Electric Reliability Corporation (NERC) provides a forum to support the progress in Synchrophasor technology within North America and around the world [5].

1.4 Synchrophasor Standards

To ensure proper use and application of the data, standards have been developed for the operation and evaluation of Synchrophasor measurements. The IEEE Standards Association first developed the IEEE 1344 in 1995 as a start to the synchrophasor protocol. This standard was superseded by the IEEE C37.118, *Standard for Synchrophasor Measurements for Power Systems*, in 2005 which addressed issues concerning PMUs. This document was updated in 2011 with the IEEE C37.118.1-2011. A new standard for Synchrophasor Data Transfer for Power Systems, IEEE C37.118.2-2011, was also established at the same time. The IEEE C37.118.1-2011 and its predecessor, IEEE C37.118.1-2005 have aided PMU manufacturers in the development of "standardized" PMUs whose data can be safely and easily used with data from any other compliant device.

The IEEE C37.118.2-2011 standard establishes specific formats for data exchange among PMUs and PDCs but fails to establish sufficient requirements and testing guidelines to enable PDC manufacturers on the development of a "standardized" PDC whose operation can be fully understood and trusted by the utilities and developers of Wide Area Data Applications. The IEEE Standards Association is developing the IEEE Draft Guide for Phasor Data Concentrators, which describes the performance, functional and communication needs of PDCs [4]. The IEEE Power and Energy Society have formed a working group under the Power System Relaying Committee (PSRC) to create a guide for Phasor Data Concentrator Functions for Power System Monitoring, Protection and Control. This guide is still a work in progress and the first draft is IEEE Standard C37.244-2013.

The NASPI Performance and Standards Task Team (PSTT) has prepared two comprehensive documents to comply with the IEEE C37.118 Standards in PMU testing and characterization of PMUs and instrumentation channels [5]. To successfully adopt and rely on Synchrophasor technology for observability analysis, state estimation, controlling relays and many other applications, a thorough set of guidelines and standards must be established for all devices in Wide Area Monitoring and Control (WAMC) Systems. There are well established constraints on PMUs; however, the boundaries for PDCs are yet incomplete.

1.5 Thesis Outline

One main objective of this thesis is to describe the tests developed and implemented at Virginia Tech for the existing data transfer guidelines illustrated in IEEE C37.118.2-2011 and to show how current PDCs meet these standards. Another focal point of this thesis is to derive new data communication tests with the aim of further evaluating and comparing the performance of PDCs for wide area measurement applications. These tests will lead to establishing new standards that will allow power utilities and authorities to fully utilize the data collected by the synchrophasors. This research is partially funded by and conducted under the authority of Pennsylvania Jersey Maryland (PJM) Interconnection which is a Regional Transmission Organization (RTO) part of the Eastern Interconnection grid.

Chapter 2 provides a more detailed description of PDC technology and of the three IEEE Synchrophasor standards. This chapter also explains how the standards can be interpreted to assemble a list of properties to be tested. Chapter 3 illustrates a sample network and provides a description of each test. Chapter 4 displays the testing equipment and setup along with the test results. Chapter 5 is a summary of the test results and how they relate to the current and future standards and requirements. This chapter also suggests future work in the area of Synchrophasor communication guidelines and overall Wide Area Measurement Systems.

Chapter 2

Standards and Guidelines of Phasor Data Concentrator

2.1 Phasor Data Concentrator

A Phasor Data Concentrator is a PMU data gathering device that synchronizes the measurements taken at every time instant independent of when the data was received. Similar to the PMU, the PDC time needs to be synchronized. This type of alignment removes errors caused by the physical distance between PMU and PDC and other network and communication timing issues. The PDC aims to reduce the data processing time and computational requirements needed to time align, translate, error check, and/or change the data rate of PMU data from multiple PMUs. Once these measurement frames are processed by the PDC, they are sent to the SCADA and other monitoring and control applications. They are also sent to additional PDCs connected to other power utilities [4]. A network of distributed PDCs may be formed to serve a hierarchy of systems: substation, utility, control area, reliability coordinator, and interconnection level. Distributed PDCs may also interact with each other on a peer-to-peer basis among utilities, control areas, and reliability coordinators. Each layer in the hierarchy may be serving different requirements depending on applications. These requirements include latency, quality and resolution of data along with archival, event triggering and data capturing [7].

Based on their development, PDCs can be divided into two categories, PDC-only devices and devices with added PDC functionality. The PDC-only devices are designed to work only with PMU data. The devices with added PDC functionality are data gathering devices developed for other utility applications where the PDC function has been added to the existing functions. Due to the limitations set by other applications and their existing standards, this research focuses on PDC-only devices. Based on their implementation, PDC-only devices can be divided into hardware and software PDC types. A hardware PDC is a complete device with limited number of inputs and usually aim**s** for applications with a small number of PMUs such as at substations. A software PDC is a software package implemented in recommended commercial hardware (high end PCs and Servers) and whose hardware size and structure is determined by the size of the

wide area network. This type of device can also be used at a substation with multiple PMUs. The tests and procedures discussed in this thesis apply to both hardware and software PDCs unless specified.

PMUs are connected to a single PDC via Ethernet cable which currently has a maximum downlink speed of up to 1 gigabit. The data is transmitted using standard network protocols, User Datagram Protocol (UDP)/Internet Protocol (IP) or Transmission Control Protocol (TCP)/IP. Figure 2 below illustrates the communication differences between the two protocols. When UDP/IP is used, the source broadcasts the data to the destination IP address without handshaking and predetermining the transmission channel. There is no way to check whether the data was received at the destination so it is common for packets to be dropped with UDP/IP; however, the data stream following a dropped packet will be unaffected. If TCP/IP is used for communication, there is handshaking between the source and destination node and packets are only sent once a data path has been determined and acknowledged. If a packet is not received at the destination, then it will be retransmitted from the origin. This provides error checking and correction over the data frames but make TCP/IP significantly slower than UDP/IP.

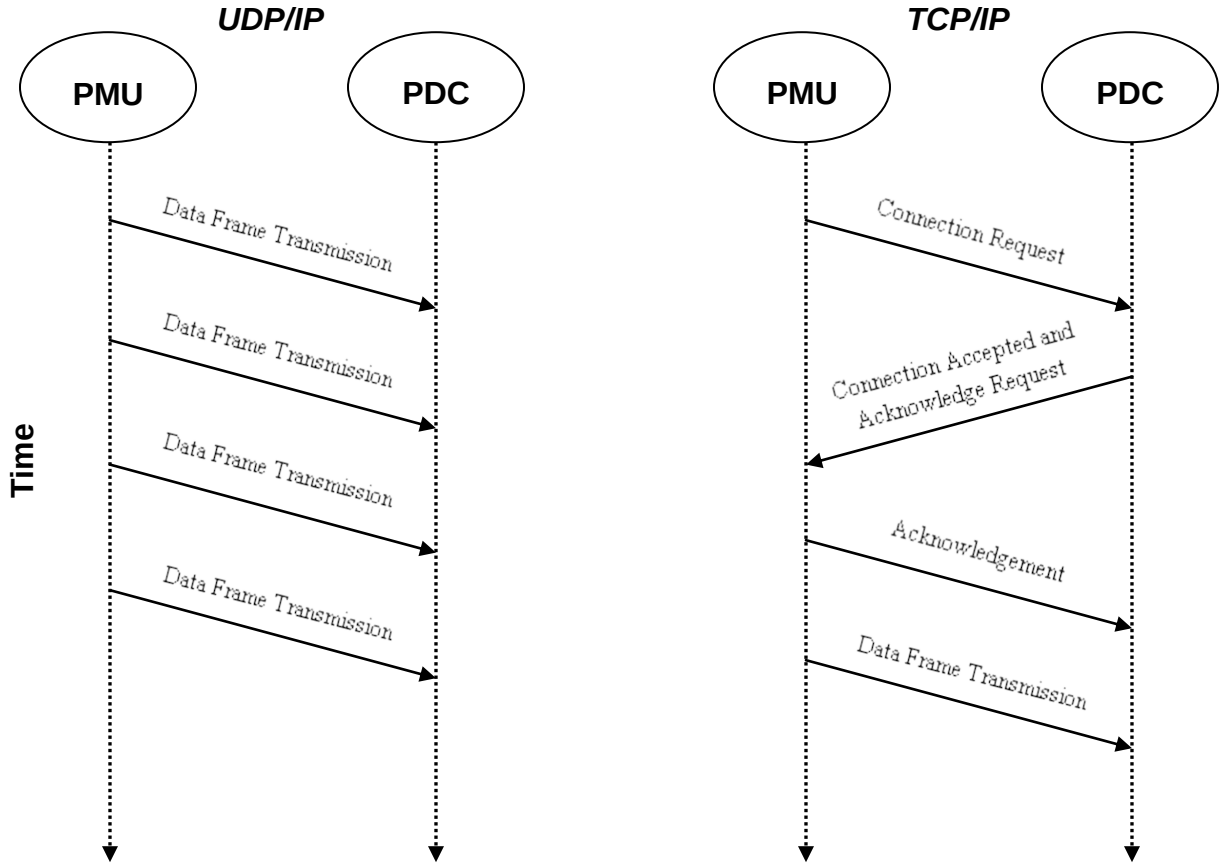

Figure 2.1: a) UDP/IP Data Stream and b) TCP/IP Data Stream

2.2 Data Transfer Standard

2.2.1 Reporting Rate and Time Tag

The standard established for synchrophasor data transfer is the IEEE C37.118.2-2011 and this is the only document that can be used as a guideline for PDC communication. This standard defines the reporting rate of the incoming PMU data frames as sub-multiples of the system frequency such as 10, 25 and 50 Hz for a 50 Hz system and 10, 12, 15, 20, 30 and 60 Hz for a 60 Hz system. The PDC must send the time aligned frame at an equal, lower or higher data rates using down-sampling or up-sampling methods. The measurement time tag of the synchrophasor

consists of three numbers; the Second-of-Century (SOC), the Fraction-of-Second (FRACSEC) and the time quality flag. The SOC is a binary count of seconds from UTC midnight (00:00:00) of January 1, 1970 [8] to the current second. Each device has a TIME_BASE integer and the actual fraction of the second count is an integer representing the FRACSEC divided by the TIME_BASE of the device so that each frame is evenly spaced. Both components combine to form the actual time as shown in equation (1). The FRACSEC is always zero at the turn of a second.

$$\text{Time} = \text{SOC} + \text{FRACSEC}/\text{TIME\_BASE} \qquad (2.1)$$

The time quality flag indicates the maximum uncertainty in the time stamp at the time of measurement. This value is based on the time source and includes the uncertainties in the PMU measuring process. The synchrophasors must be synchronized to UTC time to meet the accuracy requirements of IEEE Standard C37.118.1. The maximum steady-state error allowed by the standard can cause up to 1% total vector error (TVE) which is equivalent to a phase error of 0.01 radians. This corresponds to a time error of +/-26 microseconds for a 60Hz system and +/- 31 microseconds for a 50 Hz system. These limits require a highly reliable time source to be connected to the synchrophasor devices and this is why GPS clocks are used to provide the required time accuracy.

2.2.2 Message Framework

Important criteria outlined in the IEEE C37.118.2-2011 standard for PMU data exchange include Synchrophasor message format, GPS clock synchronization, Time Quality flag, and missing or late data handling. There are four types of frames that are transmitted between PMU/PDC and PDC. These include the data, configuration, header and command frames. Each frame constitutes of a SYNC word, FRAMESIZE word, IDCODE, and time stamp numbers followed by the data and CHECK word. The standard defines the synchrophasor message format in a configuration frame and data frame. The configuration frames 1 and 2 (CFG-1 and CFG-2) are divided into 21 fields as shown in Table 2.1 and configuration frame 3 (CFG-3), which was recently added to the new standard, has 29 fields including all those in CFG-1 and 2 plus a few

additional fields to define PMU characteristics and quantities. Configuration management is an important and essential activity of a PDC. It is designed to assure availability of appropriate data for the local functions of the PDC as well as other applications that receive data from the PDC. Configuration information is used to separate signal identities, data format and other metadata from synchrophasor data transmissions [7]. Separating this information from the data frame permits better use of the communication bandwidth.

Table 2.1: Configuration Frame 1 and 2 Organization [8]

No	Field	Size (bytes)	Short description
1	SYNC	2	Sync byte followed by frame type and version number.
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	Stream source ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Message Time Quality, defined in 6.2.
6	TIME_BASE	4	Resolution of FRACSEC time stamp.
7	NUM_PMU	2	The number of PMUs included in the data frame.
8	STN	16	Station Name—16 bytes in ASCII format.
9	IDCODE	2	Data source ID number identifies source of each data block.
10	FORMAT	2	Data format within the data frame.
11	PHNMR	2	Number of phasors—2-byte integer (0 to 32 767).
12	ANNMR	2	Number of analog values—2-byte integer.
13	DGNMR	2	Number of digital status words—2-byte integer.
14	CHNAM	16 × (PHNMR + ANNMR + 16 × DGNMR)	Phasor and channel names—16 bytes for each phasor, analog, and each digital channel (16 channels in each digital word) in ASCII format in the same order as they are transmitted. For digital channels, the channel name order will be from the least significant to the most significant. (The first name is for bit 0 of the first 16-bit status word, the second is for bit 1, etc., up to bit 15. If there is more than 1 digital status, the next name will apply to bit 0 of the second word and so on.)
15	PHUNIT	4 × PHNMR	Conversion factor for phasor channels.
16	ANUNIT	4 × ANNMR	Conversion factor for analog channels.
17	DIGUNIT	4 × DGNMR	Mask words for digital status words.
18	FNOM	2	Nominal line frequency code and flags.
19	*CFGCNT*	2	Configuration change count.
	Repeat 8–19		Fields 8—19, repeated for as many PMUs as in field 7 (NUM_PMU).
20+	DATA_RATE	2	Rate of data transmissions.
21+	CHK	2	CRC-CCITT.

The configuration frames and the data frames contain the time tag fields, SOC, FRACSEC and TIME_BASE; they also contain the IDCODE of the device. These fields must be checked for the correct data.

Table 2.2: Data Frame Organization [8]

No.	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number.
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	Stream source ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2, for all measurements in frame.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2, for all measurements in frame.
6	STAT	2	Bit-mapped flags.
7	PHASORS	4 × PHNMR or 8 × PHNMR	Phasor estimates. May be single phase or 3-phase positive, negative, or zero sequence. Four or 8 bytes each depending on the fixed 16-bit or floating-point format used, as indicated by the FORMAT field in the configuration frame. The number of values is determined by the PHNMR field in configuration 1, 2, and 3 frames.
8	FREQ	2 / 4	Frequency (fixed or floating point).
9	DFREQ	2 / 4	ROCOF (fixed or floating point).
10	ANALOG	2 × ANNMR or 4 × ANNMR	Analog data, 2 or 4 bytes per value depending on fixed or floating-point format used, as indicated by the FORMAT field in configuration 1, 2, and 3 frames. The number of values is determined by the ANNMR field in configuration 1, 2, and 3 frames.
11	DIGITAL	2 × DGNMR	Digital data, usually representing 16 digital status points (channels). The number of values is determined by the DGNMR field in configuration 1, 2, and 3 frames.
	Repeat 6–11		Fields 6–11 are repeated for as many PMUs as in NUM_PMU field in configuration frame.
12+	CHK	2	CRC-CCITT

The data frame is separated into 12 fields as shown in Table 2.2 [8] above, and it is the most frequent type of frame that is transmitted. The data frame includes all the PMU/PDC input measurements in separate data blocks. The data frame outputs a 16 bit STAT word for each PMU data block which provides the complete status for that specific PMU block. These status bits must be checked to see if the PDC is processing the PMU inputs correctly. The PDC waits a fixed maximum amount of time to receive all its PMU data, and if this wait time is exceeded before all the data is received, the PDC will substitute the missing PMU data block with filler data. Bits 14-15 are the Data Error Indicator, and if the PMU data block consists of filler data

then these bits will be set to 10. The actual data can also be set to show that it is invalid such as with floating-point data, NaN (Not a Number), can be inserted and for fixed-point data, -32768 can be inserted for both polar and rectangular forms. When the PMU has lost synchronization lock with its external time source or when the PDC detects a synchronization error in the incoming PMU input, then the PDC sets bit 13, PMU Sync Error, to 1. There are uncertainties in the measurement time at the actual time of measurement especially if the external time source is locked and then unlocked frequently. Bits 6-8 indicate the PMU Time Quality and the range of codes used are shown in Table 2.3 below.

Table 2.3: 3-bit PMU Time Quality Indication Codes (PMU_TQ) [8]

BINARY	HEX	VALUE (worst-case accuracy)
111	7	Estimated maximum time error > 10 ms or time error unknown
110	6	Estimated maximum time error < 10 ms
101	5	Estimated maximum time error < 1 ms
100	4	Estimated maximum time error < 100 µs
011	3	Estimated maximum time error < 10 µs
010	2	Estimated maximum time error < 1 µs
001	1	Estimated maximum time error < 100 ns
000	0	Not used (indicates code from previous version of profile)

If the PMU loses time synchronization, the amount of time that has passed since the loss of synchronization is represented in bits 4-5, Unlocked Time. Table 2.4 shows the unlocked time ranges and bit codes. The standard also includes the Header frame and Command frame organization provided in appendix A.1 and A.2, respectively.

Table 2.4: 2-bit Unlocked Time Bit Code [8]

Bit code	Indication
00	Locked or unlocked less than 10 s
01	Unlocked 10 s or longer but less than 100 s
10	Unlocked 100 s or longer but less than 1000 s
11	Unlocked 1000 s or more

The header and command frames contain fields similar to that of the configuration and data frames such as SYNC, FRAMESIZE, IDCODE, SOC and FRACSEC. The data field in the header frame contains source information such as the MAC address of each input, scaling, filtering, and other algorithms [9]. The command frame is used by the PDC to send commands to the data sending device (PMU or PDC). The IDCODE represents the type of command request from the PDC, and the data sending device shall execute the command using a matching IDCODE. Tables 2.5 and 2.6 below show the organization of each field in the header frame and command frame, respectively.

Table 2.5: Header Frame Organization [8]

No	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number (AA11 hex).
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	PMU/PDC data stream ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2.
6	DATA 1	1	ASCII character, 1st byte.
K+6	DATA k	1	ASCII character, Kth byte, K>0 is an integer.
K+7	CHK	2	CRC-CCITT.

Table 2.6: Command Frame Organization [8]

No	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number (AA41 hex).
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	PMU/PDC ID data stream number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2.
6	CMD	2	Command being sent to the PMU/PDC (0).
7	EXTFRAME	0–65518	Extended frame data, 16-bit words, 0 to 65518 bytes as indicated by frame size, data user defined.
8	CHK	2	CRC-CCITT.

2.3 Requirements for Applications

PDCs are set up at multiple utilities and power authorities; they need to be able to communicate and send data to one another seamlessly to encompass analysis of large measurement systems. If these devices are not working similarly when processing and sending/receiving data, unnecessary problems may arise. Therefore it is important that there is an expectation as to how they should work. However, what is expected from these devices is not entirely clear. For this reason, standards such as IEEE C37.118.2-2011 [8] have been proposed to help make operation as uniform as possible. Unfortunately, the standards for PDCs are lacking and tend to only set up communication protocols. To help normalize PDCs, these standards need to be expanded to include all PDC operations and leave little room for discrepancy as to what a PDC should do in any given situation.

2.3.1 Latency

For any wide area measurement application one important limit that needs to be known is the acceptable range of latency associated with a packet of data. Latency can be described as the time difference between the start of creating the data at the source and the end of processing it at the destination. There are many sources of delay within a network and they can be classified into stages such as data frame construction at the source, data transmission through the network and packet processing at the destination. One example of a wide area network configuration is having a PMU is situated in a local substation and the PDC is located at a remote substation or control center as shown in Figure 2.2 a). After processing the data, the PDC transmits the packets to applications at the control center as shown in Figure 2.2 b).

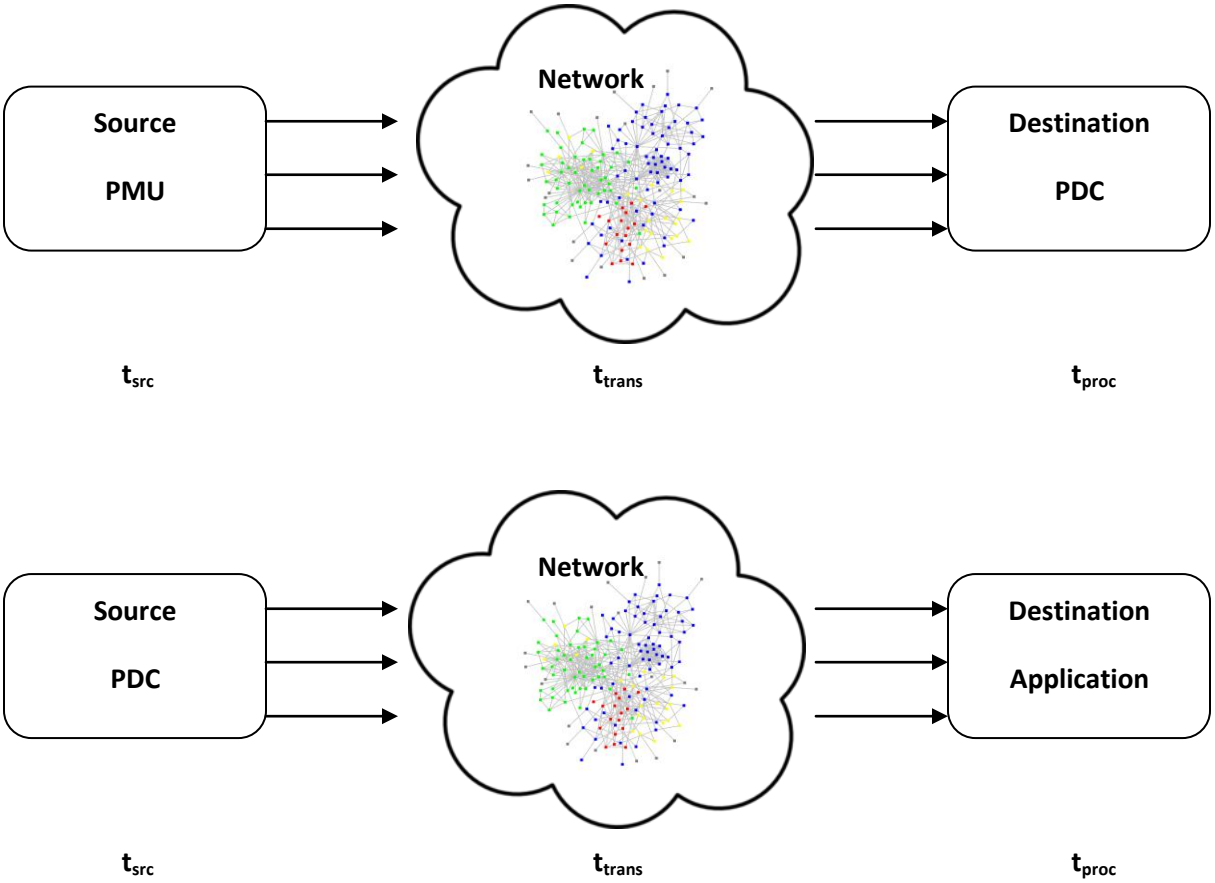

Figure 2.2: Time Delays in Wide Area Measurement Network. a) Delays between PMU at substation and PDC at control station (top). b) Delays between PDC at substation and application at control station (bottom).

