

A QUANTITATIVE STUDY OF MUSICAL INSTRUMENT DIGITAL INTERFACE  
(MIDI) OVER INTERNET PROTOCOL (IP) PROTOCOLS

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

---

James Pate Williams, Jr.

Certificate of Approval:

---

Homer Carlisle  
Associate Professor  
Computer Science and Software  
Engineering

---

Richard Chapman, Chair  
Associate Professor  
Computer Science and Software  
Engineering

---

Saad Biaz  
Assistant Professor  
Computer Science and Software  
Engineering

---

Chung-wei Lee  
Assistant Professor  
Computer Science and Software  
Engineering

---

Stephen L. McFarland  
Dean  
Graduate School

A QUANTITATIVE STUDY OF MUSICAL INSTRUMENT DIGITAL INTERFACE  
(MIDI) OVER INTERNET PROTOCOL (IP) PROTOCOLS

James Pate Williams, Jr.

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama

December 16, 2005

A QUANTITATIVE STUDY OF MUSICAL INSTRUMENT DIGITAL INTERFACE  
(MIDI) OVER INTERNET PROTOCOL (IP) PROTOCOLS

James Pate Williams, Jr.

Permission is granted to Auburn University to make copies of this dissertation at its discretion, upon request of individuals or institutions and at their expense. The author reserves all publication rights.

---

Signature of Author

---

Date of Graduation

## VITA

James Pate Williams, Jr., son of James Pate Williams, Sr. and Avon Jordan Williams was born on December 8, 1953, in LaGrange, Georgia. He graduated from LaGrange High School in 1971. He graduated from LaGrange College with a Bachelor of Arts degree in chemistry in 1979. He attended graduate school at the Georgia Institute of Technology from 1980 to 1983 without attaining a degree. He received a Bachelor of Science degree in computer science from La Grange College in 1994. He was awarded a Masters of Software Engineering from Auburn University in 2000. He has been programming computers virtually continuously since 1977.

DISSERTATION ABSTRACT  
A QUANTITATIVE STUDY OF MUSICAL INSTRUMENT DIGITAL INTERFACE  
(MIDI) OVER INTERNET PROTOCOL (IP) PROTOCOLS

James Pate Williams, Jr.

Doctor of Philosophy, December 16, 2005  
(M.S.W.E., Auburn University, 2000)  
(B.S., LaGrange College, 1994)  
(B.A., LaGrange College, 1979)

410 Typed Pages

Directed by Richard O. Chapman

The research, which is discussed in this dissertation, consists of the development and testing of a suite of ten Transmission Control Protocol (TCP) and reliable Real Time Protocol (RTP) MIDI over IP (MOIP) protocols, and the subsequent implementations of musical duet collaboration systems based on the MOIP protocols. These MOIP protocols were subjected to a quantitative and statistically significant set of experiments using two experimental metrics or performance measurements. The statistical protocol winner of these experiments was used in the duet systems. We implemented the musical duet systems on two different hardware platforms with different and competing operating systems. The general hardware and software architectures of the musical duet collaboration systems were essentially platform independent. The procedural programming language C and the object-oriented programming language Java were utilized. Before a path leading to a modicum of

success was found a number of roads to unsuitable protocols were explored and these lanes to nowhere are also discussed extensively in this dissertation.

Style manual or journal used: Transactions of the Institute of Electrical and Electronics

Engineers Inc (IEEE)

Computer software used: Microsoft Word

## TABLE OF CONTENTS

LIST OF FIGURES.....	ix
LIST OF TABLES .....	xi
CHAPTER 1 INTRODUCTION .....	1
CHAPTER 2 LITERATURE REVIEW.....	5
CHAPTER 3 UNSUITABLE PROTOCOLS.....	30
CHAPTER 4 RTP AND TCP PROTOCOLS.....	70
CHAPTER 5 MUSICAL DUET SYSTEM.....	88
CHAPTER 6 CONCLUSIONS .....	93
REFERENCES.....	98
INDEX.....	100
APPENDIX A LAN EQUATION (1) CHAPTER 4 GRAPHS .....	101
APPENDIX B LAN PAIRED MEANS COMPARISON GRAPHS.....	122
APPENDIX C WAN EQUATION (1) CHAPTER 4 GRAPHS .....	213
APPENDIX D WAN PAIRED MEANS COMPARISON GRAPHS .....	234
APPENDIX E LAN PAIRED MEANS COMPARISON STATISTICS.....	325
APPENDIX F WAN PAIRED MEANS COMPARISON STATISTICS.....	330
APPENDIX G MIDI INSTRUMENTS .....	335
APPENDIX H MIDI INSTRUMENT GROUPINGS.....	336
APPENDIX I MIDI META-MESSAGES AND MIDI CONTROLLERS .....	337
APPENDIX J UT-RTP-ND & UT-RTP-NE VERSUS SN-TCP-ND & SN-TCP-NE...	338
APPENDIX K ATCP-32 VERSUS SN-TCP-ND AND SN-TCP-NE .....	354
APPENDIX L ATCP-40 VERSUS SN-TCP-ND, SN-TCP-NE, AND ATCP-32.....	364
APPENDIX M ATCP-TCP-ND VS SN-TCP-NX AND ATCP-X .....	374
APPENDIX N ATCP-TCP-NE VS SN-TCP-NX, ATCP-X, AND ATCP-TCP-ND.....	385



## LIST OF FIGURES

Figure 2-F-1 RMCP Gateway Model and Its Protocols .....	21
Figure 2-F-2 Young and Fujinaga MIDI Message Format.....	23
Figure 2-F-3 <i>Aura</i> Name Format .....	25
Figure 2-H-1 Classical Time/Space Matrix.....	29
Figure 3-A-1 CW and YF MIDI Message Format.....	31
Figure 3-A-2 CW and YF Datagram Format.....	31
Figure 3-A-3 Trace Route from WAN Client to WAN Server.....	36
Figure 3-C-1 <i>ATCP-x</i> MIDI Short Message Format .....	48
Figure 3-C-2 <i>ATCP-x</i> Datagram Format .....	49
Figure 3-D-1 Musician Registration Dialog .....	60
Figure 3-D-2 Music Studio (House) Metaphor Client .....	60
Figure 3-D-3 Music Studio after a Musician Has Entered a Room .....	61
Figure 3-D-4 Music Room with a Piano Keyboard for MIDI Input.....	61
Figure 3-D-5 Standard Java Open File Dialog .....	62
Figure 3-F-1 MIDI Hardware Configuration.....	68
Figure 3-F-2 MIDI Software Configurations.....	69
Figure 4-A-1 MIDI Short Message Format and Delta-Time.....	72
Figure 4-B-1 Inter-Departure Time and Inter-Arrival Time Temporal Relationships.....	74
Figure 4-C-1 PC-TCP-ND VS SN-TCP-ND $m = 1$ .....	75
Figure 4-C-2 PC-TCP-ND VS SN-TCP-ND $m = 2$ .....	76
Figure 4-C-4 PC-TCP-ND VS SN-TCP-ND $m = 4$ .....	77
Figure 4-C-5 PC-TCP-ND $m = 1$ .....	81
Figure 4-C-7 PC-TCP-ND $m = 3$ .....	82
Figure 4-C-8 PC-TCP-ND $m = 4$ .....	83
Figure 4-C-9 Equation (1) Plot for LAN PC-TCP-ND $m = 1$ .....	83
Figure 4-C-11 Duet System Main Window.....	85
Figure 4-C-12 Duet System Hardware Configurations .....	86
Figure 4-C-13 Duet System Software Configurations .....	87
Figure 5-1 Main Windows.....	90

Figure 5-2 Virtual Keyboard.....	91
Figure 5-3 Channel Map Windows .....	91
Figure 5-4 Peer-to-Peer Duet Architecture.....	92

## LIST OF TABLES

Table 3-A-1 Preliminary Protocols .....	35
Table 3-A-2 Points Awarded to Algorithms and Overall Ranks .....	37
Table 3-B-1 Control and Data Channels and Transport Layer Protocols for RTP .....	38
Table 3-B-2 Partial Protocol Stack for UT-RTP.....	39
Table 3-B-3 TT-RTP ND/NE Relationships ND = Nagle Disabled and NE = Nagle Enabled .....	40
Table 3-B-4 XY-RTP SN-TCP Source Code Files and Lines of Code where X = {T, U} and Y = {T, U} .....	41
Table 3-B-5 RTP and TCP Number of MIDI Short Messages and Packet Lengths in Bytes .....	41
Table 3-B-6 Trippygaia1.mid UT-RTP-ND Actual Runtime Hours .....	42
Table 3-B-7 Trippygaia1.mid UT-RTP-NE Actual Runtime Hours.....	42
Table 3-B-8 Trippygaia1.mid SN-TCP-ND Actual Runtime Hours.....	42
Table 3-B-10 Trippygaia1.mid TNDTND-RTP Actual Runtime Hours.....	43
Table 3-B-11 Trippygaia1.mid TNETND-RTP Actual Runtime Hours .....	43
Table 3-B-12 Trippygaia1.mid TNDTNE-RTP Actual Runtime Hours .....	44
Table 3-B-13 Trippygaia1.mid TNETNE-RTP Actual Runtime Hours .....	44
Table 3-B-14 M=1 Sign of t-Statistic * Statistical Significance.....	46
Table 3-B-15 M=2 Sign of t-Statistic * Statistical Significance.....	46
Table 3-B-16 M=3 Sign of t-Statistic * Statistical Significance.....	47
Table 3-B-17 M=4 Sign of t-Statistic * Statistical Significance.....	47
Table 3-C-1 ATCP-x Source Code Files and Lines of Code.....	52
Table 3-C-2 ATCP-32 Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File..	52
Table 3-C-3 SN-TCP-ND Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File .....	53
<i>Table 3-C-4 SN-TCP-NE Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File .....</i>	<i>53</i>
Table 3-C-5 ATCP-40 Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File..	54
Table 3-C-6 ATCP-TCP Source Code Files and Lines of Code.....	55

Table 3-C-7 ATCP-TCP-ND Ending Date/Time for Trippygaia1.mid Standard MIDI Type 0 File .....	55
Table 3-C-8 ATCP-TCP-NE Ending Date/Time for Trippygaia1.mid Standard MIDI Type 0 File .....	55
Table 3-C-9 Sign of t-Statistic * Statistical Significance m = 4 .....	56
Table 3-C-10 Sign of t-Statistic * Statistical Significance m = 5 .....	57
Table 3-C-11 Sign of t-Statistic * Statistical Significance m = 6 .....	57
Table 3-C-12 Sign of t-Statistic * Statistical Significance m = 7 .....	57
Table 3-C-13 Sign of t-Statistic * Statistical Significance m = 8 .....	58
Table 3-F-1 MIDI Sequence Playing Time Locally.....	68
Table 3-F-2 MIDI Sequence Playing Time on a LAN .....	68
Table 4-A-1 Channel and Transport Layer Protocols for RTP.....	71
Table 4-C-1 Combined Sign of t-Statistic and Statistical Significance Table m = 1.....	78
Table 4-C-2 Combined Sign of t-Statistic and Statistical Significance Table m = 2.....	78
Table 4-C-3 Combined Sign of t-Statistic and Statistical Significance Table m = 3.....	79
Table 4-C-5 Combined Sign of t-Statistic and Statistical Significance Table m = 1.....	80
Table 4-C-6 Combined Sign of t-Statistic and Statistical Significance Table m = 2.....	80
Table 4-C-7 Combined Sign of t-Statistic and Statistical Significance Table m = 3.....	80
Table 4-C-8 Combined Sign of t-Statistic and Statistical Significance Table m = 4.....	81
Table 5-1. Source Code Files and Lines of Code.....	89

## CHAPTER 1 INTRODUCTION

This research was aimed at providing another means for musicians to interact with one another using computers and the Internet. Musical collaboration via the Internet in itself is not a new or novel concept, and as we shall read in Chapter 2, efforts in this direction can be traced to the early 1990s. Within this research we did introduce some previously unknown networking protocols, some of which were unsuitable protocols while others were moderately successful. Another somewhat radical departure from prior research was the idea of playing a duet over a network.

The basic idea behind this research was to find a good, i.e. reliable, networking protocol for transmitting and receiving Musical Instrument Digital Interface (MIDI) data over a TCP/IP network, whether it was a local area network (LAN) or a wide area network (WAN) like the general Internet. Such protocols are known as MIDI over IP protocols, or for short MOIP protocols. In order to realize this dream, we had to explore the space of known MOIP protocols then create some new protocols, and compare the two protocol sets quantitatively. The desired fundamental goal of the research was to be able to conduct a MOIP musical duet on a real network.

In the course of this research many pieces of software were written primarily in two languages, namely, the procedural language C, and the object-oriented language, Java, both of which are declarative rather than functional languages like LISP and its dialect Scheme. An imperative language tends to have syntax closer to regular English

with infix mathematical expressions than a functional language that typically uses the more arcane prefix notation. Our experience was to prefer C over Java due to programming language latency issues, however, as computers become faster and faster and software engineering techniques mature then this negative side effect of Java may disappear. Where ever possible the networking protocols were implemented in both C and Java. Another good language choice for this type of research is C++, which probably has a performance profile much closer to C than to Java on many systems. We developed several new networking protocols in C and Java. Also, we designed and implemented a number of musical collaboration systems in both of the previously mentioned primary languages.

A few operating systems were used for both the qualitative and quantitative aspects of this research. We utilized Windows 98, Windows XP and the UNIX based Apple's OS X. OS X on a Power Mac G4 or G5 system seemed to have the lowest MIDI latency of all the systems in the operating system and hardware suite. In many respects, the Apple MIDI subsystem of the OS X audio system appears more robust than the corresponding Windows multimedia system in the humble opinion of the primary researcher.

This research went down several avenues some of which lead to unsuitable protocols. Applied computer science is more an empirical and exploratory science rather than a rigorous mathematical science, and hence, we as applied computer scientists are more apt than mathematicians to discuss research dead ends. Such discussions are necessary, and in theory, sufficient, to discourage other researchers from going down the same paths that lead to nowhere.

In Chapter 2 we will discuss the foundations of this research which include: MIDI; a competing open architecture specification by the Gibson Instrument Company known as MAGIC; the aspects of TCP and RTP that are important to MOIP; C, Java, and JINI, MIDI systems devised by Apple and Microsoft; other notable MOIP protocols and systems; sources of latency in MOIP based systems; and computer supported collaborative work (CSCW). This chapter comprises a literature review.

Chapter 3 is dedicated to all the research we performed in this project, which resulted in unsuitable protocols. As was stated a little earlier, it is important in science to illustrate research paths that did not pan out or led to dead ends, so that other researchers will not duplicate work that was not completely successful. The first notable failure was an attempt to devise a UDP based MOIP protocol that used the central concept of data redundancy to attempt to correct for the unreliability of the UDP. This minor catastrophe leads us in the direction of the development of reliable protocols later in the course of this research. Another failure, which was a good idea that was attempted on antiquated hardware, was the development of a Java musical collaboration system, which had a music studio metaphor. We used JINI within the framework of another Java musical collaborative system that could also be deemed a failure on some hardware and operating systems. An alternative musical duet system for the Windows platform also was a failure in many senses.

In Chapter 4 we discuss the successful quantitative experiments involving reliable versions of RTP and TCP protocols without or with the Nagle algorithm enabled. The experimental procedure and results are briefly outlined. A number of graphs and tables that represent the data are presented. The two metrics that were used in the experiments

and their subsequent analysis are introduced. The most successful duet system for any platform that we developed is illustrated in Chapter 5. Our conclusions are enumerated in Chapter 6.



## CHAPTER 2 LITERATURE REVIEW

This chapter is divided into six sections that discuss: MIDI; an alternative open protocol stack to MIDI named MAGIC; TCP and RTP; C, Java, and JINI; MIDI systems under two operating systems; previous MOIP research; the latency issues involved in MOIP; and computer-supported cooperative work (CSCW). It is hoped that this short literature review will be all the information required to understand this research at a fundamental level.

### A. MIDI

“MIDI is an acronym for ‘Musical Instrument Digital Interface [1]’. MIDI is analogous to sheet music in that it consists of a set of instructions, which tell an electronic musical instrument how to play a piece of music [2]. MIDI is an electronic musical device and instrument manufacturer standard and is a set of specifications that allows devices and instruments of different makers to communicate with one another using a common digital language [1].

The hardware component of the MIDI specification consists of the definition of MIDI ports, cables, and the electronic signals sent over the cables [1]. There are three different types of MIDI ports in the specification, namely, in, out, and thru ports. Each MIDI port is a female jack to receive the five-pin DIN (Deutsche Industrie Norm) MIDI cable connector [2, 3]. Currently, the specification only uses three of the five pins [2]. Pins 1 and 3 are not used, pin 2 is shielding, pin 4 is grounding, and pin 5 is for MIDI data [4].

MIDI cables are usually no longer than fifty feet and are typically much shorter than the maximum length. The best quality cables have some sort of shielding to prevent

unwanted stray electrical signals from interfering with the MIDI transmission [2]. The MIDI specification calls for the serial digital transmission of messages using a start bit, an octet of eight message bits, and a stop bit. The start bit is a logical 1 bit and the stop bit is a logical 0 bit [1] or vice versa [2]. The send and receive data rates are set at 31,250 baud, which is 31,250 bits per second [1]. This means that 3,125 10-bit MIDI messages can be sent or received each second [1]. This particular baud rate was chosen since it is a divisor of 1,000,000, and 1 MHz was a typical clock frequency for early PCs [3].

MIDI messages can be broken into five different groups: “channel voice messages, channel mode messages, system common messages, system real-time messages, and system exclusive messages” [1]. The first four groups of the preceding listed groups can be categorized as MIDI short messages.

MIDI short messages consist of a status byte and zero, one, or two data bytes [1]. An octet, which is more commonly known as a byte, can have 256 different values. A status byte is in the range 128 to 255 (80H to FFH in hexadecimal) and a data byte is in the range 0 to 127 (00H to 7FH) [1]. This means that a status byte has a one high order bit and a data byte has a zero high order bit [1].

The channel voice group of MIDI short messages consists of: note on, note off, polyphonic key pressure, channel pressure, program change, control change, and pitch bend change [1]. Program change messages have one data byte so there can be 128 instruments active at one time. See Appendix G for a list of the standard instruments. The instruments are organized into sixteen groups of eight instruments per group. See Appendix H for the names of the groups. The channel mode group of MIDI short messages is comprised of: local control, all notes off, omni mode off, omni mode on,

mono mode on, and poly mode on [1]. A good introduction to control change messages can be found online [5]. The system common messages are: song position pointer, song select, time request, and EOX (end of exclusive) [1]. The system real-time messages are: timing clock, start, stop, continue, active sensing, and system reset [1]. The system exclusive messages are sometimes used to transfer parameter settings from one MIDI enabled device to another that are both by the same manufacturer such as Yamaha or Roland.

MIDI controllers include drum controllers, guitar controllers, keyboard controllers, and wind controllers [2]. A drum controller is vastly different from actual drums and consists of one or more pads. A guitar controller is usually retrofitted to a standard electric guitar via special pickups [2]. Wind controllers are usually specially designed wind instruments that resemble a futuristic clarinet [2].

There are three MIDI file formats. Format zero files consist of a single track. Format one files consist of a number of tracks to be played simultaneously. Format two files consist of a number of tracks to be played independently [5].

MIDI files contain chunks. Each chunk has a type that is four octets in length, a length that is four octets, and data that has length octets. There are two types of chunks a header chunk and a track chunk. A header chunk has the type "MThd" and a track chunk has the type "MTrk". A header chunk has length equal to six. The data in a header chunk consists of format, tracks, and division each of which are sixteen bits in length and are in big endian (most significant octet first) format. The format can be zero, one, or two. If the high order bit of the division is zero then the division is the number of ticks per quarter note. If the high order bit of division is one then the bits fourteen to eight are the negative

of the number of frames per second and bits seven to zero are the ticks per frame. The track chunk consists of length MIDI events. A MIDI event consists of delta-time in ticks and either a sysex-message, meta-message or a MIDI short message. The possible meta-messages are given in Appendix I [5]. A few good online sites for general MIDI discussions are [6-8]. There is an excellent source of information on MIDI programming using the Java language in [9] by Sun Microsystems.

## B. MAGIC

Gibson Guitar Corporation has proposed an alternative musical instrument digital interconnection technology known as Media-accelerated Global Information carrier (MAGIC) to replace as well as incorporate the aging Musical Instrument Digital Interface (MIDI, 1983) standard [10]. According to the specification the motivations behind the development of the MAGIC protocol stack were as follows: “enhanced real-time sonic fidelity, interoperability, complete digital solution, simple installation, and ease-of-use” [10]. A MAGIC link is bi-directional and carries fixed-length data and control information as well as power in real-time [10].

MAGIC is able to transmit up to 32 channels of up to 32-bit audio at sampling rates of up to 192 kHz [10]. This is much better than CD quality audio, which consists of two channels of 16-bit audio at a sampling rate of 44.1 kHz. Since MAGIC is based upon the IEEE 802.3 Ethernet standard that has a baud rate of 100 Mbps, you are probably limited to sampling rates of 100 kHz if you are using all 32 channels and 32-bit data. However, this is a definite improvement over CD quality audio.

Gibson hopes that amplifier, instrument, and guitar effects manufacturers will readily embrace the MAGIC technology so that its adoption and usage will become universal.

Currently, Gibson is mainly an instrument manufacturer with the Baldwin line of pianos, the Gibson line of acoustic and electric guitars, and the Goldtone line of tube guitar amplifiers. Perhaps Gibson has the political savvy and clout to push through a standard to a diverse and highly competitive set of manufacturers. Gibson seems fairly committed to MAGIC since Gibson's current Chief Executive Officer (CEO), Henry Juskiewicz, was instrumental in the development of MAGIC.

Many instruments that musicians use today are either analog or require a lot of analog to digital (A/D) conversion or digital to analog (D/A) conversion. A/D and D/A conversions introduce latencies into a performance. These conversions introduce latencies of 3,000 to 10,000 microseconds [10]. Most modern recording equipment is digital. Gibson wants to create a totally digital environment.

It is very easy to connect a computer with an IEEE 802.3 Ethernet standard network adapter to a local area network (LAN). It is Gibson's vision that connection of amplifiers, instruments, and effects to a musical network will be as seamless as connecting a computer to a LAN.

There exists a definite cable snarl problem in performance and recording environments. This problem can cause performers and stagehands to trip over the mass of cables, interference between power carrying and signal carrying cables, and it is hard to determine if everything has been correctly connected. Gibson wants to overcome these difficulties by having each musical instrument, amplifier, or effect with at most two cables, an external power cable for devices that require more than 9 volts direct current and a MAGIC Ethernet cable. Most MAGIC compliant devices will only require the Ethernet cable. There also exists a wire snarl problem in home entertainment centers and

this problem could be in theory eliminated by the adoption of MAGIC technology by that industry.

MAGIC supports both the daisy chain and star network topologies that are popular in the MIDI world [10]. It also supports what is known as an uplink topology that consists of two switching hubs that are connected by a high-speed link [10]. This high speed link between star networks could be Gigabit Ethernet [10]. A switching hub multiplexes links from more than one device or daisy chain network [10].

The protocol stack consists of a physical layer, data link layer, and MAGIC application layer [10]. The physical layer and data link layers are compatible with IEEE 802.3 Ethernet protocol physical layer and data link layer [10]. The MAGIC application layer encapsulates its data and control information in IEEE 802.3 Ethernet data link frames [10]. Other well-known protocol stacks are the Open Systems Interconnection (OSI) reference model and the hybrid reference model introduced by Tanenbaum [11]. The OSI reference model consists of seven layers namely, a physical layer, a data link layer, a network layer, a transport layer, a session layer, a presentation layer, and an application layer [11]. The hybrid protocol stack has five layers: physical layer, data link layer, network layer, transport layer, and application layer [11].

MAGIC uses Category 5 cables and RJ-45 connectors [10]. Four of the conductors in a Category 5 cable are used for data transport and the other four are used to carry power [10]. The cable is capable of carrying at least a 9-volt direct current power supply over distances up to 328 feet [10].

The IEEE 802.3 Ethernet frame format consists of a preamble of 7 bytes, 1 byte frame delimiter, 2 or 6 byte destination address, 2 or 6 byte source address, 2 byte length of data

field, 0 to 1500 byte data, 0 to 46 byte pad, and 4 byte checksum [11]. The check sum is a cyclic redundancy code (CRC) [10, 11].

Each MAGIC application layer packet consists of 32, 32-bit data slots of 16, 24, 28, or 32 bits of Pulse Code Modulation (PCM) audio [10]. These slots can also carry arbitrary 32-bit data [10]. MIDI protocol data can be encapsulated in a packet [10]. MAGIC is similar to the Synchronous Optical Network (SONET) [11] in that it requires a system-timing master (STM). The STM is chosen using a device enumeration protocol and the process is automatic [10]. “The default MAGIC frame timing is 48 kHz with an acceptable tolerance of 80 picoseconds. This timing is locally generated by the STM, and recovered and regenerated by all other devices. The Ethernet signaling rate is asynchronous with the rate at which frames are transmitted [10]”.

The main competitors to MAGIC are the MIDI standard and IEEE 1394 FireWire standard [12-14]. MIDI devices are only able to transmit and receive control information rather than raw audio data. MIDI control messages are like sheet music telling a synthesizer what note to sound etc. FireWire is a high-speed serial bus for computer data and audio/visual data communications. FireWire does carry power for low powered devices just like MAGIC. FireWire has a higher baud rate than the current vision of MAGIC over the 100 Mbps Ethernet and the IEEE 1394b has cable lengths that equal the maximum MAGIC cable length. There is a MIDI media adaptation layer for IEEE 1394 [15].

According to the video on the MAGIC web site [16], MAGIC would be useful as a transmitting and receiving medium for telemetry from state-of-the-art medical scanning devices such as Computer Aided Tomography (CAT), Magnetic Resonance Imaging

(MRI), and Positron Emission Tomography (PET) scanners. Using a MAGIC network, medical students could view in real-time scanner data transported from a diagnostic room to a classroom. So, Gibson views MAGIC in a larger context than just in the musical performance and recording world.

It appears to this researcher that MAGIC might have some promise in the home entertainment sector; however, this area may wind up being dominated by IEEE 1394 devices since Apple, Intel, and Microsoft support this particular standard. Gibson does not have an industrial presence in the home entertainment market so this market may be lost to other standards.

Some criticisms of the MAGIC proposal are that possibly Gibson does not have the corporate strength in the instrument market to push a standard onto the rest of the industry. Also, musicians tend to be extremely conservative with respect to their instruments, so widespread acceptance of the standard might be an opium pipe dream. MAGIC does have a lot of promise as a musical recording studio standard. MAGIC compliant digital guitars could be very useful in a recording environment. Many guitar effects are becoming digital so removing the A/D and D/A conversions currently required could be advantageous. Most modern audio recording equipment is digital. A good online summary of MAGIC can be found in [17].



### C. TCP and RTP

There are essentially two architectural reference models in the networking world, namely, the Open Systems Interconnection (OSI) reference model and the TCP/IP reference model. The OSI reference model has seven layers: the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the application layer. The TCP/IP reference model has four layers: the host-to-network layer, the Internet layer, the transport layer, and the application layer [11].

The TCP/IP reference model has two transport layer protocols, the User Datagram Protocol (UDP) [18] and the Transmission Control Protocol (TCP) [19]. UDP is a connectionless, best effort, and unreliable transport protocol. TCP is a connection-oriented and reliable transport protocol. TCP uses sequence numbers and acknowledgements to insure that each packet is delivered in the order sent. UDP typically involves a lesser amount of overhead than TCP. The primary reason for lost data-grams or packets on the Internet is congestion which can be contrasted with the wireless universe where the primary culprit for dropped data-grams or packets is the high bit error rate (BER) of the medium [11].

The prototypical programming-paradigm of TCP/IP is the client/server model. A client sends a request and the server answers the request. Servers offer services and these services could be as simple as an echo service or time of day service or as complicated as a database search service. Servers can be either UDP or TCP servers or both. Servers can be either concurrent or iterative. A concurrent server usually uses independent threads of

execution for each request and hence can handle more than one simultaneous request. An iterative server handles one request at a time [11, 20].

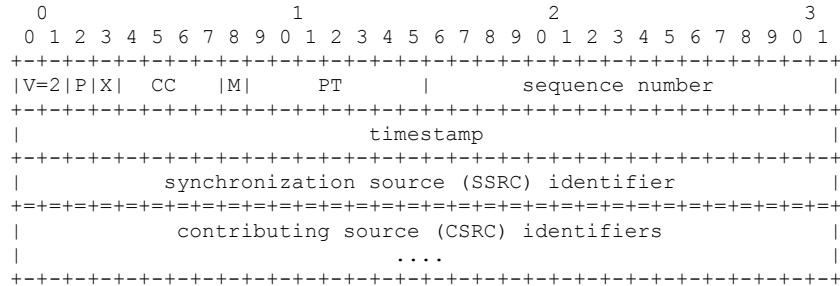
Many consider the BSD UNIX sockets library as the quintessential programming package for TCP/IP. This package provides an interface known as a socket that connects the two ends of the client/server interaction. Sockets can be UDP sockets or TCP sockets. Sockets can also support broadcasting or multicasting [20-22].

The client side of the interaction consists of creating a socket, connecting the socket, writing a request to the server, reading the response from the server, and closing the socket. On the other hand, the server side of the interaction for a TCP server involves the following steps: creating a socket, binding the socket to a port number, listening for connections, accepting the connection, reading a client's request, and writing the response to the request, and closing the accept socket [20].

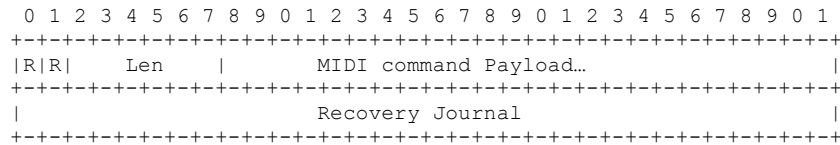
The Real Time Protocol (RTP) is designed to transmit data such as audio or video in real-time. Some of the early applications of RTP were audio and video conferencing over the Internet. RTP does not guarantee delivery or in order delivery of packets since the Internet version is based on UDP, which is an unreliable protocol. RTP does not give quality of service (QoS) assurances either [23].

RTP has a control protocol associated with it named RTCP. Usually on the Internet, RTP uses UDP for sending data-grams and for the control protocol. The RTP session has a destination IP address and destination IP port number. Typically, Internet implementations use an even port number for UDP transport and an adjacent odd port number for the RTCP port, such as 50000 and 50001. This researcher questions the use of UDP for the RTCP rather than a reliable transport protocol such as TCP. However, the

usage of UDP for both functions does not make sending and receiving data-grams or control information seamless by using `sendto` or `recvfrom` for both types of data. The RTP header has the following format [23]:



V is the two bit version field, P is the padding bit and if set the payload contains padding octets, X is the extension bit and if set the header is followed by one extension, CC is the four bit number of CSRC identifiers in the header, M is the marker bit defined by the profile, PT is seven bit payload type, the sequence number is initially random and has sixteen bits, timestamp is sampling instant of the first octet in the RTP packet, the synchronization source is chosen randomly, and the contributing source identifiers are the contributors to the RTP payload. The MIDI payload used by Lazarro et al. in their NMP system, which is mentioned in the Section F has the following format:



R are reserved bits, the Len field is six bits and is the length of the MIDI payload in octets, and the Recovery Journal is checkpoint information that allows for retransmission of lost data.

#### D. C, Java, and JINI

The computer programming language C was derived from the type-less language BCPL through the B language and is most often associated with Dennis M. Ritchie and the development of the UNIX operating system and came into existence in the period from 1969 to 1973. The C language falls into the class of imperative computer languages that includes FORTRAN and ALGOL [24], [25]

The BCPL computer programming language was a popular systems programming in the United Kingdom in the late 1960s and is the grandparent of C. The small footprint variation of BCPL, the B programming language was developed to run on Digital Equipment Corporation's (DEC's) PDP-7 minicomputer and only occupied 8K (8192) bytes of memory. B can be thought of C without data types and is considered C's immediate parent. Ken Thompson of Bell Laboratories is the inventor of the B language and its name was probably derived from the name of another programming language developed by Thompson named Bon [24].

C has some features that are very close to assembly language such as register variables and easy bit and byte manipulation. The explicit pointer and its accompanying arithmetic, which were once such a boon in programming quickly became somewhat of a bane and later completely disappeared from a programming language with the invention of the language Java. Pointers are an integral part of the C language and the only means of passing a reference to a subprogram which in C are called functions. Perhaps a drawback of C is the fact that functions can't be nested. C introduced a means for easy modularization or packaging of code and the module or package interface is known as a

header file. Decomposing a problem into modules and then functions laid the foundations for a later software engineering paradigm shift known as object-oriented programming (OOP) [25]. C is known as a weakly typed programming language.

At the Sun-World Conference on May 23, 1995, John Gage, director of the Science Office for Sun Microsystems, and Marc Andreessen, cofounder and executive vice president of Netscape announced to the audience and thus the world that the Java language existed and was to be utilized by the Netscape Navigator, which was then the most popular web browser. Only a small number of people, less than thirty, were responsible for the invention and introduction of Java technology [26].

Sound and MIDI support became available in Java rather late with Java 1.3. A glaring deficiency was the fact that only MIDI output devices were supported on many popular platforms such as the Windows platform. This problem has been rectified in 2004 with the 1.5 version.

Java is a platform independent and interpreted language [27]. Other interpreted languages include the Scheme functional language, which is the language of choice for some programming languages courses and many artificial intelligence applications [28], and Microsoft's C# language [29-30]. Java has been scaled down for use on palm tops and cellular telephones [31-32].

JINI is the name of a technology invented by Sun Microsystems in the 1990s and was publicly announced in 1998. It is a set of engineering specifications and Java code that allow computers to discover and utilize services on a network. It is similar to a distributed object naming and lookup service. The whole notion is in the standard Java tradition of potentially computing on small-embedded devices. Sun had a vision that

perhaps JINI would be a glue to hold together the embedded systems networks of automobiles and other transportation vehicles [33-34].

JINI has five key concepts that are: discovery, lookup, leasing, remote events, and transactions. Services need to be discovered by JINI-aware devices before they can be used. The discovery protocols consist of the multicast request protocol, the multicast announcement protocol, and the unicast discovery protocol. Lookup is a type of name server, but has a much richer set of semantics due to the underlying object oriented language of JINI, namely, Java. Lookup can be used to find certain types of supported objects using the inheritance hierarchy of Java. A central feature of JINI is the notion of downloadable proxies. A lookup service has an object named a service item that has a proxy object and attributes objects. A client downloads the proxy object from the lookup service and then communicates via the proxy with the service perhaps using the Remote Method Invocation (RMI) mechanism of Java. Leasing allows the detection of crashed client and services, since consumers and services are expected to renew their leases periodically. Remote events are remote asynchronous notifications which build on the idea of local events inherent in the Java language. Transactions come from the database universe. Transactions have four properties, which are sometimes represented by the mnemonic ACID, which stands for atomicity, consistency, isolation, and durability. The transaction protocol used by JINI is the two-phase commit [33-34].

#### E. MIDI Systems of Interest

The MIDI subsystem of the Microsoft Windows multimedia system has a function for determining the capabilities of a MIDI device, which can be an input or output port, a sequencer, or a synthesizer. A MIDI device is either an input device or an

output device. There exist functions for opening either a MIDI input device or a MIDI output device, which return a handle to the MIDI device. A MIDI input device can be started or stopped. Both types of devices should be stopped and closed before the program that opens the devices is exited. When you open a MIDI input device a callback function or window must be specified. Under Windows MIDI short messages are represented by double words, which are 32-bit entities. There is a lot more information on the features of the Microsoft's MIDI system in the online help of Microsoft's Visual C/C++ and Visual Studio .Net.

The architecture of the MIDI system under OS X is quite elegant and consists of a MIDI server which is built upon the MIDI driver layer which, in turn is over the I/O subsystem of the OS X kernel. Each application that desires to receive or transmit MIDI data must create a MIDI client, a MIDI destination or source, and an input or output port. MIDI messages are called MIDI packets and are encapsulated in a structure that has the unsigned integer field length, MIDI timestamp, and length data bytes. MIDI running status is not supported in the current MIDI packet structure. Another structure called a MIDI packet list allows more than one MIDI event (MIDI packet) to be transferred at one time in the system. There are functions for initializing a MIDI packet list, adding packets to the list, and iterating through the list elements. Complete descriptions of the functions and properties of the OS X MIDI system are given in [35].

#### F. Prior MOIP Research

In this section we discuss four previous studies of remote collaboration between musicians, namely, the Remote Music Control Protocol (RMCP) [36], the Young and Fujinaga version of MIDI over IP [37], the Aura system [38], and the Network Musical

Performance (NMP) system [39]. The first two systems are based on UDP, the third TCP, and the fourth on UDP and RTP.

As has been previously stated RMCP is connection-less and based on UDP/IP. Since it uses UDP, broadcasting is available without the overhead of multiple transmissions [36]. RMCP was originally intended for a lossless network such as some Ethernets since it does not have a mechanism for loss or out-of-order data-grams. Between 1992 and 1997, five systems using RMCP have been developed which are described in this paragraph. (1) A virtual dancer that is choreographed by musicians in real-time. (2) A virtual jazz session between a pianist, a bassist, and a drummer with accompanying computer graphics for gestures. (3) Multiple musicians interacting via the Ethernet. (4) Improvisation, a system in which two untrained people can create improvisational music and interact with each other. (5) RemoteGIG, a remote session over the Internet between musicians [36].

RMCP is based on the client/server architectural model of the Internet. There are four servers, specifically, the sound server which transmits “MIDI messages of received packets to a MIDI instrument”, the display server which “visualize MIDI messages of received packets in the form of a piano keyboard”, the animation server which “generate music-driven real-time computer graphics corresponding to received packets”, and the recorder server which “record all received packets with the received timestamps, in a RMCP Packet Record File”. There are four types of RMCP clients, in particular, the MIDI receiver which “receive MIDI from MIDI instruments”, the MIDI station client which “substitute a computer keyboard and mouse for a MIDI keyboard instrument”, the



Standard MIDI File (SMF) player client which plays “a standard MIDI file”, and the player client which plays “a RMCP packet record file” [36].

RMCP requires distributed clock synchronization. The system has one RMCP time synchronization server. RMCP packets either have a timestamp or no timestamp. If the timestamp is not present the message in the packet is executed as soon as the packet arrives [36].

RMCP was originally designed for use on a reliable LAN. The extension to a WAN involves using RMCP gateways that connect two LANs using TCP/IP, the reliable and connection-oriented Internet transport protocol [36]. See Figure 4.1 on the next page for a visualization of the RCMP system and its networking connections and protocols.

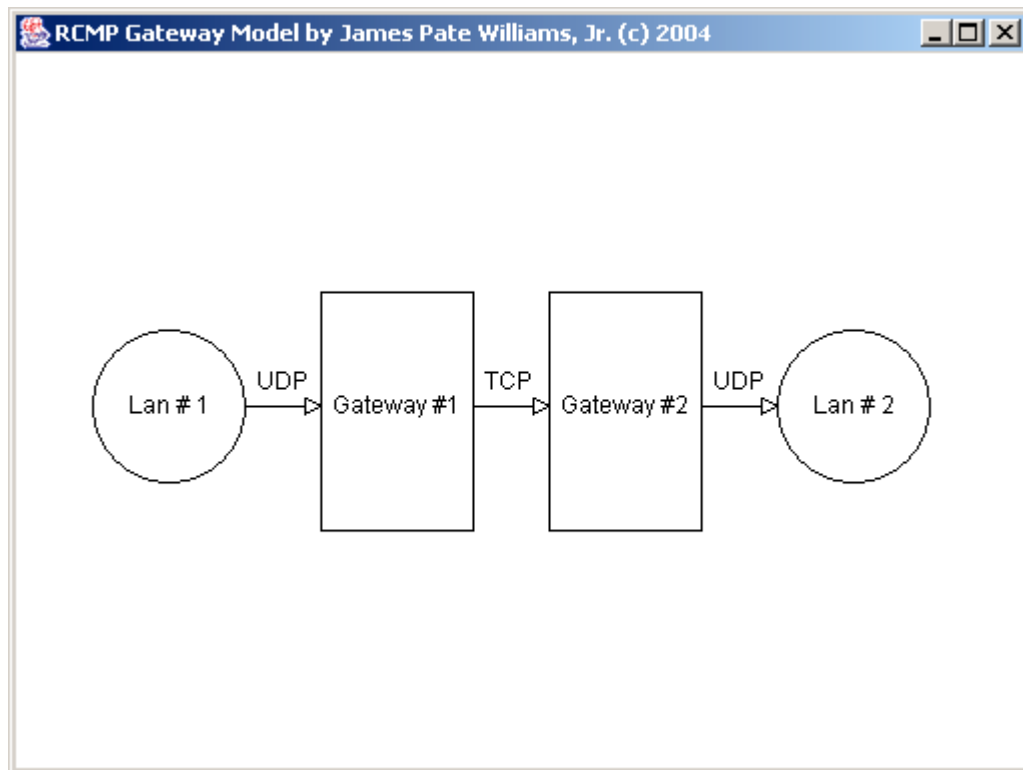


Figure 2-F-1 RCMP Gateway Model and Its Protocols

The duet system developed by this researcher is reminiscent of RMCP. The similarities include a piano keyboard for visualizing MIDI data that comes over the network and both systems can play sequences over the network. The differences between the two systems are that RMCP has more MIDI musical visualization aids and the networking protocols are not the same. RMCP uses a combination of UDP and TCP for MIDI data transport whereas the duet system uses TCP and a homegrown protocol based on TCP. Both systems are capable of sending and receiving standard MIDI type 0 or type 1 file and of recording a session. RMCP uses the predominant Internet client/server architecture whereas the duet system uses the new peer-to-peer (P2P) Internet paradigm.

There are essentially two methods of transmitting music over the Internet. The first method is to transfer audio data via the Internet. The second method is to transmit musical gesture information such as the data encapsulated in the MIDI specification. Apple, Microsoft, Sun, et al. commercial software vendors have been working on systems for streaming audio for music and for teleconferencing. Streaming audio requires relatively large bandwidth, in order to sound reasonably good uses an initial buffering mechanism, and there can be pauses in the audio stream. Typically streaming audio requires two different protocols: one for a low bandwidth connection and one for a high bandwidth connection.

MIDI and standard written notation are universal representations of musical gesture. MIDI can be good and faithful for a piano performance. Sending MIDI messages to remotely perform on an instrument can create a unique remote performance environment. MIDI requires less bandwidth than streaming audio [37].

Young and Fujinaga chose UDP as their basic transport protocol. To overcome the unreliability of UDP, they transmitted multiple copies of each MIDI message. A unique index was used with each message to ensure that duplicates were discarded and messages were played in the right sequence. They also used a buffer of a few seconds, which makes real-time musician-to-musician interaction virtually impossible [37].

Young and Fujinaga cite three reasons for not utilizing the reliable Internet transport layer protocol TCP. These reasons were retransmissions and their associated latencies, in order delivery of packets and the required latency to enforce this policy, and the extra bandwidth for reliability. They were particularly concerned with the stopping and starting of the music due to retransmissions. This researcher encountered the starting and stopping TCP problem with most sequences, but the fault is not so prevalent in the duet system where there is a somewhat limited amount of data being transported on the network. Although Young and Fujinaga did not give the exact number of bits used in their datagram format, this researcher interpreted their description as follows in Figure 2-F-2. A datagram consisted of one or more MIDI messages.



Figure 2-F-2 Young and Fujinaga MIDI Message Format

Richard O. Chapman and this researcher developed a protocol that uses some redundancy and does not require buffering of data-grams before playback. We call this protocol the CW protocol. The idea is to send multiple copies of MIDI messages spread

over several data-grams to ensure delivery of most of the MIDI messages. We describe this protocol in detail in Chapter 7.

Local area networks (LANs) provide an economical and high-speed means of connecting personal computers (PCs). MIDI networks use one specialized transmission protocol whereas LANs may use many different digital protocols. Dannenberg and van de Legeweg built a system named *Aura* at Carnegie Mellon that takes advantage of low cost LANs for implementing real-time music programs [38].

At first Dannenberg and van de Legeweg used UDP as their transport level protocol since it seemed good for real-time applications and seemed relatively reliable on LANs. Previous research by Goto et al. into RMCP was done using UDP. However, Dannenberg and van de Legeweg were getting dropped data-grams on their LANs so they switched to TCP. The key ideas for this researcher to come from the *Aura* work are the usage of TCP\_NODELAY to disable the Nagle algorithm and the utilization of multiple threads of execution [38]. By disabling the Nagle algorithm TCP does not delay until it has certain minimum size packet to transmit, instead the protocol sends smaller packets at more frequent intervals [11].

*Aura* is a distributed system for communicating musical data in real-time. It uses the object oriented programming paradigm. The *Aura* system consists of spaces, zones, objects, and names. A machine is a space or address space. A space consists of one or more threads of execution. A zone is a collection of objects that are shared by a single thread of execution within an address space. There can be as many zones in an address space as there are threads. Objects are entities that can transmit and receive musical data in the form of asynchronous messages. Objects are differentiated by their real-time

requirements with low latency objects going into a particular zone with other such objects. Names are unique 64-bit integers with the format given in Figure 5-3. And this figure represents the end of the paragraphs on the third previous research system.

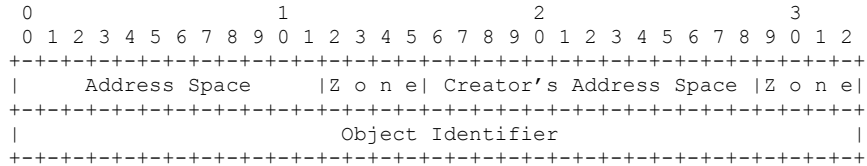


Figure 2-F-3 *Aura* Name Format

*Aura* appears to be a much more general and versatile system for asynchronous communication of musical data than this researcher’s vision of MOIP. However, with this generality and versatility there are prices to be paid such as code and system complexity and clock synchronization issues. This researcher used the object-oriented programming paradigm utilizing Java for the code to compare the different MOIP transport level protocols introduced in this dissertation, however, straight C was used the final duet system since it is closest to the Microsoft Windows native Application Programming Interface (API).

A network musical performance (NMP) occurs between musicians that are playing musical instruments at different locations that are connected by a computer network. Ideally, we would like to use real-time audio to send the actual music that a given musician is playing, however, bandwidth considerations and latencies may make this impossible. The next best thing is to use musical gestures such as MIDI. Lazzaro and Wawrzynek used RTP to send MIDI commands over the Internet [39].

There are two classes of delay in NMP, namely, network delay and local delays. Network delay on the Internet is usually associated with congestion. The local delays

include “computational delay, audio and control I/O delay, and perhaps local acoustic delay.” Via minimization of each of the preceding type of latency perhaps a viable NMP system can be achieved [39].

The NMP system uses the standard client-server architecture that is so prevalent in many Internet applications. The NMP client used the IETF Real Time Protocol (RTP) under the Audio/Video Profile (AVP) [28] to transfer MIDI data between network endpoints. A mirror server was also developed to reflect the gesture information back to the client [39].

The NMP researchers used a recovery journal mechanism that is similar to forward error correction (FEC) and reliable multicast transport (RMT). The researchers noted three qualitative artifacts associated with their attempts to build error resilience and reliable into the system:

1. Occasionally a depressed key does not create a corresponding note
2. Noticeable jitter in the sounding of notes
3. A released key sometimes continues generating sound for a fairly short time.

The NMP research team measured late and lost packets on a relatively high speed California instate network. Their late and lost packet data had a bi-modal distribution [39]. The time 12:30 PM was very good and the time 7:30 PM was very bad. It is common knowledge that Internet Service Providers (ISPs) peak times are 7:00 PM to 10:00 PM local time.

#### *G. Sources of Latency in MOIP*

There are five sources of latency in distributed audio systems: the finite speed of sound in air, the network, the operating system, the sound card, and the implementation

language. In the following paragraphs we will give a little discussion of the preceding forms of latency.

The speed of sound in air is given approximately by the following formula:

$$v = 331.4 + 0.6T \text{ (m / s)}$$

The temperature,  $T$ , is in Celsius [40]. For  $T = 21.11$  degrees Celsius = 70 degrees Fahrenheit  $v = 344.28 \text{ m / s} = 1129.52 \text{ ft / s} = 1.13 \text{ ft / ms}$ . So a percussionist who is fifty feet from the violin section in an orchestra would experience a delay of about forty-four milliseconds in the sound of the violins using the previous data. Musicians are accustomed to latencies of the order of ten milliseconds or about the delay that occurs between pressing a key on a MIDI keyboard and getting an audio response. Supposedly some very gifted individuals are able to detect delays of the order of one millisecond [41].

Network latency is hard to quantify. There have been some papers such as [42] that attempt to model TCP latency; however, it is hard to deduce ballpark estimates of the TCP latency in a distributed audio system from these models. Due to retransmissions and the enforcement of the receive in-order policy TCP has a greater latency on the general Internet than UDP. In the absence of retransmissions and out-of-order data-grams in this researcher's experience TCP and UDP have similar latencies.

In a comparison of the latencies in off-the-shelf operating systems for audio systems comparing Windows 95, NT 4, Windows 98, and NT 5, it was found that Windows 98 had the lowest worst-case latency of about twelve milliseconds. Windows 95, NT4, and NT5 had worst-case delays of around fifty milliseconds [43].

Sound card latency is most pronounced when recording audio from one of the input-sources on the sound card into software being executed on the computer. Some recording software such as SONAR 3 by Cakewalk allows this latency to be reduced with degradation of the quality of the recorded sound [44]. Another way of reducing sound card latency is to purchase a high-end sound card.

All implementation languages have associated latencies. An interpreted language such as Java is expected to have a greater inherent latency than a compiled language such as C since interpretation involves conversion from an intermediate language to native code, whereas the C compiler outputs native code.

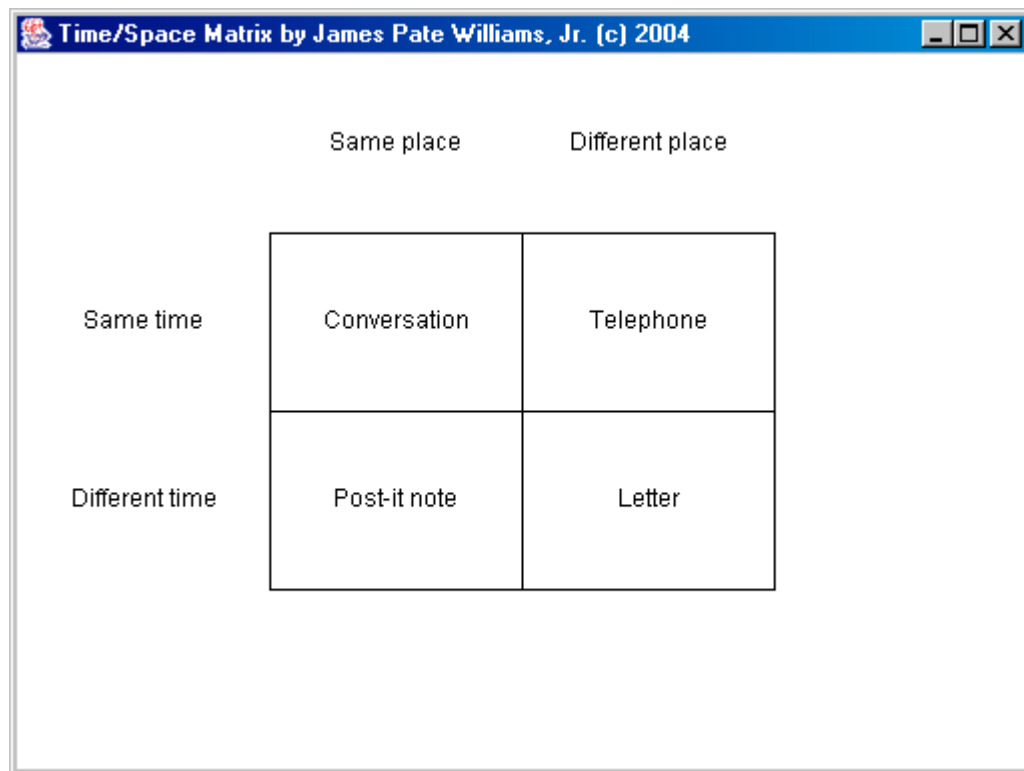
#### *H. Computer-Supported Cooperative Work (CSCW)*

Human-computer interaction (HCI) involves psychology and the computer, whereas CSCW is more related to sociology and the computer. However, CSCW generally comes under the auspices of HCI in the scientific literature [45]. Groupware is the common name given to software, which allows the interaction of two or more individuals via computers [45].

Groupware can be differentiated according to the standard time/space matrix whose axes consist of time that is divided into same time or different time and space that is divided into same place or different place [45]. An alternative formulation uses the time axis labels synchronous (same time) and asynchronous (different time) and space axis labels co-located (same place) and remote (different place) [45]. Examples of groupware include extreme two programmer programming teams (synchronous and co-located), chat also known as instant messaging (synchronous and remote), electronic or classical bulletin boards (asynchronous and co-located), and email (asynchronous and



remote). Figure 2-H-1 shows a classical time/place matrix in which by conversation we mean face-to-face conversation [45].



	Same place	Different place
Same time	Conversation	Telephone
Different time	Post-it note	Letter

Figure 2-H-1 Classical Time/Space Matrix

## CHAPTER 3 UNSUITABLE PROTOCOLS

This chapter is divided into six sections that give synopses of the initial research that resulted in failures to achieve the desired goals. The sections are: a discussion of the preliminary set of protocols, an outline of the initial quantitative and statistically significant collection of experiments that is divided into two sections, a brief description of an initial collaborative system, and finally an adumbration of another Java and JINI duet system which did not pan out due to latency problems and an alternative duet system.

### *A. Preliminary Protocols and Their Implementations*

The protocols used in the preliminary set of experiments were a UDP based protocol with a modicum of redundancy developed by Professor Richard O. Chapman and this researcher (*CW*), a RTP based protocol without error recovery (*RTP*), a simple and naive TCP protocol (*SN-TCP*), and Young and Fujinaga's UDP based protocol (*YF*). In this chapter we will present brief outlines of the protocol, talk about their implementation, and finally, discuss a set of experiments performed on a Wide Area Network (WAN).

As far as MIDI data was concerned, only MIDI short messages such as channel pressure, control change, key pressure, note off, note on, pitch-bend, and program change were transmitted and received. MIDI meta-messages were not included in the transmission stream. MIDI meta-messages such as tempo changes can be incorporated

into the protocols. The MIDI short message data was placed in a structure consisting of a unique index, a delta time in MIDI ticks, and the MIDI short message. This structure was based on the structure that Young and Fujinaga utilized and is shown in Figures 3-A-1 and 3-A-2.

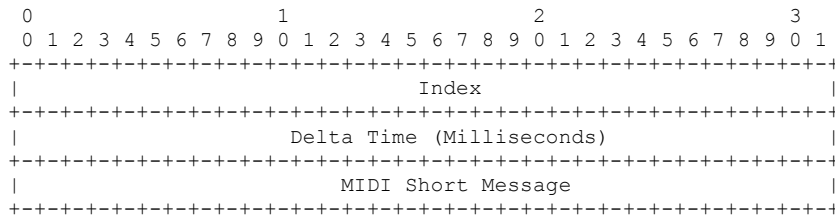


Figure 3-A-1 CW and YF MIDI Message Format

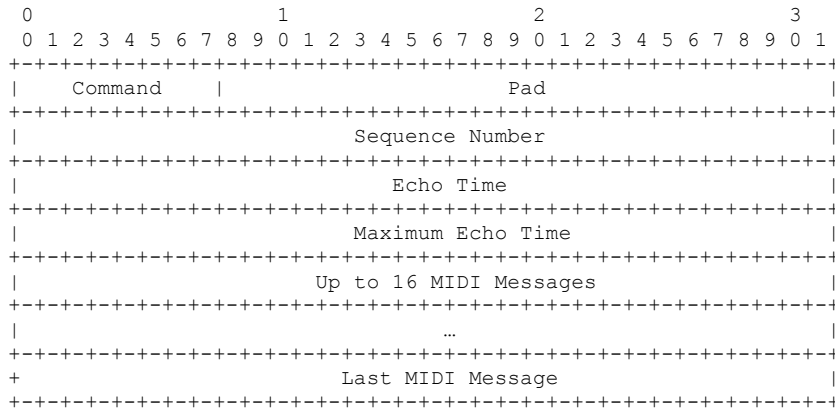


Figure 3-A-2 CW and YF Datagram Format

The command is either OPEN (0) or CLOSE (1), which either opens or closes the TCP connection between the client and server. The sequence number is unique to each UDP client/server connection and limited by the precision of a 32-bit integer. The echo time and maximum echo times are calculated by the client and communicated to the server in the datagram.

With respect to the delta times there are two policies, which this researcher calls lying and honesty. In the lying policy a client reports a zero delta time to the server in hopes that the network latency is so low that the client and server will be essentially

synchronized, and no buffering is done by the server. In the honesty policy case the client reports the delta time that it used to play the MIDI short message. The lying policy case was used in most of the preliminary protocols.

One parameter was common to all the protocols, namely, the number of MIDI short message structures per datagram or packet,  $m$ . Another parameter that was shared by the  $CW$ ,  $SN-TCP$ , and  $YF$  protocols was the number of data-grams between datagram echoes,  $n$ . Datagram echoes were used to get a rough estimate of the data-grams or packets that were being lost during transmission. Of course, for  $SN-TCP$  no packets are lost, but we still monitored the *round-trip-times* ( $RTTs$ ) found via packet echoes. As will be seen the  $RTTs$  were used by  $CW$  and one variant of  $YF$  to get an estimate of the dynamic buffer size. A third parameter was the number of datagram copies sent by the  $YF$  protocol,  $k$ . In the  $CW$  protocol and the dynamic buffer size  $YF$  protocol ( $dYF$ ), the server's dynamic buffer size is calculated via the formula:

$$BufferSize = m * MaximumEchoTime / EchoTime$$

The dynamic buffer holds structures of the type shown in Figure 8-1. The dynamic buffer size is computed after each received datagram.

The  $CW$  protocol uses the following transmission scheme for  $m = 3$ . It first sends MIDI messages 1, 2, and 3 then it sends 2, 3, and 4, and then 3, 4, and 5, et cetera. Thus, the number of data-grams transmitted by the  $CW$  algorithm is the total number of MIDI messages minus  $m$  plus one.

The dynamic buffer variation of the  $dYF$  algorithm differs from the  $CW$  algorithm in the number of copies of the datagram transmitted.  $CW$  only sends one copy.  $dYF$  sends  $k$  copies. The large buffer version of  $YF$  uses a static buffering scheme. Both the  $CW$  and

*dYF* used a variable named the *currentIndex*. If the incoming datagram has MIDI short messages with *indices* less than the *currentIndex* then it must be a copy of the datagram and is ignored. If the incoming datagram's MIDI short messages have *indices* equal to the *currentIndex* then the datagram's MIDI short messages are played immediately. Lastly, if the incoming datagram's *indices* are greater than the *currentIndex* then the datagram's MIDI data is buffered. Each time a datagram is received, the buffer is checked to see if it is full, and if it is full then the buffer's MIDI short message data is played. The buffer is maintained in sorted order on the *index* of each MIDI short message structure in ascending order. After the buffer is played the *index* variable is set to last buffered MIDI short message structure's index value plus one.

The *static buffer YF (sYF)* protocol abandons the idea of unique *indices* and uses a saner unique datagram *sequenceNumber*. The protocol uses a variable named the *expectedSequenceNumber*. If an incoming datagram's *sequenceNumber* is less than the *expectedSequenceNumber* then it is ignored since it must be a datagram copy. If the incoming datagram's *sequenceNumber* is equal to *expectedSequenceNumber* then its MIDI short message structures are played immediately and the *expectedSequenceNumber* is incremented by one and then the buffer is searched for more data to play if it is nonempty. In the last case of the incoming datagram's *sequenceNumber* being greater than the *expectedSequenceNumber* then the whole datagram is buffered in ascending order in buffer with the *sequenceNumber* as a key. In the last case after addition of a datagram, the buffer is checked to see if it is full, and if it is full then the data-grams' MIDI short message data is played immediately. *sYF* uses the honesty policy case with respect to delta times.

There are two types of RTP protocols available either with or without error recovery. By error recovery we mean building reliability into RTP by retransmission of lost data-grams. An error is the loss of a datagram. We used RTP without error recovery in the preliminary experiments. The *RTP* protocol used the *index* idea of *CW* and *dYF*.

The *SN-TCP* protocol used the same data structures as the dynamic buffer algorithms *CW* and *dYF* and *RTP*. However, in reality, the idea of an index or packet sequence number is not needed by *SN-TCP* since it is reliable and all packets are received in the order transmitted. We used the same *index* arithmetic with the TCP server as with the other algorithms that used *indices* so as not to give *SN-TCP* an unfair advantage.

The echoing process used *CW*, *SN-TCP*, and *YF* was to echo each *n*th datagram or packet. The client generates a timestamp for the datagram or packet with its current real time clock value. Upon receipt of an echo the client checks to make sure the echoed datagram or packet's data is the same as the datagram or packet transmitted for echo and if it was the same then it computes the *RTT* otherwise the client would cause an exceptional condition. The client would maintain the *minimum*, *average*, and *maximum RTTs*, and in the dynamic buffer cases would send the *average* and *maximum echo time* to the server.

The four protocols and one protocol variation were implemented in the C language using the Microsoft Visual C/C++ 6.0 compiler. As alluded to earlier in the description of the algorithms client/server architecture was used. The client is a MIDI file format 0 or 1 sequencer. The sequencer transmits data as soon as a datagram or packet becomes full with *m* MIDI short message structures.

The *CW* and *YF* server implementation has three sockets: a TCP socket for receiving control information, a UDP socket for receiving data-grams, and a UDP socket for sending and receiving echoed data-grams. The TCP socket and UDP receiving socket are handled by the select system call. The server blocks until one of the sockets receives data. There are two types of TCP messages either an OPEN message or a CLOSE message. An OPEN message sends the client's parameters such as *m*, *k*, and *n*. The CLOSE message tells the server to shutdown the TCP socket and go back to the accept socket system call to wait on another connection by a client. The server has two threads of execution an echo thread and a main MIDI message processing thread. Table 3-A-1 summarizes the protocols.

<i>Name</i>	<i>Buffer</i>	<i>Data Transport</i>	<i>Control Transport</i>
<i>CW</i>	Dynamic	UDP	TCP
<i>dYF</i>	Dynamic	UDP	TCP
<i>sYF</i>	Static	UDP	TCP
<i>SN-TCP</i>	Not Applicable	TCP	Not Applicable
<i>RTP</i>	Not Applicable	UDP	UDP

Table 3-A-1 Preliminary Protocols

The server machine was a Windows 98 personal computer (PC) with a Pentium 2 450 MHz central processing unit (CPU), 128 MB SDRAM, a Turtle Beach Montego A3D 64 voice PCI sound card, and Altec Lansing ACS 295 speakers with subwoofer. The client machine was a Windows XP Home Edition PC with a Pentium 4 2.26 GHz CPU, 512 MB RDRAM, a Turtle Beach Santa Cruz DSP sound card, and Harman/Kardon HK-695 speakers with subwoofer. The server machine had a 31.2 Kbps

dial-up link to the Internet and the client machine had an Asymmetric Digital Subscriber Line (ADSL) link to the Internet. Although the computers were only a few rooms apart in a residence, the network formed was a Wide Area Network (WAN). The dialup-link went along the analog part of the telephone line to West Point, GA, a trip of about 18 miles then on the Earthlink network to Atlanta, GA, then back to the house in LaGrange, GA, via the Earthlink/Bell South ADSL network. As was mentioned earlier, the client sends MIDI short messages to the server to be played. The primary metric was the runtime at the server as measured to the nearest second. Trace route information is given below.

```
Tracing route to user-2inid4o.dialup.mindspring.com [165.121.52.152]
over a maximum of 30 hops:

  0  <1 ms    <1 ms    <1 ms    172.16.0.254
  1  15 ms    18 ms    14 ms    user-1120k01.dsl.mindspring.com [66.32.80.1]
  2  14 ms    14 ms    15 ms    acr01-v1-3.ga-atlanta0.ne.earthlink.net [207.69.143.1]
  3  16 ms    15 ms    15 ms    cor02-v1-11.ga-atlanta0.ne.earthlink.net [207.69.223.190]
  4  14 ms    14 ms    14 ms    dir10-g12-0-0.ga-atlanta0.ne.earthlink.net [209.165.96.18]
  5  23 ms    21 ms    21 ms    cisco-h0.wp-lag.mindspring.net [207.69.230.226]
  6  23 ms    22 ms    21 ms    acn02a.ga-westpoint1.ne.earthlink.net [207.69.144.222]
  7  236 ms   190 ms   195 ms   user-2inid4o.dialup.mindspring.com [165.121.52.152]

Trace complete.
```

Figure 3-A-3 Trace Route from WAN Client to WAN Server

The experiments are scored using a system that awards four points for first place, three points for second place, two points for third place, and one point for fourth place. When calculating the total points one point was awarded for each experiment instance in which the protocol was reliable. Two standard MIDI files were utilized, namely, one MIDI type 0 files: *Trippygaial.mid* and one MIDI type 1 file: *Flourish.mid*.



<i>File</i>	<i>M</i>	<i>CW</i>	<i>RTP</i>	<i>SN-TCP</i>	<i>dYF</i>
<i>Trippygail.mid</i>	1	2	1	3	4
	2	1	4	2	3
	3	1	4	2	3
	4	1	4	2	3
<i>Flourish.mid</i>	1	2	1	3	4
	2	2	1	4	3
	3	2	1	4	3
	4	1	4	2	3
<i>Points</i>	-	12	20	22	26
<i>Zero Loss Points</i>	-	0	4	8	5
<i>Total Points</i>	-	12	24	30	31
<i>Overall Rank</i>	-	4 <sup>th</sup>	3 <sup>rd</sup>	2 <sup>nd</sup>	1 <sup>st</sup>

Table 3-A-2 Points Awarded to Algorithms and Overall Ranks

As can be seen from Table 3-A-2 *SN-TCP* is clearly the only reliable protocol which is very important for MIDI over IP since as can't be overemphasized MIDI is intolerant with respect to lost or out-of-order data. The *CW* protocol performed so poorly that it was dropped from further consideration.

#### *B. RTP and TCP Protocols*

In this section and the next section we make a transition away from protocols implemented in the programming language, C, and we utilize the interpreted platform independent language, Java. The reason we change languages is to take advantage of a unique design feature of Java Media Framework's (JMF) implementation of RTP, which is further explained the next paragraph. Java has more inherent latency in most cases than C on the machines chosen for the subsequent experiments however Java is utilized

because of the elegance of the RTP implementation in this researcher’s opinion. The experimental work associated with these protocols was done in a statistically significant number of trials. All of the subsequent protocols used the honesty delta-time policy.

The Real-Time Protocol (RTP) implementation in the JMF allows the user to abstract away the underlying transport protocol from the RTPManager object via the implementation of the interface RTPConnector. This means that the user can utilize either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) as the transport layer protocol. Since RTP has two channels, namely, a control channel and a data channel, and there are two Internet transport layer protocols, four possibilities exist for the channels and transport protocols of RTP as illustrated by the following table.

Control	Data
TCP	TCP
UDP	TCP
TCP	UDP
UDP	UDP

Table 3-B-1 Control and Data Channels and Transport Layer Protocols for RTP

Our usage of an alternative transport protocol to UDP with JMF’s implementation of RTP does not appear to be novel since previous researchers used the Stream Control Transport Protocol (SCTP) in a similar manner to our usage of TCP, but without the four cases of Table 3-B-1 [34].

These four RTP variations have been nicknamed *TT-TCP*, *UT-RTP*, *TU-RTP* and *UU-RTP* where the first letter is an abbreviation of the control channel transport protocol and the second letter is an abbreviation for the data channel transport protocol.

Practical MIDI streams such as those generated by real musicians or sequencers differ from real-time audio in that the MIDI streams consist of discrete or quantized events instead of continuous bit streams. The MIDI specification calls for a stream with a baud rate of 31250 bps, however, it is generally fairly rare to generate MIDI short messages at the maximum MIDI short message rate of less than or equal 3125 MIDI short messages per second. Suppose Eve is playing a MIDI instrument at a metronome setting of 120 beats per minute (BPM) without using control messages or pitch-bend messages. Then at most Eve is generating two MIDI short messages every half-second, a previous note off message and a next note on message. This is an ideal situation for reliable protocols that send a single packet at a time, since there is probably enough time between the send for the return acknowledgment before the next send.

<i>Application</i>
RTP
UDP/TCP
IP

Table 3-B-2 Partial Protocol Stack for UT-RTP

The various RTP protocols were implemented using JMF 2.0 and Java 1.4.0\_01 on Windows operating systems machines. We translated the simple and naive TCP protocol, SN-TCP, from the C version of the previous chapter without the echoing feature. All the protocols were implemented using the same version of Java. The Nagle algorithm was disabled in some variations of *TT-RTP*, *UT-RTP-ND*, *TU-RTP-ND*, and *SN-TCP-ND* by setting the TCP\_NODELAY socket option to true. The Nagle algorithm was enabled in some variations of *TT-RTP*, *UT-RTP-NE*, *TU-RTP-NE*, and *SN-TCP-NE*

by setting the TCP\_NODELAY socket option to false. Table 3-5 illustrates the various Nagle disabled/enabled relationships for TT-RTP.

<i>TT-RTP Protocol</i>	<i>Control</i>	<i>Data</i>
<i>TNDTND-RTP</i>	<i>ND</i>	<i>ND</i>
<i>TNETND-RTP</i>	<i>NE</i>	<i>ND</i>
<i>TNDTNE-RTP</i>	<i>ND</i>	<i>NE</i>
<i>TNETNE-RTP</i>	<i>NE</i>	<i>NE</i>

Table 3-B-3 TT-RTP ND/NE Relationships ND = Nagle Disabled and NE = Nagle Enabled

Client/server architecture was used. The clients were both Musical Instrument Digital Interface (MIDI) sequencers that play and send a stream of MIDI short message to the server to be played. Since we are disabling the Nagle algorithm a relatively short byte stream was sent from client to server. Each RTP and TCP MIDI short message consisted of twelve bytes: a MIDI channel byte, MIDI command byte, two MIDI data bytes, and eight bytes of information that represented the delta time in milliseconds. A RTP or TCP packet consisted of  $20 + 12 * m$  bytes where 20 is the number of bytes in the TCP header and m is the number of MIDI short messages per packet. Next are a table of source code files and lines of code (LOC), and also a table of packet lengths.

<i>Source Code File</i>	<i>Lines of Code</i>
RTPMIDIClient.java (TT-RTP)	599
RTPMIDIServer.java (TT-RTP)	344
RTPMIDIClient.java (UT-RTP)	643
RTPMIDIServer.java (UT-RTP)	372
RTPMIDIClient.java (TU-RTP)	643
RTPMIDIServer.java (TU-RTP)	372
RTPMIDIClient.java (UU-RTP)	336
RTPMIDIServer.java (UU-RTP)	401
TCPMIDIClient.java	400
TCPMIDIServer.java	247
<i>Total</i>	<i>4357</i>

Table 3-B-4 XY-RTP SN-TCP Source Code Files and Lines of Code where X = {T, U} and Y = {T, U}

<i>M</i>	$20 + 12 * M$
1	32
2	44
3	56
4	68

Table 3-B-5 RTP and TCP Number of MIDI Short Messages and Packet Lengths in Bytes

Some of the experiments were carried out over a period of days between March 14, 2004 and March 28, 2004 and the other experiments were carried out over a period of days between October 10, 2003 and October 13, 2003. It is to be hoped that the Internet was relatively stable during the time frame of the experiments. The following tables show the value of  $m$ , the starting time and date, ending time and date of the sixty trials per

experiment, and the actual number of hours. One standard MIDI files was used:

Trippygaia1.mid a MIDI type 0 file.

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	10/10/2003 05:09 PM EDT	10/10/2003 07:16 PM EDT	2.1157
2	10/11/2003 08:05 AM EDT	10/11/2003 10:12 AM EDT	2.1160
3	10/12/2003 10:14 PM EDT	10/12/2003 12:21 PM EDT	2.1205
4	10/13/2003 07:12 AM EDT	10/13/2003 09:19 AM EDT	2.1216
<i>Totals</i>			8.4738

Table 3-B-6 Trippygaia1.mid UT-RTP-ND Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	10/10/2003 02:42 PM EDT	10/10/2003 04:49 PM EDT	2.1245
2	10/11/2003 01:04 PM EDT	10/11/2003 03:11 PM EDT	2.1259
3	10/12/2003 02:51 PM EDT	10/12/2003 04:58 PM EDT	2.1214
4	10/13/2003 11:42 AM EDT	10/13/2003 01:49 PM EDT	2.1204
<i>Totals</i>			8.4922

Table 3-B-7 Trippygaia1.mid UT-RTP-NE Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	10/10/2003 08:52 PM EDT	10/10/2003 10:59 PM EDT	2.1197
2	10/11/2003 10:16 AM EDT	10/11/2003 12:23 PM EDT	2.1158
3	10/12/2003 12:39 PM EDT	10/12/2003 02:46 PM EDT	2.1170
4	10/13/2003 09:22 AM EDT	10/13/2003 11:29 AM EDT	2.1178
<i>Totals</i>			8.4703

Table 3-B-8 Trippygaia1.mid SN-TCP-ND Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	10/10/2003 12:18 PM EDT	10/10/2003 02:25 PM EDT	2.1249
2	10/11/2003 03:20 PM EDT	10/11/2003 05:27 PM EDT	2.1214
3	10/12/2003 05:01 PM EDT	10/12/2003 07:08 PM EDT	2.1270
4	10/13/2003 02:03 PM EDT	10/13/2003 04:10 PM EDT	2.1237
<i>Totals</i>			8.4970

Table 3-B-9 Trippygaia1.mid SN-TCP-NE Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	03/14/2004 04:49 PM EST	03/14/2004 06:30 PM EST	1.6789
2	03/15/2004 01:01 AM EST	03/15/2004 02:45 AM EST	1.6797
3	03/15/2004 12:21 PM EST	03/15/2004 02:02 PM EST	1.6761
4	03/16/2004 10:14 AM EST	03/16/2004 11:55 AM EST	1.6747
<i>Totals</i>			6.7094

Table 3-B-10 Trippygaia1.mid TNDTND-RTP Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	03/25/2004 01:49 PM EST	03/25/2004 01:49 PM EST	1.6738
2	03/28/2004 02:28 PM EST	03/28/2004 02:28 PM EST	1.6790
3	03/28/2004 04:14 PM EST	03/28/2004 04:14 PM EST	1.6748
4	03/28/2004 06:37 PM EST	03/28/2004 06:37 PM EST	1.6788
<i>Totals</i>			6.7064

Table 3-B-11 Trippygaia1.mid TNETND-RTP Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	03/24/2004 01:44 PM EST	03/24/2004 03:25 PM EST	1.6759
2	03/24/2004 03:46 PM EST	03/24/2004 05:27 PM EST	1.6762
3	03/24/2004 05:30 PM EST	03/24/2004 07:11 PM EST	1.6762
4	03/25/2004 09:03 AM EST	03/25/2004 10:44 AM EST	1.6756
<i>Totals</i>			<i>6.7039</i>

Table 3-B-12 Trippyaia1.mid TNDTNE-RTP Actual Runtime Hours

<i>M</i>	<i>Starting Time/Date</i>	<i>Ending Time/Date</i>	<i>Actual Hours</i>
1	03/21/2004 04:55 PM EST	03/21/2004 06:36 PM EST	1.6790
2	03/22/2004 01:29 PM EST	03/22/2004 03:10 PM EST	1.6759
3	03/22/2004 03:36 PM EST	03/22/2004 05:17 PM EST	1.6761
4	03/22/2004 05:25 PM EST	03/22/2004 07:06 PM EST	1.6767
<i>Totals</i>			<i>6.7077</i>

Table 3-B-13 Trippyaia1.mid TNETNE-RTP Actual Runtime Hours

The same client and server setup mentioned previously was utilized in the experiments covered in this section. There are two hundred and twenty four tables appended to this dissertation, namely, one hundred and twelve pairs of tables based on the hundred and twelve separate experiments can be found in Appendix J. The first table in a pair has the protocol, one of *TNDTND-RTP*, *TNETND-RTP*, *TNDTNE-RTP*, *TNETNE-RTP*, *UT-RTP-ND*, *UT-RTP-NE*, *SN-TCP-ND* or *SN-TCP-NE*, the number of trials, *N*, which is always sixty, the standard deviations, and the standard error means. The second table in a pair has the means difference, the standard deviation of the means difference, the standard error mean of the means difference, the Student's *t*-statistic,  $(Protocol\ 1\ mean - Protocol\ 2\ mean) / standard\ error\ mean$  of the means difference, the



degrees of freedom,  $DF$ , which are always  $N - 1$  that equals 59, and the two-tailed significance of the  $t$ -statistic. A two-tailed significance of  $\leq 0.05$  means that one of the protocols outperformed the other statistically speaking. The better of the two protocols is determined by the sign of the  $t$ -statistic: - indicates that first protocol wins and + means that the second protocol wins.