The first step in the measurement process is the PMU. The PMU must take accurate voltage and current phasor measurements at specific time intervals according to the external time source; it must convert these values and organize them in a data frame. This accounts for the delay at the source caused by creating the data frame, t_{src}. Acceptable delays for this process are defined in IEEE C37.118.1-2011 [9].

There are many causes for latency during transmission through a network. . The transmission delays vary depending on the communication protocol chosen such a TCP/IP or UDP/IP as explained in section 2.1. Depending on the size of the network and the distances

between PMU and PDC, the largest source of latency can be the transmission along the Ethernet wire. The wire-line latency equation for a 100 km fiber optic cable from [10] is:

$$t_{WL} = \frac{1x10^5 m}{0.67 * 3x10^8 \frac{m}{s}} \approx 500\mu s$$

$$t_{WL} = \frac{1x10^3 m}{0.67 * 3x10^8 \frac{m}{s}} * Number\ of\ km$$

$$t_{WL} \approx 5\mu s\ x\ Number\ of\ km \quad (2.2)$$

Each node the packet travels through will add some amount of latency. Routers and switches are common elements in a network and the time they each take to receive the packet and then process it before transmitting it is an additional delay. There are other types of information that may use the same network for communication and cause traffic and congestion; this may hold the packets in a queue or in worse situations lose the packet completely. The term, queue, used in communication networks, refers to a sequence of packets stored in a type of line up before being processed. In reality, all queues have a limited size and any packets that arrive after the queue size has been exceeded will be discarded. Queues can be quite complex and many different types of ordering algorithms can be applied on them depending on the priority given to the data being transmitted. Queuing algorithms and latency of a queue is already a specialized research area involving Queuing Theory, Network Calculus and Quality of Service (QoS) and will not be discussed in much more detail in this paper. The latency of the network cannot be controlled by the PDC, but an acceptable range of network latency should be set to ensure successful operation.

The PDC has a latency associated with processing incoming PMU data and outputting a corresponding synchronized frame. Figure 2.3 demonstrates the individual stages of phasor measurement alignment at the PDC terminal. PMU data may be transmitted from within the same substation as the PDC or it may travel many miles within a network to reach the PDC. The PDC must aggregate the PMU inputs into a buffer before processing. The PDC holds the data in the buffer up to a maximum period of time to accommodate for variations in network delays. After the last PMU input has arrived or the maximum wait time has been reached, the data will

be time-aligned and validated. This is followed by data conversion, placement in the output frame and transmission. The START is labeled as the time at which the last PMU input has arrived or the maximum wait time has been reached for that individual time tag. The END is when the output frame of that time tag has been transmitted by the PDC. The period between the START and the END is referred to as PDC Latency, and it is an important requirement that must be set according to the application's needs. The PDC Wait Time is an additional delay to the PDC Latency.

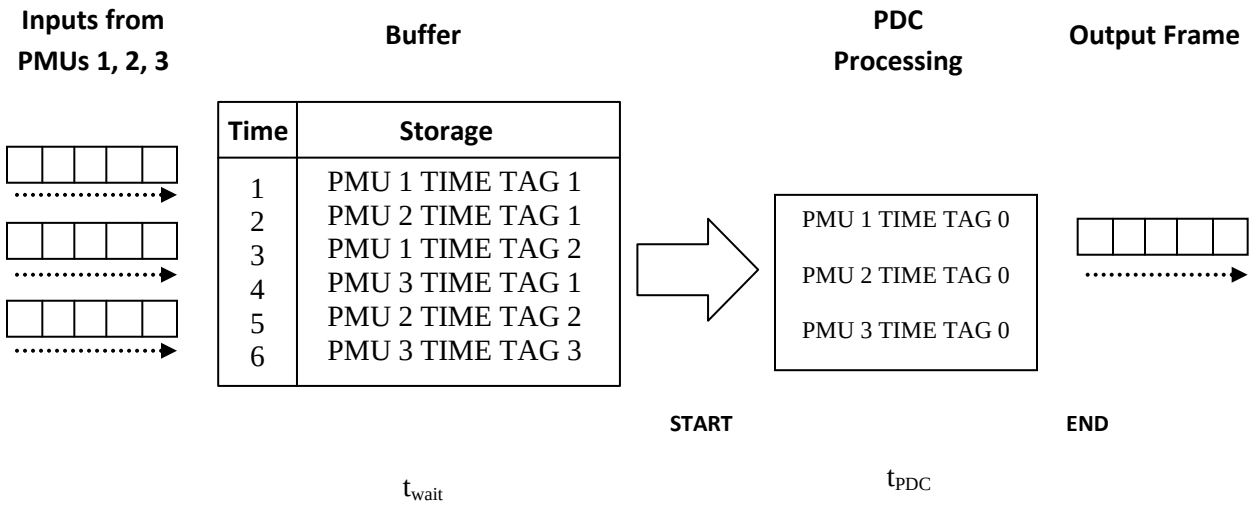

Figure 2.3: Phasor Data Concentrator Latency

2.3.2 Maximum Wait Time and Buffer Size

As described in the previous section, the PDC buffer is a temporary storage capacity for PMU data before it is processed into an output frame. The buffer size can be defined as the maximum length of time the PDC can store data before discarding the oldest frame. For example, if the time is currently *t =10* seconds and the buffer size is 5 seconds, then all data that arrived before *t - 5* seconds will be discarded if it has not been processed already. The buffer size is very important and is dependent upon the hardware limitations of the PDC. A very short buffer

size may result in continuous loss of data and a high buffer size can be quite expensive or incompatible with the hardware. The buffer size must be significantly larger than the PDC processing time so that new incoming data can be stored while the PDC is processing the output frame corresponding to the previous time tag. Taking specific application requirements into account, an ideal buffer size must be determined and set so that the PDCs can behave predictably.

The PDC wait time has been mentioned several times in the last few sections, it is an integral constant in determining PDC behavior. As described earlier, the PDC waits a maximum amount of time to receive an input from all PMUs for each individual time tag. If the PDC does not receive all PMU inputs, then it should label the missing PMU data block with a flag and substitute the data frame with filler measurements. If the PDC receives the missing PMU input after the maximum wait time, this frame is considered late and should be discarded. Ideally, a long wait time would accommodate all PMU input network delays so that few measurements are lost before reaching the PDC. However, a long wait time could result in significant delays for the data to reach a critical application. The wait time must also be much shorter than the buffer size since the buffer must store the data of all PMUs while the PDC is waiting. The maximum wait time should be set so that it can accommodate the PMU input that is usually received last under normal network delays. This setting should be higher than the average time difference between the latest arriving PMU input and the soonest arriving PMU input.

2.3.3 PMU Inputs in Incorrect Order

Up until this point, all PDC behaviour has been based on the assumption that it receives time synchronized PMU data frames in a consecutive order; so that the newest measurement of an individual PMU will always arrive after its predecessors. Consider a scenario in which this assumption doesn’t always hold true; what if an older measurement from PMU 1 arrives after its newest measurement? The PDC will receive the PMU data in the incorrect order; however, the input frames will still hold the correct time tag of their measurement time. The PDC must be capable of handling this type of unpredictable situation. If the PDC recognizes the mis-ordered input frames, there are several options for how it may respond. The simplest option would be to

accept the newest measurement and discard the late older measurement by treating it as a missing frame. Another option would be to store the complete set of data from the newer time tag, if possible under buffer limitations, and process the data from the late arriving older time tag. After the older set of data is outputted, the newer measurements can be released from the buffer and processed. A third possible option would be where the PDC may treat the newest measurement as invalid data and discard this data frame since it was expecting to receive the previous PMU input. The PDC will process the older measurement frame if it arrives before the maximum wait time limit. All of these options can become more complicated when multiple PMU sources are transmitting to the PDC. The PDC's response must be standardized, independent of the type of application(s) its output is being used for.

2.3.4 Up-sampling and Down-sampling of Data

The reporting rate of a PMU must be a sub-multiple of the power line system frequency. In a 60 Hz system, common reporting rates are 30 samples/second and 60 samples/second. There is no standard for the PDC reporting rate; however, it would be logical to follow similar report rates as PMUs. If a PDC has a different reporting rate than a PMU, the PDC would need to sample its incoming data to compensate for the discrepancy in rates. There are two types of sampling performed by the PDC, up-sampling and down-sampling. Up-sampling occurs when the PMU reports at a lower rate than the PDC and the PDC must increase or "up" the number of samples it receives to transmit at a higher rate. Down-sampling is the opposite situation where the PDC has a lower reporting rate than the PMU and the PDC must reduce or "down" the number of samples it transmits. The method with which the PDC samples its data must be complaint for its applications and it must be common amongst all utilities so that the received data can be validated.

A simple example of down-sampling would include a PMU input with a reporting rate of 60 samples/second sending to a PDC with an output rate of 30 samples/second. The PDC must re-sample the data and not just discard every other input frame. The data must be sampled in such a way that it prevents aliasing and distortion of data. The PDC must do the same if the rates

were reversed and the PDC reported at twice the input rate. Conversely, if the PDC reporting rate was not a simple sub-multiple of the PMU rate, a much more robust re-sampling method must be used.

If many PMUs were transferring data at multiple data rates to a single PDC, the sampling method becomes much more complicated. The PDC must be capable to handle the multiple data rates and not lose necessary phasor measurements. A sampling scheme must be developed to handle all the above scenarios and successfully be implemented in all types of PDCs.

2.3.5 Internet Protocol

PMUs communicate data using network protocols such as TCP/IP and UDP/IP. The differences between the two protocols were described in detail in section 2.1. One of the important differences, one that is most significant to phasor measurement applications is the amount of time the data requires for transmission using these protocols. UDP/IP is considerably faster than TCP/IP due to its lack of handshaking and reducing measurement time is one of the main objectives of wide area measurement systems. For this reason, UDP/IP should be selected as the industry standard.

2.3.6 Input Capacity

The input capacity of a PDC is defined as the number of PMU/PDC inputs it can successfully handle and process. Each input can contain a variable number of phasor measurements. The saturation point is the maximum number of inputs the PDC can handle before there is a significant change in latency/processing speed or loss of data. This capacity of determined based on the needs of the application and the PDC should be designed according to these requirements.

All the examples and PDC properties described in this section suggest new guidelines that should be created for the PDC and its applications. These guidelines will assist in

maintaining uniformity amongst power utilities and ISOs while taking advantage of synchrophasor technology.

Chapter 3

Development of Tests for Current Standards and Future Guidelines

The application of PDCs for wide area measurement requires minimum delays, precise alignment and time tagging of the reported data, and consistent performance across all units in a power transmissions system. These requirements demand a set of functional and performance tests to ensure that an installed PDC system can meet its desired objectives. The tests developed in this thesis were orchestrated at Virginia Polytechnic Institute's Power Systems Laboratory. The testing was conducted under the guidance and resources of Pennsylvania Jersey Maryland (PJM) Interconnection which is a regional transmission organization (RTO).

PDCs are part of PJM's three year project to implement a wide area monitoring system. To guarantee correct operation of this system, the individual PDCs must undertake various types of tests to ensure its reliable and accurate operation. A portion of the tests described here were developed according to the PJM requirements for minimum PDC performance and are partially based on the *Guide for Phasor Data Concentrator Requirements for Power System Protection, Control and Monitoring* [11].

3.1 Conformance Tests

The IEEE C37.118.1 and C37.118.2 standards define certain conformance requirements for any PMU data receiving device and the PDC must meet these specifications completely. Tests are developed to check how the PDC handles PMU data under normal operation and during measurement or communication contingencies. The requirements being tested to meet the standards include message framework and configuration, data aggregation and synchronization, data validation, data format and conversion, time quality and missing or late data.

3.1.1 Message Framework and Correct Configuration

The message framework includes the header, command, configuration and data frames. To meet basic requirements, the sequence of the frames and the appropriate response from the sending and receiving devices must be tested. The location and size of each field in their corresponding frames must be checked. The configuration frame must be correct so that it translates the information in the data frame accurately. A simple network of one PMU and one PDC is sufficient to test for this standard. The PMU must be configured to transmit data to the PDC using its IP address and port number. The PDC must be setup to receive the PMU input and also transmit an output after processing the data. Network data must be collected in real-time to verify the inputs and outputs from both, the PMU and PDC. The output frames must be decoded to synchrophasor format before checking the specific fields. If the transmitting device sends out a configuration frame in regular time intervals then its corresponding data frames can be checked. The commands received by the PMU and PDC should be tracked to ensure that the device is executing actions according to these messages. Examples of such command messages include, “Start Data Transmission” and “Request Configuration Frame”. Commands that are not understood by the receiving device such as an unidentified IDCODE should be ignored by the device.

3.1.2 Data Aggregation and Synchronization

The primary purpose of a PDC is to collect phasor measurements from multiple devices and time synchronize them before separately processing and transmitting the time tagged frames. Modifying the setup described above by adding multiple PMUs that will transmit data to a single PDC on the same network, the PDC can be tested for data aggregation and time synchronization. In a real communications network where each PDC does not have an exclusive channel to receive data, there may be errors during transmission that may cause changes in the message frames or even loss of packets. The data being sent from the PMUs may not always match what is being received at the PDC. To ensure that the PDC is processing the data correctly, we must know what data it is receiving and not hold it accountable for errors that occurred during

transmission from the source. This is why it is important for the incoming and outgoing network traffic at the PDC node to be analyzed instead of the outgoing data at the PMU nodes. In each PDC output, the time stamp can be found in the synchrophasor message; the time is presented as the second of century (SOC) and fraction of second (FRACSEC). The data blocks will display the input data that was processed by the PDC. All the PMU inputs with the same time stamp that were received by the PDC can be found and matched to the PDC data blocks as shown in Figure 3.1 below. The actual measurement information in the data blocks within the output frame can be compared to the corresponding input frames to verify that the correct packets were aggregated and time synchronized. The reporting rate of the PDC can also be verified by calculating the time difference between every consecutive FRACSEC value.

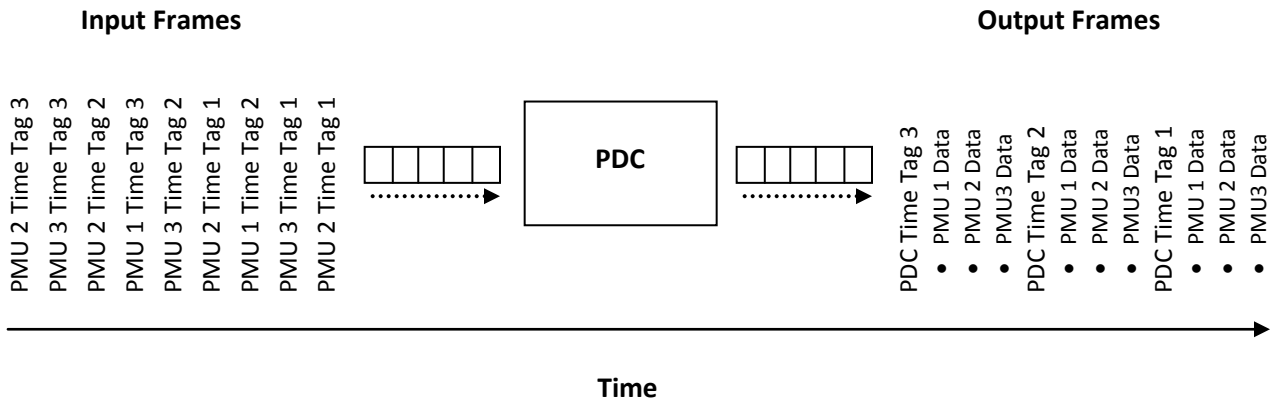

Figure 3.1: Data Frame Aggregation and Alignment

3.1.3 Data Validation

The PDC performs several functions while processing the phasor data it has received. One of the steps it applies to each input is data validation. This comprises of basic data verification and checking of data status flags, time quality of all PMUS and data integrity. Each measurement block within a PDC output frame has a status field in addition to the phasor measurements. The status uses flags to tell the user or the application the quality of the data it is

receiving. The flags include a single bit each for data validation, PMU error, time synchronization and data sorting technique. They also include two bits for time locked quality. Using a similar setup to the data aggregation test, multiple PMUs will be transmitting measurements to a single PDC. All PMUs will be time synchronized to UTC time through an external time source. Only the output frames from the PDC need to be captured for this test.

When data is arriving as expected by the PDC and all data is valid; assuming there are no problems with time synchronization these flags should be set to zero. However, when there are problems with the incoming PMU data, the PDC must reflect that correctly using these flags. The two main tests that can be performed to check data validity involve PMU time quality and missing data.

The first test, PMU time quality check, can easily be achieved by tracking the status field of one PMU data block. This PMU should start off under normal operation and then the time source must be quickly disconnected from it. The PMU data block should reflect a data invalid flag, a PMU error, a loss of time synchronization and unlocked time value to indicate the range of seconds the loss of sync was detected. The status field of other PMU data blocks should also be checked to ensure that their flags are not set due to one PMU error. The PMU that was disconnected from its time source should be reconnect during the same test and the status field should be checked to ensure that the flags are reset to default (non error) values.

The second test, which is for missing or late data, should be conducted using two types of test setups. One model involves only one PMU transmitting to a single PDC and the other setup has multiple PMUs transmitting to the PDC. In each type of test, only the output frames of the PDC are collected. Similar to the PMU time quality check, the status field is tracked during normal operation and all flags should be set to their default values. After a certain period, one of the PMUs in the multiple PMU test setup and the one PMU in the standalone PMU test setup should skip or delay a data frame by more than the wait time of the PDC. The status flags should change for the time stamp that was skipped or delayed. In either case, the PDC will treat this PMU input as missing data and flag the corresponding PMU block with invalid data and PMU error. However, there should still be phasor measurements in the data frame. These measurements are filler data and may even be set to zero by the PDC. All data frames sent after the missing frame should be processed correctly by the PDC and status flags should be set back

to their default values. The reason for having two separate test setups is to notice a difference, if any, between how the PDC handles late or missing data with one input compared to having multiple inputs. The PDC may not respond to both scenarios in the same manner. The PDC may not process an output frame at all for the missing PMU data while only receiving inputs from one device. If the frame is late (beyond the wait time) but not missing, the PDC may still process this input similar to normal operation, without any status flags, even though this response does not conform to the IEEE standards.

3.1.4 Data Format and Coordinate Conversion

The data format refers to the way the measurements are represented in the data frame such as fixed-integer or floating point. The coordinate refers to the type of measurement such as polar or rectangular form and these can be represented in either data format. The PMUs may send data in one type of format or multiple PMUs may send data in different formats; the PDC will specify the type of format it will use for its output depending on application needs. The output data must be collected and verified against the PMU data to ensure the correct format is used and the measurements are converted correctly. This test is also run using the two setups described under Data Validation to compare the PDCs response with a single input with that of multiple inputs.

3.2 Functional and Performance Tests

In addition to the conformance standards described in IEEE C37.118.1 and C37.118.2, there are other functional requirements that the PDC must always meet in order to operate properly. These requirements can be derived as an extension of the IEEE standards as they consider the PDC's response under non-ideal operation conditions. Also, depending upon the type of application the measurements are being used for, certain performance requirements must be established for the PDC. These performance requirements will be based on the application needs, the network the PDC is receiving data from and delivering data to, and the number of

inputs it is expected to process. Tests must be developed to examine the PDC's ability to handle these conditions.

3.2.1 Re-sampling Test

The PDC may receive data at different reporting rates than itself and it must re-sample this data correctly during processing. To cover all up-sampling and down-sampling scenarios, this function will be tested using four different configurations which can be separated into two equipment setups. This test is developed to meet an extension of the IEEE standards involving reporting rates and reporting times.

The first setup is the most simple as it includes one PMU and one PDC. During the up-sampling test the PMU will have a lower reporting rate than the PDC, and in the down-sampling test the PMU will have a higher reporting rate than the PDC. Common PMU/PDC reporting rate combinations are 30fps/60fps and 60fps/30fps. The data received by the PDC will be compared to the data sent out by the PDC. First, the actual reporting rate must be checked using the FRACSEC value in the output frame to ensure that the PDC is not skipping any outputs. The time difference between consecutive FRASEC values can be calculated, it should always be equivalent and equal to the inverse of the reporting rate (1/reporting rate). Secondly, depending upon the type of measurements the PMU is transmitting, such as steady state values or a ramp-up frequency, the output data should represent the best time-aligned estimate that minimizes any re-sampling error. The phasor measurements and frequency for every device should be compared.

In the second setup, multiple PMUs will be connected to one PDC and each of the two configurations will test for both up-sampling and down-sampling. Every PMU will have a unique reporting rate. To test whether the PDC re-samples in both directions, the PDC will have a reporting rate equal to approximately the average of the PMU reporting rates. In the first test, the PDC reporting rate will exactly match the reporting rate of one of the PMUs, the closest in value to the average PMU reporting rate. In the second test, the PDC reporting rate must not match the reporting rate of any of the PMUs but still be close to the average of the PMU reporting rates. In this setup, a minimum of 3 PMUs are required to provide the appropriate range of PMU

reporting rates. Similar to the previous setup, the data receiving by the PDC and transmitted by the PDC can be compared to test whether it was re-sampled correctly. As described earlier, the PDC reporting rate can be checked using the FRACSEC values.

For both re-sampling test setups, the following information can be checked for in the PDC data frame:

- The IEEE Standard C37.118.2 describes the STAT word bit 9 as a flag that data was modified but does not indicate the nature of this modification since a great number of possibilities exist. This bit alerts the user to the fact that modifications have been made and measurement characteristics of the PMU may not be exactly applicable. This bit shall be set to 1 when the PDC is required to re-sample data.
- The PDC may use an absent data tag, such as NaN, insertion in an up-converted data stream [7]. The test in this case could consist of verifying the existence of this tag, at the correct timestamps in the PDC output stream.