We do not report any experimental results with  $TU-RTP$  and  $UU-RTP$  since these protocols were so unreliable that they failed in every attempted experiment. We also created a hybrid protocol between  $UU-RTP$  and the Young and Fujinaga ( $YF$ ) protocol of chapter 6 and this protocol ( $YF/UU-RTP$ ) also failed for choices of the number of copies of each datagram transmitted equal one, two, and four. By failure, we mean at least one datagram was lost.

The following tables distill the information from the two hundred twenty four tables mentioned above. The numbers in the row and column headings stand for a protocol such as 1= $TNDTND-RTP$ . The other letters have the following meanings: N=Not applicable, D=Does not count, 0=row and column protocols are statistically equivalent, row protocol is statistically better than column protocol if  $\leq -0.05$ , and column protocol is statistically superior to row protocol  $\leq +0.05$ .

P	P#	1	2	3	4	5	6	7	8	9	10	11
TNDTND-RTP	1	N	0.08	0.42	-0.91	0.22	-0.49	D	D	D	0.84	-0.30
TNETND-RTP	2		N	-0.03	-0.01	-0.83	-0.04	D	D	D	-0.03	-0.00
TNDTNE-RTP	3			N	-0.29	0.06	-0.10	D	D	D	-0.19	-0.00
TNETNE-RTP	4				N	0.12	-0.46	D	D	D	0.79	-0.22
UT-RTP-ND	5					N	-0.04	D	D	D	-0.04	-0.00
UT-RTP-NE	6						N	D	D	D	0.29	-0.92
TU-RTP-ND	7							N	D	D	D	D
TU-RTP-NE	8								N	D	D	D
UU-RTP	9									N	D	D
SN-TCP-ND	10										N	-0.04
SN-TCP-NE	11											N

Table 3-B-14 M=1 Sign of t-Statistic \* Statistical Significance

<i>P</i>	<i>P#</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>
<i>TNDTND-RTP</i>	<i>1</i>	N	0.32	0.68	0.46	0.32	-0.51	D	D	D	0.29	-0.98
<i>TNETND-RTP</i>	<i>2</i>		N	0.53	0.49	0.32	-0.42	D	D	D	0.28	-0.86
<i>TNDTNE-RTP</i>	<i>3</i>			N	0.04	0.11	-0.09	D	D	D	0.00	-0.00
<i>TNETNE-RTP</i>	<i>4</i>				N	0.18	-0.08	D	D	D	0.00	-0.00
<i>UT-RTP-ND</i>	<i>5</i>					N	-0.04	D	D	D	0.90	-0.00
<i>UT-RTP-NE</i>	<i>6</i>						N	D	D	D	0.03	0.34
<i>TU-RTP-ND</i>	<i>7</i>							N	D	D	D	D
<i>TU-RTP-NE</i>	<i>8</i>								N	D	D	D
<i>UU-RTP</i>	<i>9</i>									N	D	D
<i>SN-TCP-ND</i>	<i>10</i>										N	-0.00
<i>SN-TCP-NE</i>	<i>11</i>											N

Table 3-B-15 M=2 Sign of t-Statistic \* Statistical Significance

<i>P</i>	<i>P#</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>
<i>TNDTND-RTP</i>	<i>1</i>	N	0.47	-0.96	0.99	-0.45	-0.06	D	D	D	0.69	-0.08
<i>TNETND-RTP</i>	<i>2</i>		N	-0.00	-0.00	-0.20	-0.00	D	D	D	-0.00	-0.04
<i>TNDTNE-RTP</i>	<i>3</i>			N	0.66	-0.40	-0.00	D	D	D	0.00	-0.07
<i>TNETNE-RTP</i>	<i>4</i>				N	-0.38	-0.00	D	D	D	0.01	-0.07
<i>UT-RTP-ND</i>	<i>5</i>					N	-0.79	D	D	D	0.27	-0.01
<i>UT-RTP-NE</i>	<i>6</i>						N	D	D	D	0.00	-0.27
<i>TU-RTP-ND</i>	<i>7</i>							N	D	D	D	D
<i>TU-RTP-NE</i>	<i>8</i>								N	D	D	D
<i>UU-RTP</i>	<i>9</i>									N	D	D
<i>SN-TCP-ND</i>	<i>10</i>										N	-0.04
<i>SN-TCP-NE</i>	<i>11</i>											N

Table 3-B-16 M=3 Sign of t-Statistic \* Statistical Significance

<i>P</i>	<i>P#</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>
<i>TNDTND-RTP</i>	<i>1</i>	N	-0.31	-0.70	-0.02	-0.14	-0.00	D	D	D	-0.08	-0.02
<i>TNETND-RTP</i>	<i>2</i>		N	0.05	0.53	-0.83	-0.99	D	D	D	0.52	-0.52
<i>TNDTNE-RTP</i>	<i>3</i>			N	-0.50	-0.33	-0.20	D	D	D	0.84	-0.11
<i>TNETNE-RTP</i>	<i>4</i>				N	-0.40	-0.03	D	D	D	0.62	-0.10
<i>UT-RTP-ND</i>	<i>5</i>					N	0.76	D	D	D	0.32	-0.67
<i>UT-RTP-NE</i>	<i>6</i>						N	D	D	D	0.00	-0.30
<i>TU-RTP-ND</i>	<i>7</i>							N	D	D	D	D
<i>TU-RTP-NE</i>	<i>8</i>								N	D	D	D
<i>UU-RTP</i>	<i>9</i>									N	D	D
<i>SN-TCP-ND</i>	<i>10</i>										N	-0.07
<i>SN-TCP-NE</i>	<i>11</i>											N

Table 3-B-17 M=4 Sign of t-Statistic \* Statistical Significance

### C. *ATCP* and *ATCP-TCP* Protocols

The motivation for developing more protocols is to attempt to find reliable protocols, which with certain choices of the parameters can beat *SN-TCP* as far as the critical variable runtime at server, is concerned. This author developed a new protocol called the *Almost TCP (ATCP)* protocol, which can be characterized as a stop and wait and selective repeat quasi-transport level protocol. It uses the User Datagram Protocol

(UDP) as its official transport level protocol. *ATCP* gets its name from being very close in performance to TCP for the MIDI over IP application. *ATCP* does share some other similarities with TCP such as they both use acknowledgments, sliding windows with advertisements, are both reliable and deliver data in the order transmitted, and both use Jacobson's algorithm for computing the acknowledgment timeout (see *Computer Networks Third Edition* by Andrew S. Tanenbaum page 541 for a good description of Jacobson's algorithm and TCP in general). *ATCP* sends a stream of UDP data-grams that ultimately consists of a byte stream. TCP transmits a stream of IP-packets that in the final analysis is a byte stream. An *ATCP sequence number* refers to a given UDP datagram, whereas a TCP acknowledgment and *sequence number* refer to a single byte in the transmission byte stream. *ATCP* uses a datagram buffer size (window) advertisement that is equivalent to the number of data-grams between acknowledgments. The current version uses a fixed number of data-grams between acknowledgments which we will designate by  $x$  and call the protocol *ATCP-x* where currently  $1 \leq x \leq 40$ . The MIDI short message and datagram format are given in Figure 3-C-1 and Figure 3-C-2.



Figure 3-C-1 *ATCP-x* MIDI Short Message Format

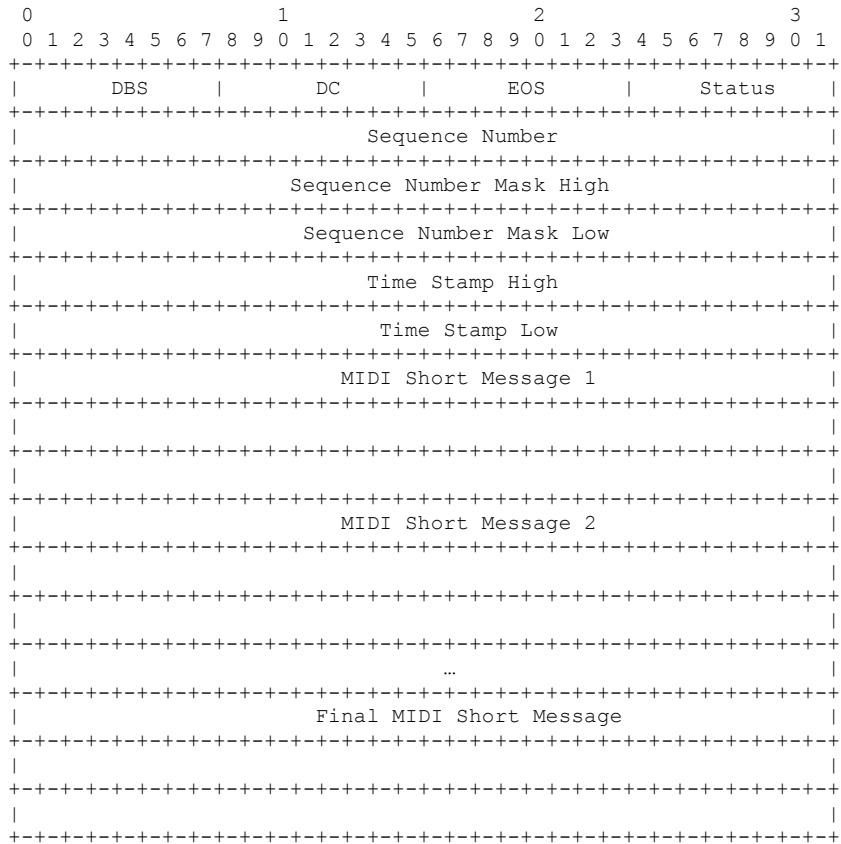


Figure 3-C-2 *ATCP-x* Datagram Format

Figure 10-2 requires further elaboration. DBS is the *Data Buffer Size*, which is an integer in the range 1 to 64. DC is the *Datagram Count*, which tells the server how many datagrams the client is currently buffering, e. g. the total number of data-grams the server should receive. EOS is the *End of Stream* flag that is 0 if not end of file and 1 to indicate the end of the file. Status is an enumeration that represents a data datagram or acknowledgement datagram or a negative acknowledgement datagram. The MIDI short message is the same as Figure 3-C-1.

In some preliminary experiments we used a variation of the protocol that utilized a variable number of data-grams between acknowledgments with a slow start algorithm

and exponential back-off mechanism, somewhat similar to TCP and TCP's congestion window mechanism. However, this variant was later abandoned in favor of a simpler and error free version. The protocol sends  $x$  data-grams then waits for an acknowledgment or a timeout period to expire. If the timeout period expires up to  $n$  negative acknowledgments are sent before the protocol signals failure and terminates. For now the value of  $n$  is sixteen. The timeout period is variable (dynamic) and is calculated by Jacobson's algorithm that is based on the round-trip times. *ATCP-x* also uses a bit vector or bit mask that is named by us the sequence number mask which tells which data-grams have been received by the receiver. Suppose the current datagram buffer size is 32 and the receiver did not receive data-grams 2 and 4 then the *sequence number mask* would be as follows in binary and then hex: 11010111111111111111111111111111 (base 2) = D7FFFFFF (base 16). The *datagram count* returned to the receiver along the *sequence number mask* would be equal to 30. A significant way that *ATCP-x* differs from TCP is that *ATCP-x* is asymmetric and has a strict sender and receiver relationship. TCP can piggyback data to the original sender on each acknowledgment, so that the receiver can also function as a sender. Another way that *ATCP-x* and TCP can be differentiated is that *ATCP-x* is essentially a connectionless protocol like the underlying UDP protocol.

However, TCP is a connection-oriented protocol.

*ATCP-x* was implemented in Java using Java version 1.4.0\_01. The fundamental data objects in the program were *MyATCPShortMessage* and *ATCPPacket*. The former data object encapsulated the data structure in Figure 3-C-1 and the latter data object was an implementation of the data structure in Figure 3-C-2. *MyATCPShortMessage* has two constructors a default constructor and a constructor that allows initialization of all the

data fields, getters for all the data fields, a *getBytes* method, and a *fromBytes* method. The *getBytes* was used to convert a *MyATCPShortMessage* object into a byte stream and the *fromBytes* was utilized to convert a byte stream to a *MyATCPShortMessage* object. *ATCPPacket* has two constructors one that has an integer parameter that is  $m$ , the number of MIDI short messages per *ATCPPacket*, and another for fully populating the packet with all its data members. *ATCPPacket* has getters and setters for all its data fields. It also has the *getBytes* and *fromBytes* methods.

We used two threads in both the client and the server. One thread was a producer of MIDI short messages and the other was a consumer of MIDI short messages. A MIDI short message vector was shared by both threads so synchronized code blocks had to be used to read or write to the vector, which was, in reality, a first-in first-out (FIFO) queue. In the server the networking thread was the producer of the MIDI short messages and the player thread was the consumer. Contrary to the server case, the sequencer was the producer in the client and the networking thread was the client's consumer. One or more active sensing MIDI short messages were used as the end of file indicator or sentinel flag message. An active sensing message in the MIDI world is somewhat analogous to a no operation op-code in the universe of computer assembly languages. We used busy – sleep loops in the networking threads and the player thread. The more elegant Java object notification was later implemented. Each datagram or packet contained  $m$  MIDI short messages where  $1 \leq m \leq 16$ . Below is a table of the *ATCP-x* source code files and the lines of code.

<i>Source Code File</i>	<i>Lines of Code</i>
ATCPMIDIClient.java	519
ATCPMIDIServer.java	543
<i>Total</i>	<i>1062</i>

Table 3-C-1 ATCP-x Source Code Files and Lines of Code

Fifteen experiments of sixty trials per experiment were carried over the period from November 2 to November 10, 2003. The next three tables contain the value of  $m$  used in the experiment and the ending time of the experiment for each of the three protocols *ATCP-32*, *SN-TCP-ND*, and *SN-TCP-NE*. One standard MIDI type 0 file named *Trippygaia1.mid* was used through out the experiments. Remember that a standard MIDI type 0 file consists of a single track. The same experimental setup of clients and servers were used in this report as those reported in the pervious chapters. We again give the starting and ending times of the experiments, and the actual runtimes in hours remind the reader that the Internet is typically thought of as being diurnal with peak times between 9:00 AM and 12:00 PM and 7:00 PM and 10:00 PM.

<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
4	11/09/2003 10:35 AM	11/09/2003 12:42 PM	2.1167
5	11/09/2003 12:52 PM	11/09/2003 02:59 PM	2.1167
6	11/09/2003 05:25 PM	11/09/2003 07:31 PM	2.1000
7	11/10/2003 10:34 AM	11/10/2003 12:40 PM	2.1000
8	11/10/2003 12:49 PM	11/10/2003 02:55 PM	2.1000
<i>Totals</i>			<i>10.5334</i>

Table 3-C-2 ATCP-32 Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File



<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
4	11/02/2003 05:59 PM	11/02/2003 08:06 PM	2.1167
5	11/03/2003 11:28 AM	11/03/2003 01:35 PM	2.1167
6	11/04/2003 10:46 AM	11/04/2003 12:53 PM	2.1167
7	11/06/2003 10:59 AM	11/06/2003 01:06 PM	2.1167
8	11/08/2003 02:39 PM	11/08/2003 04:46 PM	2.1167
<i>Totals</i>			<i>10.5835</i>

Table 3-C-3 SN-TCP-ND Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File

<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
4	11/02/2003 08:18 PM	11/02/2003 10:25 PM	2.1167
5	11/03/2003 01:44 PM	11/03/2003 03:51 PM	2.1167
6	11/04/2003 01:08 PM	11/04/2003 03:15 PM	2.1167
7	11/06/2003 02:56 PM	11/06/2003 05:03 PM	2.1167
8	11/08/2003 05:05 PM	11/08/2003 07:12 PM	2.1167
<i>Totals</i>			<i>10.5835</i>

Table 3-C-4 SN-TCP-NE Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File

Appended to this dissertation are thirty tables and fifteen graphs in Appendix K that cover the *ATCP-32* experiments. We compared *ATCP-32* to *SN-TCP-ND* and *SN-TCP-NE*, and *SN-TCP-ND* to *SN-TCP-NE*. We found that in the  $m = 6$  and 8 cases that *ATCP* statistically outperformed *SN-TCP-ND* and that in the  $m = 6, 7$ , and 8 cases was the statistical winner versus *SN-TCP-NE*. In all the other cases the protocols were statistically equivalent. *ATCP-x* does not perform that well against *SN-TCP* for values of  $m$  less than three or equal 3.

<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
4	11/15/2003 12:13 PM	11/15/2003 02:20 PM	2.1167
5	11/15/2003 02:26 PM	11/15/2003 04:32 PM	2.1000
6	11/16/2003 03:22 PM	11/16/2003 05:28 PM	2.1000
7	11/19/2003 01:24 PM	11/19/2003 03:30 PM	2.1000
8	11/19/2003 03:34 PM	11/19/2003 05:40 PM	2.1000
<i>Totals</i>			<i>10.5167</i>

Table 3-C-5 ATCP-40 Runtime Data for Trippygaia1.mid Standard MIDI Type 0 File

Table 3-C-5 displays the facts that the *ATCP-40* experiments were conducted from November 15, 2003 to November 19, 2003. There are thirty tables and fifteen graphs for the *ATCP-40* results in Appendix L. We compare *ATCP-40* to *SN-TCP-ND*, *SN-TCP-NE*, and *ATCP-32*. *ATCP-40* statistically beat *SN-TCP-ND* in the  $m = 5$  and  $m = 8$  cases, and *SN-TCP-NE* in the  $m = 5, 6, 7,$  and  $8$  cases. *ATCP-40* and *ATCP-32* statistically tied in all cases.

Figure 3-C-4 shows a *statistical significance* versus  $m$  graph for *ATCP-40* versus *SN-TCP-ND* and *ATCP-40* versus *SN-TCP-NE*. The results for *ATCP-40* versus *ATCP-32* are not shown due to the fact that they tied and there is a sign reversal in one of the means differences and the graph would not be consistent.

The *ATCP-TCP* protocol is a multiply threaded version of *SN-TCP* using the *ATCP* notion of MIDI producer and consumer threads. There are two variations of *ATCP-TCP*, namely, *ATCP-TCP-ND* and *ATCP-TCP-NE* with the now usual *ND* standing for Nagle algorithm disabled and *NE* standing for the Nagle algorithm enabled. Again we used busy – sleep loops rather than the more elegant object notification or event-handling

scheme of Java for the same reasons as in the *ATCP-x* case. Table 3-C-6 shows the source code files for *ATCP-TCP*.

<i>Source Code File</i>	<i>Lines of Code</i>
TCPMIDIClient.java	458
TCPMIDIServer.java	457
Total	915

Table 3-C-6 ATCP-TCP Source Code Files and Lines of Code

<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
5	11/22/2003 01:45 PM EST	11/22/2003 03:51 PM EST	2.1000
6	11/24/2003 07:17 AM EST	11/24/2003 09:23 AM EST	2.1000
7	12/01/2003 10:53 AM EST	12/01/2003 12:59 PM EST	2.1000
8	12/02/2003 11:57 AM EST	12/02/2003 02:03 PM EST	2.1000
<i>Totals</i>			<i>8.4000</i>

Table 3-C-7 ATCP-TCP-ND Ending Date/Time for Trippygaia1.mid Standard MIDI Type 0 File

<i>M</i>	<i>Starting Date/Time</i>	<i>Ending Date/Time</i>	<i>Actual Runtime</i>
5	12/07/2003 12:38 PM EST	12/07/2003 03:44 PM EST	2.1000
6	12/08/2003 10:41 AM EST	12/08/2003 01:47 PM EST	2.1000
7	12/13/2003 03:59 PM EST	12/13/2003 06:05 PM EST	2.1000
8	12/21/2003 11:57 AM EST	12/21/2003 02:03 PM EST	2.1000
<i>Totals</i>			<i>8.4000</i>

Table 3-C-8 ATCP-TCP-NE Ending Date/Time for Trippygaia1.mid Standard MIDI Type 0 File

Table 3-C-7 and Table 3-C-8 show the *ATCP-TCP* experiments as being over time period beginning on November 22, 2003 and ending on December 21, 2003. In Appendix M are thirty-two tables and sixteen graphs of the *ATCP-TCP-ND* experiments.

*ATCP-TCP-ND* statistically outperformed *SN-TCP-ND* and *SN-TCP-NE* in all of the experimental cases  $m = 5, 6, 7,$  and  $8$ . *ATCP-TCP-ND* was the statistical winner over *ATCP-32* and *ATCP-40* in the  $m = 7$  and  $8$  instances. From these results we can conclude that a multiply threaded TCP protocol is to be preferred to any of the previously discussed protocols. These results seem to vindicate the notion of MIDI short message consumer and producer threads.

There are forty tables and twenty graphs related to the *ATCP-TCP-NE* experiments in Appendix N. *ATCP-TCP-NE* was victorious over *SN-TCP-ND* in the  $m = 7$  case and *SN-TCP-NE* in all four cases, namely,  $m = 5, 6, 7,$  and  $8$ . *ATCP-TCP-NE* outperformed *ATCP-40* in the  $m = 6$  case. *ATCP-TCP-ND* was the statistical winner over *ATCP-TCP-NE* in the last case  $m = 8$ . The preceding results are displayed in Tables 3-C-9 to 3-C-13.

<i>Protocol/Protocol</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>1-ATCP-32</i>	N	0.34	N	N	-0.95	-0.26
<i>2-ATCP-40</i>		N	N	N	-0.53	-0.08
<i>3-ATCP-TCP-ND</i>			N	N	N	N
<i>4-ATCP-TCP-NE</i>				N	N	N
<i>5-SN-TCP-ND</i>					N	-0.07
<i>6-SN-TCP-NE</i>						N

Table 3-C-9 Sign of t-Statistic \* Statistical Significance  $m = 4$

<i>Protocol/Protocol</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>1-ATCP-32</i>	N	0.36	0.34	0.29	-0.91	-0.54
<i>2-ATCP-40</i>		N	0.73	0.32	-0.00	-0.00
<i>3-ATCP-TCP-ND</i>			N	0.33	-0.00	-0.00
<i>4-ATCP-TCP-NE</i>				N	-0.06	-0.02
<i>5-SN-TCP-ND</i>					N	-0.07
<i>6-SN-TCP-NE</i>						N

Table 3-C-10 Sign of t-Statistic \* Statistical Significance m = 5

<i>Protocol/Protocol</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>1-ATCP-32</i>	N	-0.18	-0.36	-0.33	-0.00	-0.00
<i>2-ATCP-40</i>		N	0.20	0.04	-0.74	-0.00
<i>3-ATCP-TCP-ND</i>			N	0.55	-0.00	-0.00
<i>4-ATCP-TCP-NE</i>				N	-0.19	-0.00
<i>5-SN-TCP-ND</i>					N	-0.07
<i>6-SN-TCP-NE</i>						N

Table 3-C-11 Sign of t-Statistic \* Statistical Significance m = 6

<i>Protocol/Protocol</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>1-ATCP-32</i>	N	-0.30	0.04	0.15	-0.06	-0.00
<i>2-ATCP-40</i>		N	0.00	0.08	-0.01	-0.00
<i>3-ATCP-TCP-ND</i>			N	-0.55	-0.00	-0.00
<i>4-ATCP-TCP-NE</i>				N	-0.02	-0.00
<i>5-SN-TCP-ND</i>					N	-0.53
<i>6-SN-TCP-NE</i>						N

Table 3-C-12 Sign of t-Statistic \* Statistical Significance m = 7

<i>Protocol/Protocol</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>
<i>1-ATCP-32</i>	N	-0.33	0.00	-0.72	-0.01	-0.00
<i>2-ATCP-40</i>		N	0.00	0.83	-0.00	-0.00
<i>3-ATCP-TCP-ND</i>			N	-0.00	-0.00	-0.00
<i>4-ATCP-TCP-NE</i>				N	-0.14	-0.03
<i>5-SN-TCP-ND</i>					N	-0.62
<i>6-SN-TCP-NE</i>						N

Table 3-C-13 Sign of t-Statistic \* Statistical Significance m = 8

For this chapter 1,977 lines of Java code were written and 59 hours and one minute of network time was utilized to perform the necessary illustrated experiments. Again this does not count the network time used in debugging the implementations.

*D. A First Approximation at a Collaboration System*

A Java client/server for the CW system was built early in the research and subsequently discarded. This section describes the system. Figure 3-D-1 shows the opening dialog box of the client. The user must first register a username. The server checks to see if this username is currently unused and if it is unused a message box appears stating the username is valid then the user must specify a password, a UDP port number (0 – 65535), and a musical instrument.

After the registration process is completed, four studio room frames, a chat frame, and a musician’s frame appear as in Figure 3-D-2. There are three forms of chat: broadcast, multicast, and unicast. Broadcasted chat goes to all musicians regardless of their studio room location, multicasted chat goes to a single studio room, and unicast chat goes to a single musician. The studio room frames have a “911” button and an

“Enter” button. The “911” button is to turn all MIDI notes off in case of a stuck note. The “Enter” button allows a musician to enter the given studio room.

After a musician enters a studio room the client appears as shown in Figure 3-D-3. Only the particular studio room frame, the chat frame, and the musicians-frame are open in the figure. The studio room frame has three buttons “Exit”, “Piano”, and “Send”. The “Exit” button causes the musician to exit the current studio room and the state of the client is returned to the state shown in Figure 3-D-2. The “Piano” button causes a host and port dialog to appear and after selecting a host and port a piano keyboard appears. This allows the user to send piano notes to another musician in the same studio room.

Figure 3-D-4 shows a studio room frame after the “Piano” button has been pressed. Figure 3-D-5 shows a studio room after the “Send” button has been pressed and after the host and port dialog. The figure shows a standard Java file chooser dialog from which the user can open, play, and transmit a MIDI file using the built-in MIDI sequencer.

The client/server system consists of the following Java source code files:  
Central.java 372 LOC, Client.java 1259 LOC, ClientFrameInterface.java 8 LOC, and  
RommFrameInterface.java 3 LOC for a grand total of 1642 LOC.

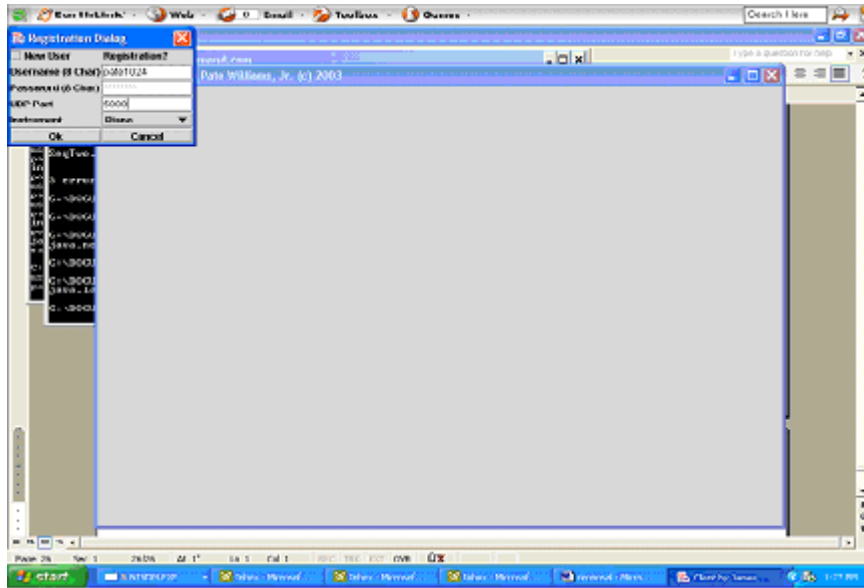


Figure 3-D-1 Musician Registration Dialog

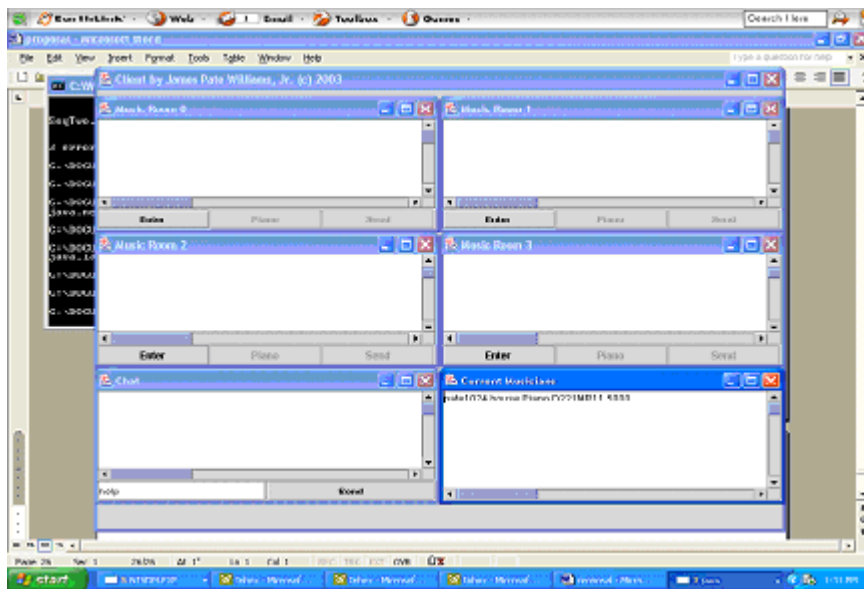


Figure 3-D-2 Music Studio (House) Metaphor Client



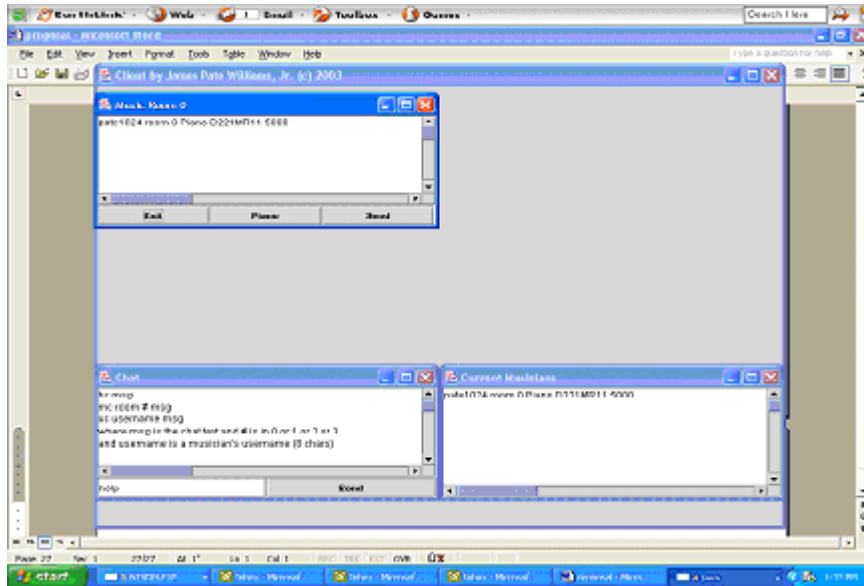


Figure 3-D-3 Music Studio after a Musician Has Entered a Room

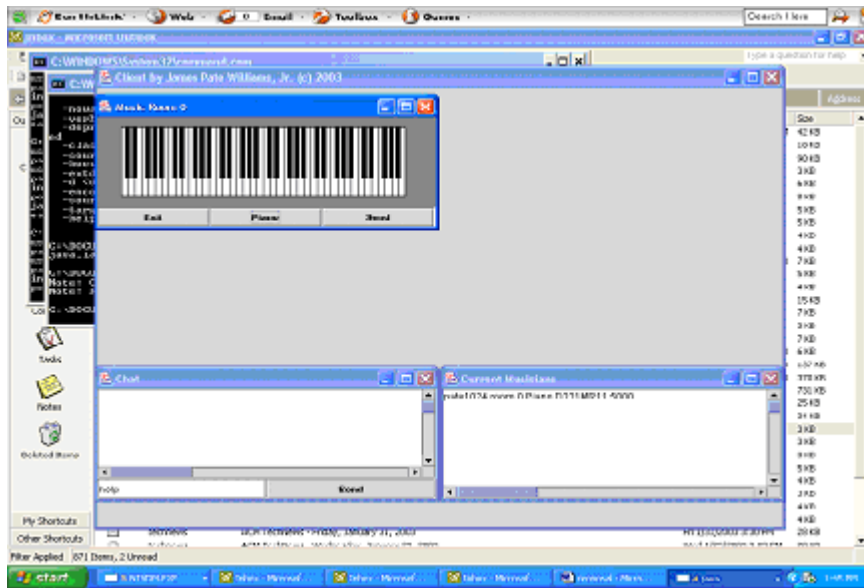


Figure 3-D-4 Music Room with a Piano Keyboard for MIDI Input

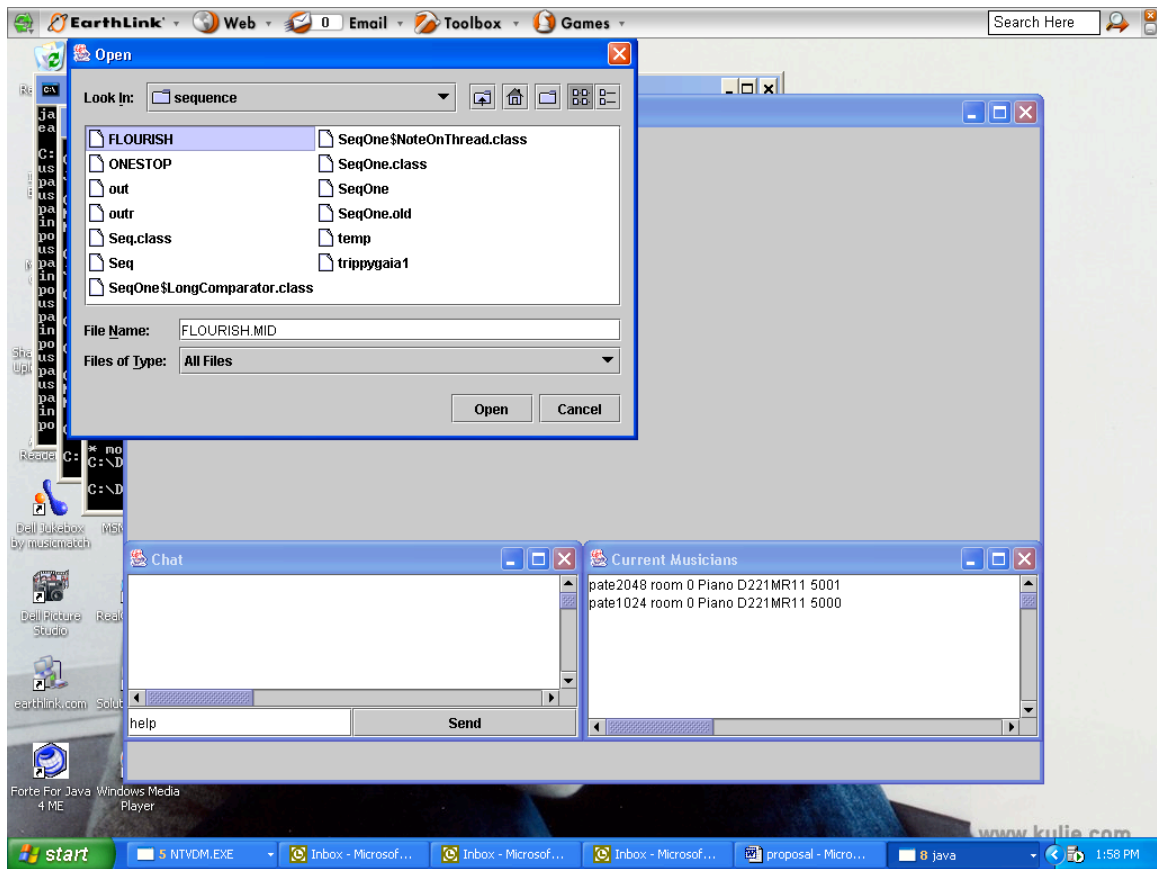


Figure 3-D-5 Standard Java Open File Dialog

### *E. Java and JINI Client/Server Duet System*

JINI is the Java based service discovery specification that was developed in the late 1990s by Sun Microsystems which was described briefly in Chapter 2. We used JINI to lookup the hostname and port of the duet system central server. This particular duet system consisted of two peers that communicate with one another and the central server. The central server relays peer IP addresses and server port numbers to interested peers. This system is very close to the Windows version of the final duet system, but has some latency saving change such as not using a virtual keyboard.

### *F. Another Musical Duet System Failure*

In this section we tried to develop another musical duet collaboration-system. The software engineering decisions required in creating a viable MIDI over IP musical duet system are as follows:

1. Choosing a network MIDI over IP protocol
2. Choosing an implementation programming language
3. Choosing the operating system(s) to be used
4. Choosing the MIDI interface hardware
5. Choosing the computer platform(s).

We used a series of quantitative experiments to automatically decide between several different MIDI over IP protocols. These protocols are described in another paper and consisted of TCP based RTP protocols, a simple and naïve TCP protocol, and a multithreaded TCP protocol [9]. Our MIDI over IP protocol is a multithreaded variation of TCP that utilizes MIDI short message producer and consumer threads. This protocol was found to be superior to the single thread TCP protocol and TCP based RTP protocols using two metrics performance that we considered useful.

We basically had two programming languages to choose from, namely, C and Java. Java became a logical choice with the advent of Java 1.5.0, which supports MIDI input on the PC platform. The programming language choice was more difficult and less straightforward than the quantitative protocol experiments. We could have used C++ also as a compromise between C and Java. In theory, C code should have the least amount of latency of the three previously mentioned programming languages. In order to determine the implementation programming language with an acceptable latency, we first

approached the problem with stripped down versions of the software that did not involve networking. We figured that if the local latency was unacceptable then there was no need to add networking to the equation. We found that both C and Java versions satisfied the local latency criterion using several different operating systems and hardware platforms. The next step was to add networking.

The operating systems that were available to us were Windows 98, Windows XP, and Mac OS X. We also had access to Sun Solaris 9 operating system; however, due to the lack of MIDI input hardware for Sun Solaris 9, that operating system was ruled out of the game. The MIDI subsystem of the audio system of OS X uses high priority kernel threads to execute the MIDI callback functions, which are very desirable low latency characteristics of OS X.

In the world of desktops and workstations there are essentially two predominant MIDI architectures: the MIDI subsystem devised by Apple for OS X and the much older and perhaps more mature Windows 95 multimedia system which has been enhanced several times in its decade long existence. First we start our discussion with the OS X MIDI subsystem. This subsystem uses client/server architecture. The first layer is the I/O toolkit of the kernel then the MIDI drivers then the MIDI server and clients, and finally the application. The subsystem has MIDI clients, MIDI sources, and MIDI destinations; MIDI input ports, and MIDI output ports. An application creates a MIDI client then it can add a MIDI input and/or output port. The MIDI input port calls back the application every time MIDI data is input into the port. MIDI data is encapsulated in a MIDI packet that has a length, timestamp, and the actual MIDI data bytes. MIDI packets are placed in a list structure that consists of one or more MIDI packets. Now onto the Microsoft MIDI

subsystem, which has MIDI input and output devices that have certain well defined capabilities. You open a MIDI input and/or output device then start the device, transmit or receive data; and then stop and close the device. This very simple architecture allows for either window or function call back entities. The fundamental unit of the MIDI transfer under Windows 9x+ is a double word, which can encapsulate all MIDI short messages and even system exclusive messages.