3.2.2 Latency Test

One of the most important requirements set by the application is the maximum latency of the PDC. Certain applications have critical timing requirements and the PDC must be configured to deliver accordingly. As described in chapter 2, there are many variables that affect the latency of the phasor data. Some of these variables cannot be accounted for by the PDC itself and therefore need to be predetermined while designing the communications network for the measurement data. The two delays that account for the PDC latency are the PDC wait time and PDC processing time. The wait time is the absolute length of time the PDC waits for its inputs at for specific time stamp before processing. The processing time is the length of time between receiving the last input and sending a data frame for a specific time tag. Both of these parameters need to be tested separately.

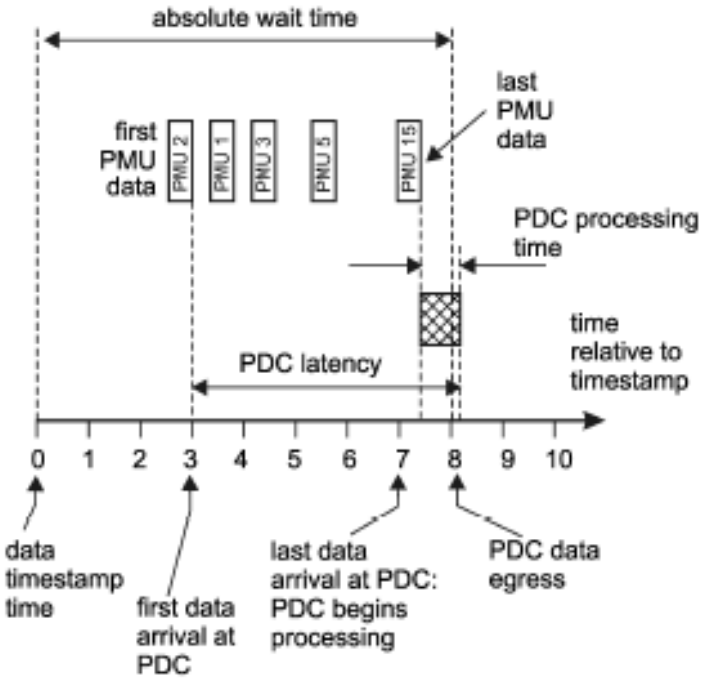

Figure 3.2: All Inputs within Wait Time [7]

In order to determine the processing latency, the PDC must receive all PMU inputs within the allocated wait time as shown in Figure 3.2 from IEEE PC37.244 D6.0 [7]. The setup for this experiment will include one PDC and several PMUs, up to the PDC capacity at equal reporting rates. The PDC receives data from one PMU for the first test. Gradually, the PDC will be configured to receive more PMUs inputs so that the change in latency due to the number of PMU inputs, if any, can be measured. In each data set, the packets arriving at the PDC and the packets leaving the PDC shall be captured. To calculate the processing time, the last arriving PMU input for a specific time tag must be found along with its corresponding output frame. This can be achieved by matching the SOC and FRACSEC time tag within both data frames. The time of capture of the last PMU frame can be subtracted from the time of capture of the PDC output; this gives you the processing time for that time stamp.

$$T_{process} = T_{output} - T_{last\ input} \quad (3.1)$$

It is a good measure to compute this latency using several sets of data for every test setup so that an average time and standard deviation can be found.

The wait time of the PDC can be tested using two different test setups. Both setups would collect data at the PDC node, similar to the PDC processing latency test. One setup would have a single PMU transmitting data to a PDC with regular delays. After a set time frame, the PMU must be forced to delay itself by more than the wait time of the PDC. This can also be achieved by disconnected the PMU from the PDC completely. In another setup, multiple PMUs will be transmitting to the PDC with regular delays, and after a set time frame only one PMU will be delayed by greater than the PDC wait time as shown in Figure 3.3 [7]. The hashed PMU block shows where the data frame was expected to be received by the PDC. The reason for testing with one PMU and with multiple PMUs is to see whether the PDC behaves in the same way in both setups. In the case of multiple PMUs, the PDC will still process the remaining PMU inputs that have not been delayed. But with only one PMU input, the PDC may respond differently.

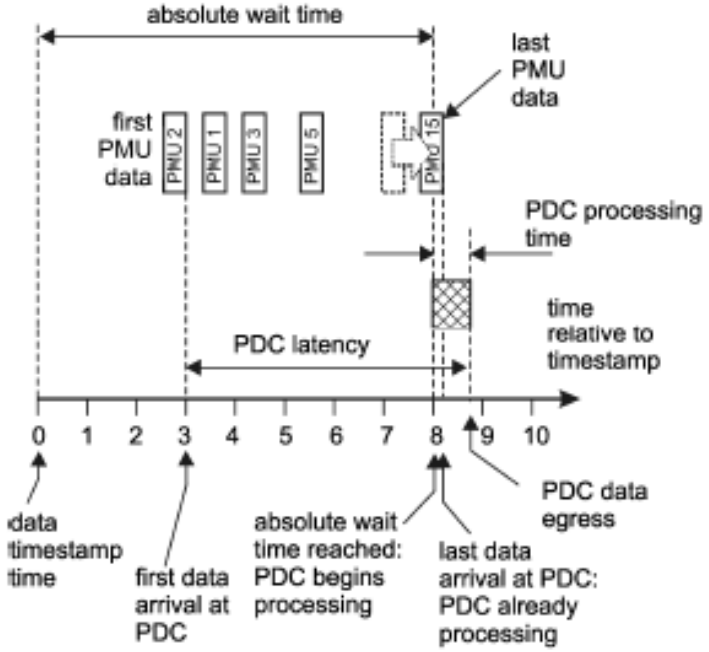

Figure 4.3: Maximum Wait Time [7]

The PDC frames that were processed when the PMU input was delayed or disconnected should be reviewed. The PDC should show an error in the missing/late PMU data block as explained under the conformance test. The total latency of the PDC can be calculated using a similar method to what was described in calculating the processing latency. However, instead of using the receiving time of the last PMU input, the receiving time of the first PMU input must be used in the calculation. The wait time can be approximated by subtracting the PDC processing time calculated in the previous test from the total latency of the PDC:

$$T_{wait} = T_{total} - T_{process} \quad (3.2)$$

In order for the PDC to operate successfully, it must process data at a rate that is higher than the data arrival rate. The latency of the PDC is significantly affected by the wait time and the frequency of the late/missing data. PDCs and their networks should be configured by minimizing its latency range which is a function of the PDC wait time, reporting rate and buffer size. Complete testing should consider the delays introduced by the added functionalities of a PDC such as data translations, reporting rate changes and data wait functions.

3.2.3 Capacity Test

The capacity of the PDC is the maximum number of inputs it can process without significant changes in its latency. The capacity test is performed similarly to the PDC processing latency test. The latency is calculated while the PDC is receiving data from the maximum number of PMUs it is capable of processing. The latency can be compared to the results from the PDC processing test to determine the impact from the increase of inputs. A few more PMUs (more than the prescribed capacity) can be added to the network to witness the PDCs response when it has exceeded its capacity. The capacity is also affected by the number of measurements that need to be processed, such as the number of phasors and analog measurements within each PMU input. The number of applications the PDC must create unique output streams for also affects the PDC's processing power.

3.2.4 Buffer Limits

The buffer is the storage capacity of the PDC. The length of time the PDC can store input data can be referred to as the buffer limit. In order to test the buffer limit, the PDC must be configured to hold data that arrived at a time equal to the current time minus the buffer size as defined in this equation:

$$T_{data} = T_{current} - \text{Buffer Size} \quad (3.3)$$

It is difficult to overload the buffer in a test environment since it would require many delayed PMUs with different time tags in a short period of time. This wouldn't be easy to achieve using even a simulated multiple PMU system since the timing of all the data is tedious to keep track of. Knowing the timing of the received data is quite crucial in this test. Alternatively, this test can be performed quite simply on PDCs that allow the user to change its wait time. The wait time should be set to a value higher than the buffer size. The test setup is similar to that of the latency tests and the two configurations include the PMU delay to be slightly less than the buffer size and the PMU delay to be slightly above the buffer size. Both configurations will be run using at least two PMUs of which one PMU will be delayed during the test. The time difference between the first PMU and the delayed PMU will be calculated by matching their FRACSEC values in their received frames and subtracting their received times. The PDC output is checked to verify whether the PDC processed the delayed input that was slightly below the buffer size and whether it discarded the delayed input that was slightly higher than the buffer size. When the PDC processes the largest acceptable delayed input, the buffer size can be determined using:

$$\text{Buffer Size} = T_{\text{delayed PMU}} - T_{\text{first PMU}} \quad (3.4)$$

Another method that can be used to calculate this limit is to perform a PDC latency calculation, specifically, the wait time calculation. The total latency is determined using the difference between the arrival of the first PMU and the PDC output and the PDC processing delay is subtracted from the total latency to find the wait time, which in this case is approximately equal to the buffer size.

3.2.5 Irregular Order of Packets

All tests developed so far have been under the assumption that the PDC receives packets in a consecutive order from each PMU whether they are delayed or not. To add further clarity, all tests so far assume that when a PMU packet has been delayed, all following packets will arrive only after the delayed packet has been received. In reality, this is not always the true. There are conditions where packets from a single PMU can arrive in a non-consecutive order. PMUs always process its measurements in order of the time they are taken, however; once these packets are transmitted, the network topology and parameters can reorder the data frames before they arrive at the PDC.

One such type of packet reordering can occur when there are multiple channels for data transmission between the PMU node and the PDC node. In communication networks, the packet will always choose the path with the shortest transmission delay. If the path with the shortest delay is no longer available, the data will travel along the path with the shortest delay from the channels available. Using these conditions, a simple network is developed to reorder the packets received by the PDC.

The network includes two PMUs, each in its own isolated section that connects it to one common PDC. One of the PMUs will only have one path to transmit data to the PDC and this PMU will always operate under normal conditions. The other PMU will have two possible routes to the PDC, one is a straight forward minimum delay path and the other is a complex route with considerable network delay. This complex path can be assumed to go along a much longer distance through many routers and switches that have traffic due to large amounts of other data traveling through. This complete network is illustrated in Figure 3.5 below. The delay along both paths should be measured before running this test. The test procedure it as follows:

1) Path 1, between PMU 1 and PDC, is connected and data is transmitted with minimum delay. Path 2 and 3, direct path and complex path, respectively, between PMU 2 and PDC, are connected and data is transmitted with minimum delay through path 2. Wait set time, t.
2) Disconnect path 2, forcing data to travel along path 3. Path 1 is still active. Wait set time, t.

3) Reconnect path 2, allowing data to stop using path 3. Path 1 is still active.

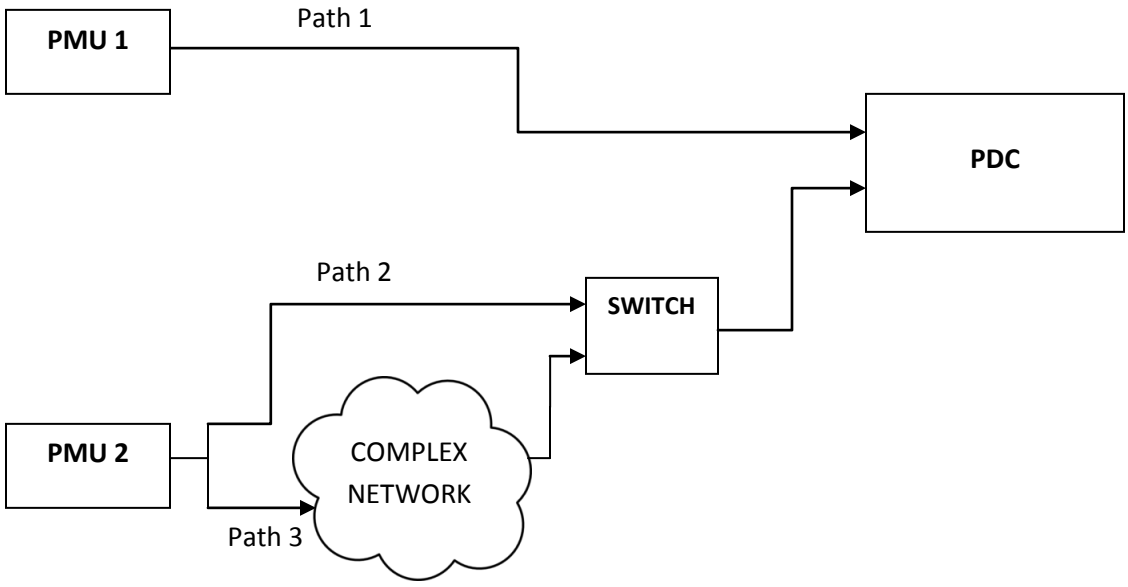

Figure 3.5: Reordered Packets - Test Network

Corresponding to the steps given above, a description of the location of the data frames in each step is as follows:

1) This step will produce results similar to the prior tests that used multiple PMUs transmitting data to one PDC under normal operation.
2) During this step, the packet that was traveling along path 2 is lost when path 2 is disconnected. Packets sent after the disconnection will arrive via path 3.
3) In this step, path 2 is reconnected. Packets that were transmitted along path 3 before the reconnect are still traveling along path 3. A few packets that were created after the reconnect will arrive at the PDC before the previous packets arrive via path 3. This is how the data frames become reordered before arriving at the PDC.

The data leaving the PMUs should be verified against the data the arriving at the PDC at all steps in this test. The data leaving the PDC must also be examined for this test. After the

reordering of the data frames is checked by comparing the outputs from the PMU with the inputs at the PDC, the PDC's response to the reordering can be analyzed. The time stamps and the measurements of the consecutive data and the reordered data should be reviewed properly at all nodes on this network to determine whether the PDC processes the reordered data correctly. The PDC may respond differently if the reordered data arrives within the wait time, after the wait time or even after the buffer length. The delay along path 3 should be varied in the separate test cases to compare the three responses. The frame order and timing of the three cases are shown in Figure 3.6 below.

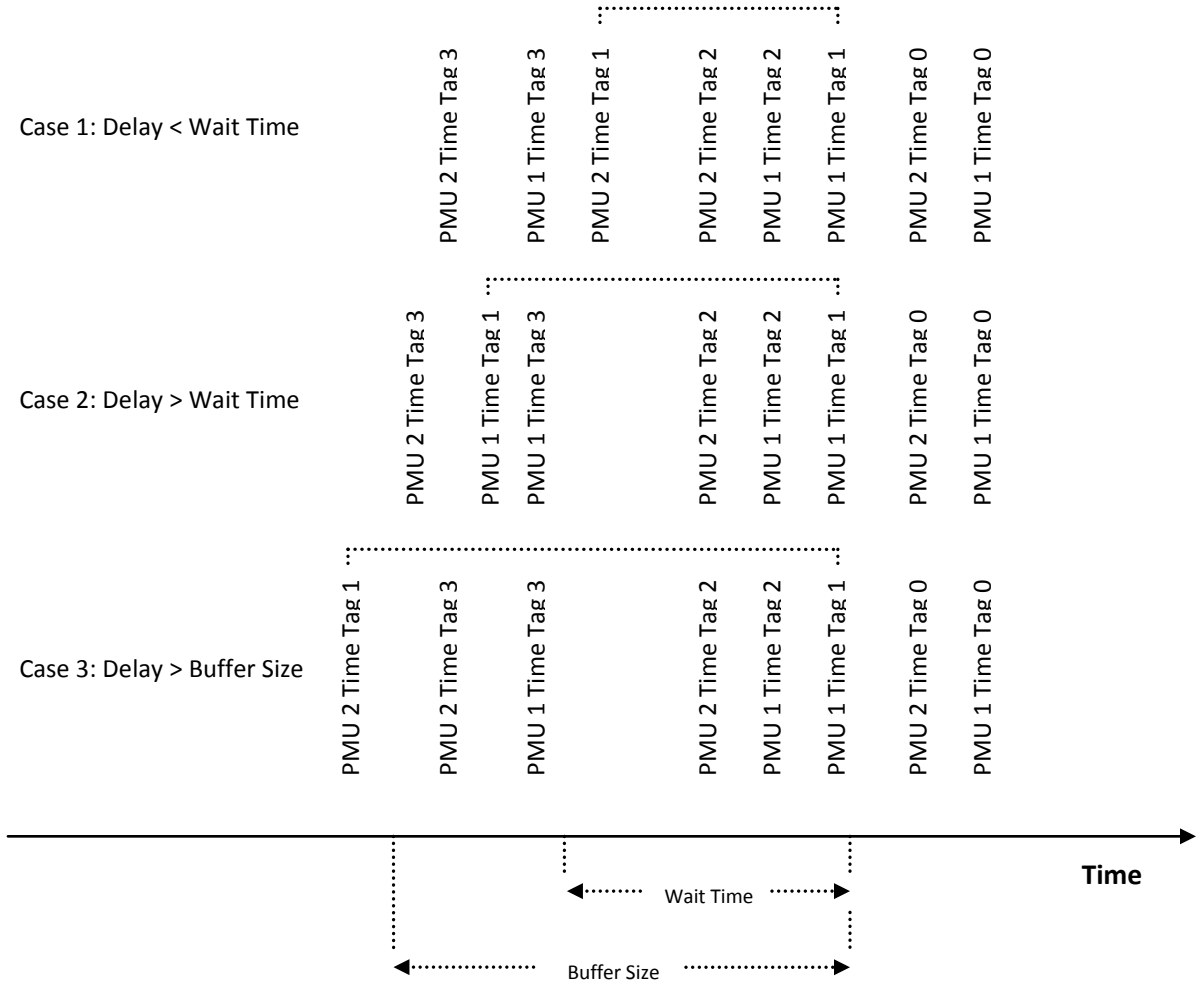

Figure 3.6: Reordered Data Frames. 1) Delay < Wait Time. 2) Delay > Wait Time. 3) Delay > Buffer Size

3.2.6 Communication Protocol Delay

The reasons for choosing UDP/IP instead of TCP/IP as the communication protocol between PMUs and PDCs have been mentioned in the previous chapters. To show the actual difference in transmission delay between the two protocols, calculate the latency of the network using TCP/IP and compare it with the latency using UDP/IP. This latency can be determined using one PMU and one PDC in an exclusive network. The time stamps in the PMU output frames can be matched with the time stamps in the PDC output frames and the difference in their corresponding system times can be computed to determine the network latency plus the PDC processing time as seen in equation (7). Since the PDC processing time can be assumed to be equal whether using TCP/IP or UDP/IP, the processing time can be ignored. The difference in total latency between the two protocols is the additional delay associated with TCP/IP transmission as shown in equation (8).

$$T_{TCP\ Total\ Latency} = T_{TCP\ Network\ Delay} + T_{PDC\ Proc} \tag{3.5}$$

$$T_{UDP\ Total\ Latency} = T_{UDP\ Network\ Delay} + T_{PDC\ Proc}$$

$$T_{TCP\ Delay} = T_{TCP\ Network\ Delay} - T_{UDP\ Network\ Delay}$$

$$T_{TCP\ Delay} = T_{TCP\ Total\ Latency} - T_{UDP\ Total\ Latency} \tag{3.6}$$

Chapter 4

Implementation of Tests for PDC

4.1 Test Systems

There are no preset systems or devices tailored for specific PDC testing. To perform the required tests, existing software and hardware need to be modified or enhanced to obtain the desired information. The equipment used at Virginia Tech for testing PDCs includes PMUs, enhanced PMU simulators, GPS antenna, PMU and PDC connection tools and a packet analyzer. MATLAB software is used to analyze the test data and for simulation purposes. The PDCs tested are not compliant to C37.118.2-2011 and C37.118.1-2011, but only to PJM Interconnection's security compliance and C37.118.1-2005.

The test set up includes a hardware-based or software based PDC that receives inputs from real PMUs or a PMU simulator through an Ethernet network. The PMUs are time synchronized to a GPS receiver using an IRIG-B connection. The PDCs are also time synchronized. The PMU and PDC data is collected using a packet analyzer at the receiving system. The primary goal of this set up is to observe if the device under test (DUT) is successfully able to process and align the data coming from the PMU(s) and output the time-aligned data to the user, which in this case is the data analyzer, without any errors.

4.1.1 Testing Hardware-based PDC

A hardware-based PDC is a complete hardware package where its sole function is to operate as a PDC. There is no software-based user interface on the device and all inputs/outputs and parameters must be configured using a PDC connection GUI installed on a computer that shares a network with the PDC. This PDC is time synchronized in the same manner as the real PMUs using an IRIG-B connection. The data from the PMUs and the PDC is transmitted to a receiving system that collects network traffic using a packet analyzer. Often, the receiving

system runs the PDC connection GUI along with the packet analyzer. If real PMUs are used in the test, then the PMU connection tool is also on the receiving system. This test system is shown in Figure 4.1 below.

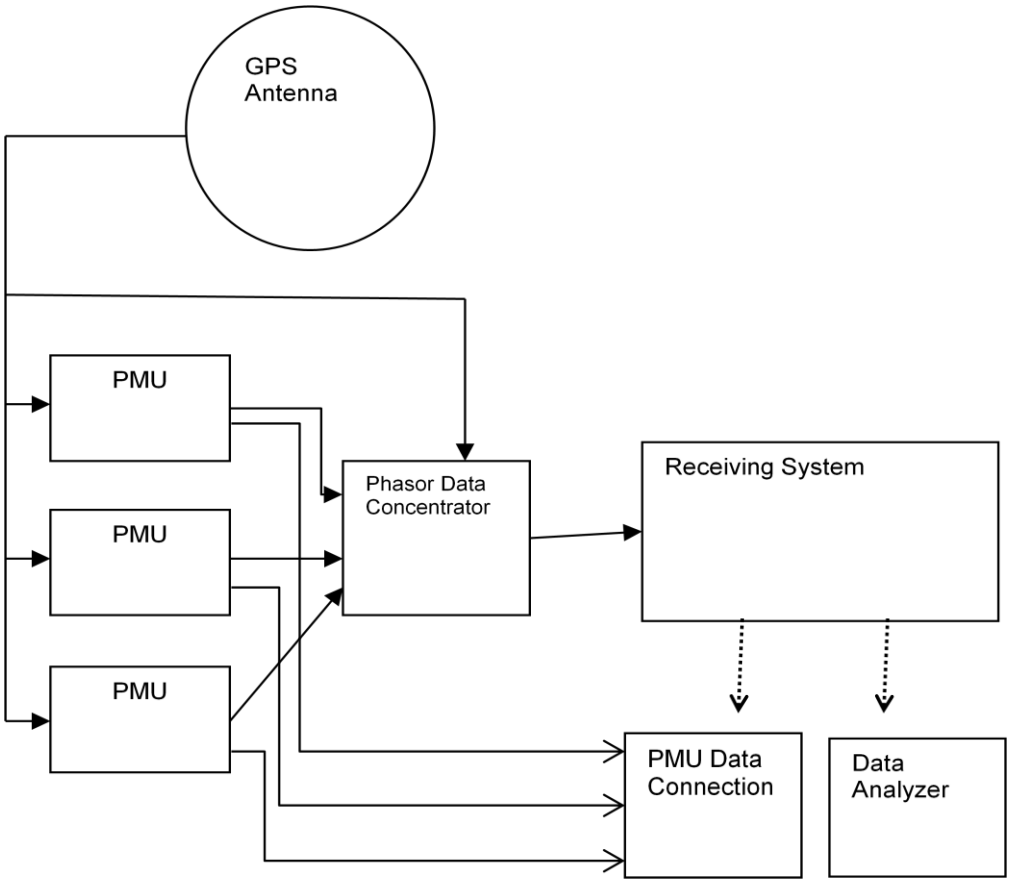

Figure 4.1: Hardware-based PDC Test System

4.1.2 Testing Software-based PDC

A software-based PDC is a software package that has the functionality of a PDC and customers may choose the hardware configuration that suits the needs of their application. The user interface is installed directly on the hardware that runs the PDC and all configurations can be made directly through this interface. This type of PDC has an internal system clock that is

periodically synchronized to the GPS clock through the server. This software package is installed on an operating system that can also be used as the receiving system. Combining the PDC with the receiving system eliminates any network delay between the two functionalities. This method also removes the need for a PMU connection tool since the PMUs are already transmitting data to the PDC at this node. This test system is shown in Figure 4.2 below.

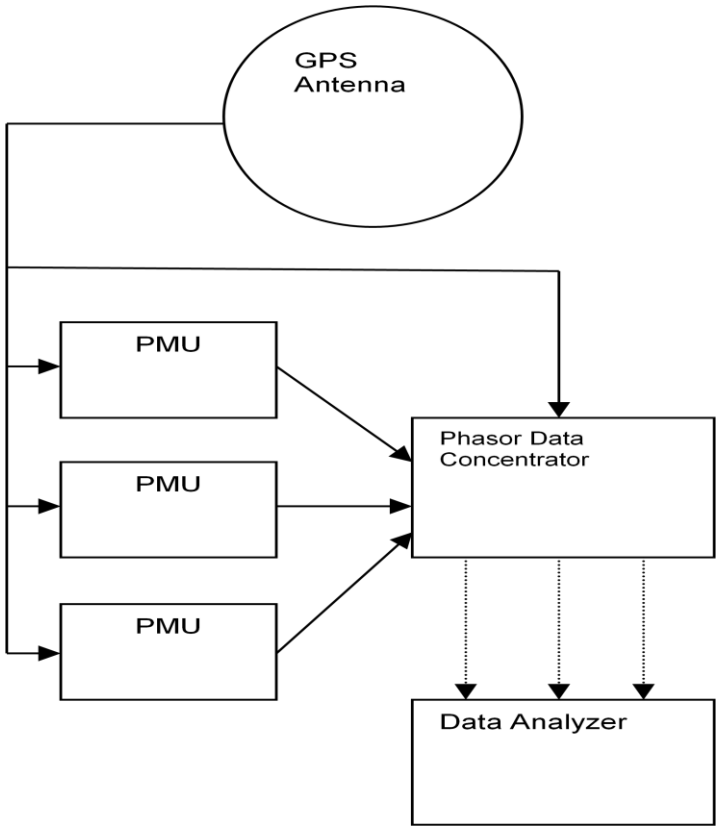

Figure 4.2: Software-based PDC Test System

4.2 Test Equipment

The specific equipment used for PDC testing is described in this section. The PDCs and real PMUs are configured and provided by PJM Interconnection. All other equipment is either open source or provided by Virginia Tech's Power Systems Laboratory. The tests conducted in

this thesis are chosen based on the capabilities and limitations of the devices being tested and the equipment used to aid the testing.

4.2.1 Phasor Data Concentrators

The two types of phasor data concentrators being tested are either hardware-based or software-based. The hardware-based PDC is developed by Schweitzer Engineering Laboratories (SEL), model 3373 shown in Figure 4.3. It is capable of accepting and processing 32 PMU inputs through its ports. Its general user interface and connection tool is through a software program, SEL PDC Assistant, shown in Figure 4.4. This GUI allows the user to configure the inputs and outputs of the PDC and check the status of incoming and outgoing data in real-time. The IP address and port number of each device communicating with the PDC must be supplied and the type of IP protocol must be selected. The reporting rate of the PDC is also selected.

Figure 4.3: SEL-3373. Hardware-based PDC [12]**; Schweitzer Engineering Laboratories, "SEL-3373," 2013. [Online]. Available: https://www.selinc.com/SEL-3373/. [Accessed 15 April 2013]. Used under fair use, 2013**

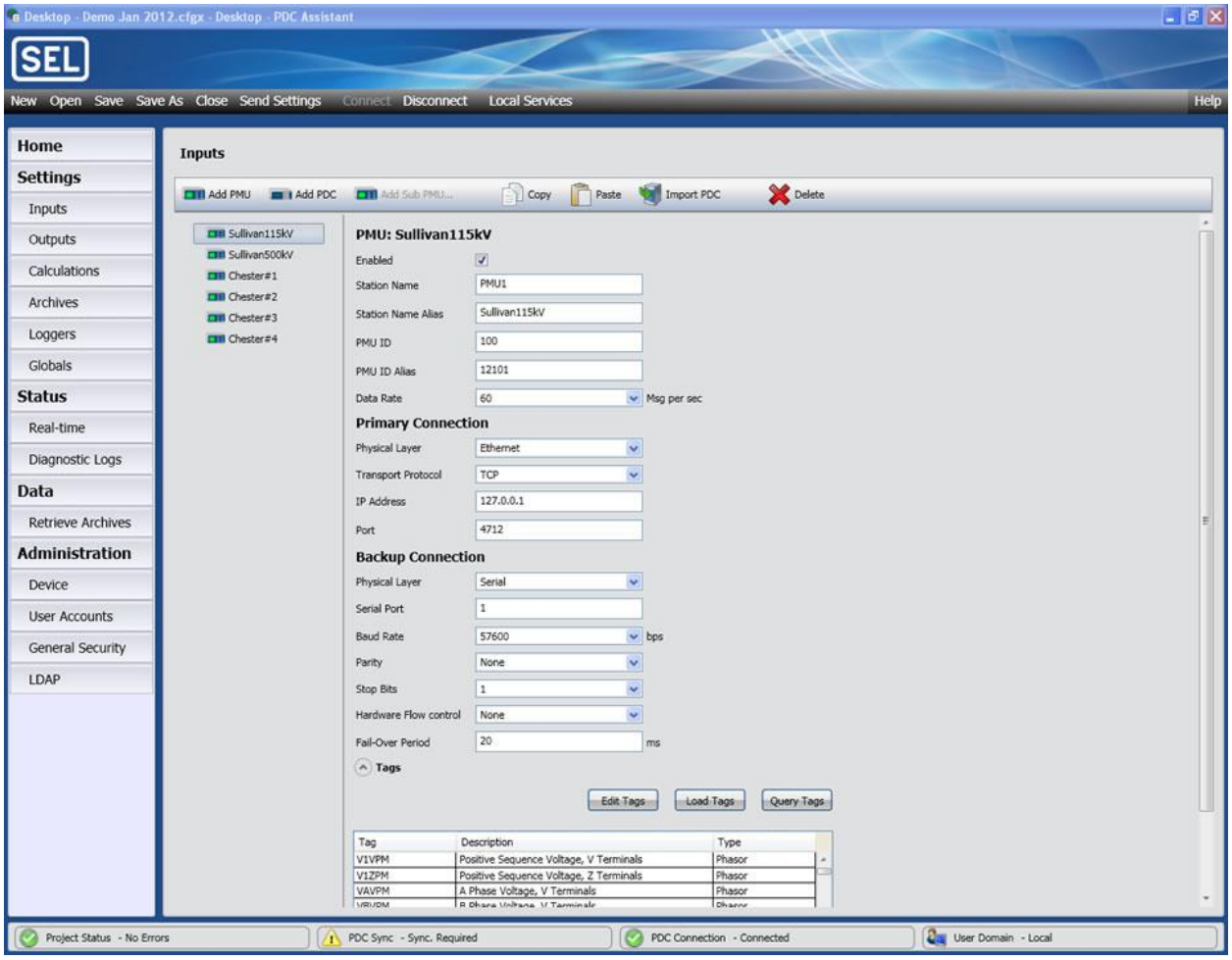

Figure 4.4: SEL PDC Assistant [12]

The software-based PDC being tested is the ePDC shown in Figure 4.5. It is developed by Electric Power Group (EPG) and is packaged according to PJM Interconnection's requirements. It is an open, platform independent software system that can be installed on Windows or UNIX/Linux systems. It supports additional functionalities compared to a traditional hardware-based PDC. These include settings such as progressive padding, data sort method, buffer size, wait time and configuration frame interval to name a few. The input and output parameters along with the additional functionalities can be accessed through the GUI shown in Figure 4.6 below.

Figure 4.5: ePDC. Software-based PDC [13]**; Electric Power Group, "Phasor Data Concentrators - ePDC & eSPDC," 2010. [Online]. Available: http://www.electricpowergroup.com/solutions/epdc/index.html. [Accessed 15 April 2013]. Used under fair use, 2013**

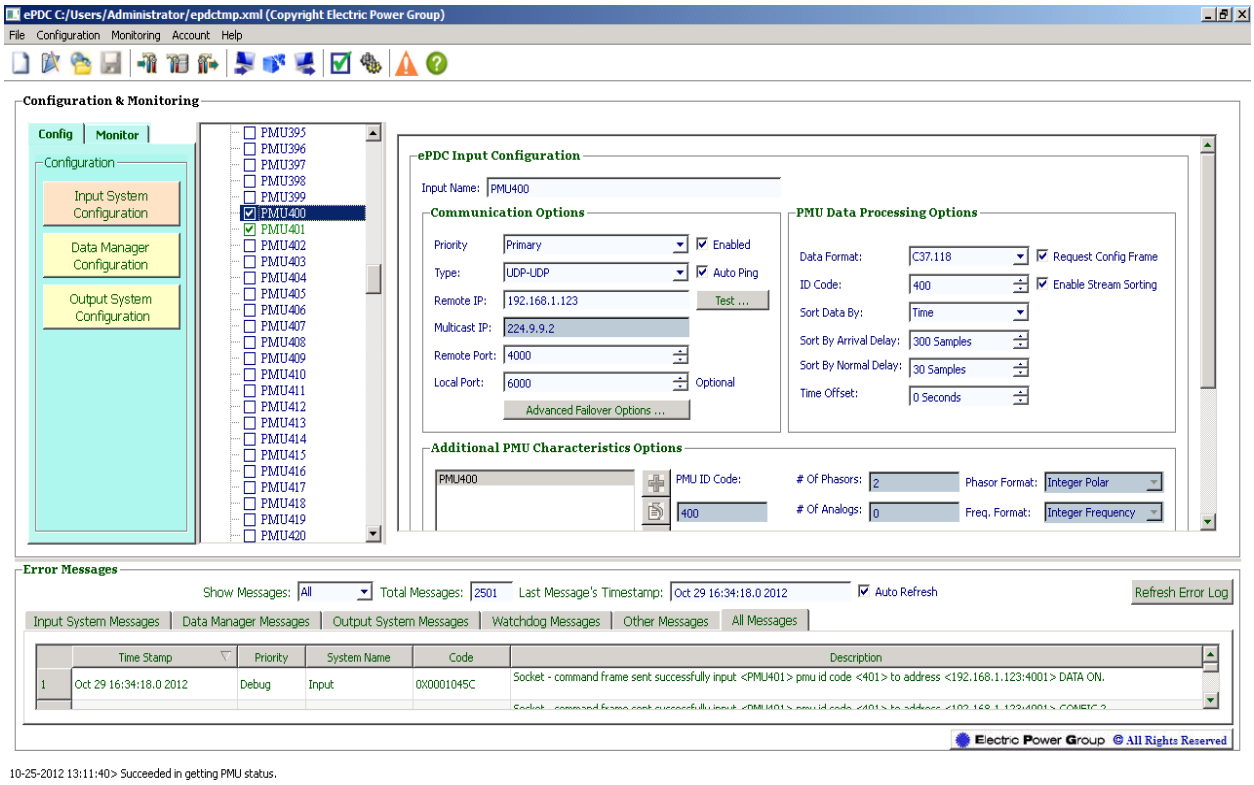

Figure 4.6: ePDC General User Interface [13]

4.2.2 Phasor Measurement Units

The Phasor Measurement Units used to test the PDCs are of two types. There are real PMUs which are developed by SEL, specifically model 421, shown in Figure 4.7 and there is a PMU simulator, called PMUSim (shown in Figure 4.8), developed by iPDC which is a software-based open source PMU and PDC developer.

Figure 4.7: SEL 421. Phasor Measurement Unit [14]**; Schweitzer Engineering Laboratories, "SEL-421," 2013. [Online]. Available: https://www.selinc.com/SEL-421/. [Accessed 15 April 2013]. Used under fair use, 2013.**

Figure 4.8: PMUSimulator by iPDC [15]

The SEL PMUs can capture real-time data to send to the PDC. Their destination port numbers and reporting rates must be preconfigured prior to testing using tools such as Accelerator or Hyper Terminal. Each unit is limited to two output streams through its ports. Each measurement is taken at a synchronized sampling pulse and time tagged accordingly. These PMUs provide a realistic measure of the capabilities of an actual PMU used in industry and limit the configurable options available through a simulated PMU. Most tests are performed using real

PMUs and simulated PMUs unless programmable modifications are necessary, in which case, only simulated PMUs are used.

The PMUSim is used in all test cases due to its flexibility and ease of use. Version 1.3.1 of this open source software has been used for the majority of the tests. The program is written in the C language for use only on Linux based platforms. The main source code that creates the PMU message framework according to IEEE Standard C37.118-2005 and controls the transmission of the data is stored in file ServerFunction.c. All modifications made to the PMUSim for testing purposes have been in this file. A few simple changes were made to the source code to adapt to the newer standard, C37.118-2-2011, such as handling a request for CONFIG frame 3. Some corrections were also added to the source code for UDP/IP communication. The PMUSim is not continuously synchronized to the GPS clock as a real PMU. It initially reads the system time which is synchronized to the GPS server and then it creates and sends out data at a precise time interval. It calculates this period using the inverse of the reporting rate:

$$Reporting\ Period = \frac{1}{Reporting\ Rate} \tag{4.1}$$

However, the reporting period of rates 60 fps and 30 fps are never ending recurring decimals and the PMUSim must round this value to determine the timing of its data frames. The simulator does not take into account the time it takes to generate a data frame, so the actual data period is higher than the inverse of the reporting rate. The newest version of this software applies a correction to the frame rate error.

There are many PDC tests that can be accomplished using a simulated PMU. The number of PMUs that can be run at one time is only limited by the processing power of the computer/operating system and not by the software. This simplifies the PDC capacity test, especially for the ePDC which has a capacity of over 100 PMUs. Another advantage of using the PMUSim is that the source code can be manipulated to simulate irregular PMU or network behaviour. One such example would be to delay one or many consecutive packets in a data stream by applying a fixed latency in microseconds. The delay could also be applied periodically so that one or more packets in every new Second of Century (SOC) can be set behind. Instead of a delay, the packet would be skipped altogether to simulate a missing data. These changes are

made in the UDP_PMU() and the TCP_PMU() functions in the ServerFunction.c file that can be found in appendix B.1.

A more advanced utilization of the PMUSimulator would be to use it as a communication emulator for the PMU data. A communication emulator can model scenarios with network issues such as noise in communication channels, loss of incoming or outgoing communication, etc. One specific test that can be simplified using an emulator is the Irregular Packet Order Test (section 3.2.5). Instead of physically creating a network with a large delay that can be difficult to accurately measure or control, this delay can be simulated in the source code using the following logic:

- Create and send data frames at the normal reporting rate
- Create data frames of PMU data that is delayed on path 3 but do not send it. Store the time stamp and measurements.
- Continue creating and sending data frames to simulate the reconnection of path 2
- Send out the delayed data that was stored using its original time stamps
- Continue creating and sending data frames according to the current time stamp

Step 2 (from 3.2.5) of this network issue involves a loss of packets due to the disconnection of path 2 and can be analyzed as a separate problem altogether. It is not necessary to simulate it here because it has already been addressed in the missing data test. The modification of the function SEND_DATA() using the above logic can be found in appendix B.2.

4.2.3 Connection Tools

Certain connection tools are used to receive data on a separate computer while testing the SEL 3373 PDC. The PDC Assistant which is a GUI for the SEL 3373 is a type of connection tool since it directs the PDC output to its destination IP address and port. It allows the tester to configure the output to be sent to the receiving system.

The tool used to allow the PMU to transmit data to the receiving system is PMU Connection Tester. This PMU/PDC connection tool is also used for capturing and validating data coming from a PDC under test. This tool may be connected to the output stream of the device under test, and will interact with the device using commands to control the output data and configuration information. The PMU/PDC connection tools are expected to be used mainly in offline and laboratory test setups. The PMU connection tester was a critical tool used to collect PMU data to compare with the PDC output for latency testing. The time of collection of the PMU data at the receiving system was used as the time the PDC received the same data. This assumption ignored the difference in negligible network delay between PMU-PDC and PMU-Receiver. After further testing and refining of testing techniques, it was determined that the PMU Connection Tester actually added its own significant delay to the arriving PMU data. During certain tests, the PDC output was being received sooner than the PMU data. This caused the latency testing of SEL 3373 to be quite inaccurate and a new method must be developed to determine the arrival time of the PMU data.

4.2.4 Time Source

The time source of all PMUs and PDCs is the GPS clock. The real PMUs and hardware-based PDC are always synchronized to the Arbiter 1094B GPS Substation Clock through an IRIG-B connection. Since the software-based PDC and the PMU Simulator are installed on a computer, they do not have IRIG-B capabilities. Instead, these devices are periodically synchronized (every 5 minutes) using the Network Time Protocol (NTP) server to receive GPS time.

4.2.5 Data Analyzer

The receiving system in any configuration must have a data analyzer to collect packets transmitted by the PDC and PMUs. The open source packet analyzer, Wireshark, is used for all tests that required data capture. This software program collects all network traffic at the node on which it is run. The data captured can be filtered according to many parameters such as the source port number using TCP/IP. The data can be reordered and viewed in many different configurations; it can also be saved in text file format. The captured data can be decoded to synchrophasor format so that it is user readable. The key information in the captured data that is used for analysis is the arrival time of the data, the decoded time stamp and the measurement data block. Figures of the captured data can be seen in the Test Result section of this chapter.

Another value tool used for data analysis is the technical computing language, MATLAB, created by MathWorks. The MATLAB code created for this project was to read the text file output from Wireshark and calculate the latency of the system. The code has been modified many times according to the type of latency being determined, such as wait time, PDC processing time, PDC total latency, or even difference in arrival time of PMU packets. The original code (appendix C.1) used a fixed base PMU as the last arriving PMU to determine the PDC processing time. The base PMU had to be determined prior to the latency calculation using another MATLAB code (appendix C.2). However, the updated code determines the last arriving PMU data for every time stamp; this provides a much more accurate latency calculation.

One other program developed using MATLAB was to demonstrate the effects of the wait time and buffer capacity on the PDC latency and to prevent loss of data. The program (appendix C.4) uses a highly simplified algorithm to simulate the incoming and outgoing packets at the PDC. The program introduces a period event which can be translated as the PDC waiting a maximum wait time at regular intervals. The algorithm uses the round robin method described in communications literature to simulate packets in a queue (buffer) before being serviced (PDC processing) by the round robin server. The service time is fixed as the PDC processing time and the packet arrival rate is the PMU reporting rate.

An example input to this program is:

- One PMU input at a data rate of 60 fps
- PDC waits maximum wait time at periodic intervals of 640 ms due to late or missing input
- Predicted PDC processing time is 11 ms
- Simulation time is 12000 ms

The packet arrival curve and departure curve is plotted in Figure 4.9 below and the wait time of all packets in the buffer is plotted in Figure 4.10. The maximum time a packet must wait in the buffer is 150 ms. As long as the arrival and departure curves have an equal slope and the wait time does not continuously increase, there will be no data loss at the PDC due to buffer limits.

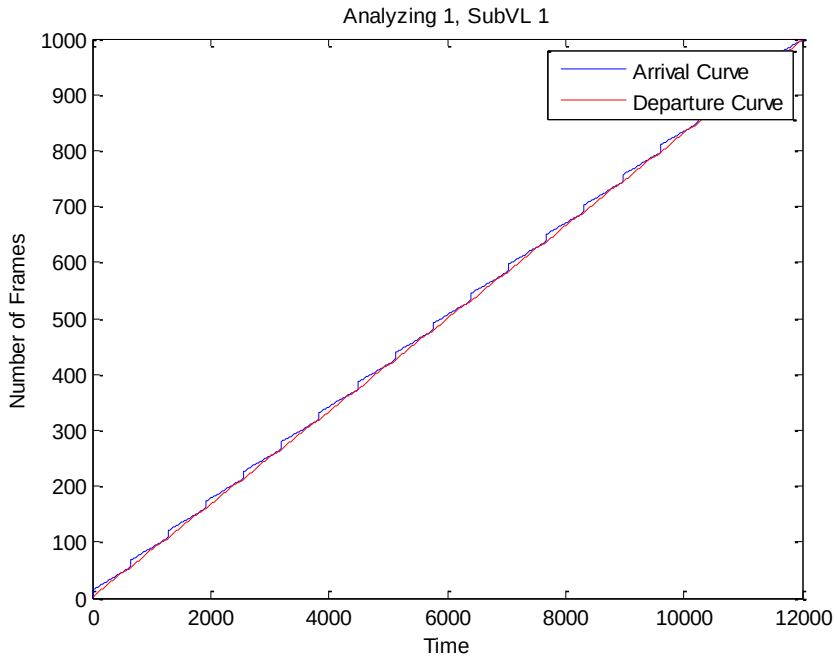

Figure 4.9: Packet Arrival and Departure Curves at PDC

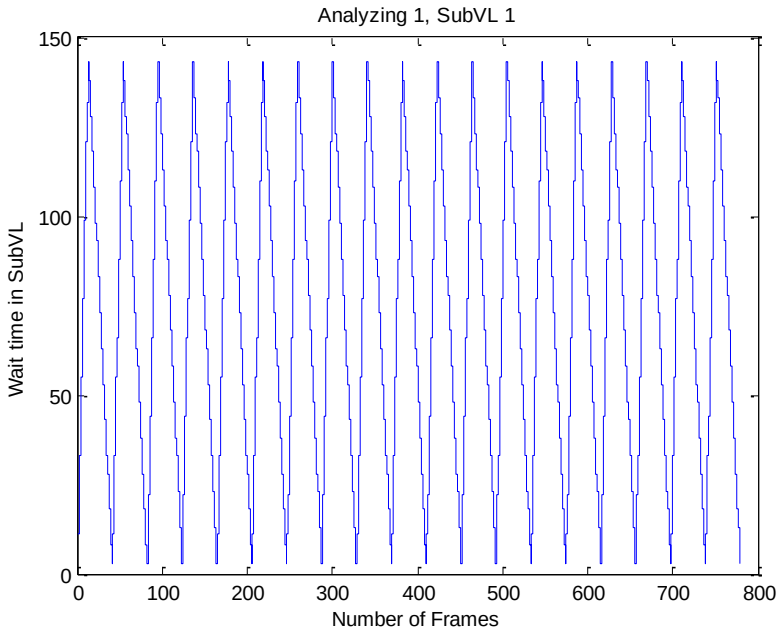

Figure 4.10: Packet Wait Time in PDC Buffer

4.3 Test Results

The tests developed at Virginia Tech's Power Lab and described in section 3 are conducted on the SEL 3373 PDC and the ePDC. The results of these tests are shown here.