On the Windows 98 and Windows XP machines we had a choice of either using a MIDI to game port sound card adapter or a MIDI-Audio MIDI-Sport 2x2 MIDI to USB interface. On the OS X machines we were forced to use the MIDI-Audio MIDI-Sport 2x2 MIDI to USB interfaces only. Using qualitative tests, we found that either of the interfaces had satisfactory latency on a Windows OS machine.

The computing platforms available to us were two Dell computers at the primary researcher's house, some laptops, some older Pentium 3 systems, and two G4 dual processor PowerMacs. One Dell computer was a 450 MHz Pentium 2 system with 128 MB of RAM, 12 GB hard-drive, and a Turtle Beach Montego sound card that ran Windows 98. The other Dell computer was a 2.26 GHz Pentium 4 with 512 MB of RAM, 80 GB hard-drive, and Turtle Beach Santa Cruz DSP sound card that ran Windows XP Home Edition.

We have isolated a number of sources of latency in MIDI over IP. Delays originate in the MIDI controller, the MIDI to computer interface, scheduling delays in the MIDI kernel of the operating system, programming language latency, sound card latency, network propagation, and sound propagation latency to the listener's ear. By carefully

choosing hardware, software, and the network most of these sources of latency can be kept to acceptable minimums.

The MIDI hardware configuration on all machines is shown in Figure 3-F-1. The MIDI data emanates from a MIDI controller such as a guitar synthesizer, MIDI keyboard, or MIDI wind controller. Then the data goes into a MIDI/USB converter via a MIDI input port into the computer through the MIDI kernel of the operating system, out a MIDI output port on the MIDI/USB converter into a tone generator, and finally via an audio connection into a set of amplified speakers or an amplifier.

The software process or thread architecture for the duet system is shown in Figure 3-F-2. Each of the six central boxes represents a heavyweight thread (UNIX process) in the user address space and the outermost boxes are the MIDI kernel threads of OS X. The figure shows two peers communicating by TCP/IP. The MIDI data flow is from the MIDI main process which is responsible for creating a MIDI client and MIDI input and output ports, and connecting the MIDI input port to a MIDI source into the MIDI kernel and vice versa. Also a MIDI destination is selected is by the MIDI main process. The MIDI data flows from the MIDI kernel into MIDI send TCP process and over the wire to a peer's MIDI receive TCP process. We used lightweight user space threads under Windows and heavyweight threads (processes) under Mac OS X.

We developed a number of Java and C prototypes of the system on the Windows platform. On this platform we included a central server to take care of registration of the peers in the peer-to-peer network. This way a musician could utilize another musician's system wide username to find a duet partner. Two of the prototypes, one in Java, and the other using a Java native method written in C utilized the JINI 1.0 specification to handle

the job the central server discovery. This made it unnecessary for the end-user to type the central server's hostname or IP address and port number into the program. JINI is a service discovery protocol that is well suited for use on a communication network for finding local services. The central server concept was also used for chatting between the duet musician pairs.

The Windows software had a graphical user interface that consisted of a virtual piano keyboard that showed the local and remote notes being played and also had a useful feature to show the instruments being played by a MIDI sequence. Each MIDI channel had its own color. The virtual keyboard could also be used as a "virtual" MIDI controller by selecting a menu item. The current software on the PowerMac platform is purely command line driven, however, this situation will be remedied in the near future.

We performed quantitative experiments using a statistically small sample space of ten experimental instances per two MIDI sequences to be transmitted to determine the time required for the sequences to be played locally on the destination machine or over a LAN on the destination machine. Table 3-F-1 shows the time in seconds required to play the MIDI sequences on the destination machine with no networking involved. Table 3-F-2 shows the playing time at the destination for two MIDI sequences over a LAN that involved the Windows 98 machine above as the destination (receiver) and the Windows XP from above as the source (sender). A MIDI standard format 0 file is a sequence, which consists of one track, whereas a MIDI standard format 1 file consists of one or more tracks to be played simultaneously. In each case the sequence required more time to play over the LAN than locally which is to be expected due to network latency.

Sequence	MIN	AVG	MAX	STD
Format 0	66.7	66.8	67.3	0.2
Format 1	89.0	89.2	90.0	0.4

Table 3-F-1 MIDI Sequence Playing Time Locally

Sequence	MIN	AVG	MAX	STD
Format 0	67.2	67.4	67.4	0.1
Format 1	89.3	89.4	89.5	0.1

Table 3-F-2 MIDI Sequence Playing Time on a LAN

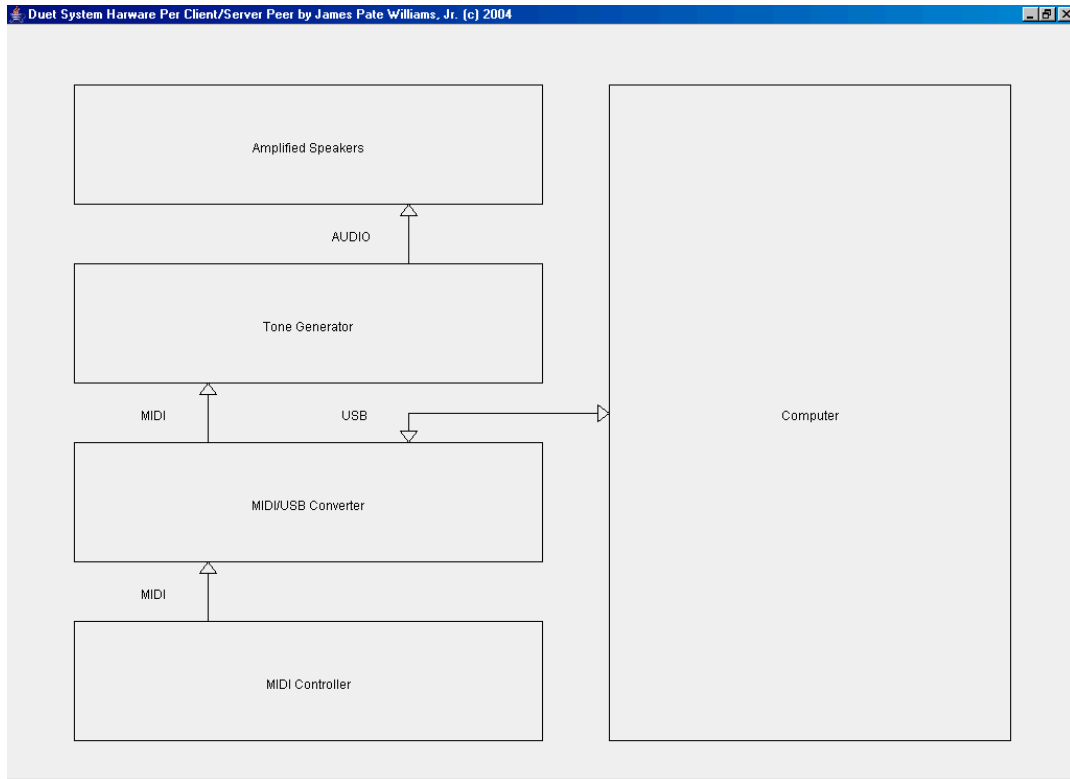


Figure 3-F-1 MIDI Hardware Configuration



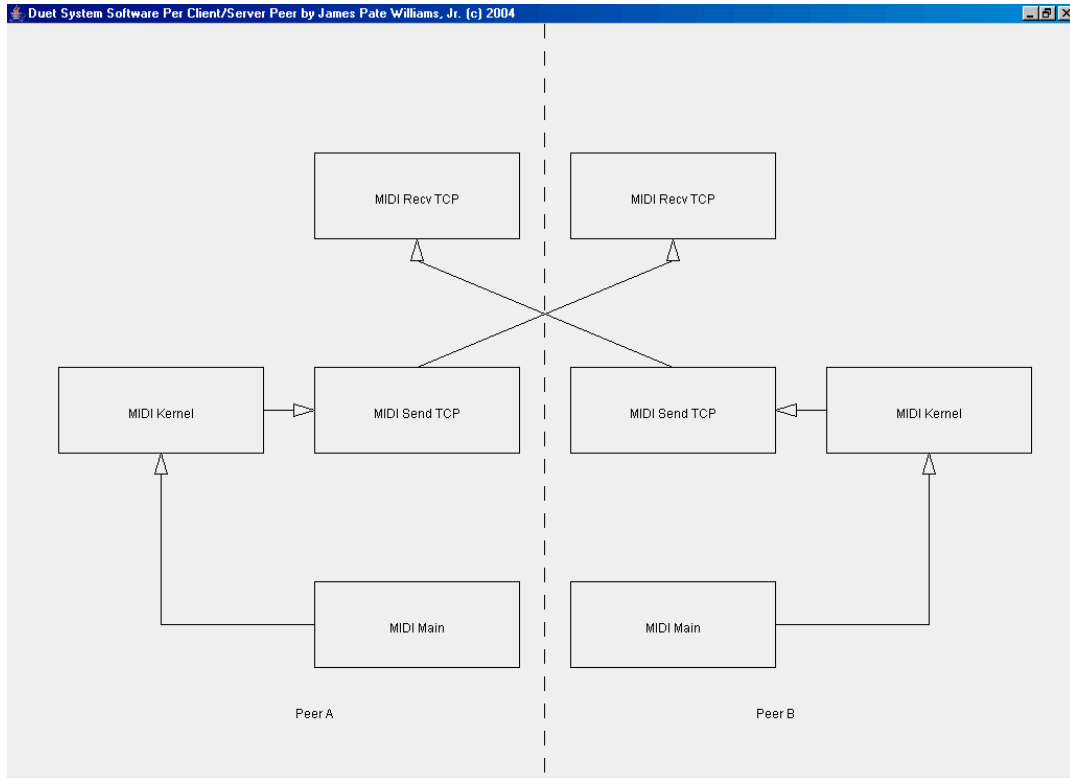


Figure 3-F-2 MIDI Software Configurations

## CHAPTER 4 RTP AND TCP PROTOCOLS

### *A. Introduction*

In this chapter we discuss the successful quantitative study of RTP and TCP protocols. All of our MOIP protocols utilized TCP as the underlying transport layer protocol due to the intolerance of MIDI for lost or out-of-order data. This inability to handle unreliable data delivery is due to the fact that a lost or out-of-order MIDI short message can have a catastrophic effect on a remote performance. Suppose the MIDI short message that turns off a certain note is lost then that note will sound indefinitely, and it is difficult for a musician to turn off a stuck note. The same situation can occur if the MIDI short message to turn a note off arrives before the MIDI short-message to turn the note on. We chose to use TCP rather than the newer reliable transport protocol the Stream Control Transport Protocol (SCTP) since TCP is ubiquitous and SCTP is just gaining acceptance [11]. A unique design feature of the Java Media Framework, which is a set of Java interfaces and objects that allow a Java application or applet to read or write streaming media such as audio or video using RTP, affords a choice of the underlying transport protocol for the JMF RTP implementation.

As was stated earlier RTP has two information channels available: one for control and one for data. Since there are two transport layer protocols that are readily available to the JMF version of RTP, namely, TCP and UDP, and there are two communication

channels in RTP, then there are four combinations of transport layer protocol and channels as shown in Table 4-1.

#	Control Channel	Data Channel
1	UDP	UDP
2	UDP	TCP
3	TCP	UDP
4	TCP	TCP

Table 4-A-1 Channel and Transport Layer Protocols for RTP

From a preliminary set of experiments, we were able to eliminate 1 and 3 due to unreliability. Also, we have two states of the Nagle algorithm: either it was enabled or disabled. We nicknamed our 6 RTP based protocols UT-RTP-ND, UT-RTP-NE, TT-RTP-NDND, TT-RTP-NDNE, TT-RTP-NEND, and TT-RTP-NENE, where the prefix UT meant UDP control channel and TCP data channel and ND stood for Nagle disabled whereas NE denoted that the Nagle algorithm was enabled. The baseline protocol was a vanilla variation of TCP, which we chose to call simple and naïve TCP, e.g. SN-TCP-ND and SN-TCP-NE. In addition another branch of the TCP tree of protocols was used which we refer to as MIDI producer and consumer thread TCP, e.g. PC-TCP-ND and PC-TCP-NE. PC-TCP is multithreaded and thus is able to play a MIDI command locally concurrently with possibly sending the command over the network. This means that our experimental protocol basis set consisted of 10 protocols.

The data structure that was transmitted and received by the protocols consisted of a MIDI short message and a delta-time in milliseconds. The MIDI short message was

composed of a channel byte, a command byte, and two data bytes. The delta-time was a Java long, which is 8 bytes or 64 bits in length. There was  $m$  of these data structures per packet where  $m$  was 1, 2, 3, or 4. The following figure illustrates the preceding data structure.

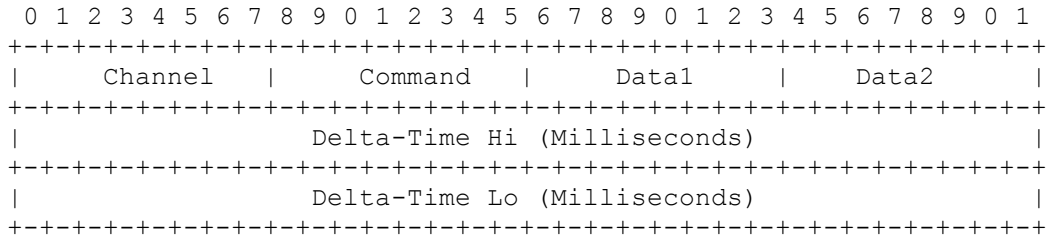


Figure 4-A-1 MIDI Short Message Format and Delta-Time

There are two different strategies that can be used as far as the delta-times that a sender reports are concerned. Either the delta-times can be initialized to zero, in which case we call this dishonesty or lying delta-time policy, or the true delta-time between MIDI short messages is specified, and this sort of policy is called the honesty delta-time policy.

*B. Experimental Procedure*

Using the 10 protocols of the previous section, we utilized two networks and conducted 60 experiments per network per protocol per value of  $m$  (which you will recall was the number of MIDI short messages per packet). We used  $m = 1, 2, 3,$  and  $4$ . So this meant we performed  $2 * 10 * 60 * 4 = 4800$  experimental instances. After consulting with a statistician, we decided to use a statistically large number of experiments [47]. In this set of experiments we utilized the dishonesty delta-time policy of the previous section.

The networks we used were a LAN and a WAN with a dialup link and an asymmetric digital subscriber line (ADSL) leg. The computers that formed the LAN and WAN were only a few feet apart in a residence, the WAN formed approximately 150 miles in wired distance. Both endpoints of the connections were Dell computers. The dialup computer was a Windows 98 machine with a 450 MHz Pentium 2 processor, 128 MB of RAM, and Turtle Beach Montego sound card. The other computer was a Windows XP Home Edition box with a 2.26 GHz Pentium 4 processor, 512 MB RAM, and Turtle Beach Santa Cruz DSP sound card. The dialup baud rate was a constant 31.2 kbps throughout the experiments.

We were interested in accumulating two metrics to measure the performance of each protocol. The first and easiest to understand metric was the time required to play a MIDI format 0 file, which had been sequenced and transmitted over the Internet, on the destination host. A MIDI (format 0) file consists of a single track, whereas the other common MIDI (format 1) file has one or more tracks, which are to be played simultaneously. We want these numbers to be close to the time required to play the sequence on the destination host without any networking. These measurements gave a rough approximation of the overall network latency. The second metric is more difficult to interpret and was rough measurement of the jitter on the networks. This metric involved gathering the inter-departure and inter-arrival times then calculating a simple function based on the absolute difference in the inter-departure time minus the inter-arrival time divided by the inter-departure time. The temporal relationships between the inter-departure and inter-arrival times are illustrated in Figure 4-B-1.

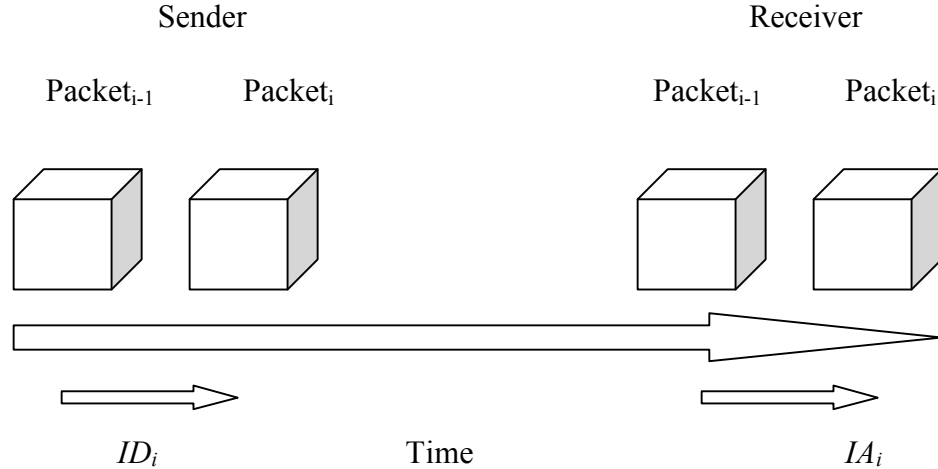


Figure 4-B-1 Inter-Departure Time and Inter-Arrival Time Temporal Relationships

The function we used for the jitter measurement is as shown in Equation (1):

$$(1) d_i = 100 \times |ID_i - IA_i| / ID_i$$

Where the index,  $i$ , runs from 2 to the number of packets. The inter-departure times and inter-arrival times are defined by Equations (2) and (3):

$$(2) ID_i = D_i - D_{i-1}$$

$$(3) IA_i = A_i - A_{i-1}$$

In Equations (2) and (3),  $D_i$  is the departure time of the  $i$ th packet and  $A_i$  is the arrival time of the  $i$ th packet and the indices are the same as in Equation (1).

### C. Experimental Results

As was previously mentioned the number of experimental instances was 4800.

We were able to distill this data into  $2 * (9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1) * 4 = 2 * 45 * 4 = 360$  graphs of mean runtime at destination paired comparison, and  $2 * 10 * 4 = 80$  graphs of the Equation (1). We also generated Student's paired means t-test data using a significance level of 5% which have been reduced to 4 tables per network, one table for each value of  $m$ . Figures 4-C-1 to 4-C-4 display histograms of the mean run time at the

destination for PC-TCP-ND (black) and SN-TCP-ND (red) on the WAN. The x-axis has the experiment number which runs from 1 to 60 and y-axis is the runtime at the destination in milliseconds. PC-TCP-ND statistically outperformed SN-TCP-ND in each of the 4 cases.

Graphs of all the experiments involving Equation (1) are to be found in Appendix A and Appendix C for the LAN and WAN, respectively. The corresponding paired comparison graphs are found in Appendix B and Appendix D for the LAN and WAN, respectively. The paired comparison statistical data is to be found in Appendix E and Appendix F for the LAN and WAN, respectively.

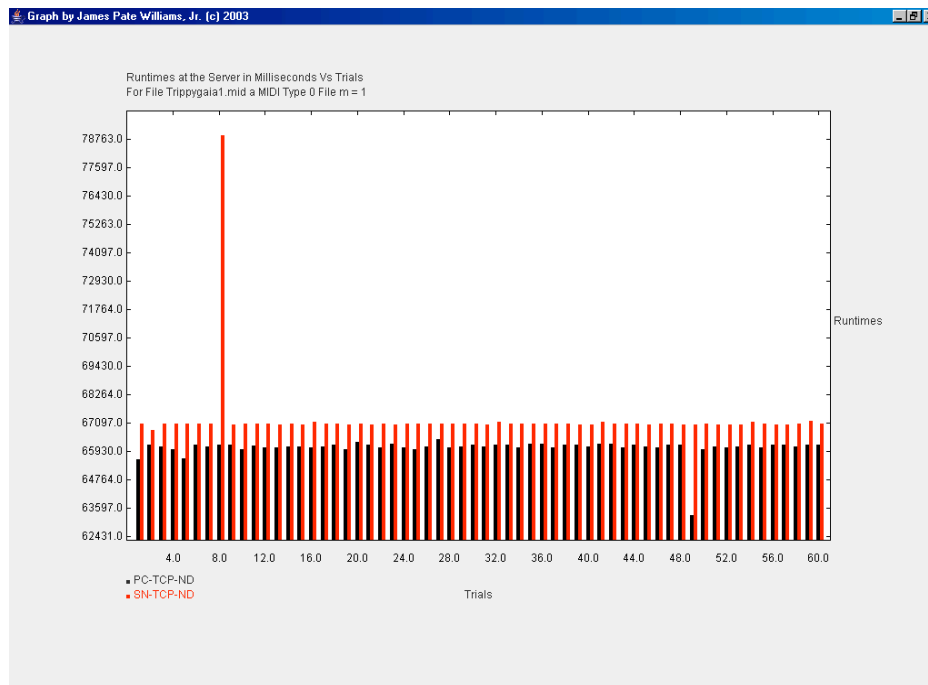


Figure 4-C-1 PC-TCP-ND VS SN-TCP-ND m = 1

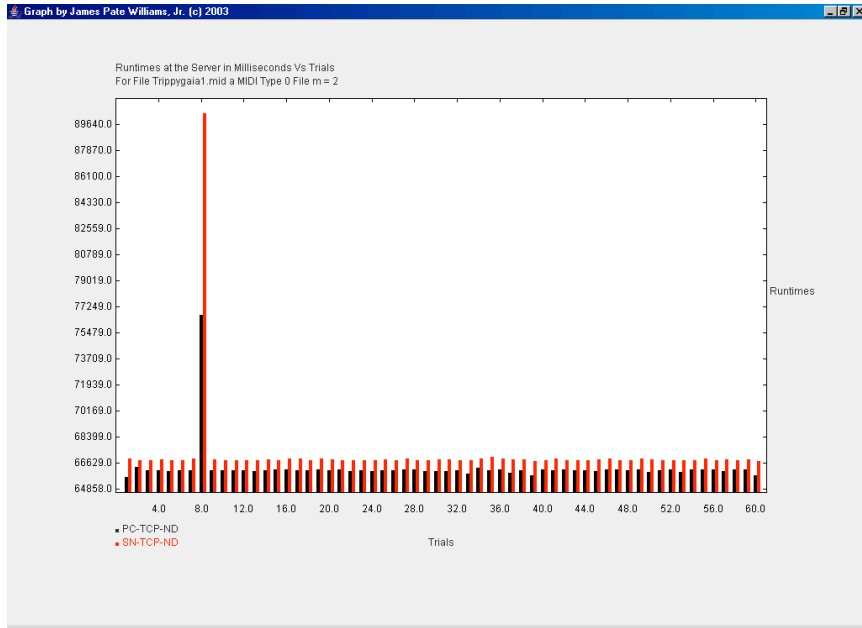


Figure 4-C-2 PC-TCP-ND VS SN-TCP-ND m = 2

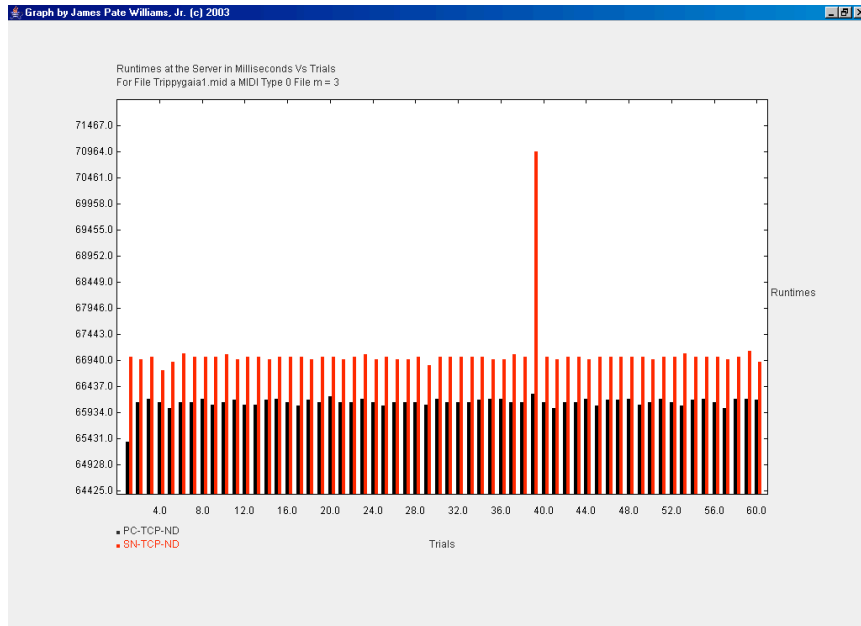


Figure 4-C-3 PC-TCP-ND VS SN-TCP-ND m = 3



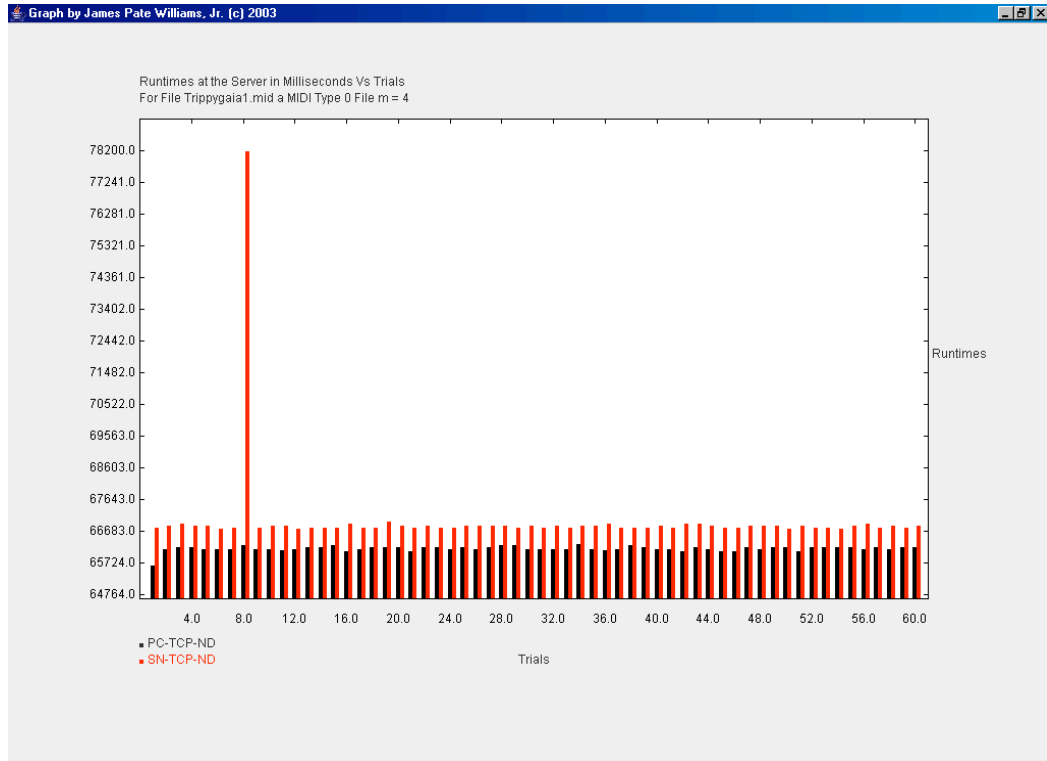


Figure 4-C-4 PC-TCP-ND VS SN-TCP-ND m = 4

Table 4-C-1 to 4-C-4 show the paired means Student's t-test signs of the t-statistics and significances for the 10 protocols on the LAN. These tables were generated from the statistical data in Appendix E. The sign is determined from the sign of the t-statistic. If the absolute value of the combined sign of the t-statistic and the significance is less than or equal 0.05 then one of the protocols in a row and column outperformed the other. If the combined sign of the t-statistic and the significance is negative and has an absolute value less than or equal 0.05 then the row protocol statistically outperformed the column protocol. On the other hand, if the combined sign of the t-statistic and the significance is positive and less than or equal 0.05 then the column protocol statistically did better than the row protocol. The protocol names have been shortened to create row and column labels as follows: NDND = TT-RTP-NDND, NEND = TT-RTP-NEND,

NDNE = TT-RTP-NDNE, NENE = TT-RTP-NENE, UTND = UT-RTP-ND, UTNE = UT-RTP-NE, PCND = PC-TCP-ND, PCNE = PC-TCP-NE, SNND = SN-TCP-ND and SNNE = SN-TCP-NE.

Looking at UTND row and the PC-TCP-NE column, we find a combined sign of the t-statistic and the significance of 0.002, which means that PC-TCP-NE statistically won the battle over UT-TCP-ND. Now look at the NENE column in the same row and the combined sign of the t-statistic and the significance is -0.164, which means that the two protocols were statistically equivalent. From the tables it is apparent that PC-TCP-ND and PC-TCP-NE were statistically the best protocols. What is surprising is that for  $m = 2$  and  $m = 3$  is that PC-TCP-NE beat PC-TCP-ND statistically.

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.673	-0.306	-0.288	0.049	-0.367	0.000	0.000	0.491	0.462
<i>NEND</i>	0.673	-	-0.315	-0.307	0.083	-0.376	0.000	0.001	0.514	0.476
<i>NDNE</i>	0.306	0.315	-	0.325	0.233	-0.986	0.000	0.000	0.187	0.062
<i>NENE</i>	0.288	0.307	-0.325	-	0.164	-0.432	0.000	0.000	0.000	-0.848
<i>UTND</i>	-0.075	-0.083	-0.233	-0.164	-	-0.292	0.000	0.002	-0.303	-0.337
<i>UTNE</i>	-0.049	0.376	0.986	0.432	0.292	-	0.000	0.000	0.294	0.224
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.000	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.001	-0.000	-0.000	-0.002	-0.000	0.000	-	-0.000	-0.000
<i>SNND</i>	-0.491	-0.514	-0.187	-0.000	0.303	0.294	0.000	0.000	-	-0.421
<i>SNNE</i>	-0.462	-0.476	-0.062	0.848	0.337	-0.224	0.000	0.000	0.421	-

Table 4-C-1 Combined Sign of t-Statistic and Statistical Significance Table  $m = 1$

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.222	0.169	0.075	0.466	0.005	0.000	0.000	0.269	0.274
<i>NEND</i>	0.222	-	0.032	0.020	-0.493	0.000	0.000	0.000	0.300	0.303
<i>NDNE</i>	-0.169	-0.032	-	0.812	0.438	0.107	0.000	0.000	-0.239	-0.244
<i>NENE</i>	-0.075	-0.020	-0.812	-	-0.433	0.225	0.000	0.000	-0.233	-0.239
<i>UTND</i>	-0.466	0.493	-0.438	0.433	-	0.407	0.000	0.000	0.950	-0.967
<i>UTNE</i>	-0.005	-0.000	-0.107	-0.225	-0.407	-	0.000	0.000	-0.208	-0.215
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	0.266	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-0.266	-	-0.000	-0.000
<i>SNND</i>	-0.269	-0.300	0.239	0.233	-0.950	0.208	0.000	0.000	-	-0.456
<i>SNNE</i>	-0.274	-0.303	0.244	0.239	0.967	0.215	0.000	0.000	0.456	-

Table 4-C-2 Combined Sign of t-Statistic and Statistical Significance Table  $m = 2$

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.673	-0.497	0.395	0.499	0.257	0.004	0.000	0.501	0.472
<i>NEND</i>	0.673	-	-0.324	0.304	0.160	0.176	0.000	0.000	0.132	0.397
<i>NDNE</i>	0.497	0.324	-	0.312	0.241	0.242	0.000	0.000	0.509	0.362
<i>NENE</i>	-0.395	-0.304	-0.312	-	-0.942	0.081	0.003	0.000	-0.247	-0.475
<i>UTND</i>	-0.499	-0.160	-0.241	-0.942	-	0.267	0.000	0.000	-0.149	-0.864
<i>UTNE</i>	-0.257	-0.176	-0.242	-0.081	-0.267	-	0.002	0.000	-0.161	-0.192
<i>PCND</i>	-0.004	-0.000	-0.000	-0.003	-0.000	-0.002	-	0.385	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-0.385	-	-0.000	-0.000
<i>SNND</i>	-0.501	-0.132	-0.509	0.247	0.149	0.161	0.000	0.000	-	0.313
<i>SNNE</i>	-0.472	-0.397	-0.362	0.475	0.864	0.192	0.000	0.000	-0.313	-

Table 4-C-3 Combined Sign of t-Statistic and Statistical Significance Table  $m = 3$

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.096	-0.315	-0.454	-0.517	0.000	0.000	0.000	0.002	0.001
<i>NEND</i>	0.096	-	-0.329	0.240	-0.612	0.000	0.000	0.000	0.052	-0.038
<i>NDNE</i>	0.315	0.329	-	0.321	0.403	0.247	0.000	0.005	0.346	0.362
<i>NENE</i>	0.454	-0.240	-0.321	-	-0.556	0.000	0.000	0.000	-0.006	-0.004
<i>UTND</i>	-0.517	0.612	-0.403	0.556	-	0.173	0.000	0.000	0.718	0.728
<i>UTNE</i>	-0.000	-0.000	-0.247	-0.000	-0.173	-	0.000	0.000	-0.000	-0.000
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.000	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.005	-0.000	-0.000	-0.000	0.000	-	-0.000	-0.000
<i>SNND</i>	-0.002	-0.052	-0.346	0.006	-0.718	0.000	0.000	0.000	-	-0.813
<i>SNNE</i>	-0.001	0.038	-0.362	0.004	-0.728	0.000	0.000	0.000	0.813	-

Table 4-C-4 Combined Sign of t-Statistic and Statistical Significance  $m = 4$

The tables Table 4-C-5 to Table 4-C-8 display WAN statistical significances, which are analogous as far as interpretation goes with the LAN values in Tables 2 to 5., and were compiled from the data in Appendix F. Please note that in the WAN case that PC-TCP-ND outperforms all of the other protocols except for PC-TCP-NE in every  $m$  case.

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	0.311	-0.016	-0.237	-0.172	0.288	0.000	0.000	-0.136	-0.209
<i>NEND</i>	-0.311	-	-0.221	-0.273	-0.223	-0.669	0.000	0.000	-0.263	-0.280
<i>NDNE</i>	0.016	0.221	-	-0.369	-0.280	0.199	0.000	0.000	-0.980	-0.774
<i>NENE</i>	0.237	0.273	0.369	-	-0.736	0.261	0.000	0.000	0.296	0.262
<i>UTND</i>	0.172	0.223	0.280	0.736	-	0.210	0.000	0.000	0.221	0.202
<i>UTNE</i>	-0.288	0.669	-0.199	-0.261	-0.210	-	0.000	0.000	-0.244	-0.262
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.088	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-0.088	-	-0.000	-0.000
<i>SNND</i>	0.136	0.263	0.980	-0.296	-0.221	0.244	0.000	0.000	-	-0.575
<i>SNNE</i>	0.209	0.280	0.774	-0.264	-0.202	0.262	0.000	0.000	0.575	-

Table 4-C-5 Combined Sign of t-Statistic and Statistical Significance Table m = 1

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.936	-0.930	-0.366	0.269	0.524	0.000	0.034	-0.744	-0.488
<i>NEND</i>	0.936	-	0.975	-0.103	0.319	0.539	0.000	0.000	-0.658	-0.047
<i>NDNE</i>	0.930	0.975	-	-0.355	0.000	0.006	0.000	0.024	-0.753	-0.466
<i>NENE</i>	0.366	0.103	0.355	-	0.175	0.235	0.000	0.000	0.268	0.179
<i>UTND</i>	-0.269	-0.319	-0.000	-0.175	-	-0.000	0.000	0.060	-0.540	-0.175
<i>UTNE</i>	-0.524	-0.539	-0.006	-0.235	0.000	-	0.000	0.042	-0.620	-0.266
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.129	-0.000	-0.000
<i>PCNE</i>	-0.034	-0.000	-0.024	-0.000	-0.060	-0.042	0.129	-	-0.000	-0.000
<i>SNND</i>	0.744	0.658	0.753	-0.268	-0.540	0.620	0.000	0.000	-	-0.943
<i>SNNE</i>	0.488	0.047	0.466	-0.179	0.175	0.266	0.000	0.000	0.943	-

Table 4-C-6 Combined Sign of t-Statistic and Statistical Significance Table m = 2

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.651	-0.489	0.593	0.138	0.282	0.000	0.000	0.595	0.388
<i>NEND</i>	0.651	-	-0.004	0.503	0.299	0.376	0.000	0.000	0.490	0.424
<i>NDNE</i>	0.489	0.004	-	0.354	0.188	0.249	0.000	0.000	0.349	0.288
<i>NENE</i>	-0.593	-0.503	-0.354	-	0.000	0.000	0.000	0.000	0.856	0.000
<i>UTND</i>	-0.138	-0.299	-0.188	-0.000	-	-0.000	0.000	0.000	0.209	-0.000
<i>UTNE</i>	-0.282	-0.376	-0.249	-0.000	0.000	-	0.000	0.000	-0.521	-0.005
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.228	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	0.228	-	-0.000	-0.000
<i>SNND</i>	-0.595	-0.490	-0.349	-0.856	-0.209	0.521	0.000	0.000	-	0.749
<i>SNNE</i>	-0.388	-0.424	-0.288	-0.000	0.000	0.005	0.000	0.000	-0.749	-

Table 4-C-7 Combined Sign of t-Statistic and Statistical Significance Table m = 3

<i>P/P</i>	<i>NDND</i>	<i>NEND</i>	<i>NDNE</i>	<i>NENE</i>	<i>UTND</i>	<i>UTNE</i>	<i>PCND</i>	<i>PCNE</i>	<i>SNND</i>	<i>SNNE</i>
<i>NDND</i>	-	-0.514	-0.002	-0.142	-0.141	-0.203	0.000	0.000	-0.712	-0.092
<i>NEND</i>	0.514	-	-0.012	-0.196	-0.204	-0.000	0.000	0.000	-0.858	-0.175
<i>NDNE</i>	0.002	0.012	-	-0.289	-0.337	0.053	0.000	0.000	0.490	-0.406
<i>NENE</i>	0.142	0.196	0.289	-	0.365	0.229	0.000	0.000	0.005	0.233
<i>UTND</i>	0.141	0.204	0.337	-0.365	-	0.247	0.000	0.000	0.001	0.334
<i>UTNE</i>	0.203	0.000	-0.053	-0.229	-0.247	-	0.000	0.000	-0.983	-0.238
<i>PCND</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	-	-0.166	-0.000	-0.000
<i>PCNE</i>	-0.000	-0.000	-0.000	-0.000	-0.000	-0.000	0.166	-	-0.000	-0.000
<i>SNND</i>	0.712	0.858	-0.490	-0.005	-0.001	0.983	0.000	0.000	-	-0.000
<i>SNNE</i>	0.092	0.175	0.406	-0.233	-0.334	0.238	0.000	0.000	0.000	-

Table 4-C-8 Combined Sign of t-Statistic and Statistical Significance Table m = 4

Figures 4-C-5 to 4-C-8 show graphs of Equation (1) on the LAN. We believe that the peak at 105% is the most significant feature of the histograms. The smaller this indicative bar then the better the protocol is with respect to transmitting a MIDI sequence over the network. Figure 4-C-9 has a three dimensional representation of the data that was used to create Figure 4-C-5.