4.3.1 Message Framework and Data Aggregation

Under the conformance tests developed in section 3, both units, SEL 3373 and ePDC, passed the message framework and data aggregation requirements. The PDCs process the time tagged inputs and output a data frame according to IEEE C37.118 specifications. A sample PDC output frame can be seen in Figure 4.11 below.

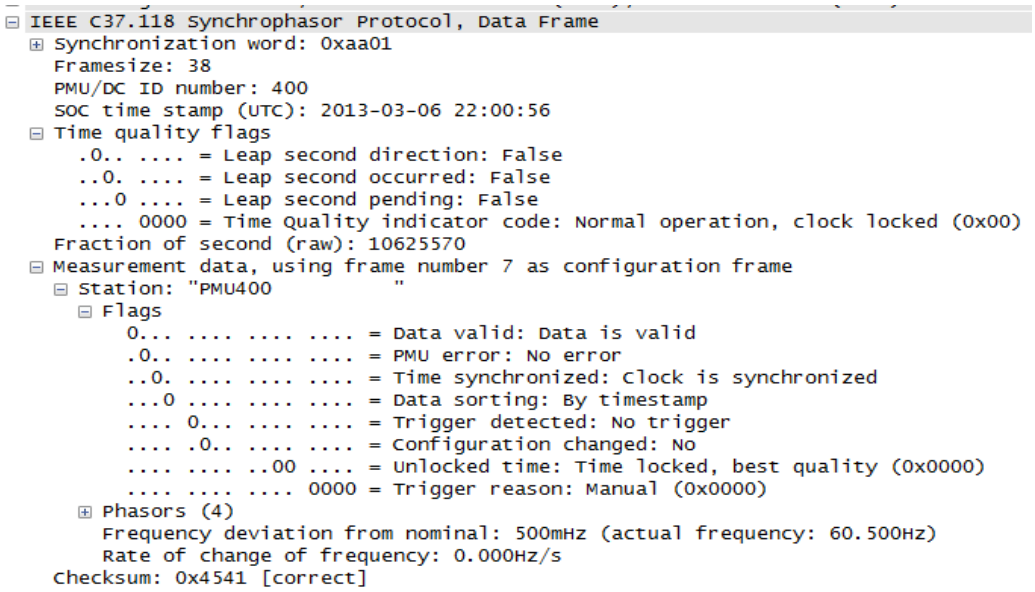

Figure 4.11: PDC Data Frame

The only part of this test that was failed by the ePDC was delivering a configuration frame in regular intervals. The ePDC only produces a configuration frame at the start of data transmission and even though its GUI allows the user to select the interval time for configuration frames, the ePDC does not actually output them. However, this requirement is not explicitly stated in the IEEE Standard but would be a common prerequisite while transmitted data to an application.

4.3.2 Data Validation Results

The Data Validation Test requires the PDC to set the Status word in every PMU data block to reflect the condition of the PMU input. When there is a missing or late frame arriving at the PDC when it is receiving multiple PMU inputs, the status word flags the data as invalid with a PMU error and time synchronization lost as shown in Figure 4.12. If the PDC is only receiving data from one PMU and it has a missing frame, the PDC does not transmit an output for this frame. When the PDC waits a fixed wait time for the missing data frame, it does not allow the delay to affect its reporting rate. All consecutive data frames are processed on time.

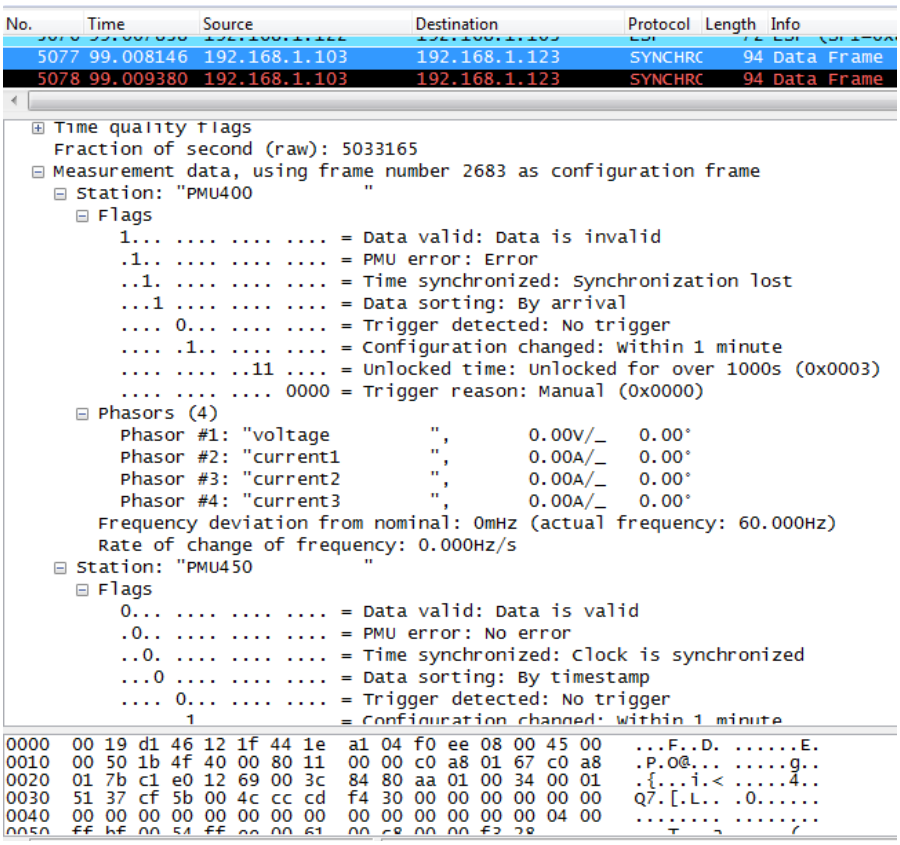

Figure 4.12: PDC Data Frame when Missing PMU Frame

When a PMU loses connection to its time source but still delivers valid data, the status flags are changed appropriately. The time since the GPS synchronization was lost is also shown under the status. Figures 4.13 (a), (b) and (c) display the change in unlock time from 10 sec to 100 sec to 1000 sec, respectively. The SEL 3373 PDC and ePDC pass all Data Validation tests.

```
User Datagram Protocol, Src Port: 54110 (54110), Dst Port: 4713 (4713)
IEEE C37.118 Synchrophasor Protocol, Data Frame
  Synchronization word: 0xaa01
  Framesize: 56
  PMU/DC ID number: 254
  SOC time stamp (UTC): 2012-04-17 19:43:39
  Time quality flags
    .0.. .... = Leap second direction: False
    ..0. .... = Leap second occurred: False
    ...0 .... = Leap second pending: False
    .... 1111 = Time Quality indicator code: Clock failure, time not reliable (0x0f)
  Fraction of second (raw): 9507089
  Measurement data, using frame number 6713 as configuration frame
    Station: "DUT1            "
      Flags
        0... .... .... .... = Data valid: Data is valid
        .0.. .... .... .... = PMU error: No error
        ..1. .... .... .... = Time synchronized: Synchronization lost
        ...0 .... .... .... = Data sorting: By timestamp
        .... 0... .... .... = Trigger detected: No trigger
        .... .0.. .... .... = Configuration changed: No
        .... .... ..01 .... = Unlocked time: Unlocked for 10s (0x0001)
        .... .... .... 0000 = Trigger reason: Manual (0x0000)
```

Figure 4.13 (a): PMU IRIG-B Disconnect. Unlock Time 10 s

```
IEEE C37.118 Synchrophasor Protocol, Data Frame
  Synchronization word: 0xaa01
  Framesize: 56
  PMU/DC ID number: 254
  SOC time stamp (UTC): 2012-04-17 19:45:17
  Time quality flags
    .0.. .... = Leap second direction: False
    ..0. .... = Leap second occurred: False
    ...0 .... = Leap second pending: False
    .... 1111 = Time Quality indicator code: Clock failure, time not reliable (0x0f)
  Fraction of second (raw): 16217975
  Measurement data, using frame number 6713 as configuration frame
    Station: "DUT1            "
      Flags
        0... .... .... .... = Data valid: Data is valid
        .0.. .... .... .... = PMU error: No error
        ..1. .... .... .... = Time synchronized: Synchronization lost
        ...0 .... .... .... = Data sorting: By timestamp
        .... 0... .... .... = Trigger detected: No trigger
        .... .0.. .... .... = Configuration changed: No
        .... .... ..10 .... = Unlocked time: Unlocked for 100s (0x0002)
        .... .... .... 0000 = Trigger reason: Manual (0x0000)
      Phasors (8)
```

Figure 4.13 (b): PMU IRIG-B Disconnect. Unlock Time 100 s

```
⊟ Time quality flags
    .0.. .... = Leap second direction: False
    ..0. .... = Leap second occurred: False
    ...0 .... = Leap second pending: False
    .... 0000 = Time Quality indicator code: Normal operation, clock locked (0x00)
  Fraction of second (raw): 633333
⊟ Measurement data, using frame number 9681 as configuration frame
  ⊟ Station: "421 VT Panel 2  "
    ⊟ Flags
        0... .... .... .... = Data valid: Data is valid
        .0.. .... .... .... = PMU error: No error
        ..1. .... .... .... = Time synchronized: Synchronization lost
        ...0 .... .... .... = Data sorting: By timestamp
        .... 0... .... .... = Trigger detected: No trigger
        .... .0.. .... .... = Configuration changed: No
        .... .... ..11 .... = Unlocked time: Unlocked for over 1000s (0x0003)
        .... .... .... 0000 = Trigger reason: Manual (0x0000)
```

Figure 4.13 (c): PMU IRIG-B Disconnect. Unlock Time 1000 s

The PDC time synchronization status is displayed in the Time Quality Flags prior to the PMU data blocks. Only the hardware-based PDC was tested by disconnecting its IRIG-B cable and watching the PDC's response. The software-based PDC was not tested because it does not have a direct connection to a time source; it also has a system time which can be used as a secondary time source. The hardware-based PDC failed this test because its Time Quality Indicator did not reflect a clock unlock status.

4.3.3 Latency Plots

The PDC processing time is a function of the number of inputs it is receiving and the number of phasors within each input. It is also dependent on the wait time that is selected based on network characteristics. These parameters set the upper boundary of the processing time. The lower boundary is a function of the physical processor speed and the buffer size. Ideally, the PDC should process its inputs before it receives any new data, and when it does have data waiting in the buffer, it should not allow the data to accumulate enough so that the buffer would need to discard it. The MATLAB program used to simulate arriving and processed packets, described in section 4.2.5, shows that the upper limit of the PDC processing latency should be approximately 2/3 of the reporting rate. This is under the assumption that the PDC does not wait the maximum wait time frequently. Figure 4.1.4 describes the methodology used to determine this upper boundary. The PMU 2 frame 1 is delayed and is received at the same time as PMU 1

frame 2, both having a reporting period of 16 ms. For the PDC to process the delayed frame and next frame but also recover from this setback, its processing speed must be at least 2/3 of its reporting rate, approximately 11 ms.

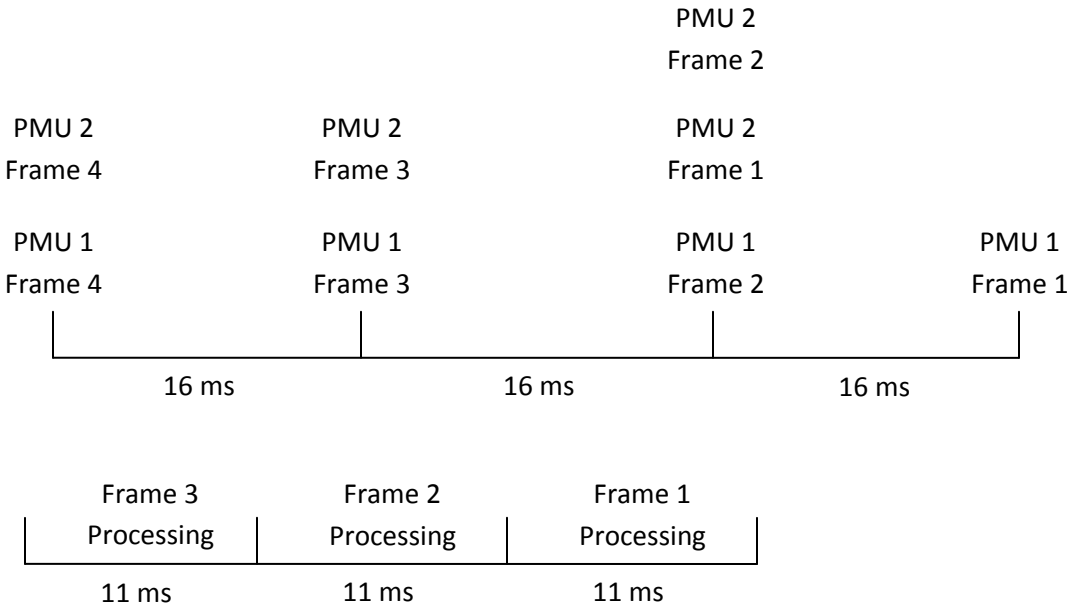

Figure 4.14: PDC Processing of Delayed Input

When the latency tests were performed using real PMUs, the MATLAB program, delayboundary.m, was run to determine which PMU was the last to arrive at the PDC. Using 12 PMUs, results of this test are shown in Figure 4.15. PMU 12 tends to be the last PMU data frame received by the PDC and this corresponds to PMU ID 254. It is used as the reference PMU in the SEL 3373 PDC latency tests using the original MATLAB latency program, delay.m. Figures 4.16, 4.17, and 4.18 show the PDC processing latency of SEL 3373 with 1PMU input, 4 PMUs and 12 PMUs, respectively. The PDC is set at a reporting rate of 30 fps. Figure 4.19 displays the median latency of the SEL 3373 while the number of PMU inputs is varied along the x-axis. The median value is the most accurate form of averaging the latencies. The mean value can be skewed due to a few high wait time values and the mode represents the most occurring value which will usually favour the lower latencies.

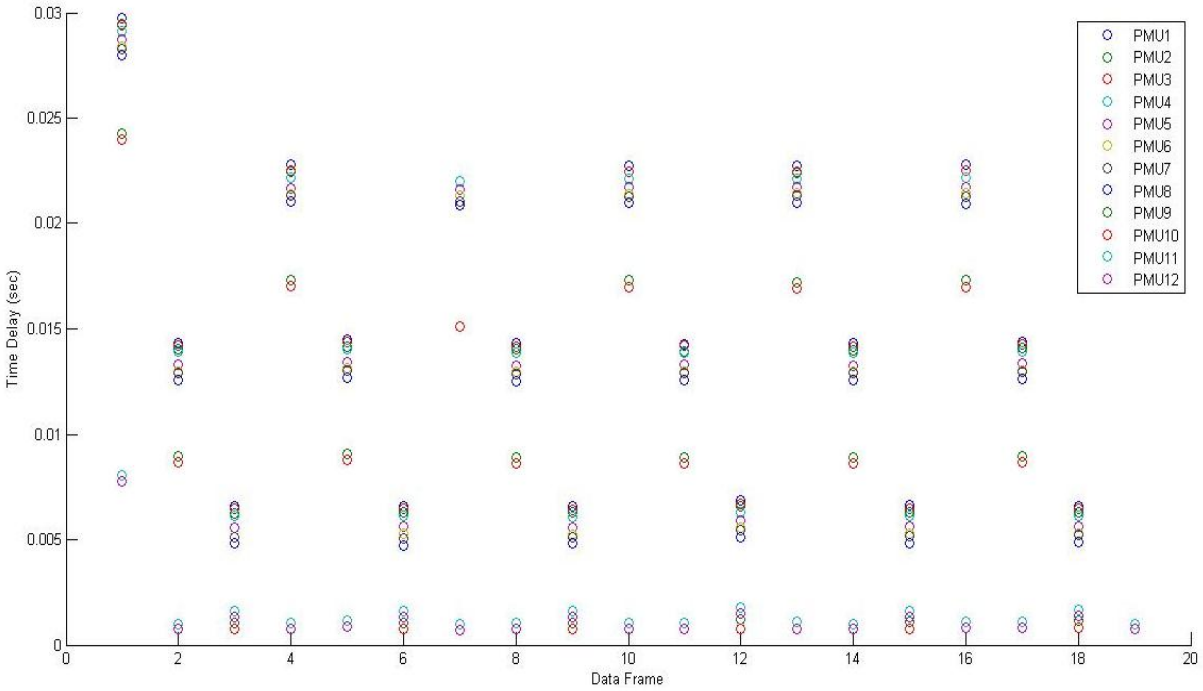

Figure 4.15: Last PMU Test [16]

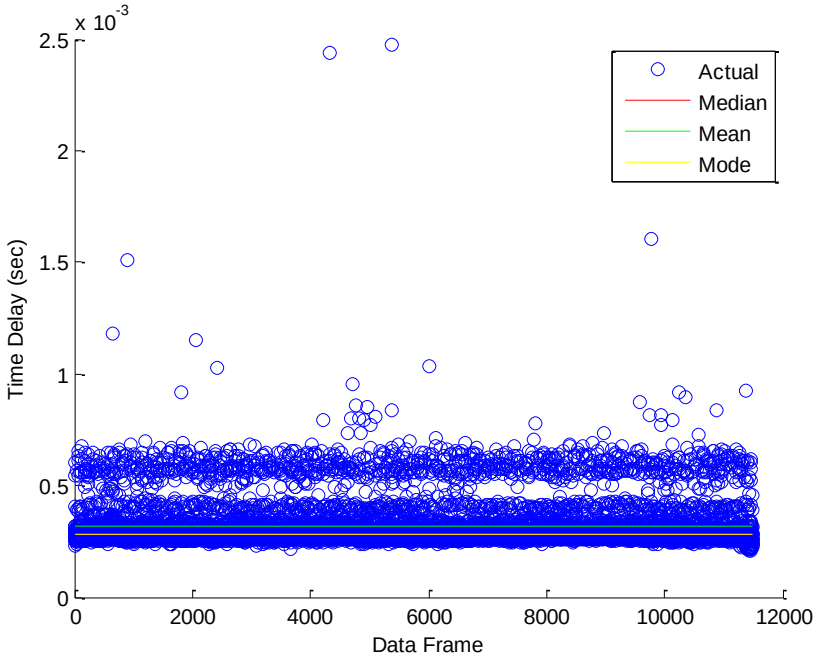

Figure 4.16: SEL 3373 - 1 PMU

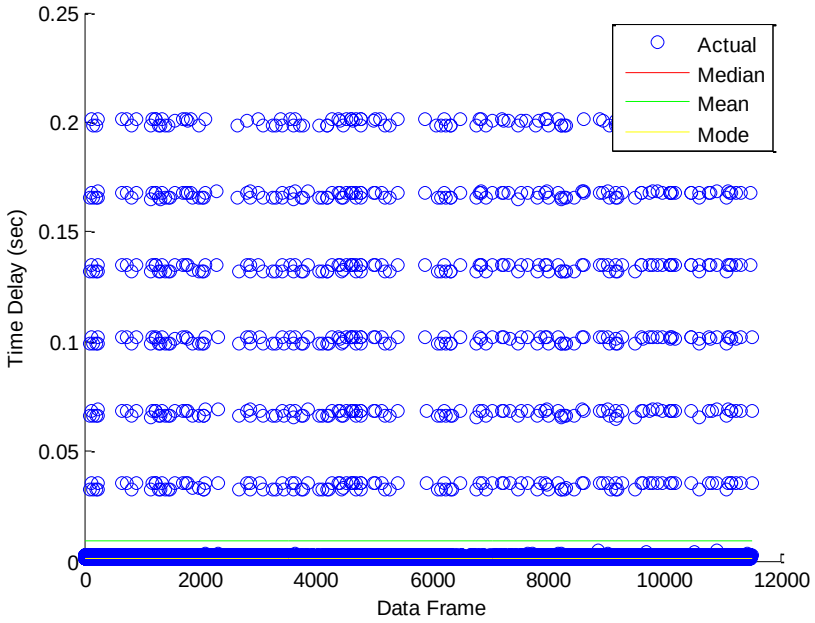

Figure 4.17: SEL 3373 - 4 PMUs

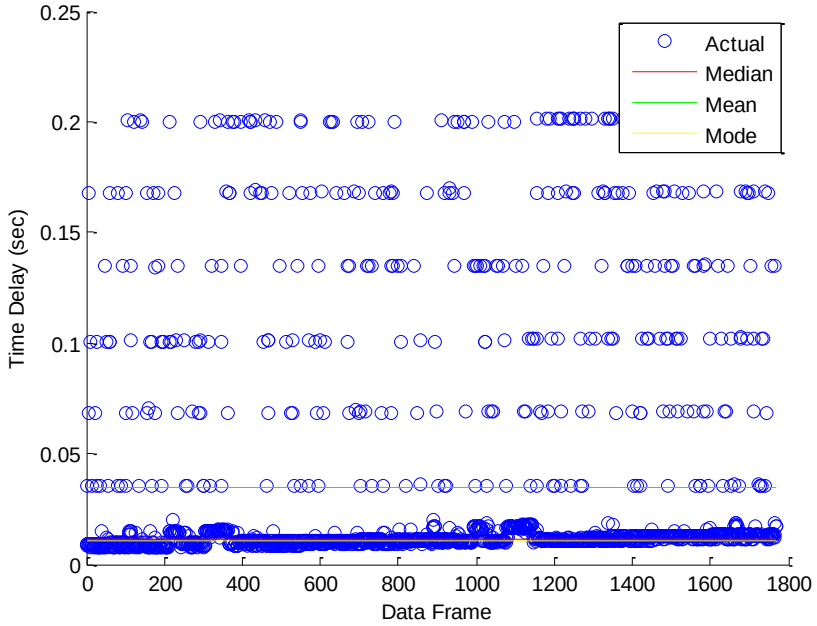

Figure 4.18: SEL 3373 - 12 PMUs

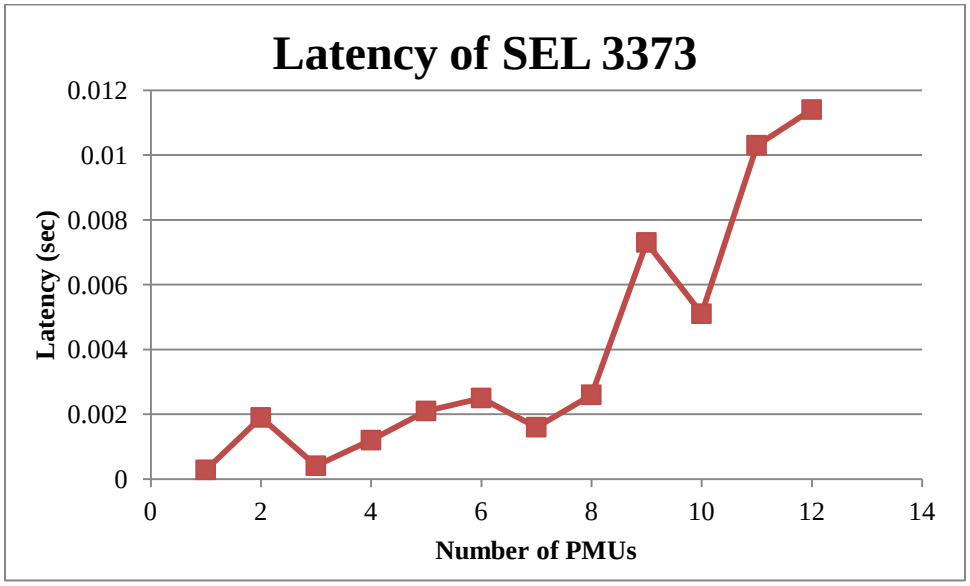

Figure 4.19: SEL 3373 – Median Latency

The ePDC latency is calculated using the updated MATLAB program that determines the last PMU input for every time tag so that the PDC processing time is a true value. The plots for ePDC set at a reporting rate of 30 fps are as follows: Figures 4.20 and 4.21 show the latency when the PDC receives 1PMU and 5 PMU inputs, respectively.

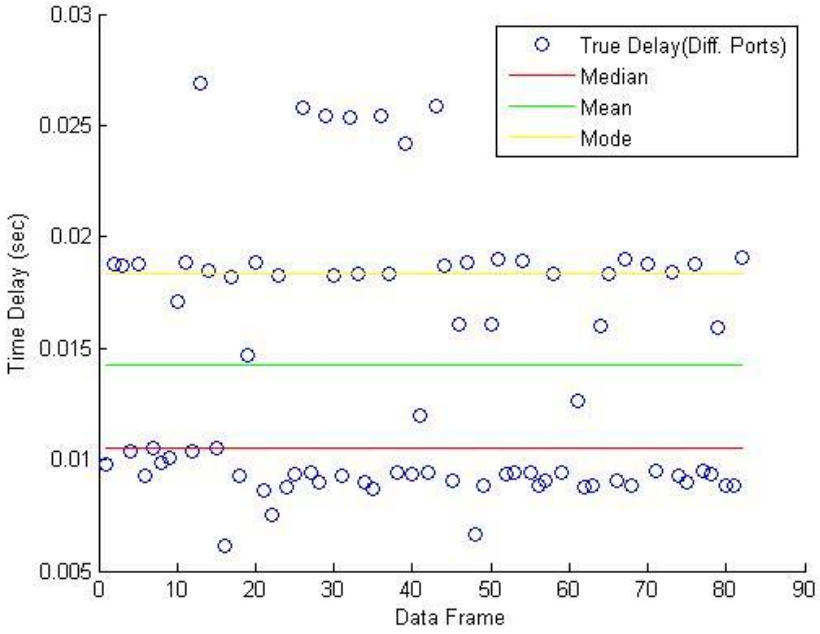

Figure 4.20: ePDC at 30 fps - 1 PMU

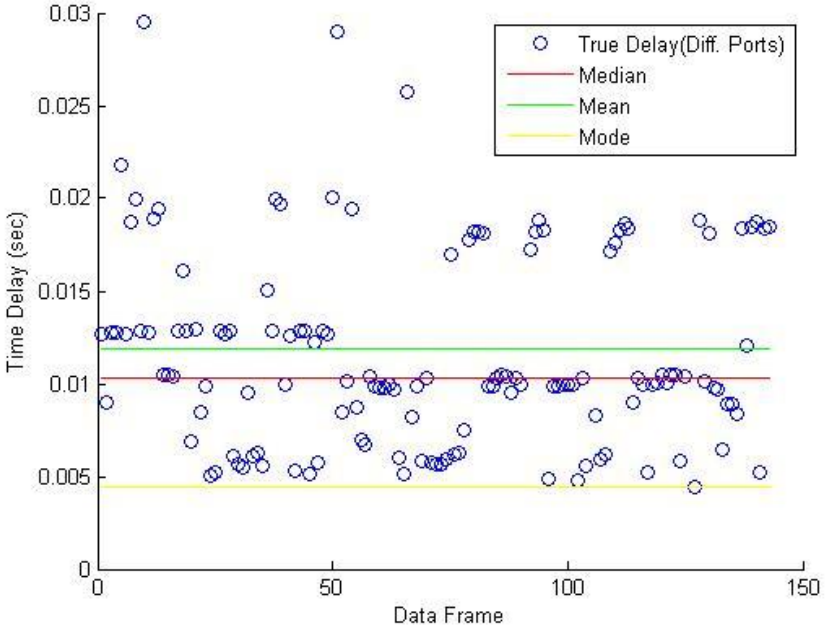

Figure 4.21: ePDC at 30 fps - 5 PMUs

The plot of the median latency versus the number of PMUs is shown in Figure 4.22 below. According to the test data, the PDC processing latency seems to be below 0.01344 sec when the reporting period is 0.033 sec. A good estimate for the maximum PDC processing time is approximately 2/3 of the reporting rate. It is impossible to determine an exact PDC processing limit without the knowledge of network characteristics and capacity of data being processed.

$$\text{PDC Latency Estimate} = (2/3) \times \text{Reporting Rate} \qquad (4.2)$$

$$= (2/3) \times 0.033 \text{ s}$$

$$= 0.022 \text{ s}$$

The actual processing time falls well below the latency estimate.

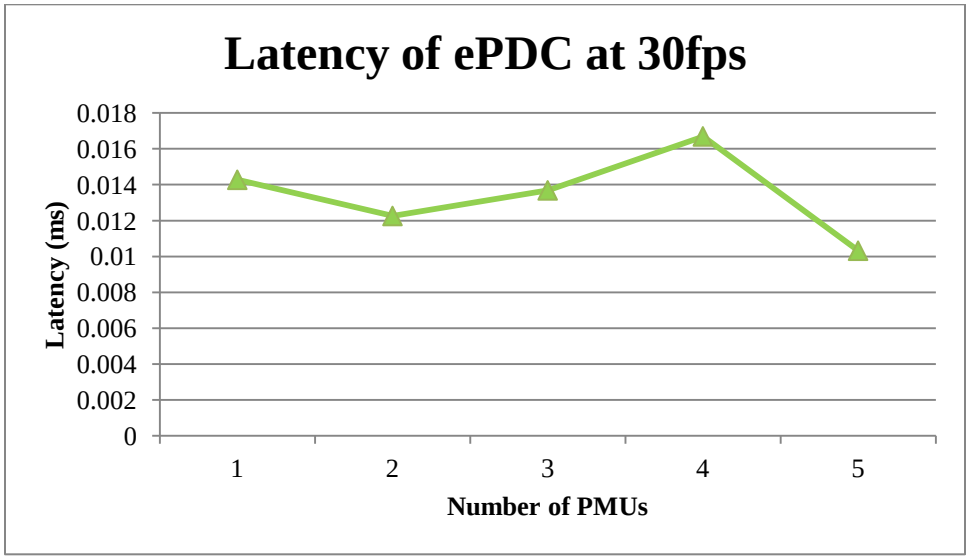

Figure 4.22: ePDC at 30 fps – Median Latency

The PDC processing latency of the ePDC at 60 fps is calculated similarly to the 30 fps data set. The latency with 1 PMU input and 5 PMU inputs are shown in Figures 4.23 and 4.24 respectively.

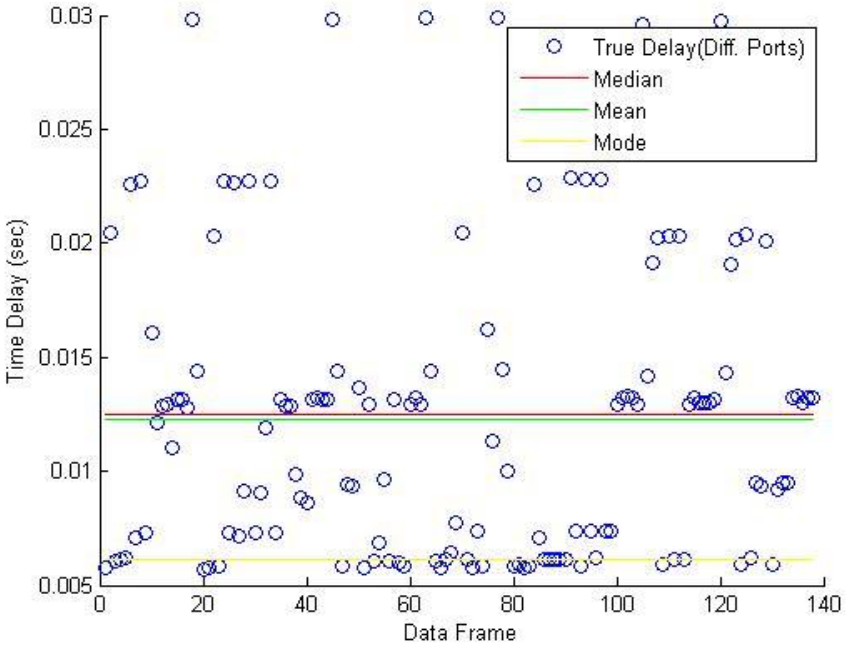

Figure 4.23: ePDC at 60 fps - 1 PMU

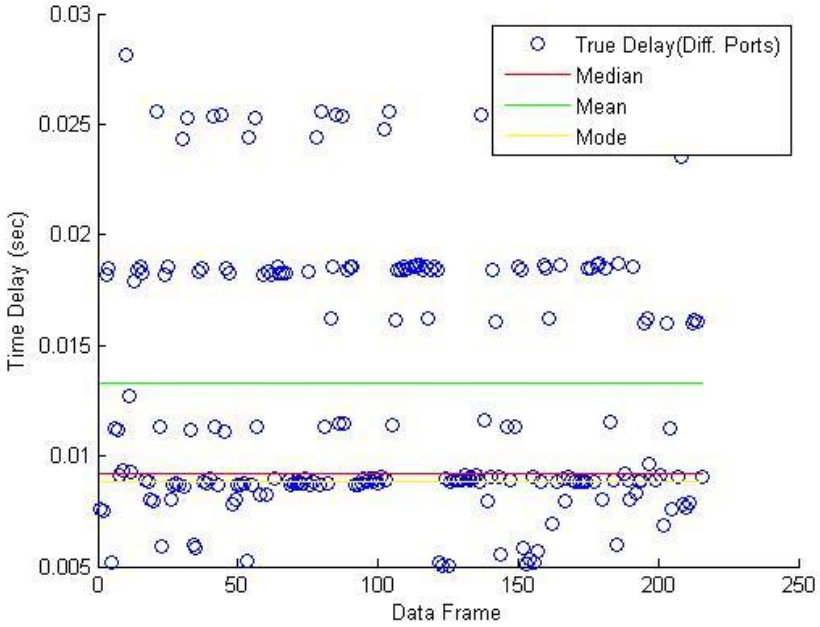

Figure 4.24: ePDC at 60 fps - 5 PMUs

The plot of the median latency versus the number of PMUs is shown in Figure 4.25 below. According to the test data, the PDC processing latency seems to be below 0.01085 sec when the reporting period is 0.0167 sec. The PDC latency estimate is calculated. The actual processing time falls below the latency estimate.

PDC Latency Estimate = (2/3) x Reporting Rate

= (2/3) x 0.0167 s

= 0.011 s

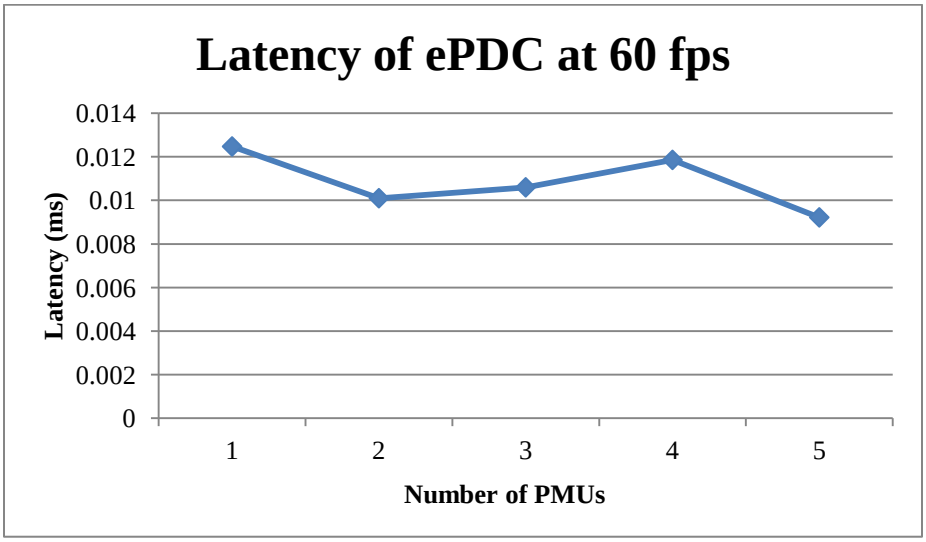

Figure 4.25: ePDC at 60 fps – Median Latency

The ePDC data shown above was collected after a recent upgrade was made to the system. The median latency of this PDC before the upgrade at 60 fps is plotted in Figure 4.26. This plot demonstrates the drastic difference in PDC processing time before and after the upgrade when compared to Figure 4.25. The latency seems to have reduced by approximately 1/3 after the upgrade. The latency shown in the plot below would be higher than the estimate of 0.011 sec.

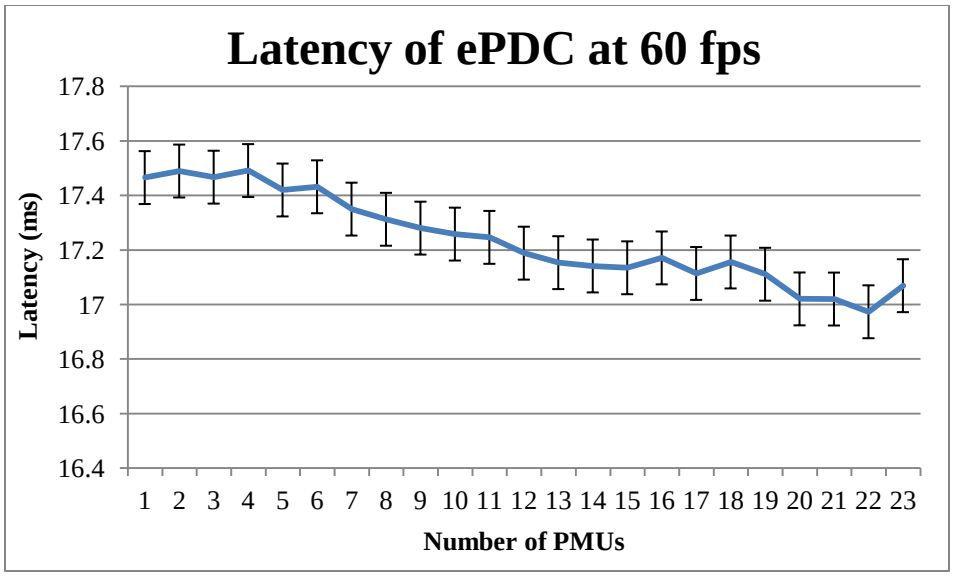

Figure 4.26: ePDC at 60 fps before upgrade – Median Latency

In order to measure the actual wait time of the PDC and compare it to the wait time setting, the latency program is slightly modified so that only delays that are +/- 50 ms from the wait time are plotted. The PMU Simulator code was used to delay packets by +/- 50 ms of the PDC wait time. Figure 4.27 shows the maximum wait time of the SEL 3373 PDC to be 200 milliseconds. The wait time of the ePDC can be changed by the user through the GUI. The actual wait time of the ePDC meets the wait time setting.

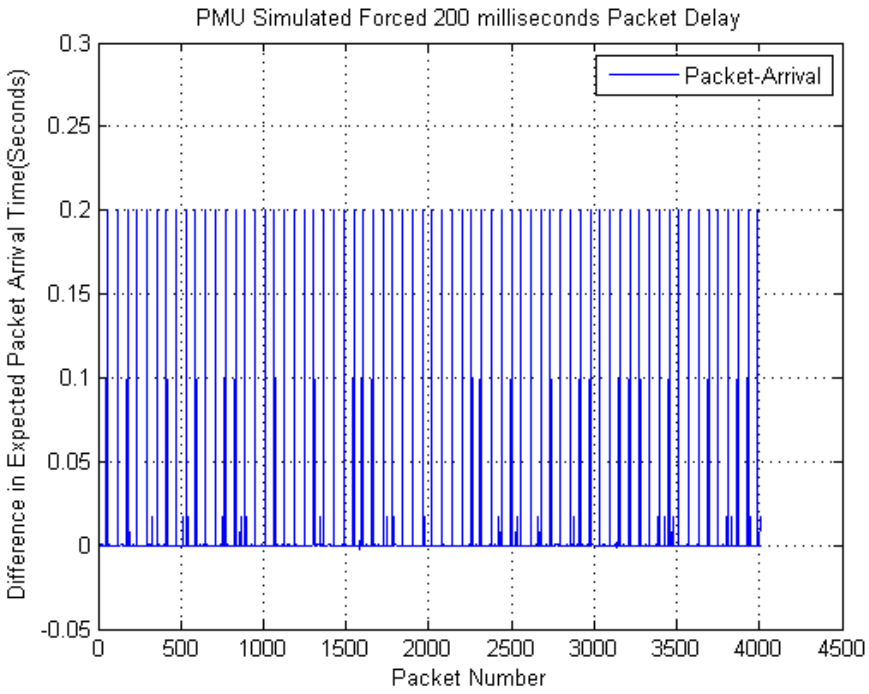

Figure 4.27: SEL 3373 – Wait Time

The results presented in this section do not take into account the latency associated with processing additional functionalities such as re-sampling or data format conversion.

4.3.4 PDC Capacity and Buffer Size Results

After applying the capacity test on the Schweitzer hardware-based PDC, the device was able to process 32 PMU inputs with no significant change in latency at this saturation point. The PMU data carried two phasor measurements each and the PDC provided one output stream shown in Figure 4.28.

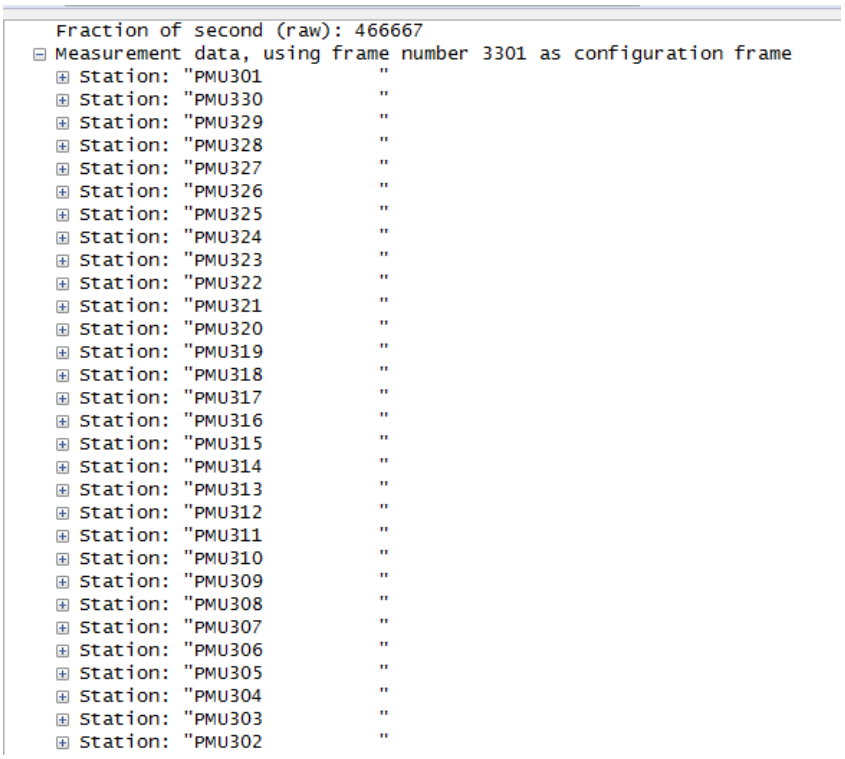

Figure 4.28: SEL 3373 Output Stream at Capacity

The ePDC is developed to handle over a 100 PMU inputs, each with over 10 phasor measurements. The unit was tested using PMU inputs with 2 phasor measurements each up to a total of 300 PMUs. At this level, only an average of 7% of the PDC's processing capabilities were being used, shown in Figure 4.29. This measurement implies that the PDC may be capable of processing approximately $\frac{300}{0.07} = 4285$ PMUs. This is a very high estimate and the variance is hard to determine with the limited available data; however, it is safe to assume that the PDC can still process well over a 1000 PMU inputs, each holding two phasor measurements. The network capacity of the PDC, Figure 4.30, is also at 7% with 300 inputs.