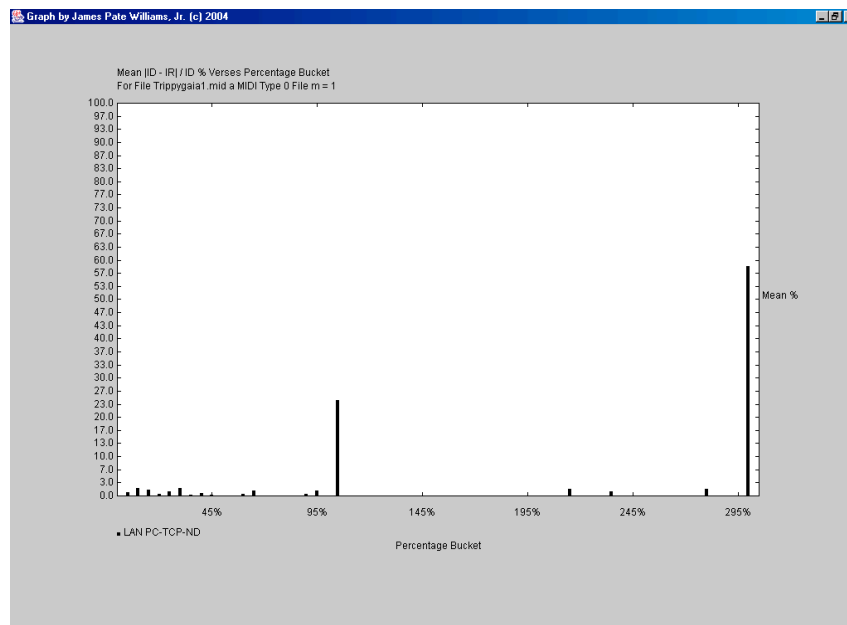


Figure 4-C-5 PC-TCP-ND m = 1

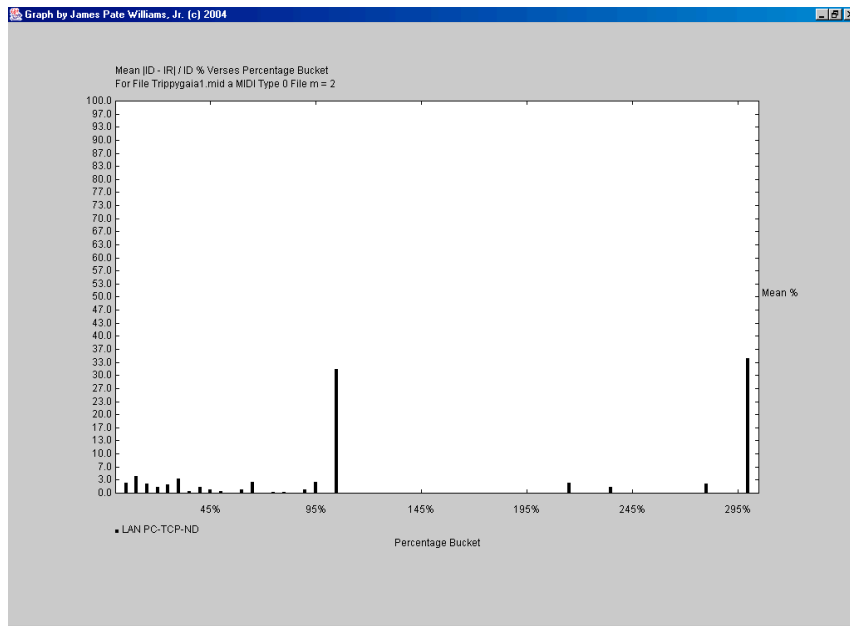


Figure 4-C-6 PC-TCP-ND m = 2

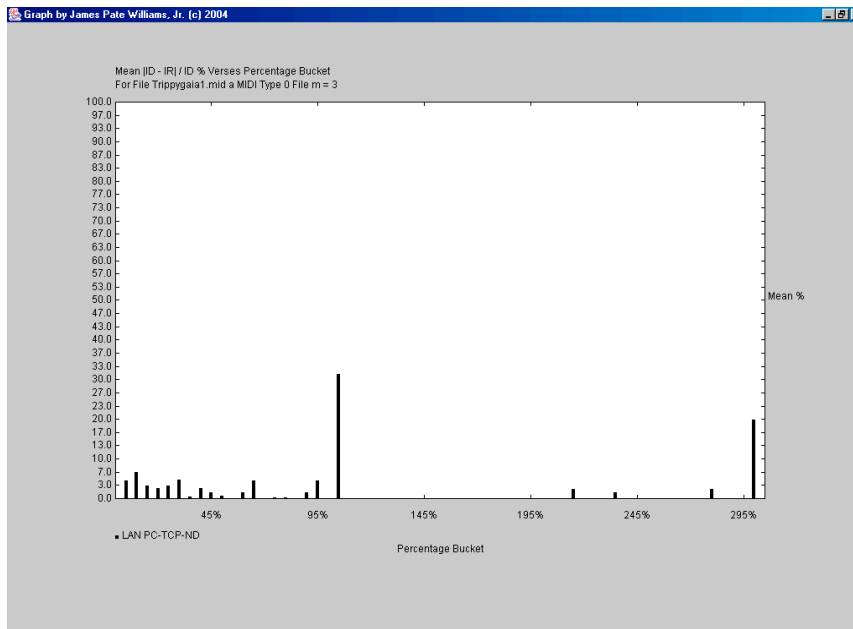


Figure 4-C-7 PC-TCP-ND m = 3

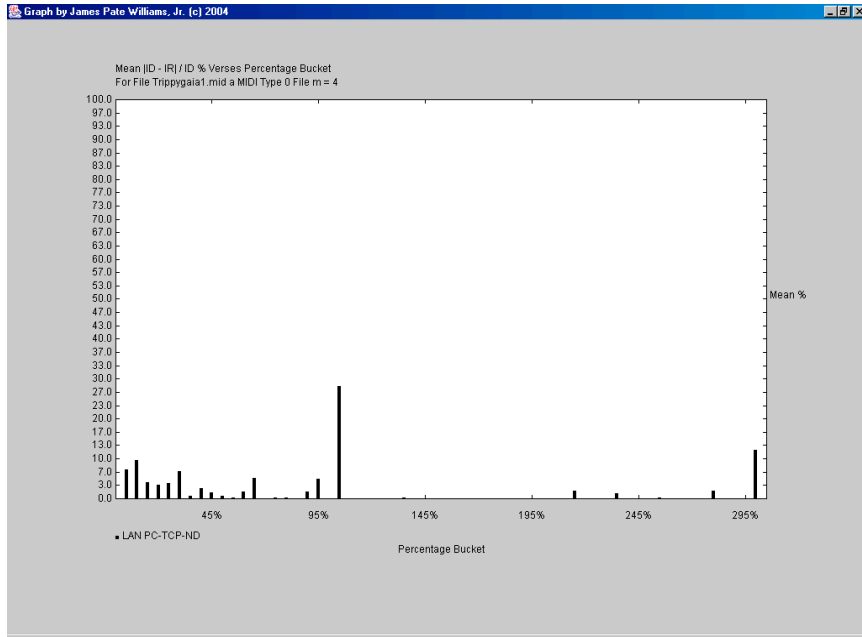


Figure 4-C-8 PC-TCP-ND  $m = 4$

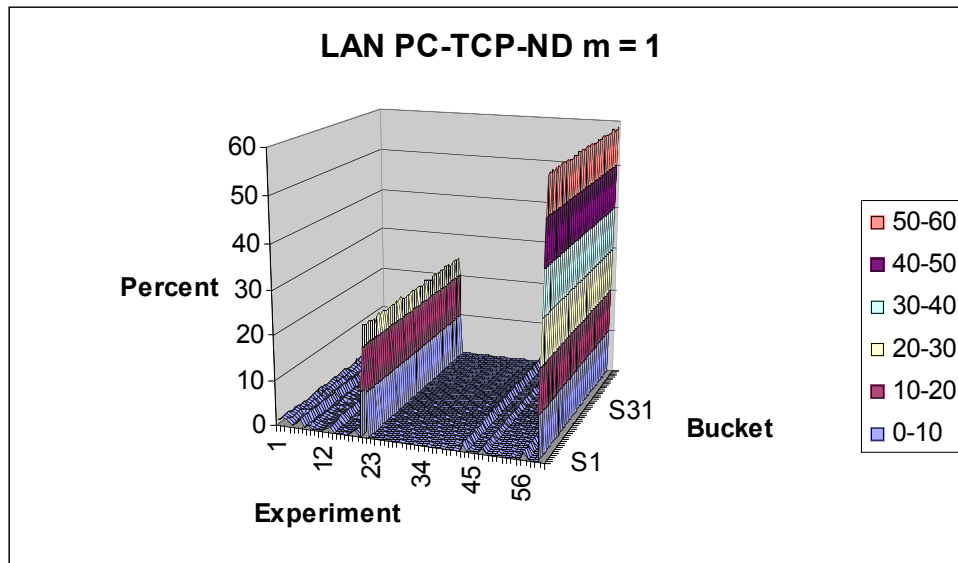


Figure 4-C-9 Equation (1) Plot for LAN PC-TCP-ND  $m = 1$

C. *Musical Duet System*

In the initial stages of the development of a musical duet collaboration system, we wanted to use Java as the implementation programming language; however, it was thought that the language had too much inherent latency. Java 1.5.0 does support both

MIDI input and output on all platforms, but the Apple version of Java 1.5.0 won't be released until the OS X Tiger becomes available in spring 2005. Our target platforms for the duet system were both Windows and OS X with ANSI C being a common language of the operating systems. ANSI C is probably lowest latency higher-level language available on both platforms since Windows and OS X extensively use C APIs.

We built a duet system on Windows, which had some common elements as the RMCP system of Goto et al. mentioned earlier. Both systems had a virtual piano keyboard for displaying keys being played or for mouse input of notes. Our display also showed the general MIDI instruments being utilized by each MIDI channel. Similarly we designed a non-GUI duet system for the OS X system. The Windows system used lightweight threads, whereas the OS X system went with heavyweight threads (processes). Both the Windows and OS X systems were peer-to-peer in nature instead of the classical client/server architecture.

The Windows system initial dialog is shown in Figure 4-C-10 and in Figure 4-C-11 is the main window.



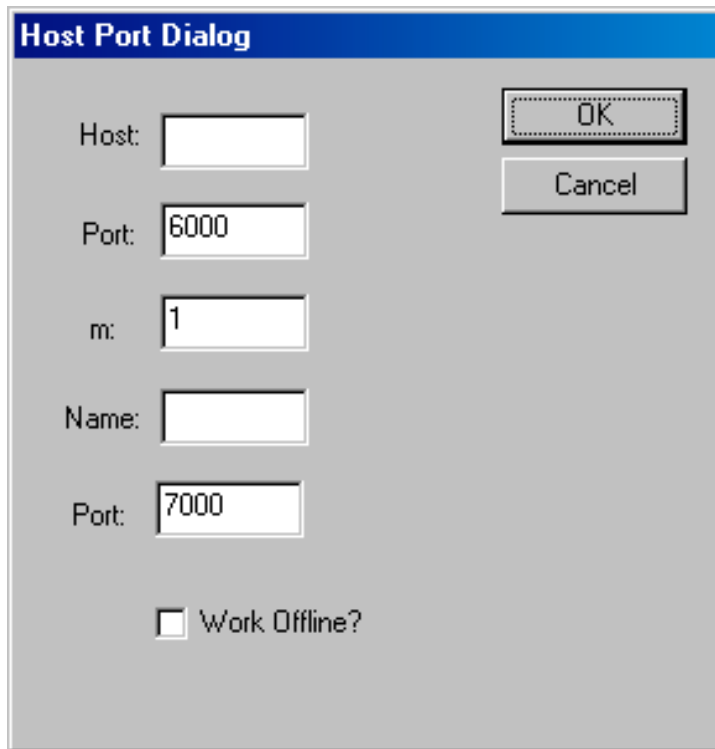


Figure 4-C-10 Duet System Initial Dialog

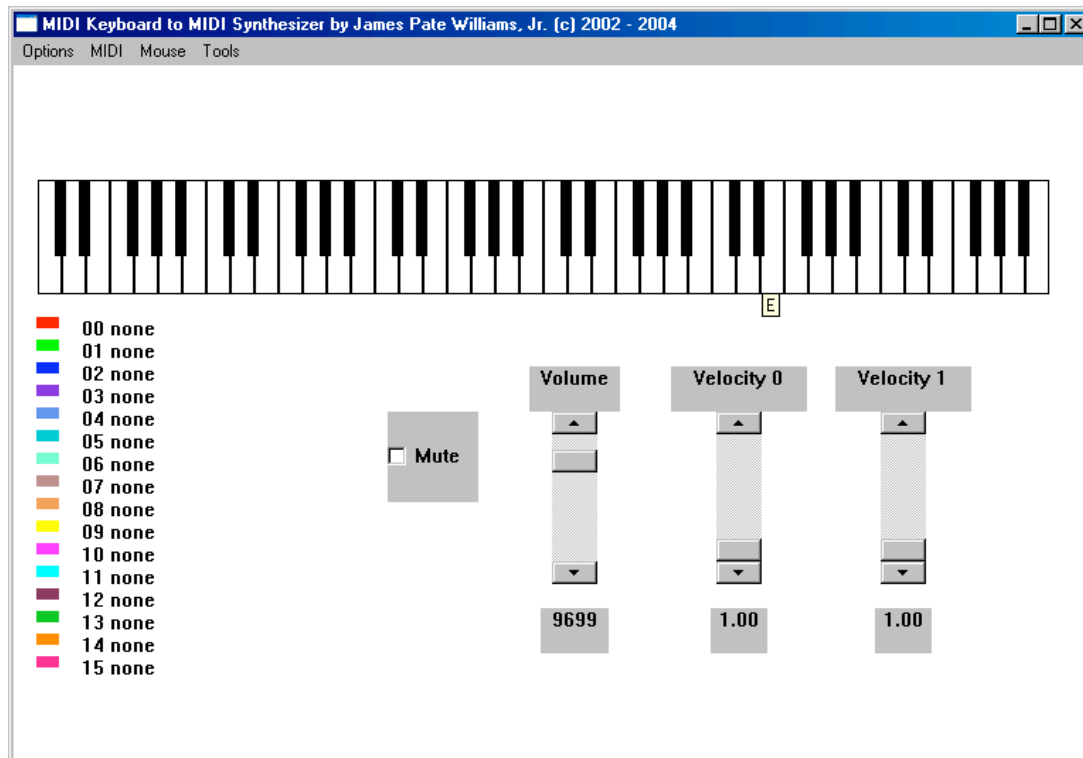


Figure 4-C-11 Duet System Main Window

We isolated a number of sources of latency in our duet systems, namely, MIDI input, programming language, operating system, network, sound card, and speakers. By carefully choosing the hardware and software the delays could be made acceptable.

The hardware configuration of the duet system consisted of five major components: computer, MIDI controller, MIDI-to-USB converter, tone generator, and speakers. We used three different types of MIDI controllers: Yamaha CBX-K2 keyboard controller, Roland GR-33 guitar synthesizer and Roland-ready Fender Stratocaster guitar, and a Yamaha WX5 wind controller. Of the three controllers utilized the keyboard controller seemed to have the most acceptable latency. On the Windows platform we tried to types of MIDI input, the direct MIDI-to-soundcard cable and the MIDI-to-USB converted. Both of these input methods appeared to be the same to us in terms of delay. Only MIDI-to-USB input was available for OS X. The hardware configuration is illustrated in Figure 4-C-12.

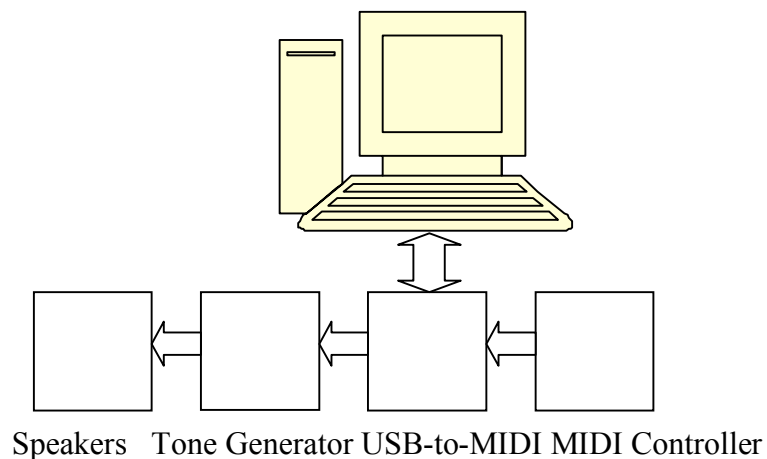


Figure 4-C-12 Duet System Hardware Configurations

The software configuration is specified in Figure 4-C-13. As has been said the networking architecture is peer- to-peer rather than the classical client/server paradigm.

Each peer consists of three processes or threads: MIDI receive, MIDI main, and MIDI send. The MIDI main entity is responsible for MIDI input and output via the MIDI-to-USB converter. The MIDI receive process or thread blocks until a packet is received then it dispatches the MIDI data in the packet to the MIDI main process or thread to be played. The MIDI send process or thread is responsible for transmitting MIDI data that from the MIDI main process or thread to the Internet.

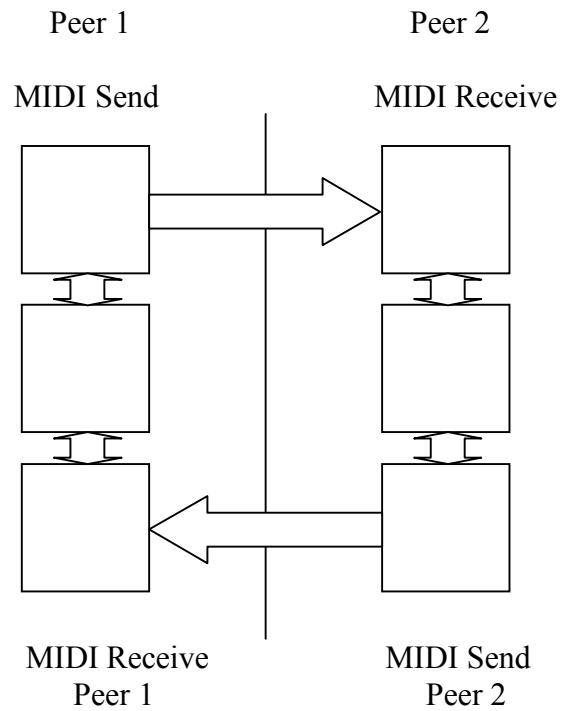


Figure 4-C-13 Duet System Software Configurations

## CHAPTER 5 MUSICAL DUET SYSTEM

Many of you may have heard the old adage that in the real estate industry everything is “location, location, location”, well in our case we substitute latency for location. We wanted to reduce the sources of latency as much as possible. As we have stated previously paper the sources of latency in this particular application area are: MIDI input and output stream latency, the hardware and software computer delays, network delays, and the latency in going from the computer speakers to listener’s ears. Our primary concern was to minimize latency.

The application evolved through many different versions from a monolithic program to an application that consists of a fair number of implementation modules that are discussed in the next section.

The Apple Carbon based MIDI duet application consists of the source code files shown in Table 5-1. Internally, the modules are pretty sparingly documented, so the number of lines of code (LOC) per module is pretty accurately portrayed in the second column of Table 5-1. The complete application has a modest total of about three thousand LOC.

Source File	Lines
Apple.h	22
Apple.c	318
Duet.h	117
Duet.c	628
Globals.h	108
Glabals.c	103
Main.c	527
MIDI.h	21
MIDI.c	732
NavFunctions.h	14
NavFunctions.c	107
TCPNetwork.h	16
TCPNetwork.c	324
Total	3037

Table 5-1. Source Code Files and Lines of Code

The graphical user interface (GUI) elements utilized by the application are: alerts, check boxes, combination (combo) boxes, edit boxes, menus, popup buttons, push buttons, radio buttons, and windows. The main window is displayed in Figure 7-1. The window has a total of seventeen GUI elements with an estimated number of states equal to  $2^4 * 2^3 * 2^2 * 2^2 * 2^1 * 2^1 * 2^3 * 2^3 * 2^3 * 2^1 * 2^1 = 2^{24} = 16,777,216$  not counting the edit box states. Obviously, this far too many states to exhaustively test by hand so it would be really nice to have some testing mechanism comparable to the Palm Operating System (OS) Gremlins for testing the interface.

The main window has check boxes for controlling the delta-time policy, which is either the honesty policy or lying policy, enabling the Nagle algorithm, local playing of a MIDI sequence, and muting of the audio system. The number of MIDI short messages per TCP packet is controlled by a popup button and defaults to one MIDI short message per packet. The peer and my ports default to TCP port number 5000. The peer host name

or Internet Protocol (IP) address is entered via an edit box. Popup buttons also allow the user to choose the MIDI input and output devices. The channel map and virtual keyboard window opening functions are implemented using check boxes. Fly over hints and their voice narration are controlled utilizing a check box whose default state is no fly over hints. The channel map and virtual keyboard windows are shown in Figures 5-2 and 5-3. The channel map window allows each local MIDI channel to be mapped to the same or different remote channel, which allows a duet to be played without the collision of local and remote MIDI short messages.

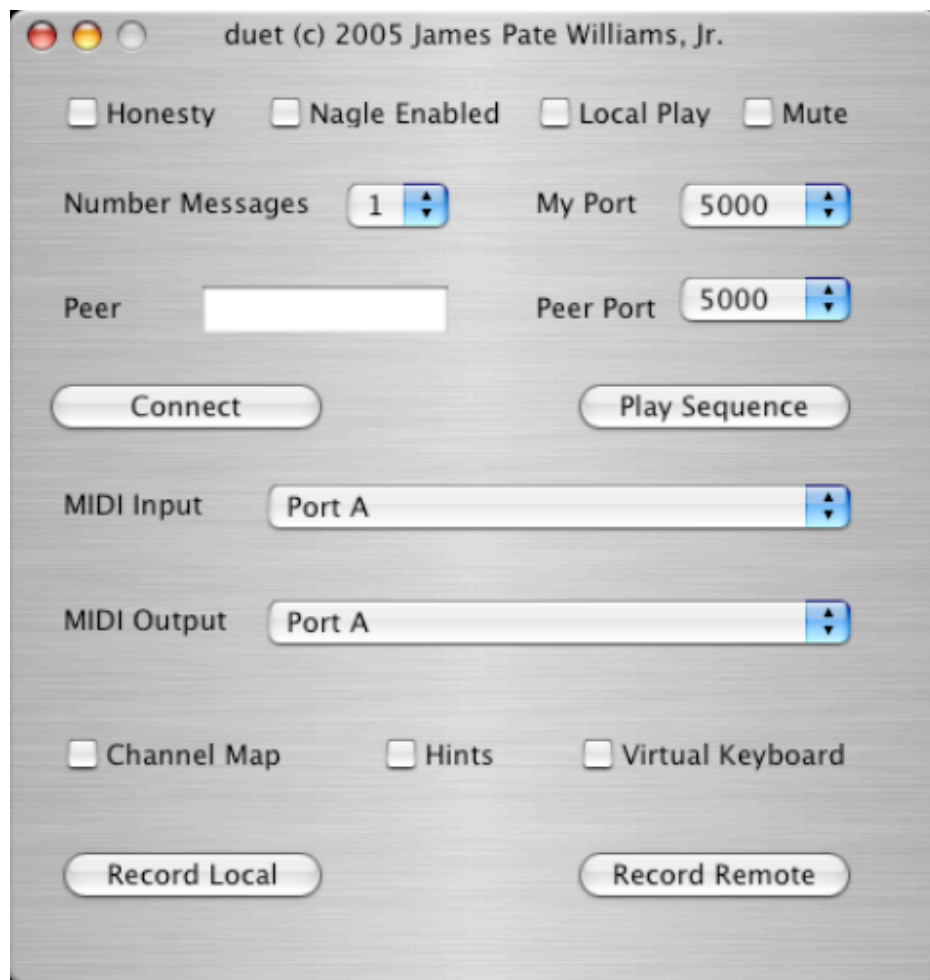


Figure 5-1 Main Windows

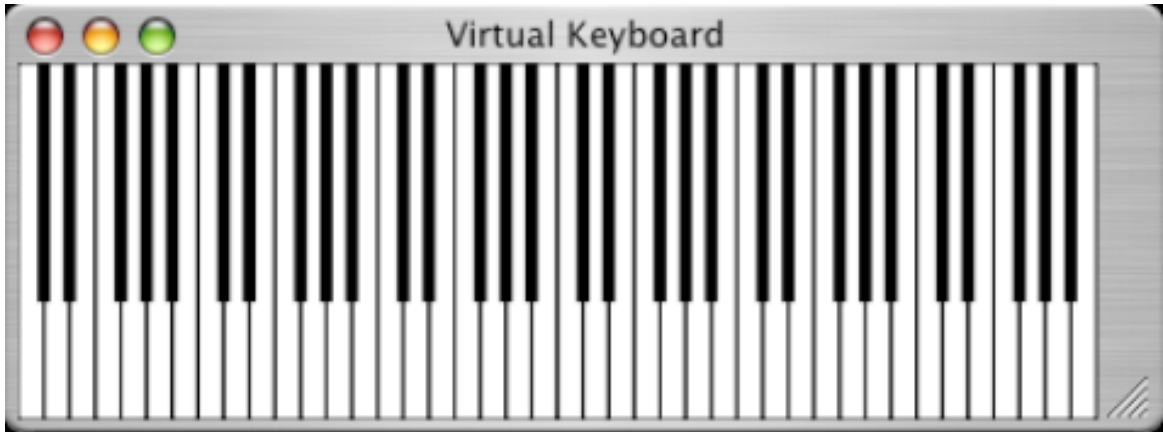


Figure 5-2 Virtual Keyboard

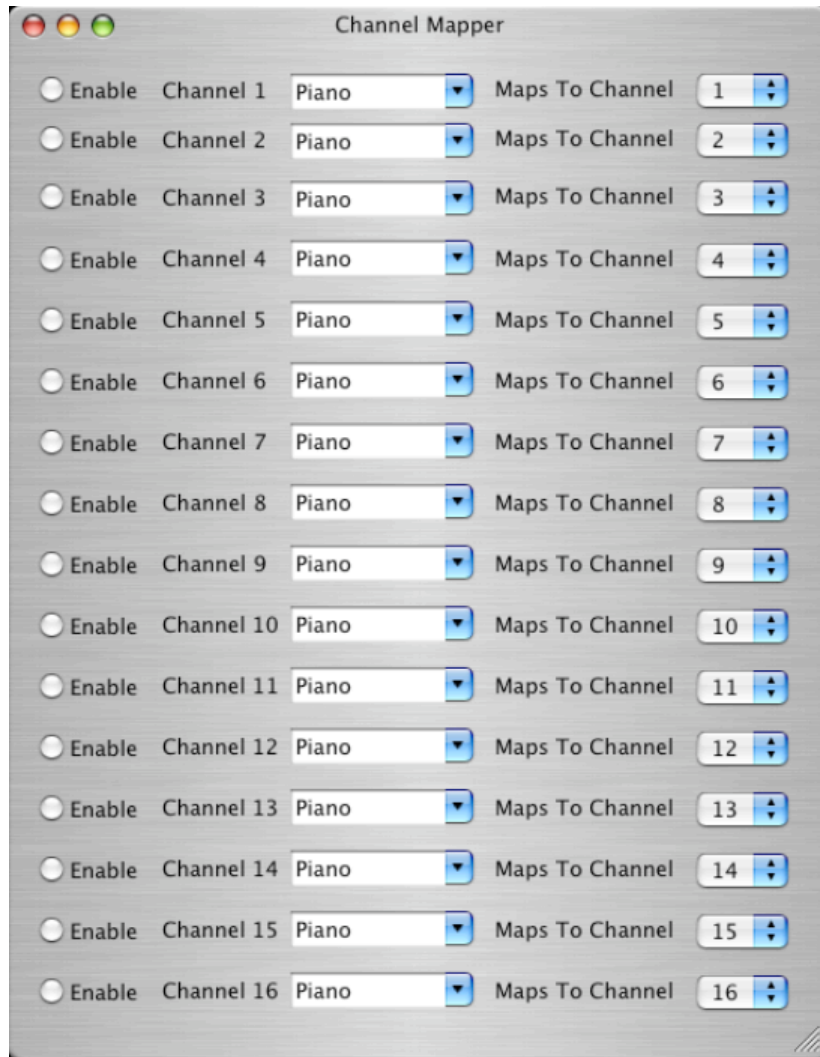


Figure 5-3 Channel Map Windows

Returning to the software, the Apple module has code for setting the text of an edit box, getting the text of an edit box, beginning and ending an open or save dialog, etc. The duet module has the definitions of many of structures used by the application and the ancillary window creation and handling functions. The Globals module has definitions and external references to all the global variables used by the application. The main module creates the main window and handles the main window's GUI elements events. The MIDI unit has the MIDI reading procedures for sequences and non-sequences. It also has the basic functions for setting up and initializing the MIDI handling procedures. The NavFunctions module has the navigation functions for the open and save navigation dialogs. The TCPNetwork program unit has the TCP server thread code and the client related TCP packet handing code.

The basic architecture of the MIDI duet application is a peer-to-peer design (P2P) that utilizes a client and server part in each of the two connected peers. This is illustrated in Figure 5-4.

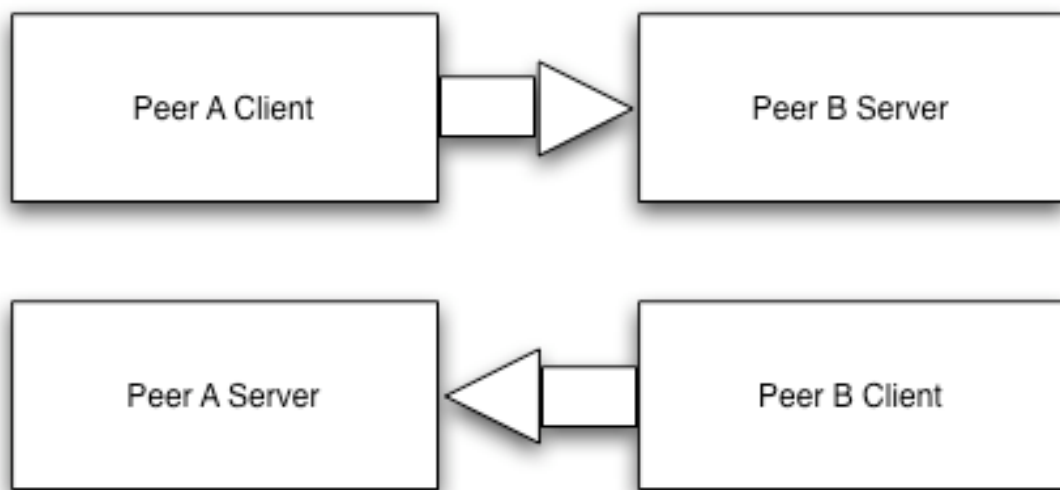


Figure 5-4 Peer-to-Peer Duet Architecture



## CHAPTER 6 CONCLUSIONS

### *A. Chapter 2 Conclusions*

The prior MOIP research was mainly an exploration of the existing protocol design space to find an effective and suitable candidate protocol. Early attempts were focused on UDP then TCP and later RTP. This research essentially followed the same line of protocol succession as the prior research. The previous studies made no attempt to quantitatively compare existing MOIP protocols. This neglect of an experimentally sound basis for choosing one MOIP protocol over another MOIP protocol seemed to be a glaring defect, and an area to be covered by this research.

Not all researchers before the current scientists came to the conclusion that a successful MOIP protocol must be, by the very nature of MIDI, a reliable protocol. A number of attempts were aimed at utilizing fundamentally unreliable protocols such as UDP and RTP. In these researchers' opinions such efforts using protocols that do not guarantee in order delivery of packets are doomed to failure.

### *B. Chapter 3 Conclusions*

The first attempt at creating a new and viable MOIP protocol by this research team was a miserable failure, however, we did learn a good lesson from this endeavor, namely, that UDP is not a suitable base protocol for the MOIP application without making extreme modifications to UDP. We also tried to utilize Young and Fujinaga's notion of adding

packet redundancy to UDP, but this fix was also unsuccessful by the experimental criteria that we were using.

A later protocol which went by the nickname ATCP which designated an Almost TCP like protocol seemed to fare somewhat better than our first flawed and unreliable protocol, but this protocol was later discovered to be fundamentally unusable for the musical duet application. In subsequent work this protocol disappeared from the mix of protocols being used experimentally.

The primordial efforts to create a musical duet system in Java were aborted due to the apparent language latency and the fact that at the time, the only way to perform MIDI input was to write native code. With the advent of Java 1.5 this glaring deficiency in the Java MIDI package was corrected. We then renewed our attempts to use Java as a basic MOIP language, but this time we were hampered by machines which were just too slow for the application. As we will see in the Chapter 5 conclusions this problem of relatively slow processors has been somewhat mitigated by currently available hardware.

The last proposed and implemented duet system introduced in this chapter represents a proof concept and is not intended to be a production system. Before an utilizable commercial product can be created more research into the issues of hardware, software, and network requirements is indicated. Also, work needs to be done on creating better user interfaces for the system with accompanying human user experimentation. Our research seems to indicate that Java involves a little too much inherent language latency to be used presently in this application area. Native or near native languages such as C seem to perform better as far as language latency is concerned. The system outlined above could be generalized to more performers than a musical duet. The extension to

trios, quartets, or ensembles is fairly straightforward. The musical duet performance system of this paper offers another way for musicians to collaborate in real-time, and unlike streaming audio the bandwidth requirements are not that great

#### *C. Chapter 4 Conclusions*

The overwhelming conclusion to come from this chapter was the fact that using the Java Media Framework (JMF), one could design and implement reliable RTP protocols using TCP as the transport protocol. The idea of using a reliable protocol at the RTP transport layer was not in itself novel, but in the MOIP area of research this notion had not been previously used. These RTP based protocols in many ways performed as well as the vanilla TCP protocols.

We devised two metrics for measuring the performance of the test suite of ten MOIP protocols which were: the runtime of a MIDI sequence on the destination host, and a metric which tended to correspond to amount of jitter in a protocol. We then used a statistically larger number of experiments to determine the best protocol in the test suite based on the previously mentioned empirical measurements. It was found that on a pair of heterogeneous Windows platforms that the producer and consumer multithreaded TCP protocol was the most efficient MOIP protocol.

#### *D. Chapter 5 Conclusions*

The primary result to be drawn from this chapter is that for consistently transmitting MIDI data over a network, a reliable transport layer protocol should be used. As was stated multiple times in this paper, MIDI is very sensitive to lost or out-of-order data, unlike audio or video transmissions which can afford to lose some data. We found that utilizing UDP for transport was, in general cases, a bad idea due to stuck notes. A simple

argument shows that the dishonesty delta-time policy is preferable to the honesty delta-time policy for  $m = 1$  in performing a musical duet. However, in the general  $m$  cases and for the transmission of MIDI sequences over a network, the honesty delta-time policy should probably be used. We will investigate the efficacy of the honesty delta-time policy in future research.

We implemented the musical duet performance system of this chapter on two different operating systems, namely, Windows and OS X using the C language. The system appeared to have a lower latency on the OS X system; however, this could be due to the fact that our Windows machines did not have the fastest x86 processors currently available.

#### *E. Overall Conclusions*

The overall conclusions to be drawn from this research is that the creation of high performance MIDI over IP protocols is a very difficult problem, and the design and implementation of a viable musical duet system using a MIDI over IP protocol is an extremely challenging software engineering task. As machines and the Internet infrastructure improve in terms of speed and bandwidth then the latencies mentioned earlier that are associated with MIDI over IP may disappear altogether. In this section we will address the overall conclusions that were derived from both of the endeavors cited immediately above in this paragraph.

New networking protocols are by the nature of the problem hard to develop and properly implement. Typically, finite state machines for both the sender and receiver are designed and implemented in some real computer language or in a simulator-type script. An alternative design strategy is the use of a Petri net. The networking protocol must be

free of deadlocks, live-locks, and improper terminations [48]. A usable musical duet system poses several problems such as having a low overall latency and proper synchronization. The latency issues were addressed as best as possible with the available software and hardware by careful design choices. Duet synchronization was performed using a simple metronome count up subsystem.

## REFERENCES

- [1] M. Boom, *Music Through MIDI*, Microsoft Press, Redmond, Washington, 1987.
- [2] J. Rothstein, *MIDI: A Comprehensive Introduction*, second edition, A-R Editions, Madison, Wisconsin, 1995.
- [3] P. D. Lehrman and T. Tully, *Midi for the Professional*, Amsco Publications, New York, New York, 1993..
- [4] J. Rona, *The MIDI Companion*, Hal Leonard Corporation, Milwaukee. Wisconsin, 1994.
- [5] G. Hansper, [http://crystal.apana.org.au/~ghansper/midi\\_introduction/midi\\_control\\_change.html](http://crystal.apana.org.au/~ghansper/midi_introduction/midi_control_change.html), 1998.
- [6] J. Glatt, <http://www.borg.com/~jglatt/>, 2003.
- [7] B. McQueer, <http://www.harmony-central.com/MIDI/Doc/primer.txt>, 1995.
- [8] MIDI Manufacturers Association. <http://www.midi.org/>, 2003.
- [9] Sun, *Java Sound Programming Guide*. [http://java.sun.com/j2se/1.4/docs/guide/sound/programmer\\_guide/contents.html](http://java.sun.com/j2se/1.4/docs/guide/sound/programmer_guide/contents.html), 2002.
- [10] Gibson Musical Instruments, *MaGIC v 2.8 Engineering Specification*, <http://www.gibsonmagic.com/magic28.pdf>, 2002.
- [11] A. S. Tanenbaum, *Computer Networks*, third edition, Prentice-Hall, Saddle River, New Jersey, 1996.
- [12] G. Hoffman and D. Moore, <http://www.skipstone.com/compeon.html>.
- [13] D. Moore and Skipstone, <http://www.skipstone.com/ss21st.html>.
- [14] G. Hoffman, <http://www.skipstone.com/newspap.html>.
- [15] Association of Musical Electronics Industry and the MIDI Manufacturers Association. [http://www.midi.org/about-midi/rp27v10spec\(1394\).pdf](http://www.midi.org/about-midi/rp27v10spec(1394).pdf).
- [16] Gibson Musical Instruments, <http://www.gibsonmagic.com/video.html>, 2002.
- [17] R. Merritt, *EE Times*. <http://www.eetimes.com/sys/news/OEG20030124S0035>.
- [18] J. B. Postel, RFC 768, <http://www.freesoft.org/CIE/RFC/768/>, 1980.
- [19] J. B. Postel, RFC 793, <http://www.freesoft.org/CIE/RFC/793/>, 1981.
- [20] D. E. Comer, *Internetworking with TCP/IP Volume I Principles, Protocols, and Architecture*, third edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1995.
- [21] B. Quinn and D. Shute, *Windows Sockets Network Programming*, Addison-Wesley, Reading, Massachusetts, 1996.
- [22] R. Stevens, *Unix Network Programming Volume 1 Networking APIs: Sockets and XTI*, second edition, Prentice-Hall, Upper Saddle River, New Jersey, 1998.
- [23] H. Schulzrinne, S. Casner., R. Frederick, and V. Jacobson, <http://www.ietf.org/rfc/rfc1889.txt>, 1996.
- [24] Dennis M. Ritchie, <http://cm.bell-labs.com/cm/cs/who/dmr/chist.html>.
- [25] Leslie B. Wilson, and Robert G. Clark, *Comparative Programming Languages*, Addison-Wesley, Wokingham, England, 1988.
- [26] John Byous, <http://java.sun.com/features/1998/05/birthday.html>, 1998.

- [27] Cay S. Horstmann, and Gary Cornell, *Core Java Volume I – Fundamentals*, Prentice-Hall, Upper Saddle River, New Jersey, 1999.
- [28] Harold, Abelson, Gerald Jay Sussman,, and Julie Sussman, *Structure and Interpretation of Computer Languages, Second Edition*, The MIT Press, Cambridge, MA, 1996.
- [29] Petzold, Charle, *Programming Windows with C#*, Microsoft Press, Redmond, Washington, 2002.
- [30] Richter, Jeffrey, *Applied Microsoft .Net Framework Programming*, Microsoft Press, Redmond, Washington, 2002.
- [31] John W. Muchow, *Core J2ME Technology and MIDP*, Prentice-Hall, Upper Saddle River, New Jersey, 2002.
- [32] Vartan Piroumian. *Wireless J2ME Platform Programming*, Prentice-Hall, Upper Saddle River, New Jersey, 2002.
- [33] W. Keith. Edwards, *Core JINI*, Prentice-Hall, Upper Saddle River, New Jersey, 1999.
- [34] Scott, Oaks and Henry Wong, *JINI in a Nutshell*, O’Reilly, Beijing, P.R.O.C., 2000.
- [35] Apple Computer, Inc. *Audio and MIDI on Mac OS X*.
- [36] M. Goto, R. Neyama, and Y. Muroka, RCMP: Remote Music Control Protocol, *Proceedings of the 1997 International Computer Music Conference*, ICMA, pp. 446 – 449, [www.etl.go.jp/~goto/PAPER/ICMC97.300dpi.ps](http://www.etl.go.jp/~goto/PAPER/ICMC97.300dpi.ps), 1997.
- [37] J. P. Young and I. Fujinaga, “Piano Master Classes via the Internet”, *Proceedings of the 1999 International Computer Music Conference*, ICMA, pp. 135 – 7, <http://www.peabody.jhu.edu/~ich/research/icmc99/icmc99.UDP.pdf>, 1999.
- [38] R. B. Dannenberg, and P. van de Lageweg, “A System Supporting Flexible Distributed Real-Time Music Processing”, *Proceedings of the 2001 International Computer Music Conference*, ICMA, pp. 267 – 270, <http://www-2.cs.cmu.edu/~rbd/papers/icmc01aura.pdf>, 2001.
- [39] J. Lazzaro and J. Wawrzynek, “A Case for Musical Network Performance”, *NOSSDAV’01*, ACM, 2001.
- [40] H. Schulzrinne , <http://www.faqs.org/rfcs/rfc1890.html>, 1996.
- [41] Souspe <http://hyperphysics.phyastr.gsu.edu/hbase/sound/souspe.html>.
- [42] Sound, *Sound on Sound Magazine* <http://www.sospubs.co.uk/sos/apr99/articles/letency.htm>, 1999.
- [43] N. Cardwell, S. Savage, and T. Anderson, “Modeling TCP Latency”, *Infocom*, pp. 1742 – 1751, <http://citeseer.nj.nec.com/cardwell00modeling.html>, 2000.
- [44] E. Brandt, and R. Dannenberg, Low-latency Music Software Using Off-the-shelf Operating Systems, *Proceedings of the 1998 International Computer Music Conference*, ICMA, <http://citeseer.nj.nec.com/309841.html>, 1998.
- [45] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*, second edition, Prentice Hall Europe, London, 1998.
- [46] X. Yin, J. Xue, and P. Stuedi, [http://www2.inf.ethz.ch/~stuedip/doc/SA\\_sctp.pdf](http://www2.inf.ethz.ch/~stuedip/doc/SA_sctp.pdf)
- [47] S. Maghsoodloo, private communication, 2/13/2004 11:03 AM, [maghsood@eng.auburn.edu](mailto:maghsood@eng.auburn.edu).
- [48] Holzmann, G. J., *Design and Validation of Computer Protocols*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

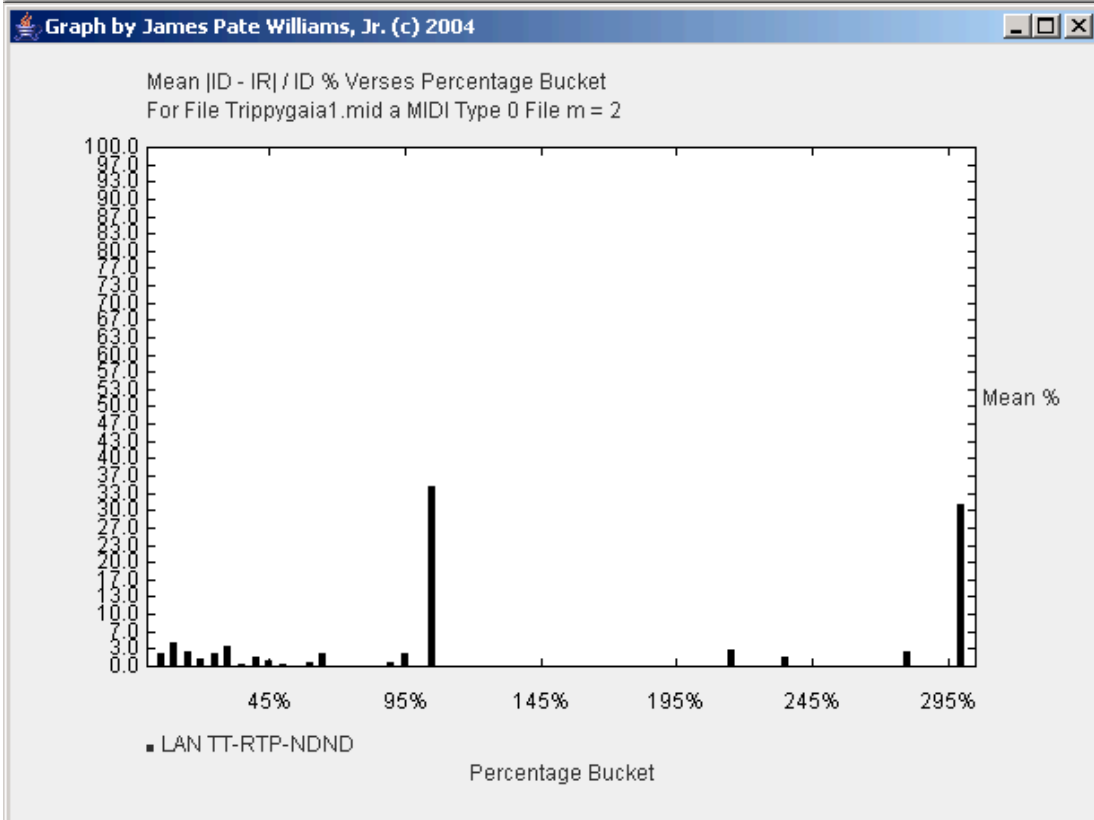
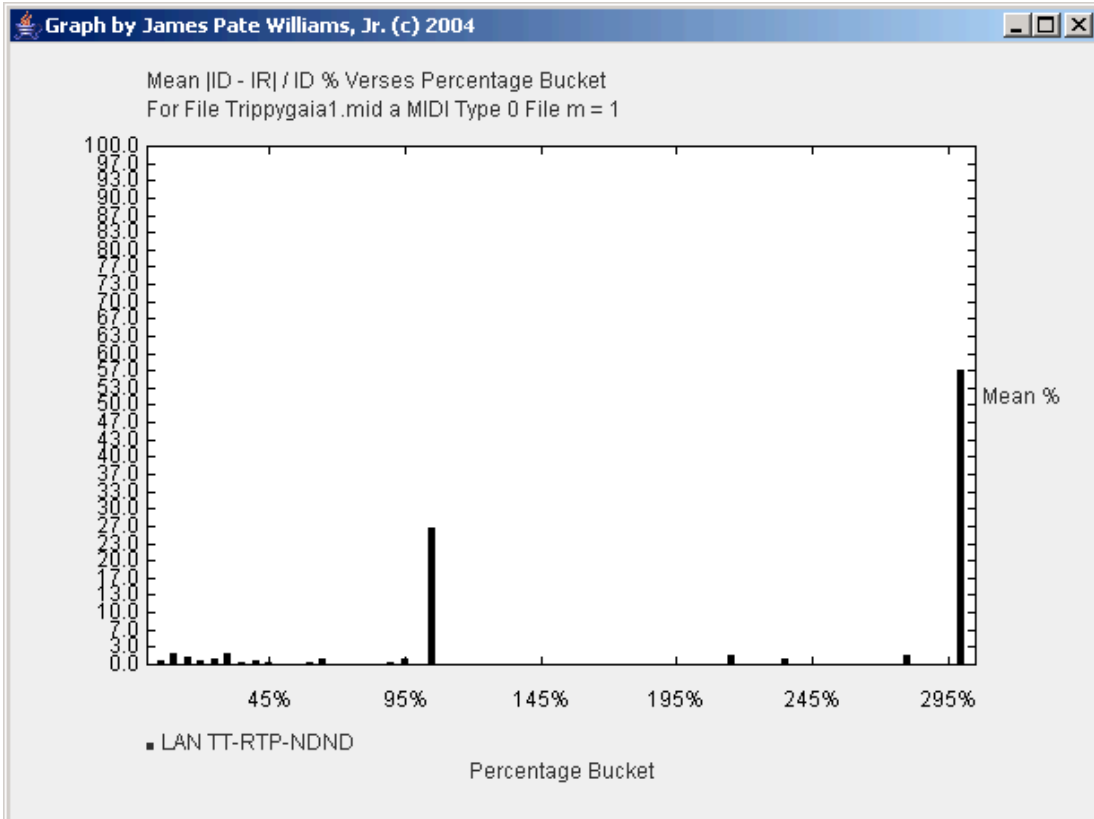
## INDEX

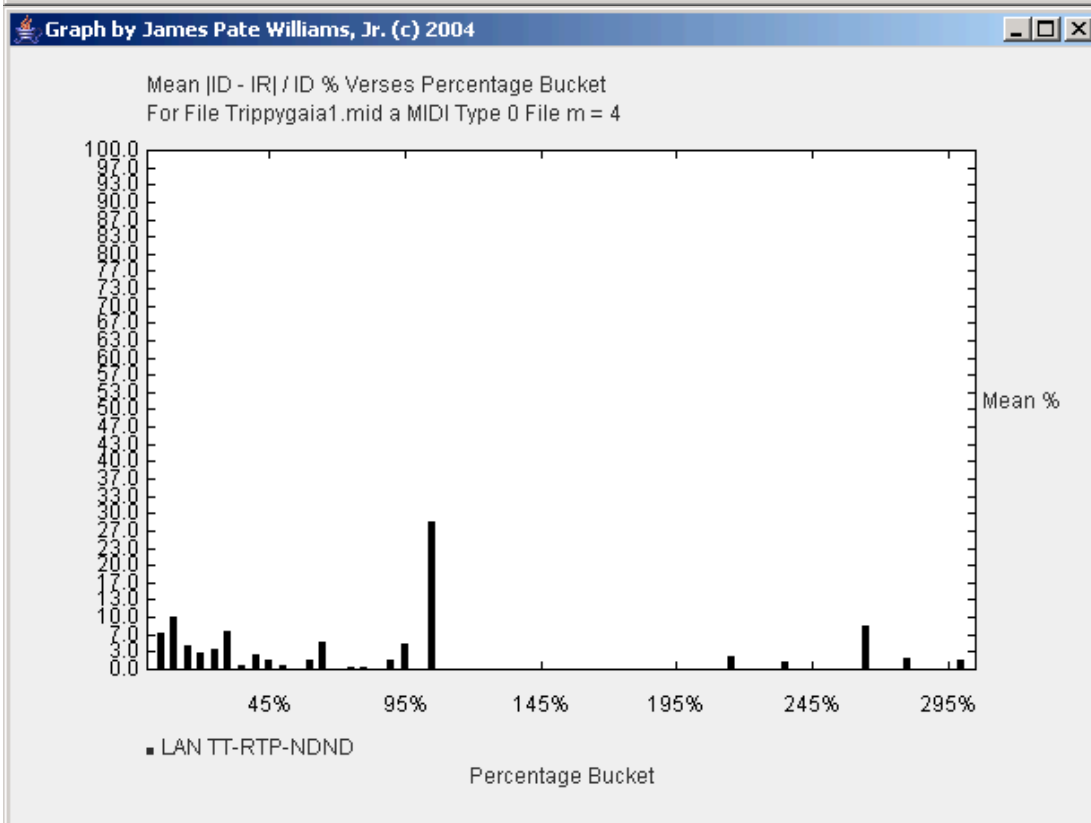
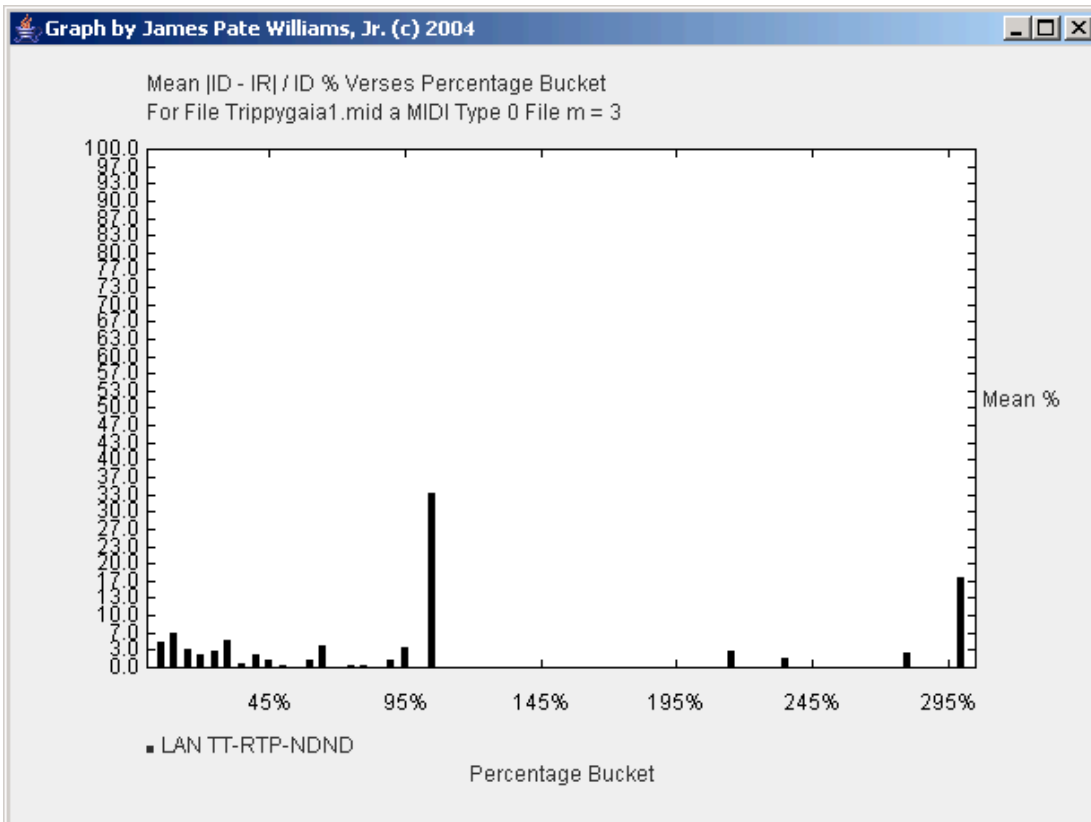
- Appendix, 6, 8, 44, 53, 54, 55, 56, 75, 77, 79
- C, v, 1, 3, 5, 8, 13, 16, 17, 19, 25, 28, 34, 37, 39, 48, 49, 50, 52, 53, 54, 55, 56, 57, 58, 63, 66, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 94, 96, 99
- C++, 2, 19, 34, 63
- collaboration, v, 1, 2, 3, 19, 63, 83
- duet, v, 1, 3, 4, 22, 23, 25, 30, 62, 63, 66, 83, 84, 86, 88, 90, 92, 94, 96, 97
- INTERNET, 1, 2, iii, v, 1, 13, 14, 20, 21, 22, 23, 25, 26, 27, 36, 38, 41, 52, 73, 87, 90, 96, 99
- Java, v, 1, 3, 5, 8, 16, 17, 18, 25, 28, 30, 37, 39, 50, 51, 55, 58, 59, 62, 63, 66, 70, 72, 83, 94, 95, 98, 99
- LAN, 1, 9, 21, 67, 68, 73, 77, 79, 81, 83
- MIDI, 1, 2, iii, v, 1, 2, 3, 5, 6, 7, 8, 10, 11, 15, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 30, 31, 32, 33, 34, 35, 36, 37, 39, 40, 41, 42, 48, 49, 51, 52, 53, 54, 55, 56, 59, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 81, 84, 86, 87, 88, 89, 92, 93, 94, 95, 96, 98, 99
- Nagle algorithm, 3, 24, 39, 40, 54, 71, 89
- OS X, 2, 19, 64, 65, 66, 84, 86, 96, 99
- protocols, v, 1, 2, 3, 13, 18, 21, 22, 24, 25, 30, 31, 32, 34, 35, 37, 38, 39, 45, 47, 51, 52, 53, 56, 63, 70, 71, 72, 77, 78, 79, 93, 94, 95, 96
- research, v, 1, 2, 3, 5, 24, 25, 26, 30, 58, 93, 94, 95, 96, 99
- TCP/IP, 1, 13, 14, 21, 66, 98
- UNIX, 2, 14, 16, 66
- WAN, 1, 21, 30, 36, 73, 75, 79
- Windows 98, 2, 27, 35, 64, 65, 67, 73
- Windows XP, 2, 35, 64, 65, 67, 73

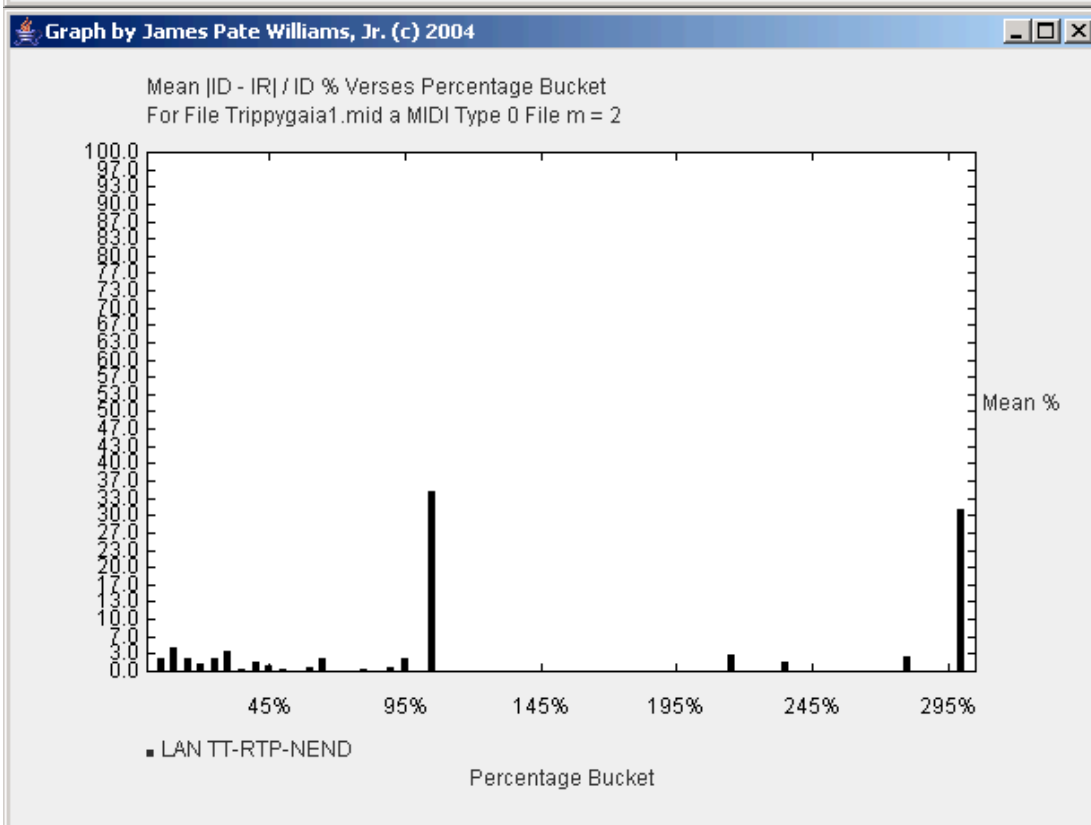
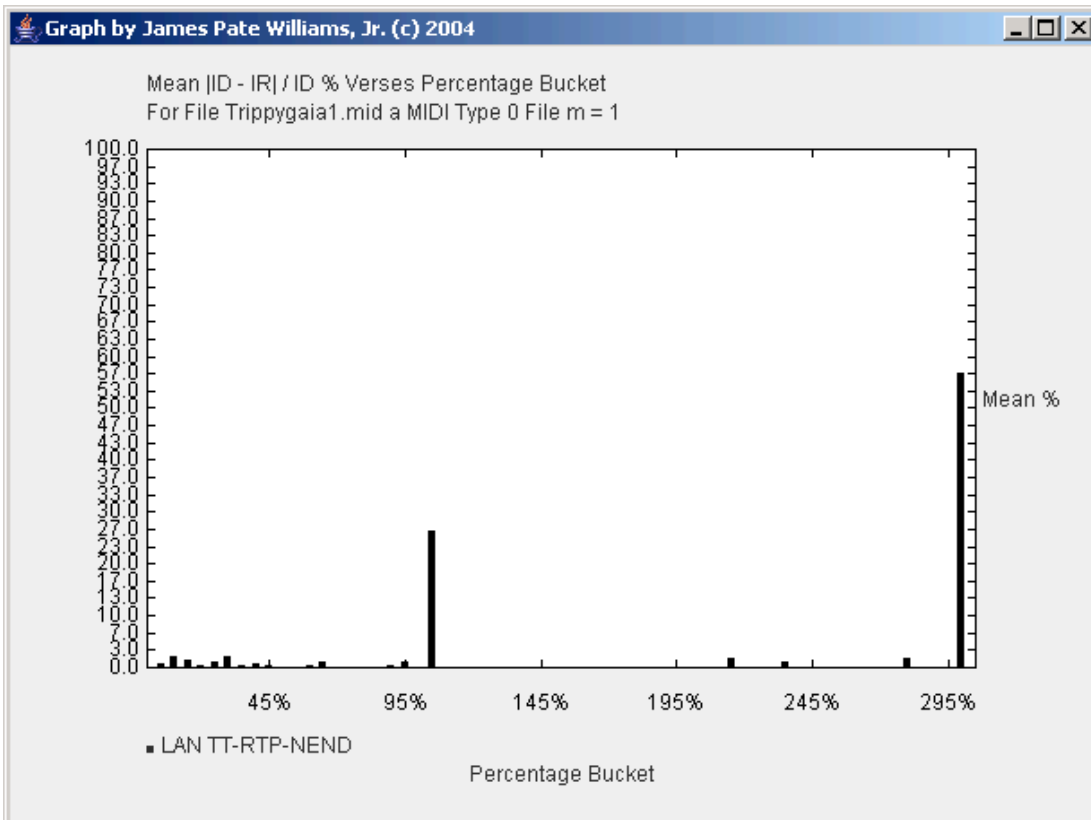


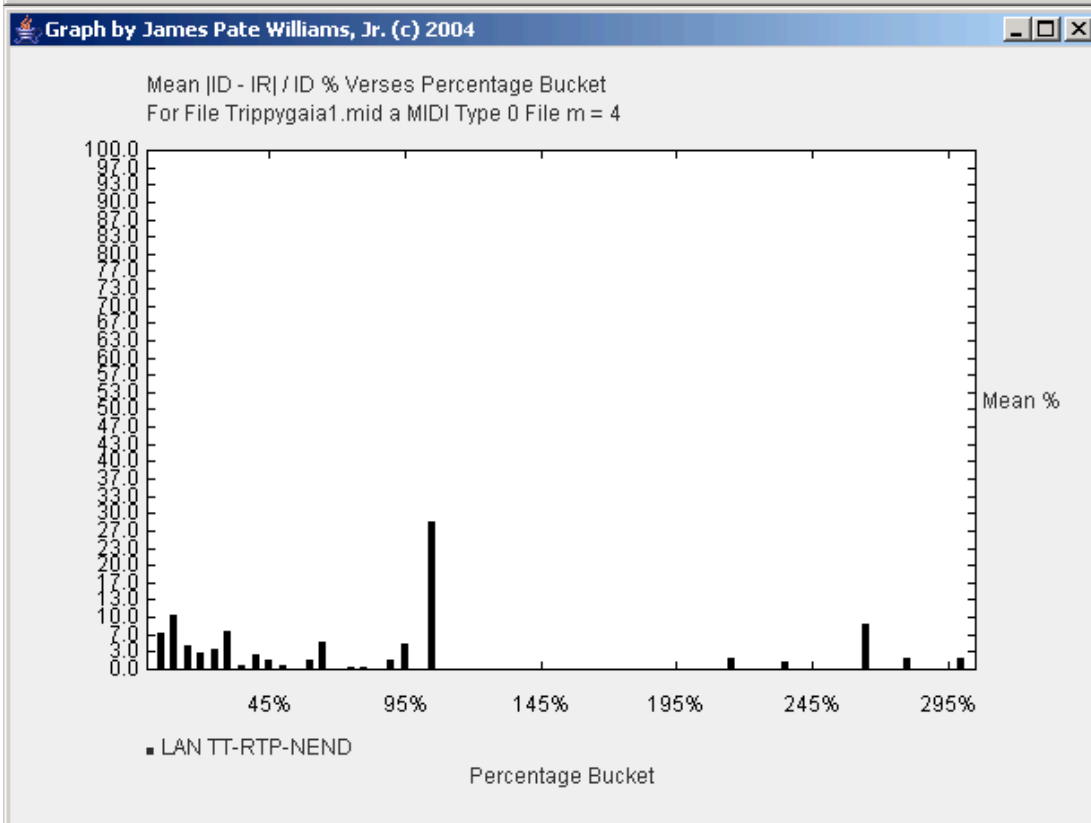
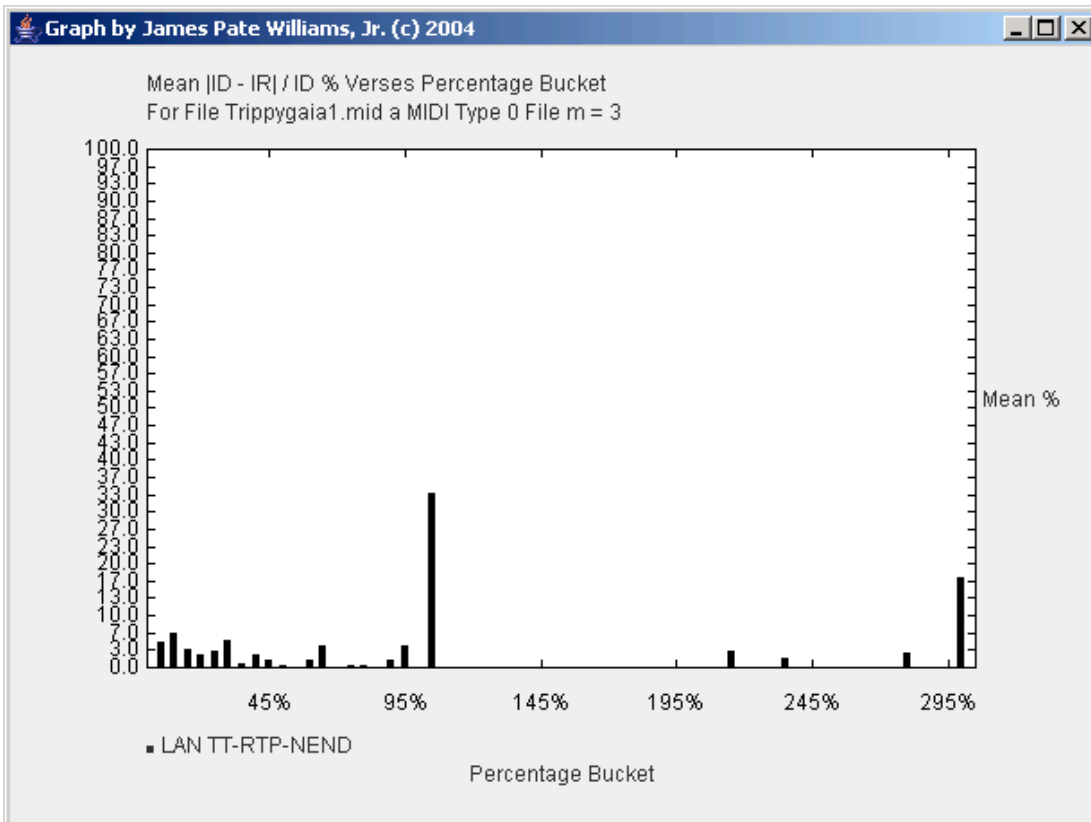
## APPENDIX A LAN EQUATION (1) CHAPTER 4 GRAPHS

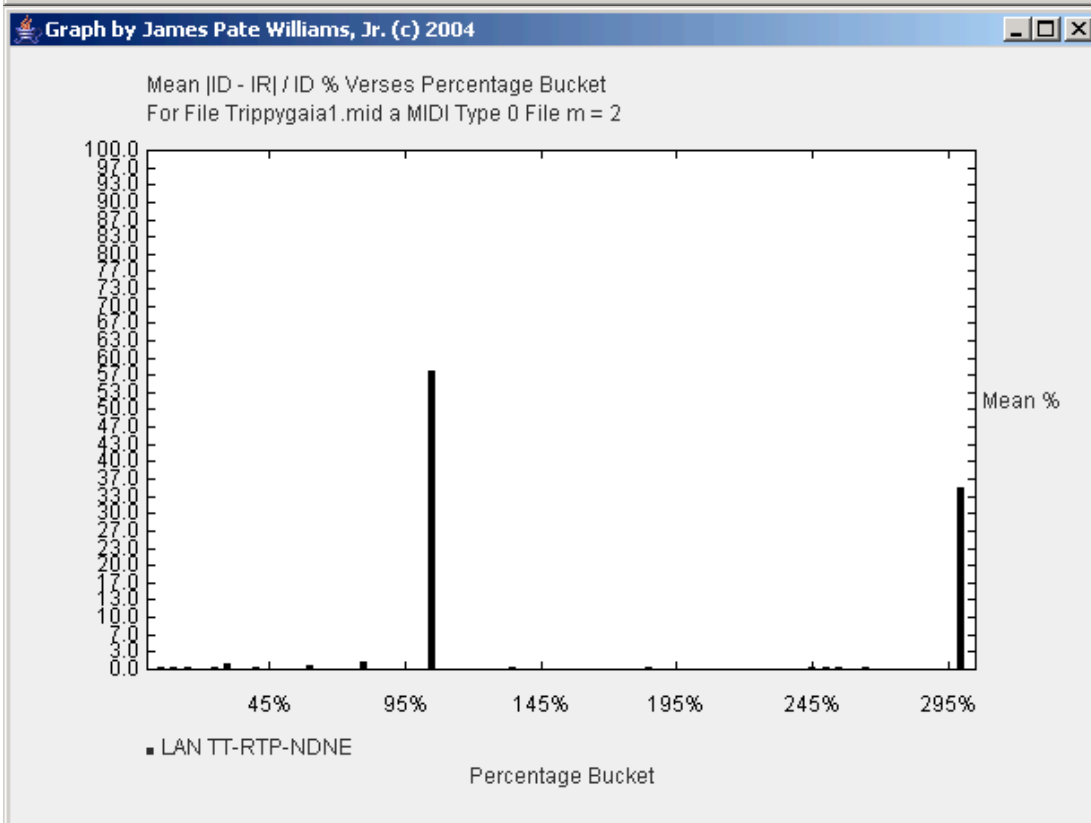
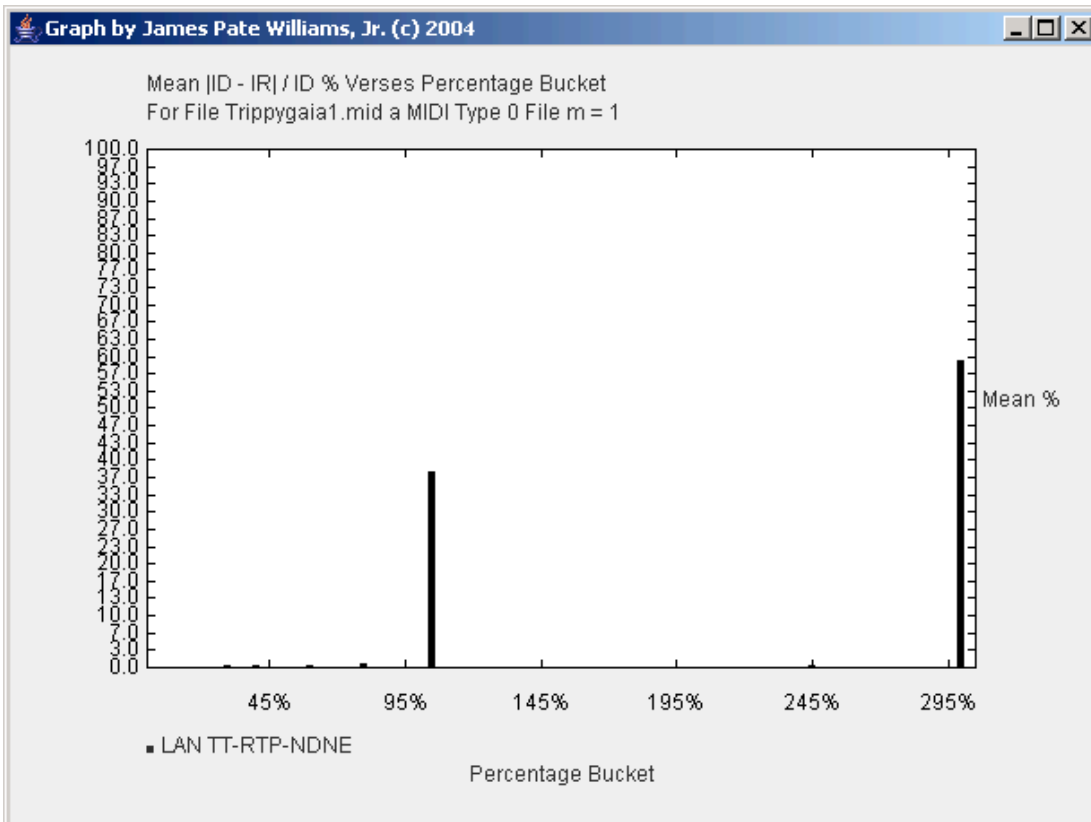
There are forty graphs in this appendix for a local area network (LAN) that were generated from Equation (1) in Chapter 4. There are four graphs each for all ten protocols in the suite of MIDI over IP protocols. To reiterate the protocols are as follows: TT-RTP-NDND, TT-RTP-NEND, TT-RTP-NDNE, TT-RTP-NENE, UT-RTP-ND, UT-RTP-NE, PC-RTP-ND, PC-TCP-NE, SN-TCP-ND, and SN-TCP-NE, where ND = Nagle algorithm disabled and NE = Nagle algorithm enabled (typically the default Internet setting).