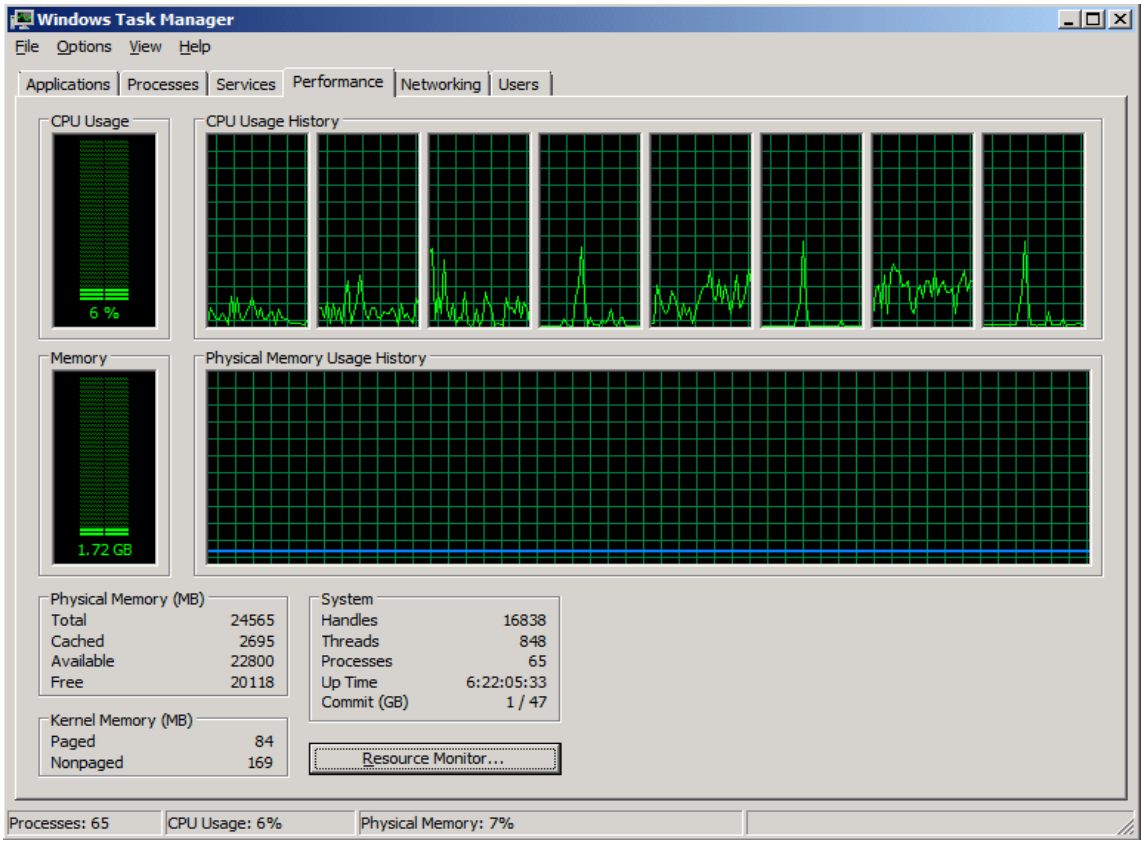

Figure 4.29: ePDC Processing Capacity

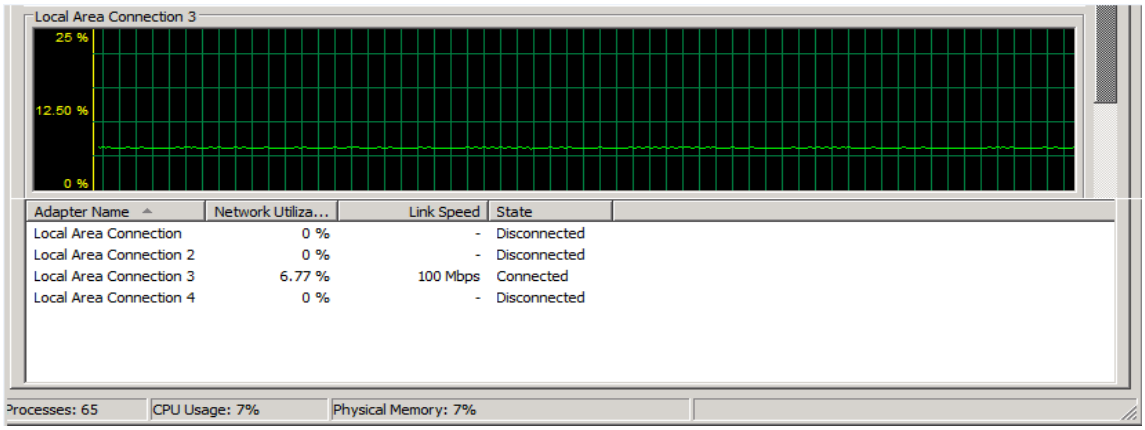

Figure 4.30: ePDC Network Utilization

The PDC processing latency of the ePDC before the upgrade with up to 303 PMU inputs can be seen in Figure 4.31. The PDC had a reporting rate of 30 fps and the maximum latency was approximately 23 milliseconds.

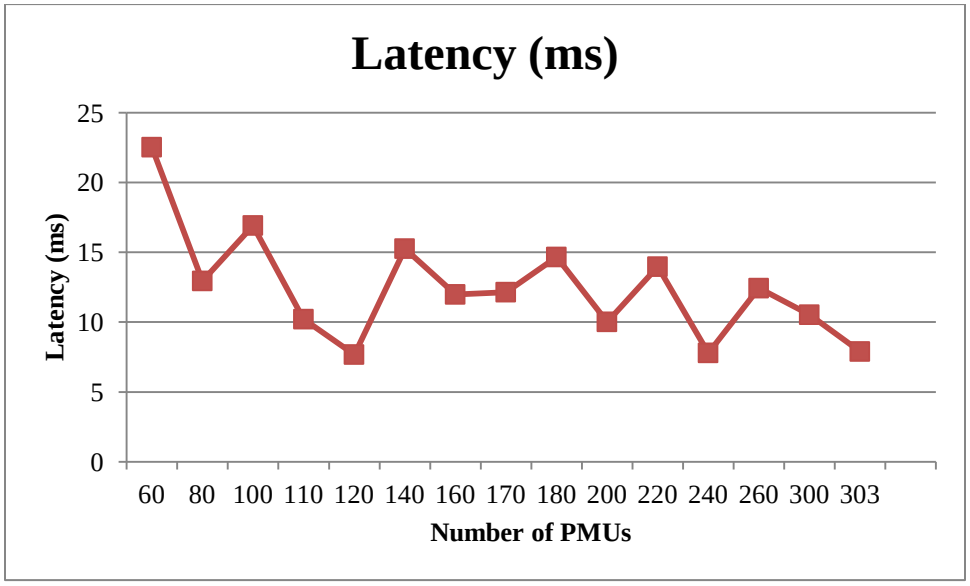

Figure 4.31: ePDC at 30 fps before upgrade – Median Latency

The buffer size test was only performed on the ePDC due to the timing sensitivity of this test. The ePDC proved that it was capable processing all data received up to its buffer limit. Data received beyond the buffer limit was discarded. This limit is a user setting and is generally set to 2 seconds during most tests.

4.3.5 Re-sampling Results

Using the PMU Simulator with the software-based PDC the three types of re-sampling tests were performed. The PDC performed appropriately in all three scenarios so that its output rate remained constant with the varying input PMU frame rates. The PDC seemed to treat up-sampling and down-sampling with the same method; the data in the last received PMU frame was found in the PDC output frame irrespective of the data rate. The algorithm the PDC uses to

re-sample the actual data cannot be tested using steady state measurements since the outputs won't reflect any changes in phasors. A ramp-up frequency can be hard-coded in the measurement data of the PMU Simulator and the PDC output should show a change in the re-sampled data due to this dynamic state.

The hardware based PDCs tested do not yet have an up-sampling or down-sampling mechanism so the PMU rate must match the PDC rate.

4.3.6 Response to Packets Out of Order

The Irregular Packet Order test performed on the SEL 3373 PDC contains the following results. The test was conducted using configurations that included just the one PMU with reordered packets or two PMUs, of which only one has reordered packets. In either test, the PDC would process the reordered packet, in the correct order, if it still arrives within the wait time. If the packet arrived after the wait time, the PDC would not process it in the one PMU test, as shown in Table 4.1. However, if the packet was delayed beyond the wait time in the two PMU test, the PDC seems to re-sample this data and provides its best estimate in the output stream. This result can be seen in Table 4.2. These tables are organized according to the time stamp of each device (they are different due to their TIME_BASE value) and the measurements of in their corresponding data stream. The PMU frame with time stamp 5033165 is the packet simulated along path 3 after path 2 is reconnected.

Table 4.1: SEL-3373 PDC Reordered Packet Data with 1 PMU

PDC Time Stamp	PMU Time Stamp	PDC Measurements	PMU Measurements	PDC Frame Received
266667	4473925	3356.52 V, 3.27 A	3356.52 V, 3.27 A	Yes
333333	5592406	3186.13 V, 3.15 A	3186.13 V, 3.15 A	Yes
300000	5033165		3112.19 V, 3.06 A	No
366667	6151646	3155.73 V, 2.95 A	3155.73 V, 2.95 A	Yes

Table 4.2: SEL-3373 PDC Reordered Packet Data with 2 PMUs

PDC Time Stamp	PMU Time Stamp	PMU Measurements	PDC Measurements	PDC Frame Received
266667	4473925	3356.52 V, 3.27 A	3356.52 V, 3.27 A	Yes
333333	5592406	3186.13 V, 3.15 A	3186.13 V, 3.15 A	Yes
300000	5033165	3112.19 V, 3.06 A	3112.17 V, 3.07 A	Yes
366667	6151646	3155.73 V, 2.95 A	3155.73 V, 2.95 A	Yes

The results of this test on the ePDC device are slightly different compared to the results of SEL-3373 PDC. The PDC processes the reordered packet when the it arrives within the wait time in the one PMU and the two PMU configuration. The PDC also processes the reordered packet in the one PMU test when it arrives beyond the wait time but sooner than the buffer limit. It does not output the frame with the 5592406 time stamp before the reordered frame with time stamp 5033165, it corrects the order, as shown in Table 4.3. In the two PMU test, the PDC does not wait beyond the wait time and it processes its data by filling the reordered PMU data block with random data and flagging the data invalid bit, similar to the missing data test. This data is displayed in Table 4.4. If the PMU data is delayed beyond the buffer limit, the PDC restarts data transmission in the one PMU case; the PDC treats the late data as missing data in the two PMU case.

Table 4.3: ePDC Reordered Packet Data with 1 PMU

PDC Time Stamp	PMU Time Stamp	PDC Measurements	PMU Measurements	PDC Frame Received
4473925	4473925	3356.52 V, 3.27 A	3356.52 V, 3.27 A	Yes
5033165	5592406	3112.19 V, 3.06 A	3186.13 V, 3.15 A	Yes
5592406	5033165	3186.13 V, 3.15 A	3112.19 V, 3.06 A	Yes
6151646	6151646	3155.73 V, 2.95 A	3155.73 V, 2.95 A	Yes

Table 4.4: ePDC Reordered Packet Data with 2 PMUs

PDC Time Stamp	PMU Time Stamp	PDC Measurements	PMU Measurements	PDC Frame Received
4473925	4473925	3356.52 V, 3.27 A	3356.52 V, 3.27 A	Yes
5592406	5592406	3186.13 V, 3.15 A	3186.13 V, 3.15 A	Yes
5033165	5033165	0 V, 0 A	3112.19 V, 3.06 A	Data Error
6151646	6151646	3155.73 V, 2.95 A	3155.73 V, 2.95 A	Yes

Chapter 5

Conclusions and Future Work

The three main research areas described in this thesis have been on Phasor Data Concentrators meeting IEEE standard C37.118 requirements, evaluating their performance when receiving non-ideal data inputs and PDC latency. Using the standards and requirements and the test results from the previous chapters, some conclusions can be drawn and feedback can be provided on the testing methods. Future work on these topics is also discussed in this chapter.

5.1 Conclusions

The tests described in this thesis demonstrate some of the functional and communication needs of the Phasor Data Concentrator. Existing tests were refined or new tests were developed based on whether the PDC is software or hardware based. Results from the IEEE standard tests prove that the tested PDCs meet most of the requirements. A few modifications still need to be made to pass all criteria outlined in the standards such as PDC clock synchronization. Updates need to be made so meet the new IEEE standard C37.118.2-2011.

Apart from the standards, the PDC must meet other requirements in order to become a reliable tool across the power grid. PDCs located at utilities and other power system authorities need to communicate data between themselves and to other applications. The type of protocol used, TCP/IP or UDP/IP, for this communication must also be set as a standard. From our testing, UDP/IP seems to be a more appropriate choice to achieve faster data transmission. The amount of time the PDC waits for all its inputs affects the delay in which applications or other PDCs receive data. Receiving devices must be configured so that they are prepared to accept data packets from the PDC, therefore this wait time should also be set as an application requirement.

Requirements that are not explicitly stated in the C37.118 have stemmed from these standards such as the PDC's time dependencies and time sources. The PDC may have more than one time source such as the GPS clock and the host machine time. The PDC must be set to

follow the primary or secondary time sources depending on the condition of the system. The PDC may also adapt to the PMU input time in certain cases. These conditions must be clearly defined as PDC requirements so that all PDCs behave in the same manner independent of manufacturer or utility. The different time sources may affect the synchronization error flags and time quality bits in the PDC frame since all sources will not have the exact same time stamp.

The latency test results show that the PDC must process its data frames within 2/3 of its reporting frame from the time is receives its last input frame for a particular time stamp. This limit is derived so that the PDC will recover from late PMU inputs. As shown in Figure 4.14, the late PMU frame 1 arrived when the PDC would normally process frame 2 and the PDC reset itself to its alignment time by the 4^{th} set of frames. From these results, the network and number of inputs to the PDC should be designed to allow to the PDC to meet this processing requirement. The wait time should also be set accordingly.

The PDC was tested to determine how it handles frames arriving in a staggered order. For example, if the PMU sent out frames A→B→C in order but the PDC received them as B→A→C, how the PDC would process its output frame and time stamps. This test replicates a scenario in which the PMU has multiple paths to transmit data to the PDC and a combination of disconnecting and reconnecting the paths can cause the PMU inputs to arrive in a different order. The two PDCs that were tested behaved in different ways under these conditions. This shows us that there needs to be a standard developed for PMU frames that arrive out of order so that data is not lost. Other application based requirements include the input capacity of the PDC, the PDC's ability to resample data correctly and the buffer size.

From the interpretation of the current PDC standards and tests described in chapter 3, some clarifications and modifications can be made to IEEE C37.118.2-2011. New guidelines based on the latter half of chapter 3 and the test results in chapter 4 can be added as application based requirements.

5.2 Future Work

Although this thesis addressed quite a few new PDC requirements, there is room for further testing and development. The current re-sampling test results show that the PDC is able to up-sample and down-sample correctly for any reporting rate combination; however, it is difficult to check whether the PDC re-samples the measurements correctly using the random PMU inputs. The PDC should be tested using known input values and ramp-up frequency so that the re-sampled output will show a change in measurements. This can be accomplished by modifying the PMUSimulator to process hard-coded measurements from a user created file. These values can be tested using a up-sampling and down-sampling algorithm and the results can be compared to the PDC's output in both conditions. In addition to processing a change in reporting rate, PDCs are capable of receiving data in Polar or Cartesian form and providing outputs in either form. The effect of the coordinate translation can be tested for correctness and its affect on latency can be measured.

Another area of research related to PDC communication is its security vulnerability and how it can affect the whole wide area measurement network it is connected to. The different levels of protection on the PDC and its network should be tested by cyber sources inside and outside the closed network.

References

[1] A. G. Phadke and J. S. Thorp, Synchronized Phasor Measurements, Springer, 2008.

[2] A. Phadke and J. S. Thorpe, "History and Applications of Phasor Measurements," Blacksburg, 2006.

[3] J. D. L. Ree, V. Centeno, J. S. Thorp and A. G. Phadke, "Synchronized Phasor Measurement Applications in Power Systems," *IEEE Trans. on Smart Grid,* vol. 1, no. 1, pp. 20-27, 2010.

[4] K. E. Martin, "Phasor Measurement Systems in the WECC," Bonneville Power Administration, Vancouver, WA.

[5] Z. Huang, B. Kasztenny, V. Madani, K. Martin, S. Meliopoulos, D. Novosel and J. Stenbakken, "Performance Evaluation of Phasor Measurement Systems," in *IEEE Power Engineering Society General Meeting*, Pittsburgh, 2008.

[6] P. &. S. T. T. W. g. C. G. A. Technical Report for the North American SynchroPhasor Initiative, "Guide for Phasor Data Concentrator Requirements for Power System Protection, Control, and Monitoring".

[7] IEEE Power System Relaying Committee, "IEEE PC37.244 D6.0 1 Draft Guide forPhasor Data Concentrator Requirements for Power System Protection, Monitoring and Control," IEEE Power and Energy Society, New York, 2013.

[8] IEEE-SA Standards Board, "IEEE Standard for SynchrophasorData Transfer for Power Systems," IEEE, New York, 2011.

[9] IEEE Power & Energy Society, "IEEE Standard for Synchrophasor Measurements for Power Systems," IEEE-SA Standards Board, New York, 2011.

[10] RUGGEDCOM Industrial Strength Networks, "Latency on a Switched Ethernet Network," Woodbridge, 2008.

[11] A. Galina - ABB, "Guide for Phasor Data Concentrator Requirements for Power System Protection, Control, and Monitoring," PJM, 2011.

[12] Schweitzer Engineering Laboratories, "SEL-3373," 2013. [Online]. Available: https://www.selinc.com/SEL-3373/. [Accessed 15 April 2013].

[13] Electric Power Group, "Phasor Data Concentrators - ePDC & eSPDC," 2010. [Online]. Available: http://www.electricpowergroup.com/solutions/epdc/index.html. [Accessed 15 April 2013].

[14] Schweitzer Engineering Laboratories, "SEL-421," 2013. [Online]. Available:

https://www.selinc.com/SEL-421/. [Accessed 15 April 2013].

[15] iPDC, "iPDC-v1.3.1," 2013. [Online]. Available: http://ipdc.codeplex.com/. [Accessed 15 April 2013].

[16] P. M. Kersey, "Applications of PMUSimulator in PDC Testing - Thesis," Virginia Tech, Blacksburg, 2012.

Appendix A: IEEE C37.118.2-2011 Message Framework

A.1 Header Frame Organization

No	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number (AA11 hex).
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	PMU/PDC data stream ID number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2.
6	DATA 1	1	ASCII character, 1st byte.
K+6	DATA k	1	ASCII character, Kth byte, K>0 is an integer.
K+7	CHK	2	CRC-CCITT.

A.2 Command Frame Organization

No	Field	Size (bytes)	Comment
1	SYNC	2	Sync byte followed by frame type and version number (AA41 hex).
2	FRAMESIZE	2	Number of bytes in frame, defined in 6.2.
3	IDCODE	2	PMU/PDC ID data stream number, 16-bit integer, defined in 6.2.
4	SOC	4	SOC time stamp, defined in 6.2.
5	FRACSEC	4	Fraction of Second and Time Quality, defined in 6.2.
6	CMD	2	Command being sent to the PMU/PDC (0).
7	EXTFRAME	0–65518	Extended frame data, 16-bit words, 0 to 65518 bytes as indicated by frame size, data user defined.
8	CHK	2	CRC-CCITT.

Appendix B: PMU Simulator Code

B.1 PMUSimulator Missing Data Test Source Code

```
/* ------------------------------------------------------------------------    */
/* FUNCTION  void* SEND_DATA
       */
/* This function run by a seprate thread only for data transmission.            */
/* Function to generate and send the data frame periodically to client's    */
/* destination address or to PDC (client).                                      */
/* ------------------------------------------------------------------------    */

void* SEND_DATA()
{
       //Added by Hema to store prev PMU data frame
       unsigned char temp_datafrm[5000];


     /* Wait till server will get Setup file path */
       while(df_data_rate == 0) usleep(1000);

     /* Calculate the waiting time during sending data frames */
       int data_waiting = 1e9/df_data_rate, i=0;
     struct PDC_Details *temp_pdc;
     send_thrd_id = pthread_self();

     struct timespec *cal_timeSpec, *cal_timeSpec1;
     cal_timeSpec = malloc(sizeof(struct timespec));
     cal_timeSpec1 = malloc(sizeof(struct timespec));

       clock_gettime(CLOCK_REALTIME, cal_timeSpec);

       while(1)
       {
              //printf("while 1 \n");
              clock_gettime(CLOCK_REALTIME, cal_timeSpec1);
              clock_gettime(CLOCK_REALTIME, cal_timeSpec);

              if (cal_timeSpec->tv_sec > cal_timeSpec1->tv_sec)
              {
                     fsecNum = 1;
                     break;
              }

       }


       while(1)
       {
       //printf("send data\n");
              if (i != 0)
              {
                     cal_timeSpec->tv_nsec += data_waiting;
              }
              else
```

```
            {
                  cal_timeSpec->tv_nsec = data_waiting;
            }
            if ((cal_timeSpec->tv_nsec) >= 1e9)
            {
                  cal_timeSpec->tv_sec++;
                  cal_timeSpec->tv_nsec-=1e9;
            }

            /* Call the function generate_data_frame() to create a fresh new Data Frame
*/
            generate_data_frame();

            clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, cal_timeSpec,
cal_timeSpec1);

          temp_pdc = PDCfirst;
       //Added by Hema to store PMU frame
         if (i==998)
         {
              B_copy(temp_datafrm, data_frm, 0, 5000);
              printf("store prev PDC. FOS is %ld \n", fsec);
         }
         else if (i==1000)
         {
              printf("data: %x %x %x %x tempdata: %x %x %x %x \n", data_frm[10],
data_frm[11], data_frm[12], data_frm[13],temp_datafrm[10], temp_datafrm[11],
temp_datafrm[12], temp_datafrm[13]);
              B_copy(data_frm, temp_datafrm, 0, 5000);
              printf("replace prev PDC. FOS is %ld \n", fsec);
              printf("data: %x %x %x %x tempdata: %x %x %x %x \n", data_frm[10],
data_frm[11], data_frm[12], data_frm[13],temp_datafrm[10], temp_datafrm[11],
temp_datafrm[12], temp_datafrm[13]);
              printf("SOC: %x %x %x %x \n", data_frm[6], data_frm[7], data_frm[8],
data_frm[9]);
         }

          pthread_mutex_lock(&mutex_pdc_object);

          while(temp_pdc != NULL)
          {
              //printf("while 3\n");
               if(!strncasecmp(temp_pdc->protocol, "UDP", 3) && (temp_pdc-
>data_transmission == 0))
               {
                  /* If STAT Word bits got changed by user */
                    if(temp_pdc->STAT_change != 0)
                    {
                         switch (temp_pdc->STAT_change)
                         {
                              case 1:
                                   data_frm[14] = 0x04;     //CFG changed
                                   data_frm[15] = 0x00;
                                   break;
                              case 2:
                                   data_frm[14] = 0x80;
                                   data_frm[15] = 0x00;
                                   temp_pdc->STAT_change = 0;
```

```
                                    break;
                                case 3:
                                        data_frm[14] = 0x40;     //PMU error
                                        data_frm[15] = 0x00;
                                    break;
                                case 4:
                                        data_frm[14] = 0x10;
                                        data_frm[15] = 0x00;
                                    temp_pdc->STAT_change = 0;
                                    break;
                                case 5:
                                        data_frm[14] = 0x08;
                                        data_frm[15] = 0x00;
                                    temp_pdc->STAT_change = 0;
                                    break;
                            }
                        }

                         /* UDP-Send the newly created data frame to connected PDC address */
                        if (sendto (temp_pdc->sockfd,data_frm, df_data_frm_size, 0,
                                                (struct sockaddr *)&temp_pdc-
>pdc_addr,sizeof(temp_pdc->pdc_addr)) == -1) {
                         printf("send UDP\n");
                                 perror("sendto");
                          }
                }
                else if(!strncasecmp(temp_pdc->protocol, "TCP", 3) && (temp_pdc-
>data_transmission == 0))
                {
                            if(temp_pdc->tcpup == 1)
                            {
                             /* TCP-Send the newly created data frame to connected PDC
address */
                                if (send(temp_pdc->sockfd, data_frm, df_data_frm_size, 0)
== -1) {

                                                perror("sendto");
                                        }
                            }
                }

                temp_pdc = temp_pdc->next;
             }
             pthread_mutex_unlock(&mutex_pdc_object);
            /*if(i==200)
            {
                usleep(2000000);     //Delay added
                printf("\nDelay 200000uS. FOS is %ld\n", fsec);
            }
            */

            i++;
                   clock_gettime(CLOCK_REALTIME, cal_timeSpec1);

        }    //while-2 ends here

} /* end of function send_data() */
```