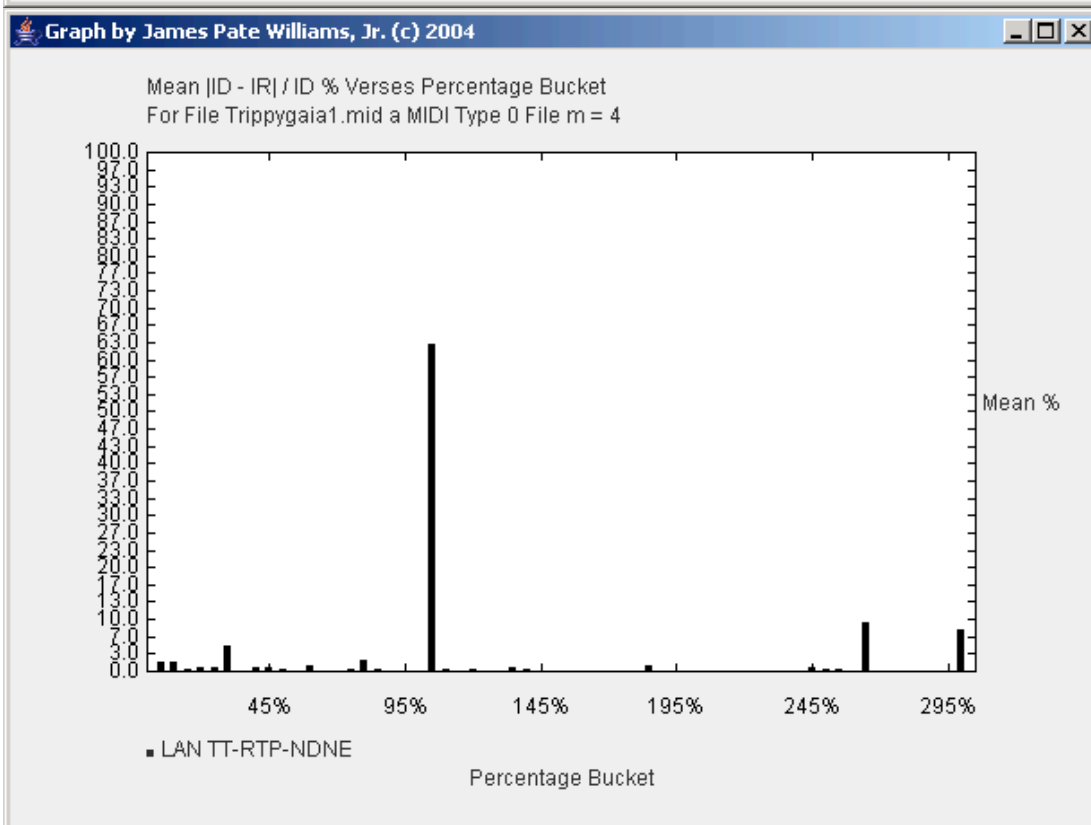
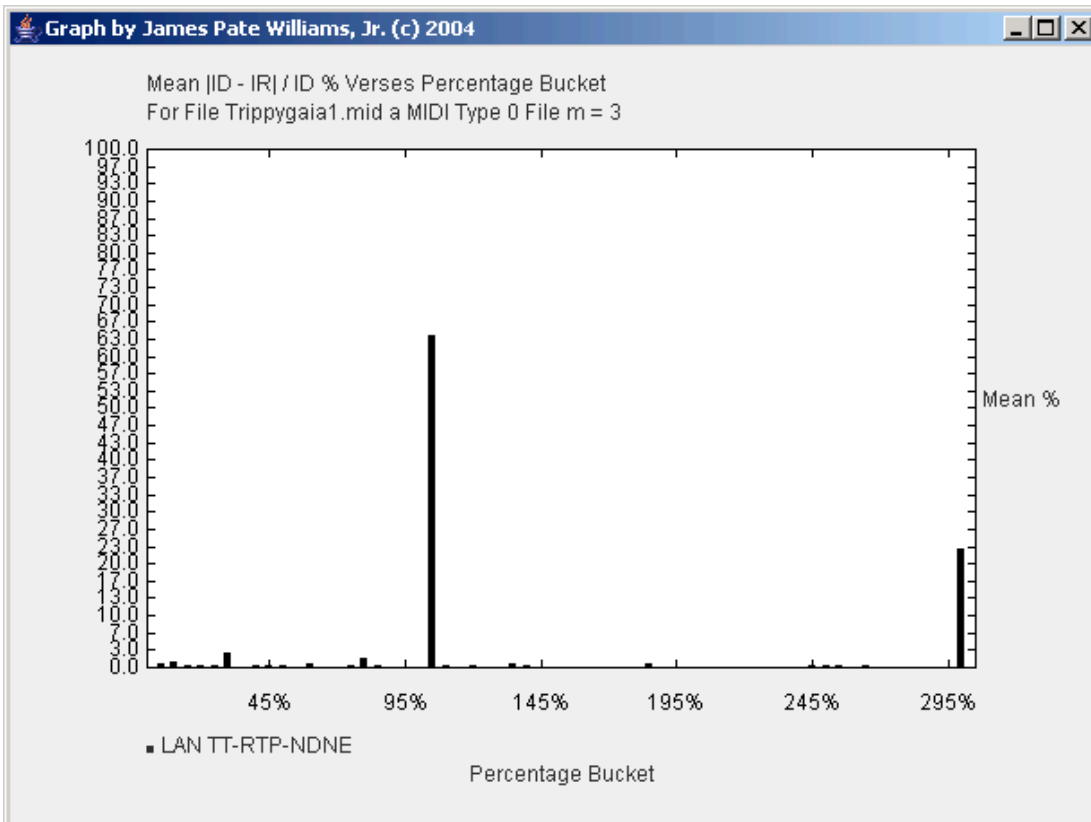


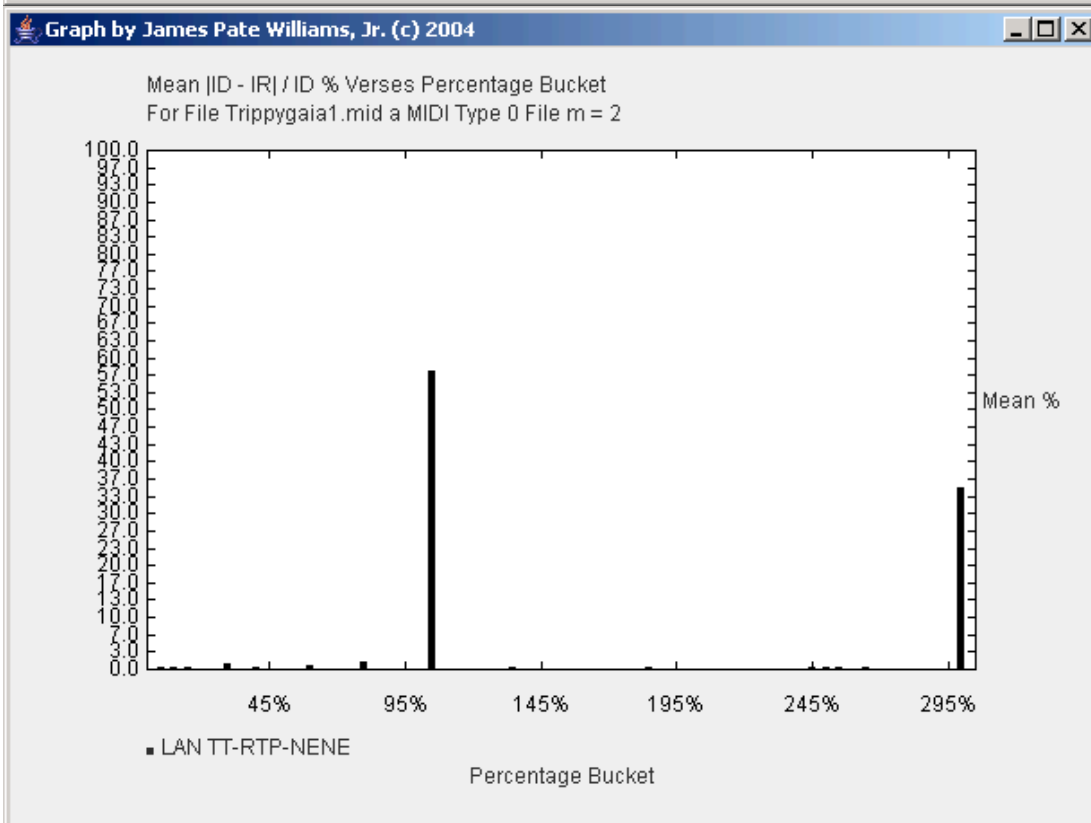
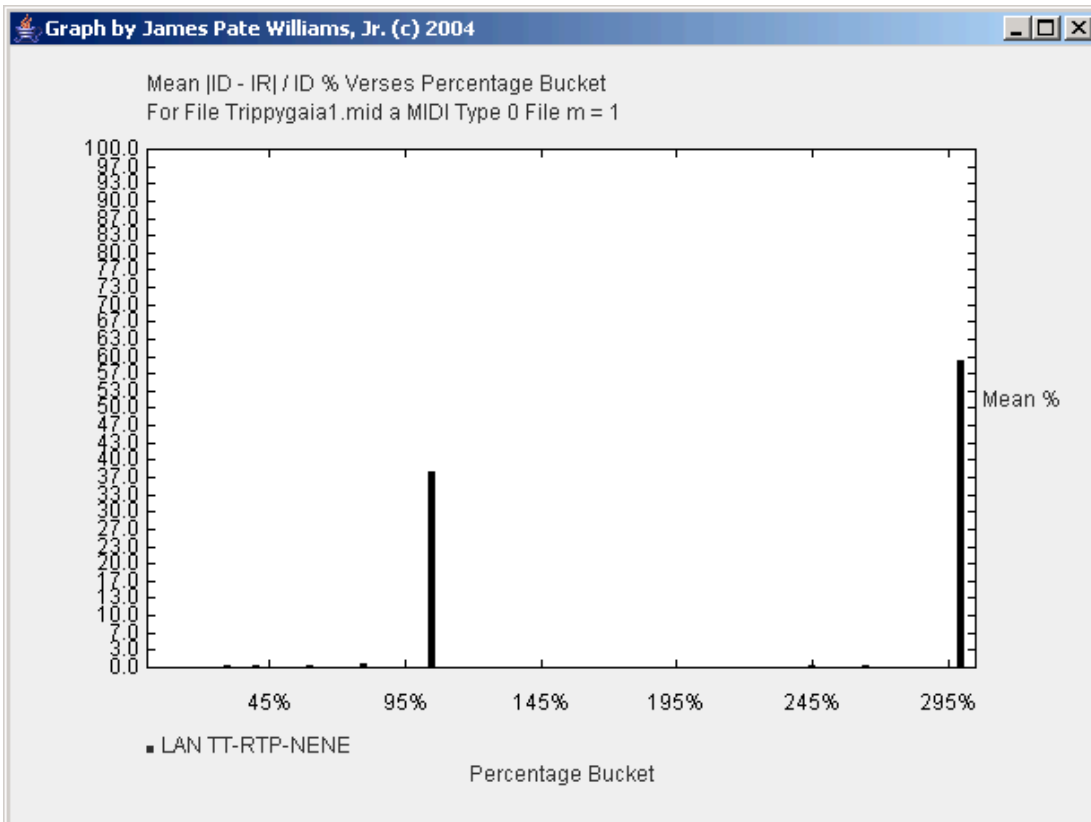




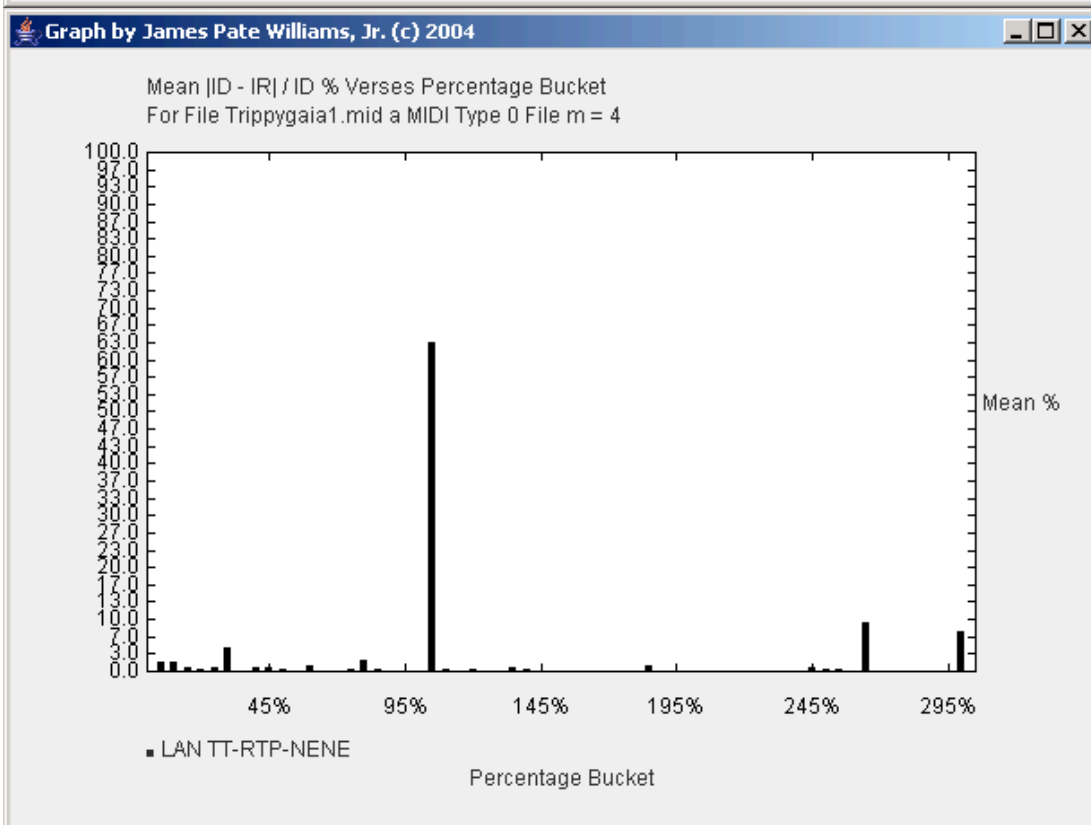
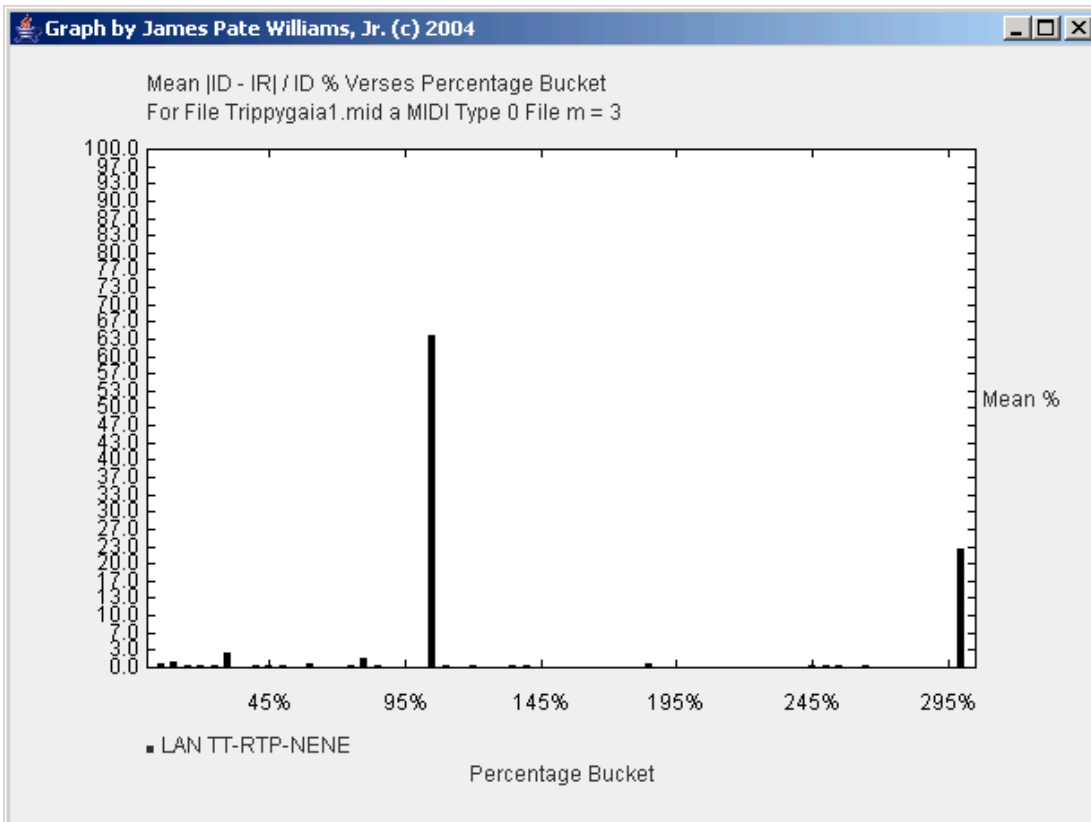


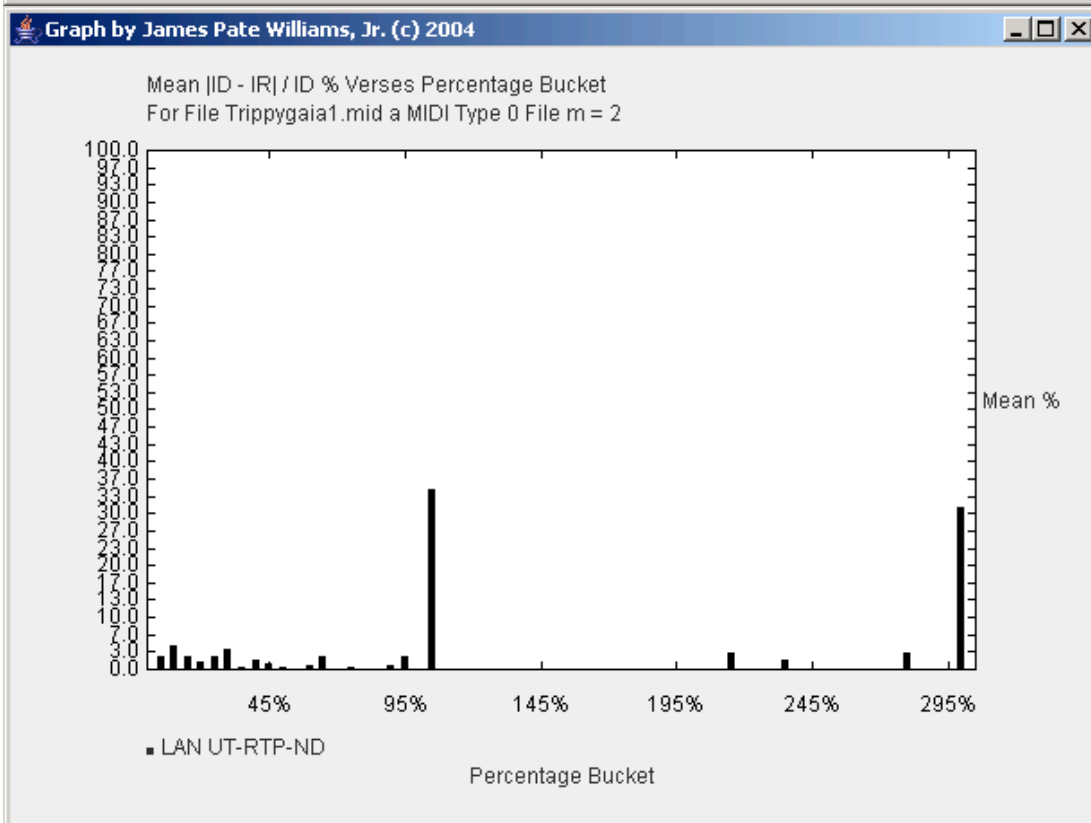
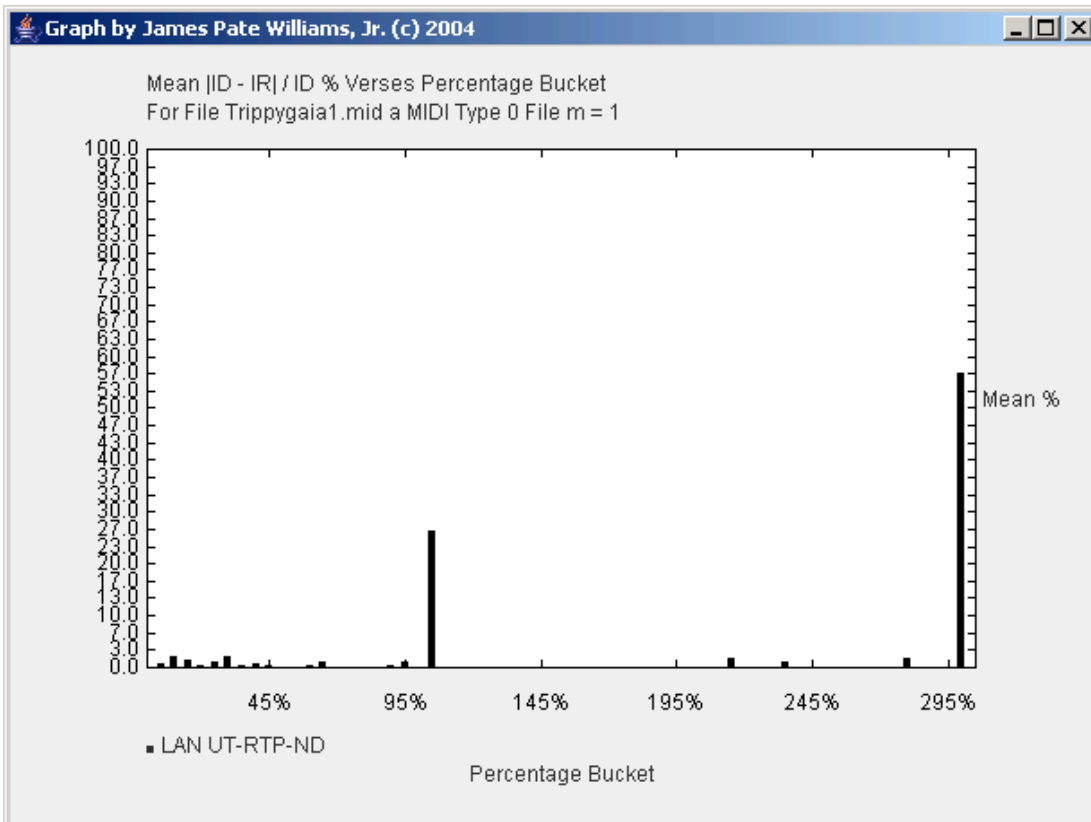


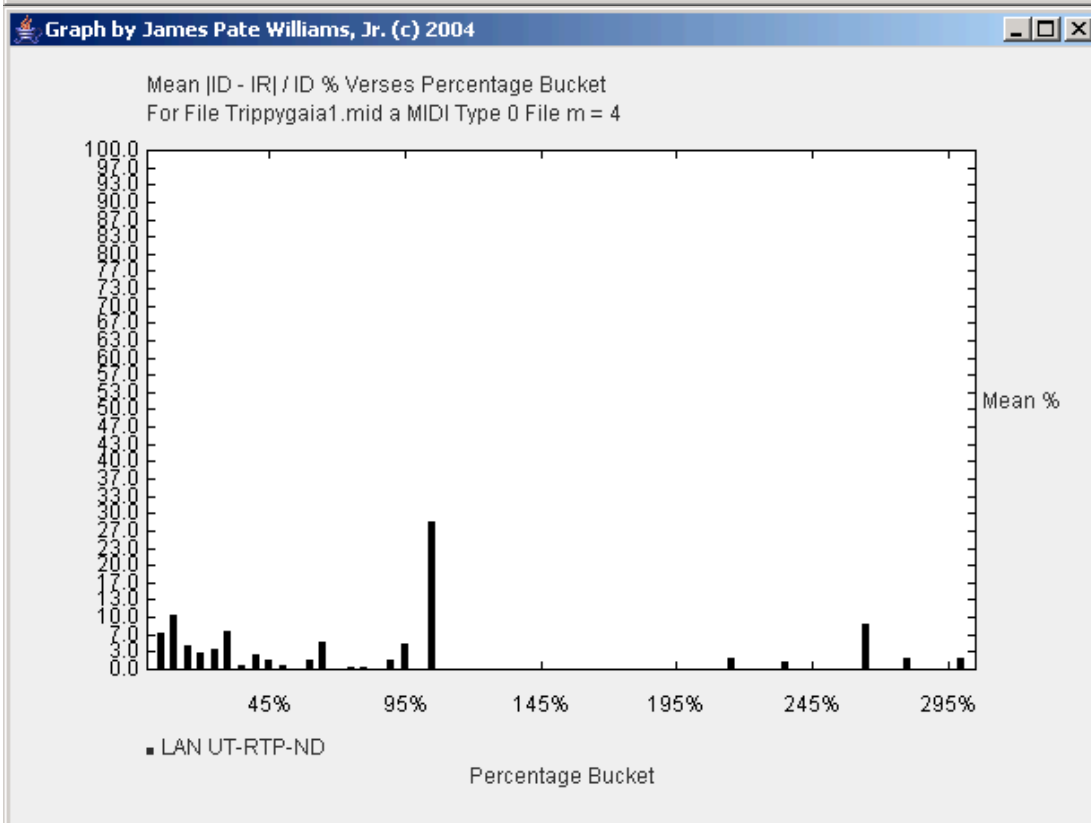
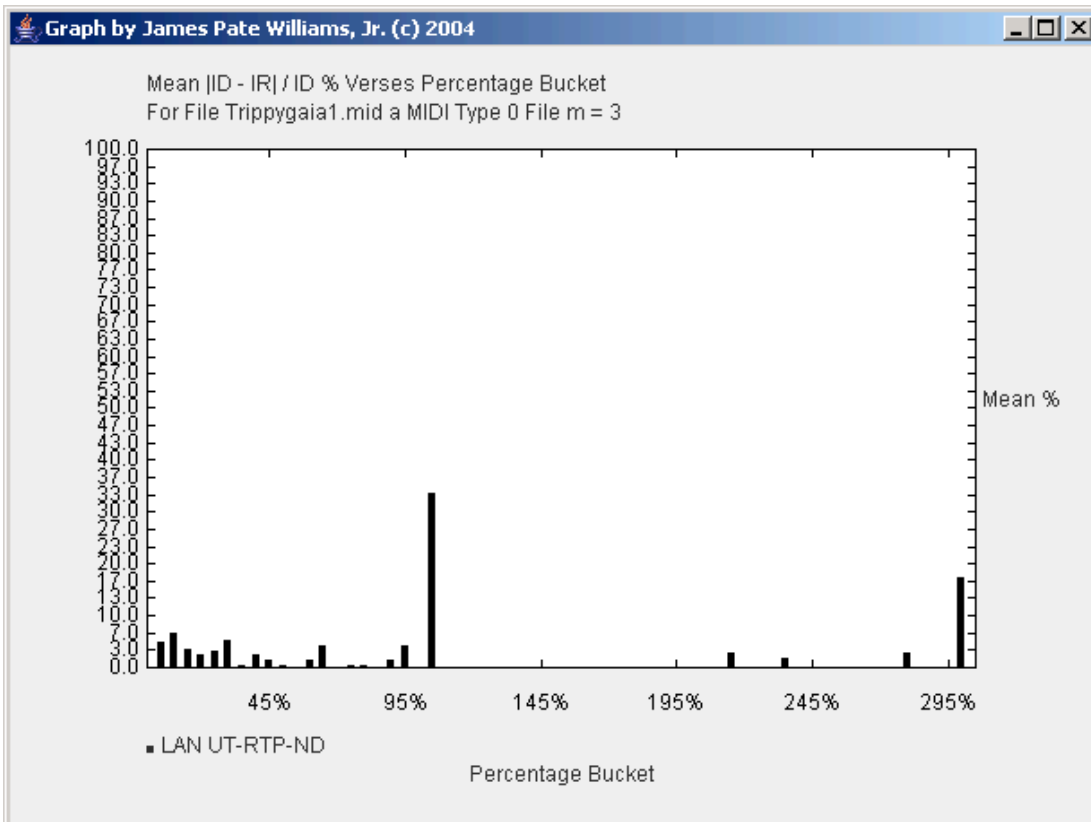


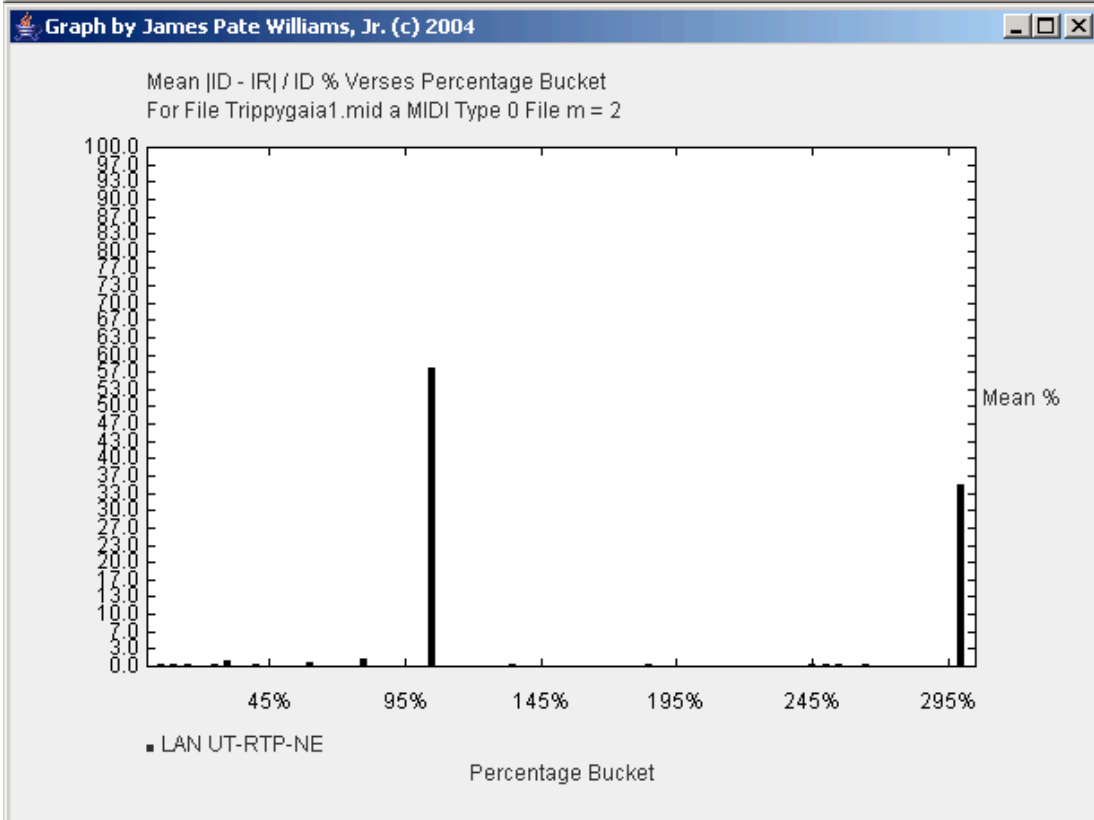
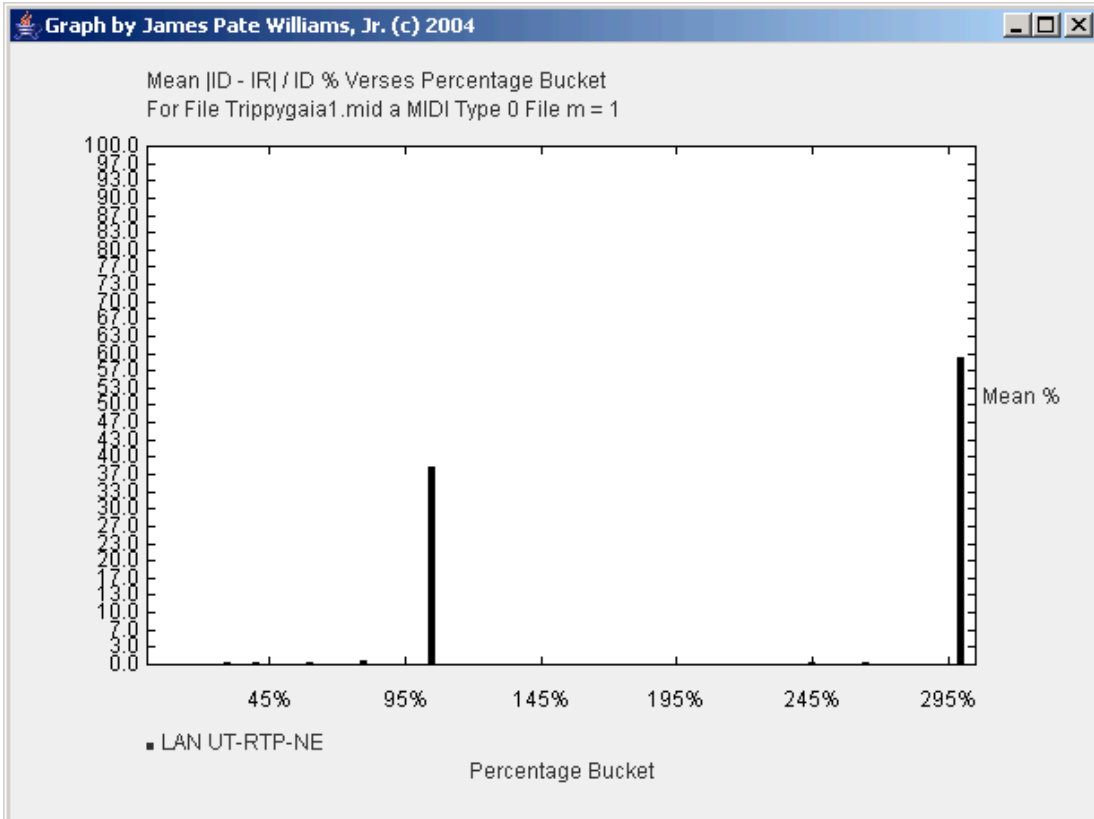


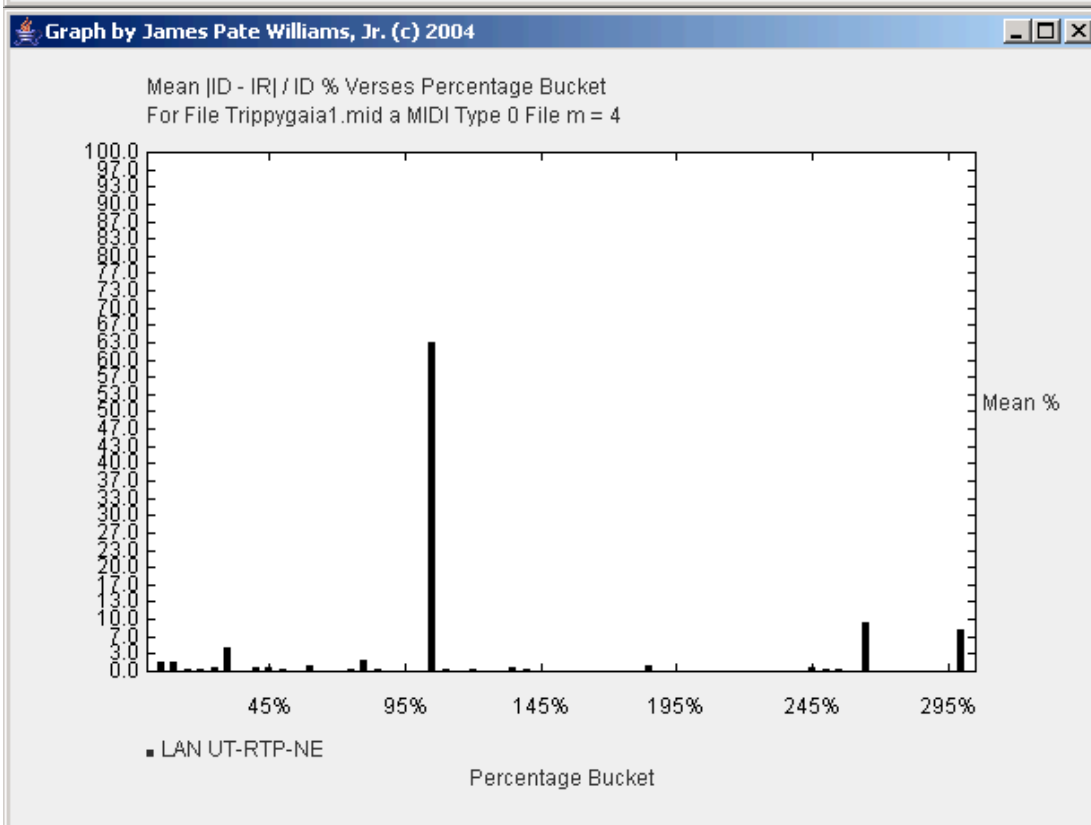
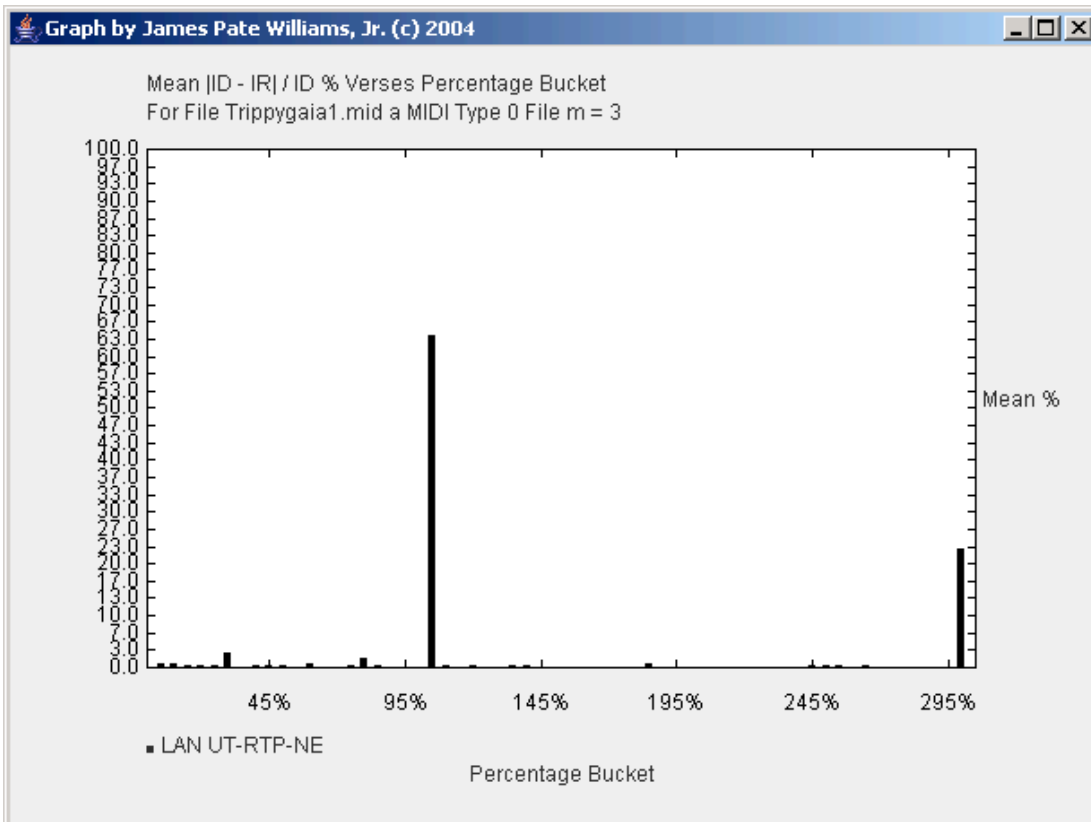