B.2 PMUSimulator Irregular Packet Order Test Source Code

```
/* ------------------------------------------------------------------------    */
/* FUNCTION  void* SEND_DATA                                                    */
/* This function run by a separate thread only for data transmission.          */
/* Function to generate and send the data frame periodically to client's    */
/* destination address or to PDC (client).                                  */
/* ------------------------------------------------------------------------    */

void* SEND_DATA()
{
       //Added by Hema to store prev PMU data frame
       unsigned char temp_datafrm[5000];


     /* Wait till server will get Setup file path */
       while(df_data_rate == 0) usleep(1000);

     /* Calculate the waiting time during sending data frames */
       int data_waiting = 1e9/df_data_rate, i=0;
     struct PDC_Details *temp_pdc;
     send_thrd_id = pthread_self();

     struct timespec *cal_timeSpec, *cal_timeSpec1;
     cal_timeSpec = malloc(sizeof(struct timespec));
     cal_timeSpec1 = malloc(sizeof(struct timespec));

       clock_gettime(CLOCK_REALTIME, cal_timeSpec);

       while(1)
       {
              //printf("while 1 \n");
              clock_gettime(CLOCK_REALTIME, cal_timeSpec1);
              clock_gettime(CLOCK_REALTIME, cal_timeSpec);

              if (cal_timeSpec->tv_sec > cal_timeSpec1->tv_sec)
              {
                     fsecNum = 1;
                     break;
              }

       }


       while(1)
       {
       //printf("send data\n");
              if (i != 0)
              {
                     cal_timeSpec->tv_nsec += data_waiting;
              }
              else
              {
                     cal_timeSpec->tv_nsec = data_waiting;
```

```
            }
            if ((cal_timeSpec->tv_nsec) >= 1e9)
            {
                   cal_timeSpec->tv_sec++;
                   cal_timeSpec->tv_nsec-=1e9;
            }

            /* Call the function generate_data_frame() to create a fresh new Data Frame
*/
            generate_data_frame();

            clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, cal_timeSpec,
cal_timeSpec1);

         temp_pdc = PDCfirst;
      //Added by Hema to store PMU frame
*/
      if (i==998)
      {
            B_copy(temp_datafrm, data_frm, 0, 5000);
            printf("store prev PDC. FOS is %ld \n", fsec);
            printf("Frame skipped\n");
      }
      else if (i==999)
      {
         pthread_mutex_lock(&mutex_pdc_object);

         while(temp_pdc != NULL)
         {
             //printf("while 3\n");
              if(!strncasecmp(temp_pdc->protocol, "UDP", 3) && (temp_pdc-
>data_transmission == 0))
              {
                /* If STAT Word bits got changed by user */
                  if(temp_pdc->STAT_change != 0)
                  {
                       switch (temp_pdc->STAT_change)
                       {
                            case 1:
                                data_frm[14] = 0x04;     //CFG changed
                                data_frm[15] = 0x00;
                            break;
                            case 2:
                                data_frm[14] = 0x80;
                                data_frm[15] = 0x00;
                            temp_pdc->STAT_change = 0;
                            break;
                            case 3:
                                data_frm[14] = 0x40;     //PMU error
                                data_frm[15] = 0x00;
                            break;
                            case 4:
                                data_frm[14] = 0x10;
                                data_frm[15] = 0x00;
                            temp_pdc->STAT_change = 0;
                            break;
                            case 5:
                                data_frm[14] = 0x08;
```

```
                                    data_frm[15] = 0x00;
                                   temp_pdc->STAT_change = 0;
                                   break;
                               }
                          }

                           /* UDP-Send the newly created data frame to connected PDC address */
                          if (sendto (temp_pdc->sockfd,data_frm, df_data_frm_size, 0,
                                              (struct sockaddr *)&temp_pdc-
>pdc_addr,sizeof(temp_pdc->pdc_addr)) == -1) {
                           printf("send UDP\n");
                                  perror("sendto");
                            }

                        B_copy(data_frm, temp_datafrm, 0, 5000);
                        usleep(200500);
                        sendto (temp_pdc->sockfd,data_frm, df_data_frm_size, 0,(struct sockaddr
*)&temp_pdc->pdc_addr,sizeof(temp_pdc->pdc_addr));
                        printf("replace prev PDC. FOS is %ld \n", fsec);
                        printf("data: %x %x %x %x tempdata: %x %x %x %x \n", data_frm[10],
data_frm[11], data_frm[12], data_frm[13],temp_datafrm[10], temp_datafrm[11],
temp_datafrm[12], temp_datafrm[13]);
                        printf("SOC: %x %x %x %x \n", data_frm[6], data_frm[7], data_frm[8],
data_frm[9]);
                    }
                    else if(!strncasecmp(temp_pdc->protocol, "TCP", 3) && (temp_pdc-
>data_transmission == 0))
                    {
                                if(temp_pdc->tcpup == 1)
                                {
                                 /* TCP-Send the newly created data frame to connected PDC
address */
                                     if (send(temp_pdc->sockfd, data_frm, df_data_frm_size, 0)
== -1) {

                                                perror("sendto");
                                         }
                                }
                    }

                    temp_pdc = temp_pdc->next;
                } //while 2 ends
                pthread_mutex_unlock(&mutex_pdc_object);

            }
            else
            {
            pthread_mutex_lock(&mutex_pdc_object);

                while(temp_pdc != NULL)
                {
                    //printf("while 3\n");
                     if(!strncasecmp(temp_pdc->protocol, "UDP", 3) && (temp_pdc-
>data_transmission == 0))
                    {
                       /* If STAT Word bits got changed by user */
                         if(temp_pdc->STAT_change != 0)
                         {
```

```
                        switch (temp_pdc->STAT_change)
                        {
                             case 1:
                                      data_frm[14] = 0x04;     //CFG changed
                                      data_frm[15] = 0x00;
                                  break;
                             case 2:
                                      data_frm[14] = 0x80;
                                      data_frm[15] = 0x00;
                                  temp_pdc->STAT_change = 0;
                                  break;
                             case 3:
                                      data_frm[14] = 0x40;     //PMU error
                                      data_frm[15] = 0x00;
                                  break;
                             case 4:
                                      data_frm[14] = 0x10;
                                      data_frm[15] = 0x00;
                                  temp_pdc->STAT_change = 0;
                                  break;
                             case 5:
                                      data_frm[14] = 0x08;
                                      data_frm[15] = 0x00;
                                  temp_pdc->STAT_change = 0;
                                  break;
                        }
                    }

                     /* UDP-Send the newly created data frame to connected PDC address */
                    if (sendto (temp_pdc->sockfd,data_frm, df_data_frm_size, 0,
                                          (struct sockaddr *)&temp_pdc-
>pdc_addr,sizeof(temp_pdc->pdc_addr)) == -1) {
                     printf("send UDP\n");
                             perror("sendto");
                      }
               }
               else if(!strncasecmp(temp_pdc->protocol, "TCP", 3) && (temp_pdc-
>data_transmission == 0))
               {
                           if(temp_pdc->tcpup == 1)
                           {
                            /* TCP-Send the newly created data frame to connected PDC
address */
                                if (send(temp_pdc->sockfd, data_frm, df_data_frm_size, 0)
== -1) {

                                                perror("sendto");
                                        }
                           }
               }

              temp_pdc = temp_pdc->next;
          } //while 2 ends
          pthread_mutex_unlock(&mutex_pdc_object);
       }

         i++;
                clock_gettime(CLOCK_REALTIME, cal_timeSpec1);
```

```
        }
} /* end of function send_data() */
```

Appendix C: MATLAB Latency Code

C.1 Original MATLAB Latency Code – delay.m

```
clear all
close all

%syms x
filename = uigetfile;
data = textread(filename, '%s', 'delimiter', '\n');
%data = textread('7PMU30000.txt', '%s', 'delimiter', '\n');
out_delay = strcat('delay_', filename);
%diary out_delay
N = size(data, 1);
%No = strfind(data, 'No.');
ieee = strfind(data, 'IEEE');
pmu = input('Input PMU IP Address \n', 's');
res_pmu = input('Input PMU Time Base \n');
pdc = input('Input PDC IP Address \n', 's');
%pmu = '192.168.1.254';
%pdc = '192.168.1.2';
%res_pmu = 16777215;
res_pdc = 1000000;
j=1;
k=1;
%res_pdc = {};
%res_pmu = {};

for i=1:1:(N)
    %dframe(1,1)=0;

    if (ieee{i}==1)
        %dframe = strfind(data{i}, 'Data Frame');
        if strfind(data{i}, 'Data Frame')
            for m=(i-7):1:i
                if strfind(data{m},'No.')
                    break
                end
            end
        end
        ip = textscan(data{m+1},'%*s %*s %s %*[^\n]');
        if (isequal(char(ip{1,1}),pmu))
          t1 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
          soc_pmu{j,1} = char(t1{1,1});
          t2 = textscan(data{i+6}, '%*s %*s %*s %*s %d');
          fos_pmu(j,1) = double(t2{1,1})./double(res_pmu);
          %t3 = textscan(data{i+7}, '%*s %*s %*s %*s %*s %d %*s %*s %*s');
          %confnum_pmu(1,1) = t3{1,1};
          t4 = textscan(data{m+1}, '%*s %f %*[^\n]');
          time_pmu(j,1) = t4(1);
          j=j+1;
        end
        if (isequal(char(ip{1,1}),pdc))
```

```
            t5 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
            soc_pdc{k,1} = char(t5{1,1});
            t6 = textscan(data{i+6}, '%*s %*s %*s %*s %d');
            fos_pdc(k,1) = double(t6{1,1})./double(res_pdc);
            %t7 = textscan(data{i+7}, '%*s %*s %*s %*s %*s %d %*[^\n]');
            %confnum_pdc(1,1) = t7{1,1};
            t8 = textscan(data{m+1}, '%*s %f %*[^\n]');
            time_pdc(k,1) = t8(1);
            k=k+1;
        end
    end

end

i=1;
m=1;
while i<=size(fos_pmu,1)
    j=1;
    while j<=size(fos_pdc,1)
        if isequal(soc_pmu{i}, soc_pdc{j}) &&
(fos_pmu(i)<=(fos_pdc(j)+0.001)) && (fos_pmu(i)>=(fos_pdc(j)-0.001))
%isequal(fos_pmu(i), fos_pdc(j))
            time_delay(m,1) = time_pdc{j} - time_pmu{i};
            if time_delay(m)==0
                time_pmu(i)
                time_pdc(j)
            end
            m = m+1;
            break
        end
        j = j+1;
    end
    i = i+1;
end
save(out_delay, 'time_delay', '-ASCII');
med = median(time_delay);
mn = mean(time_delay);
md = mode(time_delay);
median = med*ones(size(time_delay));
mean_a = mn*ones(size(time_delay));
mode_a = md*ones(size(time_delay));
scatter(1:m-1, time_delay);
hold on
plot(median, 'r');
plot(mean_a, 'g');
plot(mode_a, 'y');
xlabel('Data Frame');
ylabel('Time Delay (sec)');
legend('Actual', 'Median', 'Mean', 'Mode');
hold off
```

C.2 Last PMU MATLAB Code – delay_boundary.m

```
clear all
close all

filename = uigetfile;
data = textread(filename, '%s', 'delimiter', '\n');
out_delay = strcat('boundary_', filename);

N = size(data, 1);
%No = strfind(data, 'No.');
ieee = strfind(data, 'IEEE');
num = input('Input number of PMUs \n');
%num = 12;
pmu = cell(num, 1);
port = cell(num, 1);
soc_pmu = cell(1, num);
ip_port_track = cell(num, 2);
res_pmu = cell(num, 1);

for i=1:num
    pmu{i,1} = input(sprintf('Input PMU  %d IP Address \n', i),'s');
    port{i,1} = input(sprintf('Input PMU  %d Port Number \n', i), 's');
    res_pmu{i, 1} = input(sprintf('Input PMU %d time base \n', i));
end
pdc = input('Input PDC IP Address \n', 's');
j=1;
k=1;
res_pdc{1} = 1000000;

for i=1:1:(N)
    dframe(1,1)=0;

    if (ieee{i}==1)
        %dframe = strfind(data{i}, 'Data Frame');
        if strfind(data{i}, 'Data Frame')
        for m=(i-7):1:i
            %if No{m}==1
            if strfind(data{m},'No.')
                break
            end
        end
        ip = textscan(data{m+1},'%*s %*s %s %*[^\n]');
        port_num = textscan(data{m+6},'%*s %*s %*s %*s %*s %*s %*s %*s %*s %s
%*[^\n]');

        for n=1:num
            if (isequal(char(ip{1,1}),char(pmu{n,1})) &&
isequal(char(port_num{1,1}),char(port{n,1})))
              if cellfun(@isempty, soc_pmu(j,n))
                  %ip_port_track{n,1} = pmu{n,1};
                  %ip_port_track{n,2} = port{n,1};
              else
                j=j+1;
              end
```

```
            t1 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
            soc_pmu{j,n} = char(t1{1});
            t2 = textscan(data{i+6}, '%*s %*s %*s %*s %d');
            fos_pmu(j,n) = double(t2{1,1})/double(res_pmu{n,1});
            %t3 = textscan(data{i+7}, '%*s %*s %*s %*s %*s %d %*s %*s
%*s');
            %confnum_pmu(1,1) = t3{1,1};
            t4 = textscan(data{m+1}, '%*s %f %*[^\n]');
            time_pmu(j,n) = t4(1);

          end
        end

        if (isequal(char(ip{1,1}),pdc)) %&& ((isempty(config_check)) &&
isequal(size(dframe), [1 1]))
          if isempty(res_pdc)
              res_pdc = {input('Input PDC time base \n')};
          end
          t5 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
          soc_pdc{k,1} = char(t5{1,1});
          t6 = textscan(data{i+6}, '%*s %*s %*s %*s %d');
          fos_pdc(k,1) = double(t6{1,1})/double(res_pdc{1});
          %t7 = textscan(data{i+7}, '%*s %*s %*s %*s %*s %d %*[^\n]');
          %confnum_pdc(1,1) = t7{1,1};
          t8 = textscan(data{m+1}, '%*s %f %*[^\n]');
          time_pdc(k,1) = t8(1);
          k=k+1;
        end
        end
    end

end

i=1;
m=1;
time_delay = [];
while i<=size(fos_pdc,1)
    for k=1:num
        j=1;
        while j<=size(fos_pmu(:,k),1)
            if (isequal(char(soc_pmu{j,k}), char(soc_pdc{i}))) &&
(fos_pdc(i,1)<=(fos_pmu(j,k)+0.001)) && (fos_pdc(i,1)>=(fos_pmu(j,k)-0.001))
                time_delay(m,k) = time_pdc{i,1} - time_pmu{j,k};

                if cellfun(@isempty, ip_port_track(k,1))
                  ip_port_track{k,1} = pmu{k,1};
                  ip_port_track{k,2} = port{k,1};
                end
                break
            end
            j = j+1;
        end
    end
```

```
%   wait_time(m,1) = time_delay(m,num-(num-1)) - time_delay(m,num);
    i = i+1;
    if size(time_delay,1)>= m
        if (max(time_delay(m,:))>= 0.005)
            m = m+1;
        end
    end
end
if size(time_delay,1)>=m
    if (max(time_delay(m,:))< 0.005)
       time_delay(m,:)=[];
       m = m-1;
    end
else
    m = m-1;
end

scatter(1:m, time_delay(:,1));
hold on
if num>1
    for k=2:num
        scatter(1:m, time_delay(:,k));
    end
end
xlabel('Data Frame');
ylabel('Time Delay (sec)');
legend('PMU1', 'PMU2', 'PMU3', 'PMU4', 'PMU5', 'PMU6', 'PMU7', 'PMU8',
'PMU9', 'PMU10', 'PMU11', 'PMU12');
hold off
```

C.3 Revised MATLAB Latency Code – latency.m

```
clear all;
close all;
format long;

filename = uigetfile;                                                    %
gets the file name
disp('Busy...')
save_out = strcat('PMU_Latency_',filename);                              %
creates variables to be new filenames for data output
save_out2 = strcat('PDC_Latency_',filename);
data = textread(filename, '%s', 'delimiter', '\n');%#ok
%reads the data from the file name given
N = size(data,1);
%size of data
Packet = strfind(data, 'No.');
ieee = strfind(data, 'IEEE');

num = input('\n Input number of PMUs: \n');
pmu = cell(num, 1);
port = cell(num, 1);
```

```
soc_pmu = cell(1, num);
ip_port_track = cell(num, 2);
res_pmu = cell(num, 1);

for i = 1:num
    pmu{i,1} = input(sprintf('Input PMU %d IP Address: ', i), 's');
    port{i,1} = input(sprintf('Input PMU %d Port Number. Dest port: ',
i),'s');
    res_pmu{i,1} = input(sprintf('Input PMU %d Time Base: ', i),'s');
end

pdc = input('Input PDC IP Address: ', 's');
res_pdc = input('Input PDC time base: ');

j = 1;
k = 1;
i = 1;
while (i < N)
    if(ieee{i}==1)
        if(strfind(data{i},'Data Frame'))
            ip = textscan(data{i-6},'%*s %*s %s %*[^\n]');
            port_num = textscan(data{i-1},'%*s %*s %*s %*s %*s %*s %*s %*s
%*s %*s (%[1234567890] %*[^\n]');

            for n=1:1:num
                if(isequal(char(ip{1}),char(pmu{n,1})) &&
isequal(char(port_num{1}),char(port{n,1})))
                    if (isempty(soc_pmu{j,n}))

                    else
                        j = j+1;
                    end
                    t1 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
%gets the second of century
                    soc_pmu{j,n} = char(t1{1});
                    t2 = textscan(data{i+6}, '%*s %*s %*s %*s %f');
%gets the fraction of second
                    fos_pmu(j,n) = double(t2{1,1})/str2double(res_pmu{n,1});
%#ok<SAGROW>
                    t4 = textscan(data{i-6}, '%*s %f %*[^\n]');
%
                    time_pmu(j,n) = t4(1); %#ok
                    break
                end
            end

            if(isequal(char(ip{1}),pdc))
                t5 = textscan(data{i+4}, '%*s %*s %*s %*s %*s %s');
%gets second of century of pdc
                soc_pdc{k,1} = char(t5{1,1});%#ok
%
                t6 = textscan(data{i+6}, '%*s %*s %*s %*s %d');
%gets fraction of century of pdc
                fos_pdc(k,1) = double(t6{1,1})/double(res_pdc);%#ok
%calculates the actual fraction of second from the one gotten earlier using
base
```

```
                t8 = textscan(data{i-6}, '%*s %f %*[^\n]');
%gets wireshark time and store it
                time_pdc(k,1) = t8(1);%#ok
                k = k+1;
            end
        end
    end
    i = i+1;
    percent1 = i/N*100%#ok
end

disp('Busy... \n')
A=size(fos_pdc,1);
i = 1;
m = 1;
time_delay = [];

while i<=size(fos_pdc,1)
    for k = 1:1:num
        j=1;
        while j<=size(fos_pmu(:,k),1)
            if (isequal(char(soc_pmu{j,k}), char(soc_pdc{i})) &&
(fos_pdc(i)<=(fos_pmu(j,k)+0.001)) && (fos_pdc(i,1)>=(fos_pmu(j,k)-0.001)))
                time_delay(m,k) = time_pdc{i,1} - time_pmu{j,k};%#ok

                %                   if time_delay(m)==0
                %                       time_pmu{j,k}
                %                       time_pdc{i,1}
                %                   end
                                if (isempty(ip_port_track(k,1)))
                                    ip_port_track{k,1} = pmu{k,1};
                                    ip_port_track{k,2} = port{k,1};
                                end

                break;
            end
            j = j+1;
        end
    end

    if size(time_delay,1)>= m
        if (max(time_delay(m,:))>= 0.005)
            m = m+1;
        end
    end
    i = i+1;
    percent2 = i/A*100%#ok
end

if size(time_delay,1)>=m
    if (max(time_delay(m,:))< 0.005)
        time_delay(m,:)=[];
        m = m-1;
    end
else
    m = m-1;
```

```
end

save(save_out, 'time_delay', '-ASCII');

delay = time_delay;
delay(delay==0) = inf;

for k = 1:size(time_delay,1)
    delay(k,:) = sort(delay(k,:));
    true_delay(k) = delay(k,1);%#ok
end

save(save_out2, 'true_delay', '-ASCII');

MED = median(true_delay)%#ok
MOD = mode(true_delay)%#ok
MN  = mean(true_delay)%#ok
frame_rate = (k+1)/(time_pdc{k+1,1}-time_pdc{1,1}) %#ok

median_a = MED*ones(1,size(time_delay,1));
mean_a = MN*ones(1,size(time_delay,1));
mode_a = MOD*ones(1,size(time_delay,1));

figure
scatter(1:m, true_delay);
hold on
plot(median_a,'r')
plot(mean_a,'g')
plot(mode_a,'y')
xlabel('Data Frame');
ylabel('Time Delay (sec)');
legend('True Delay(Diff. Ports)','Median','Mean','Mode')
hold off
```