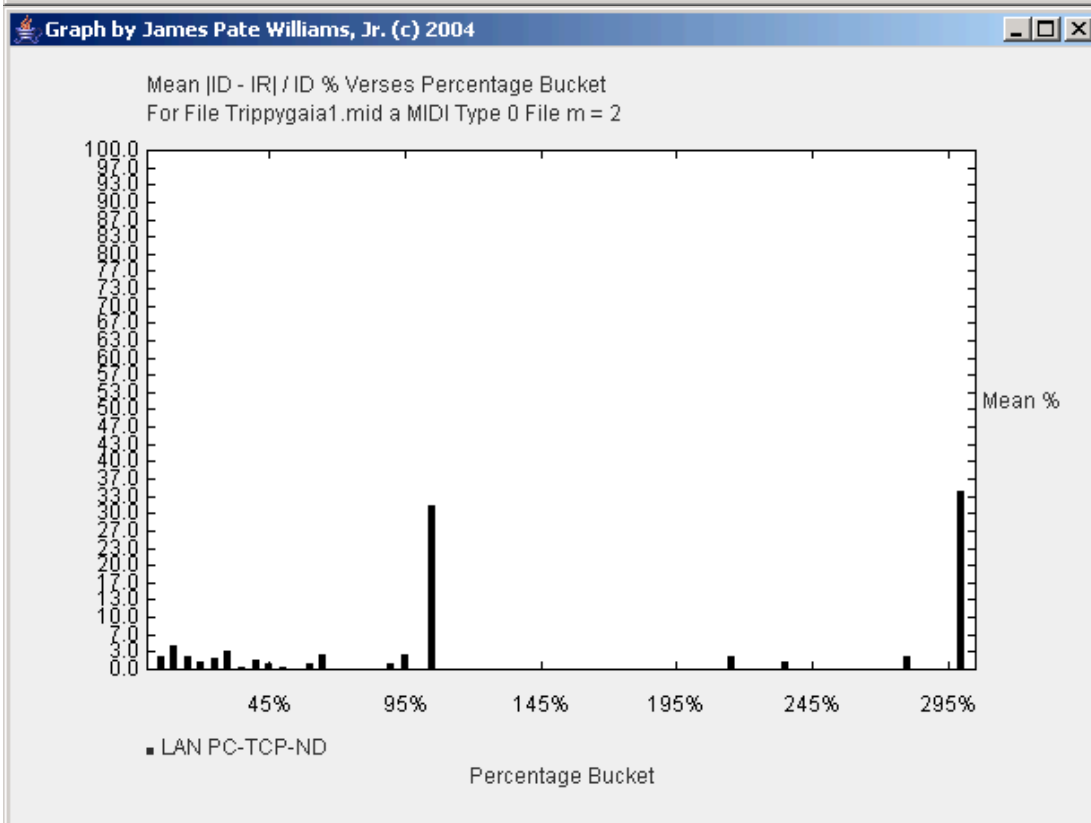
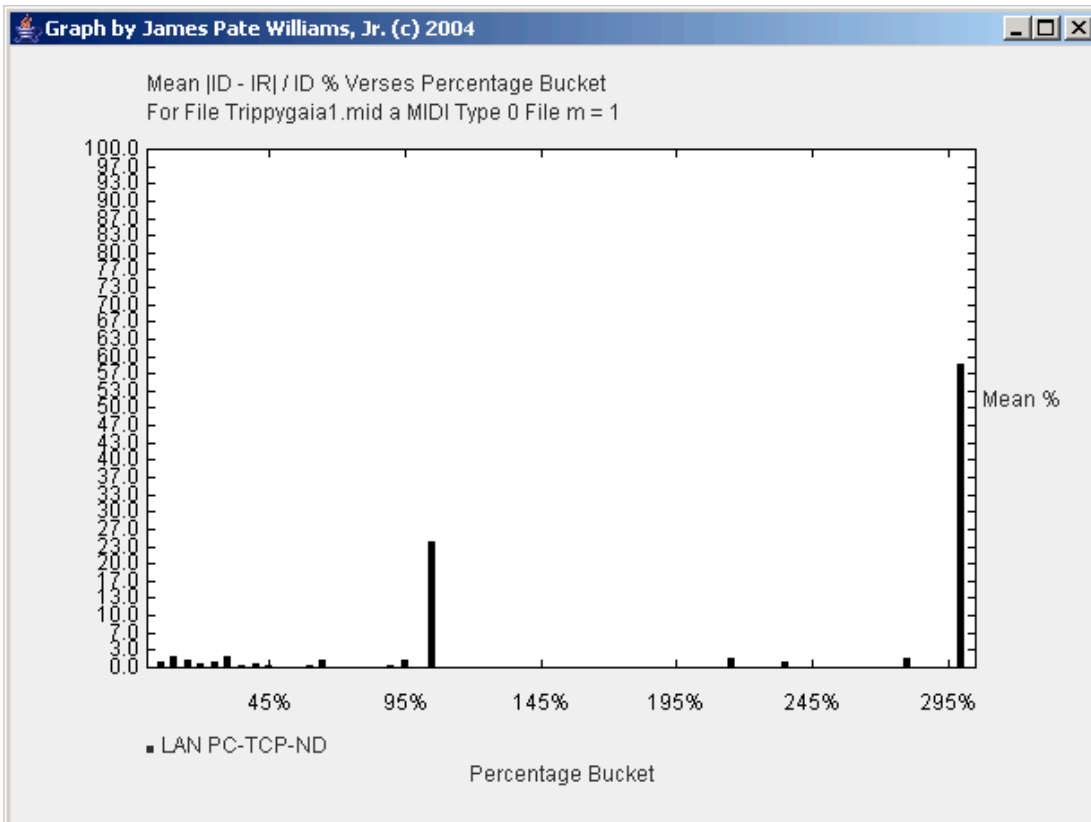


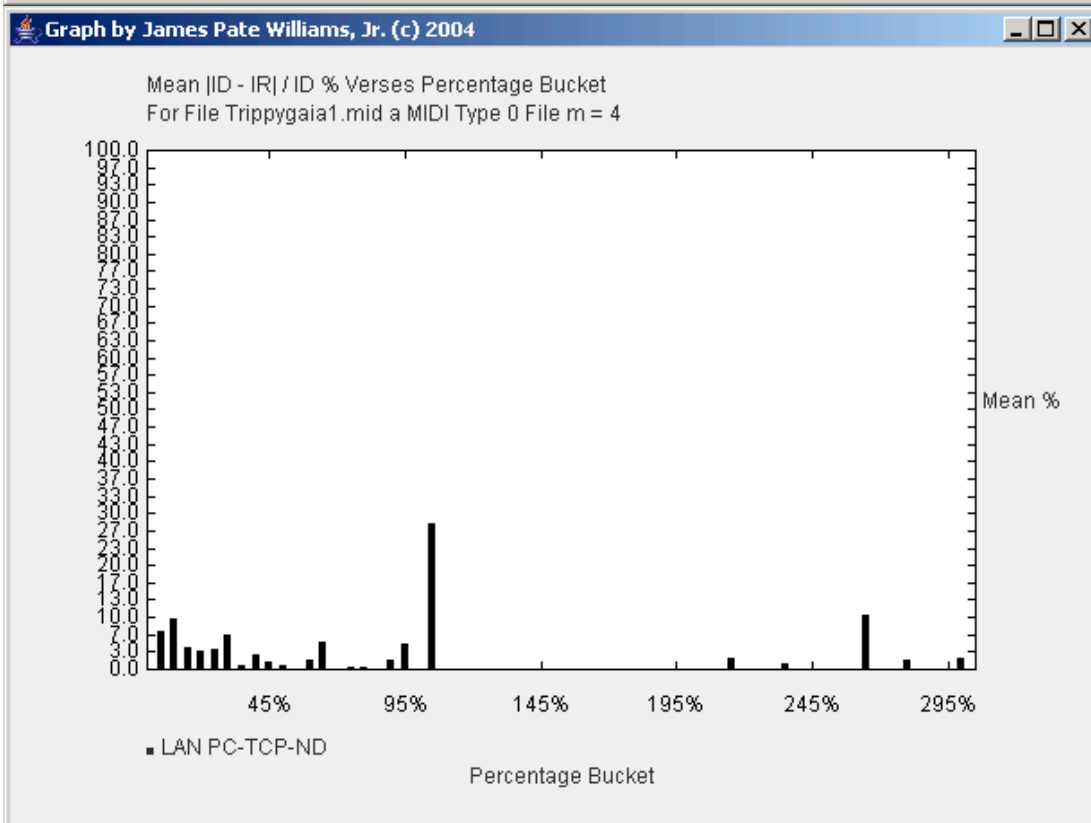
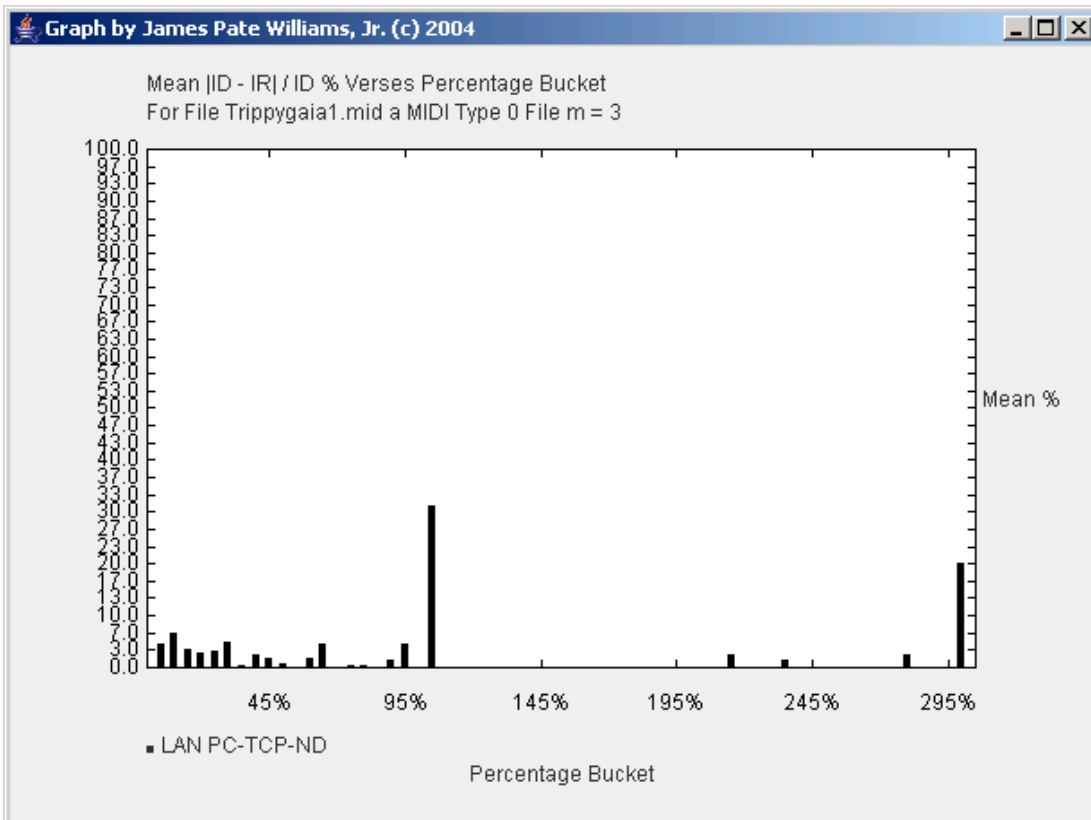


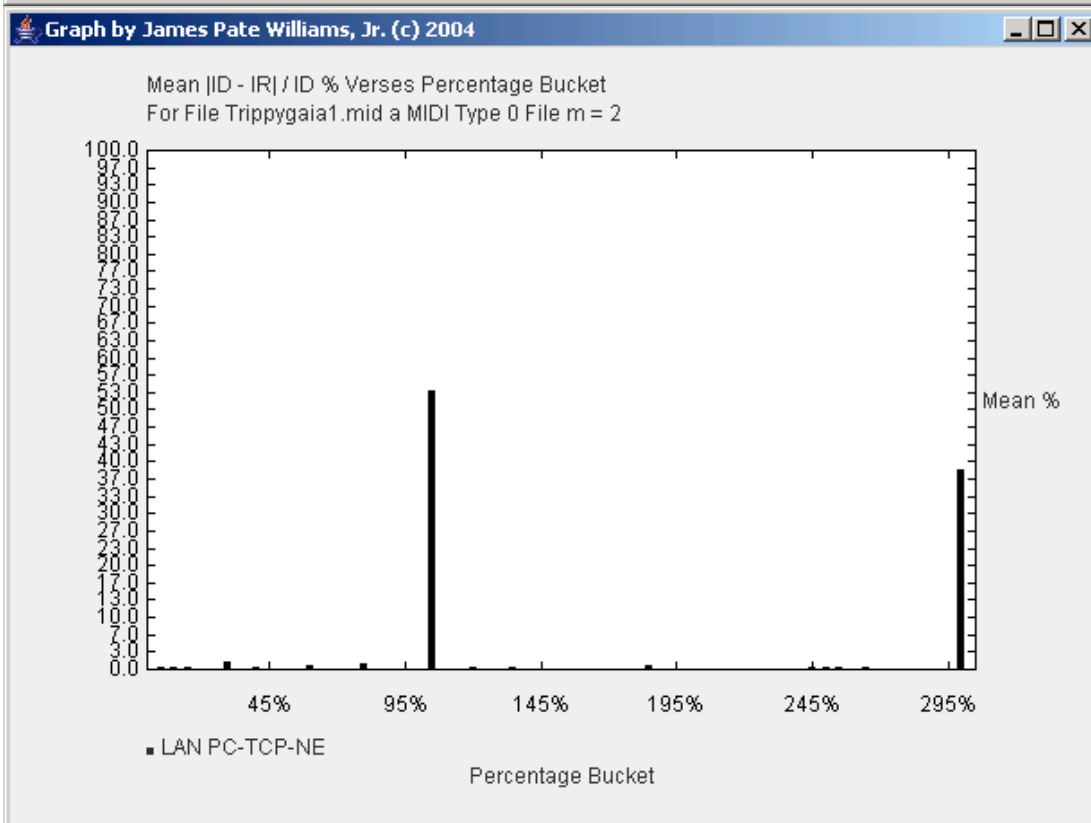
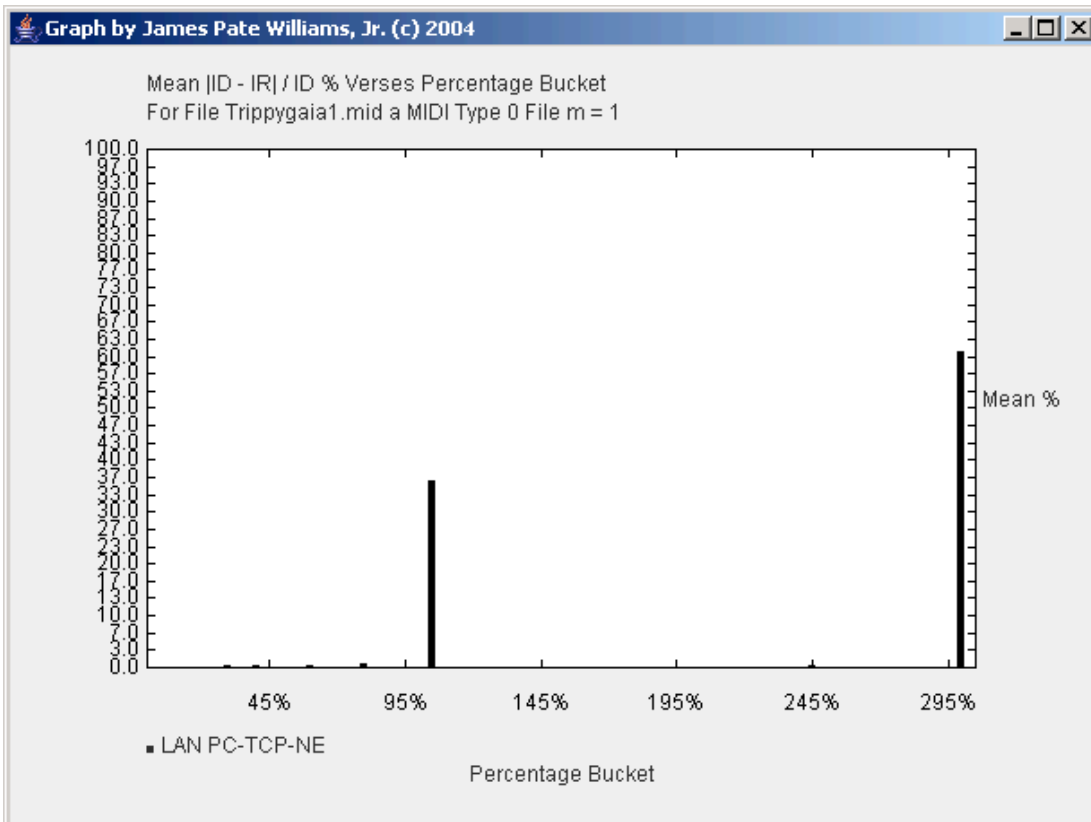




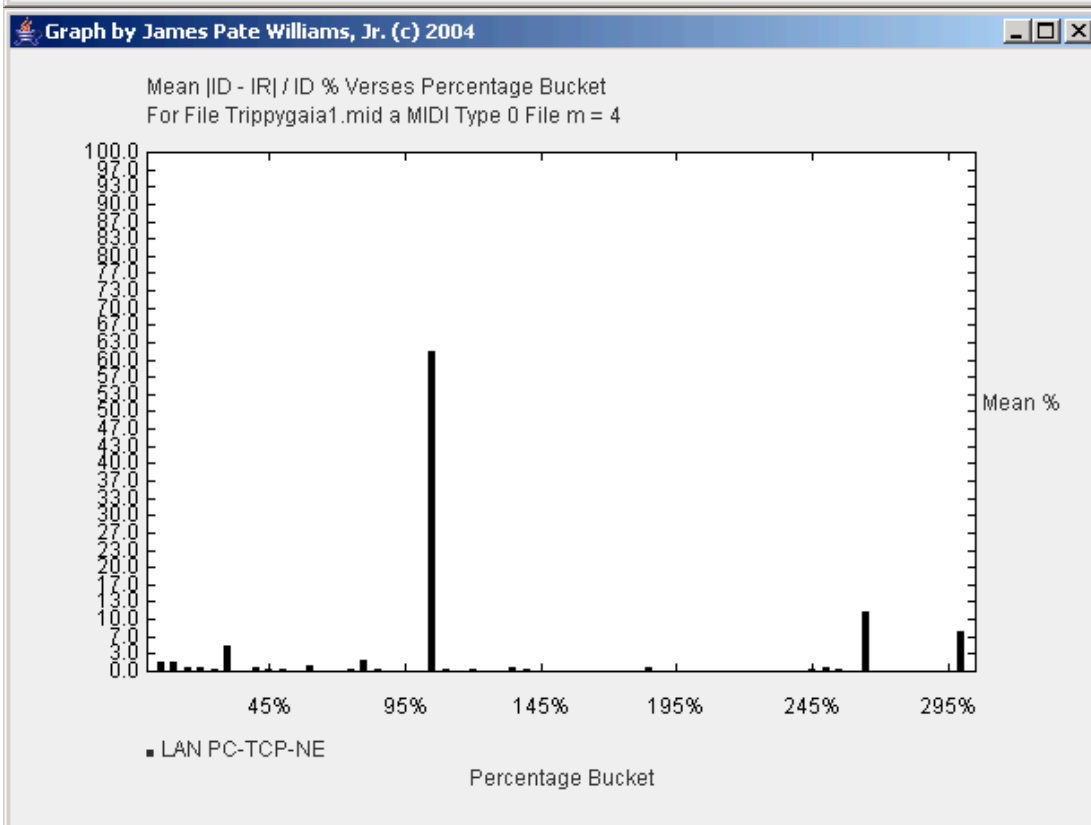
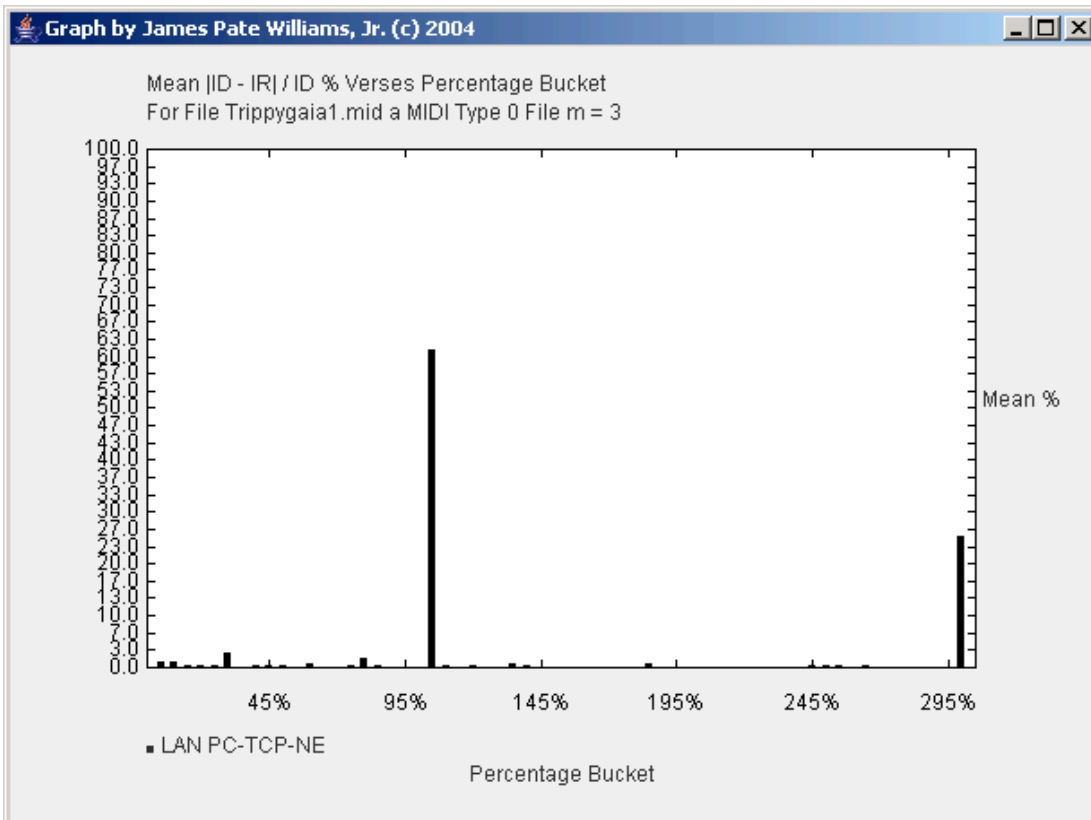


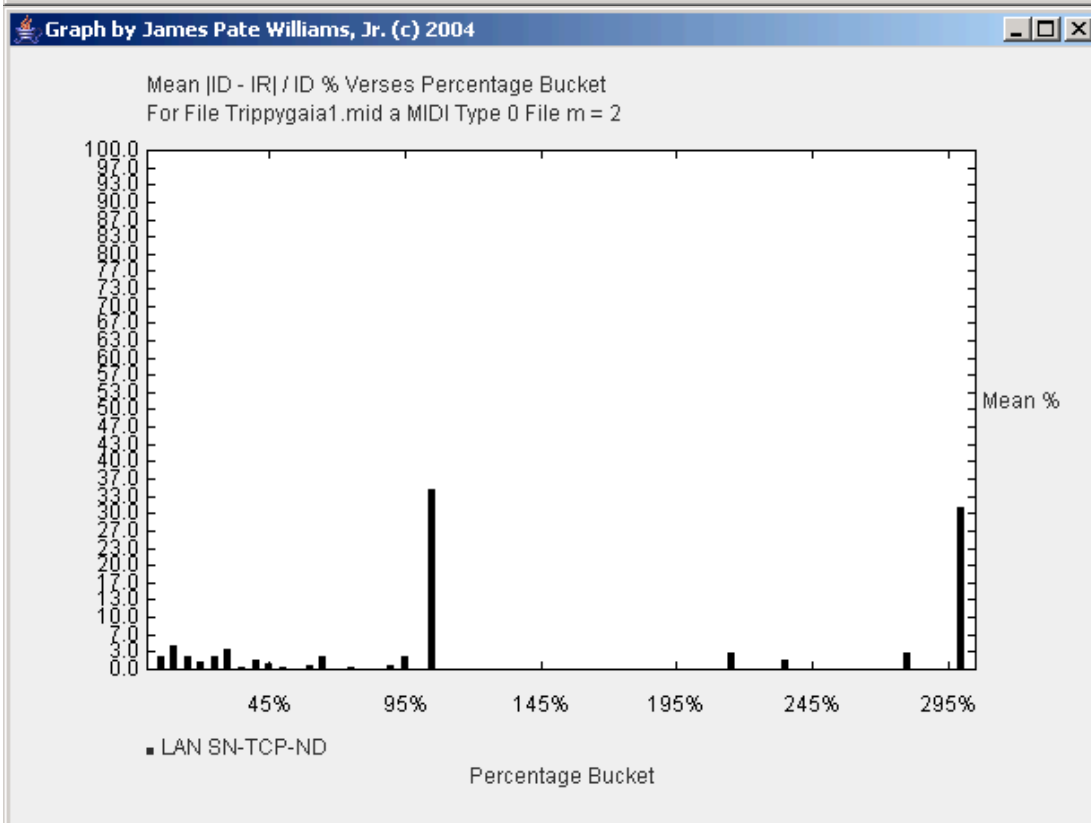
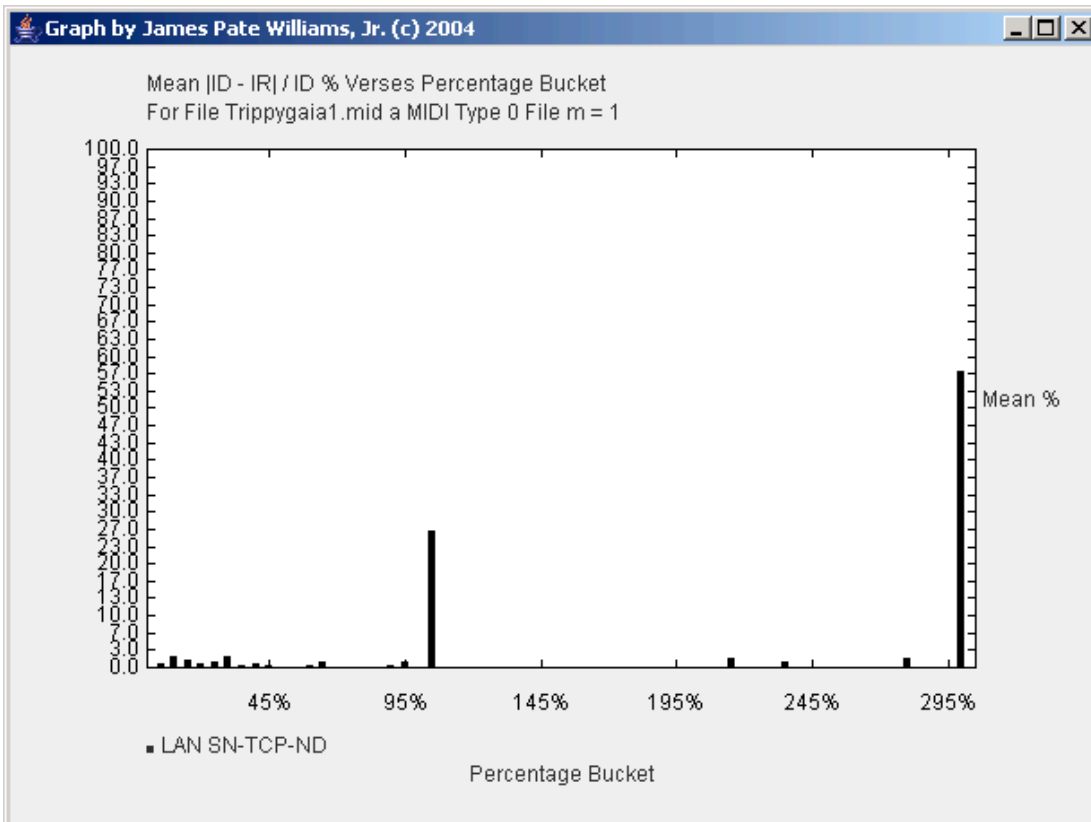


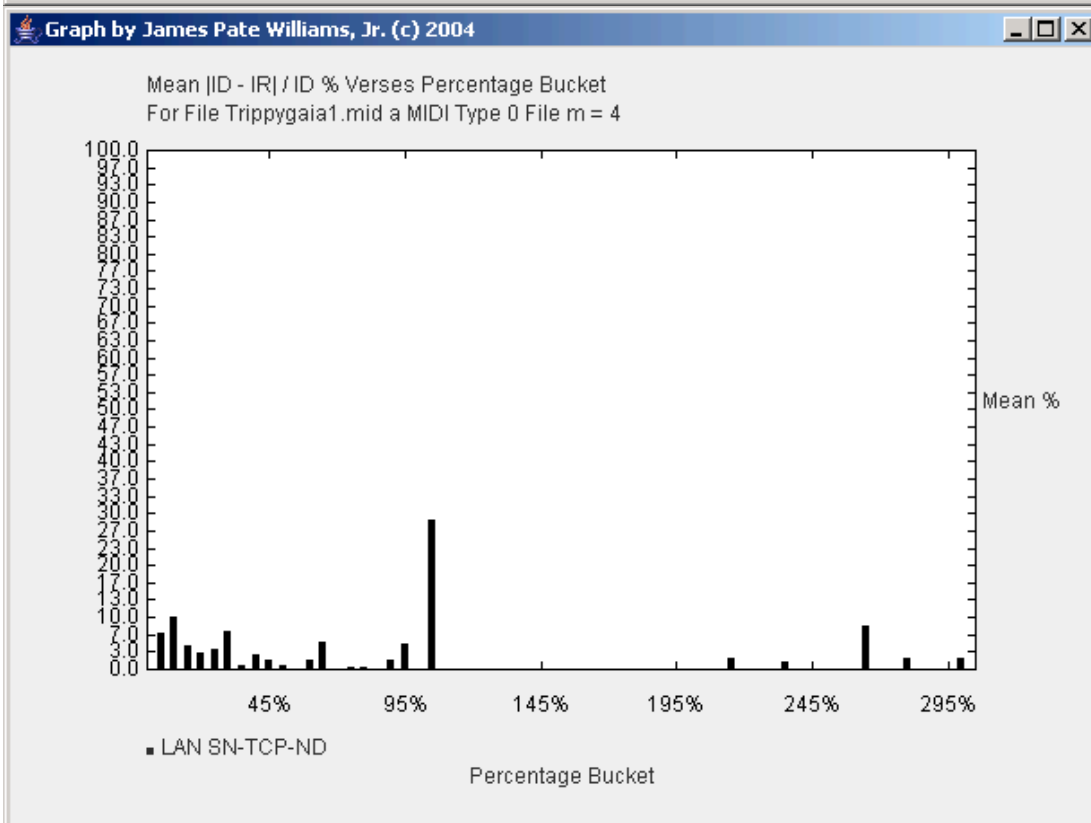
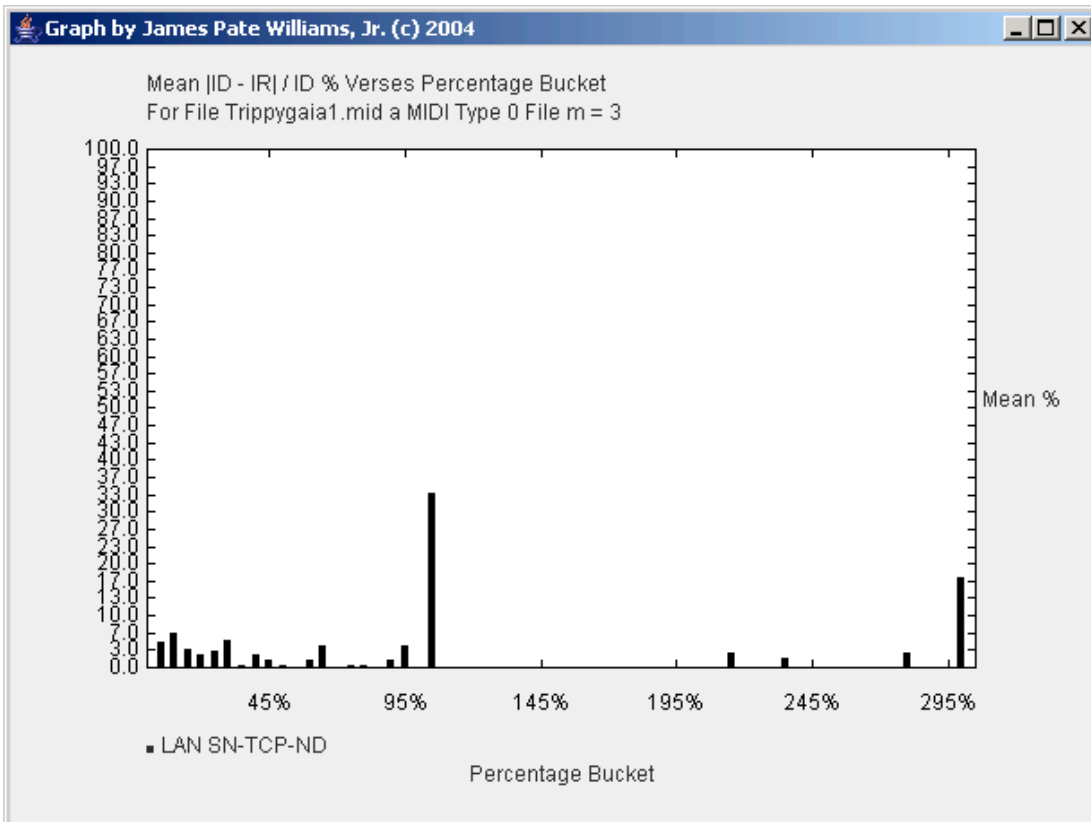


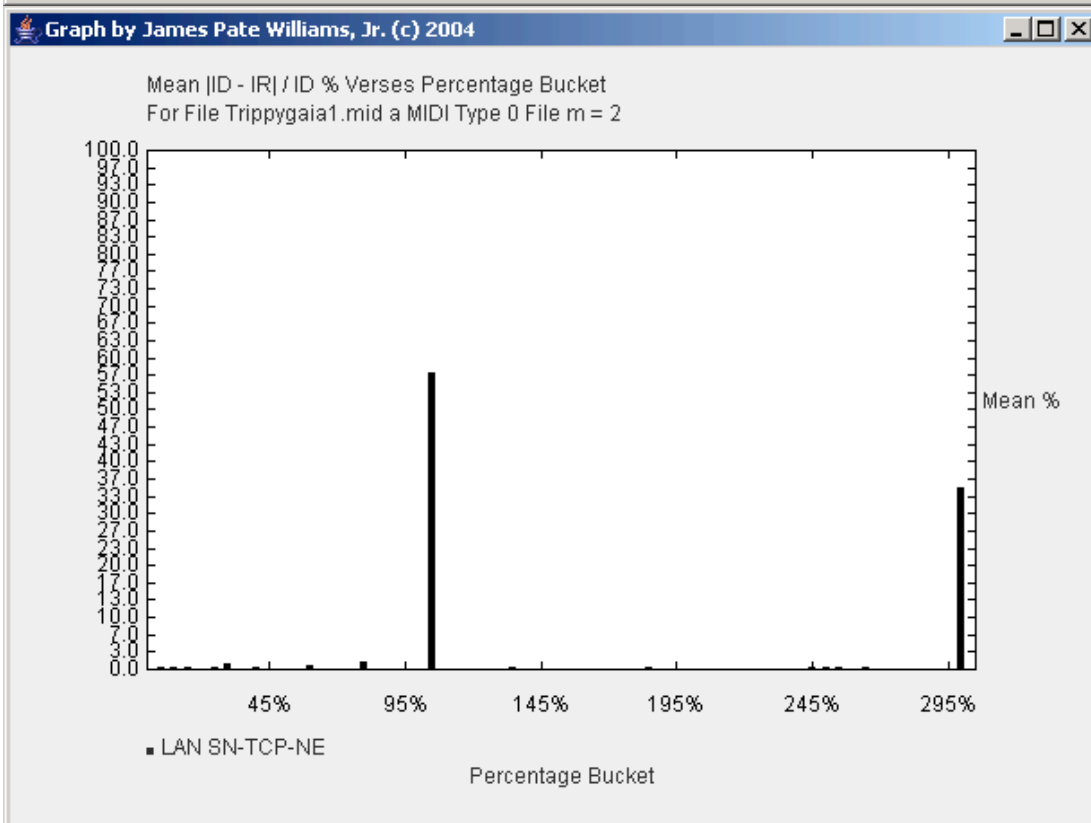
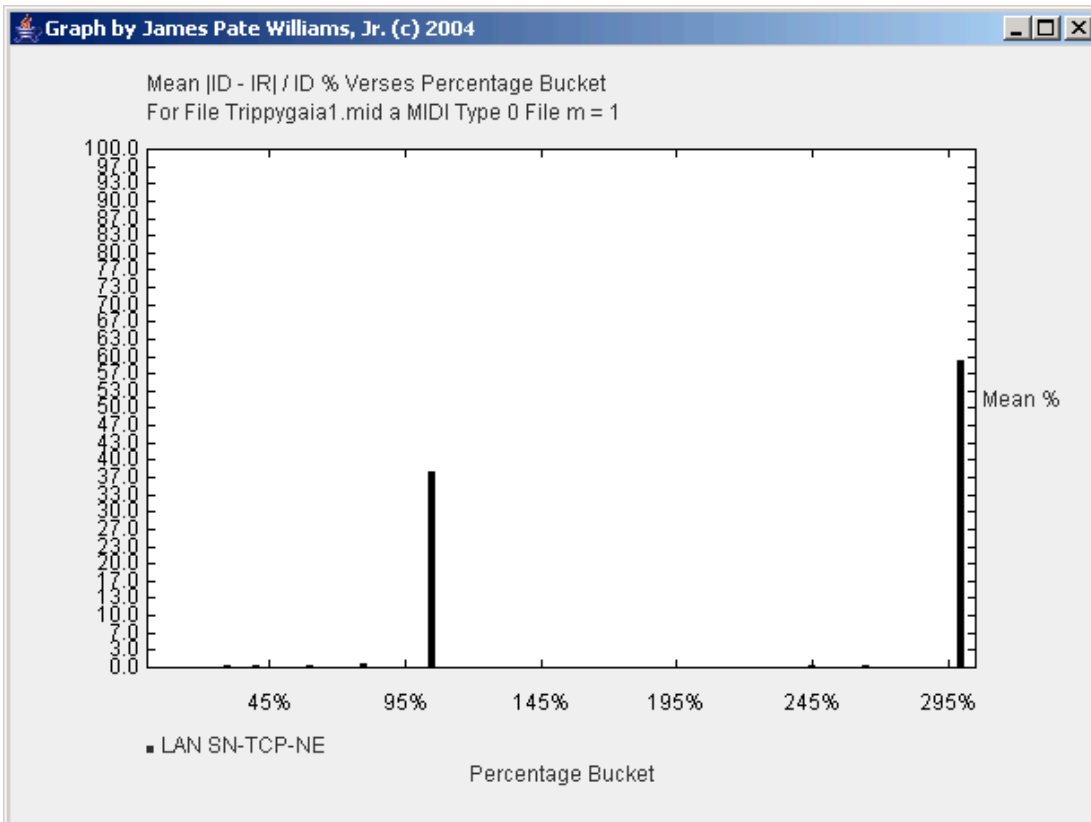


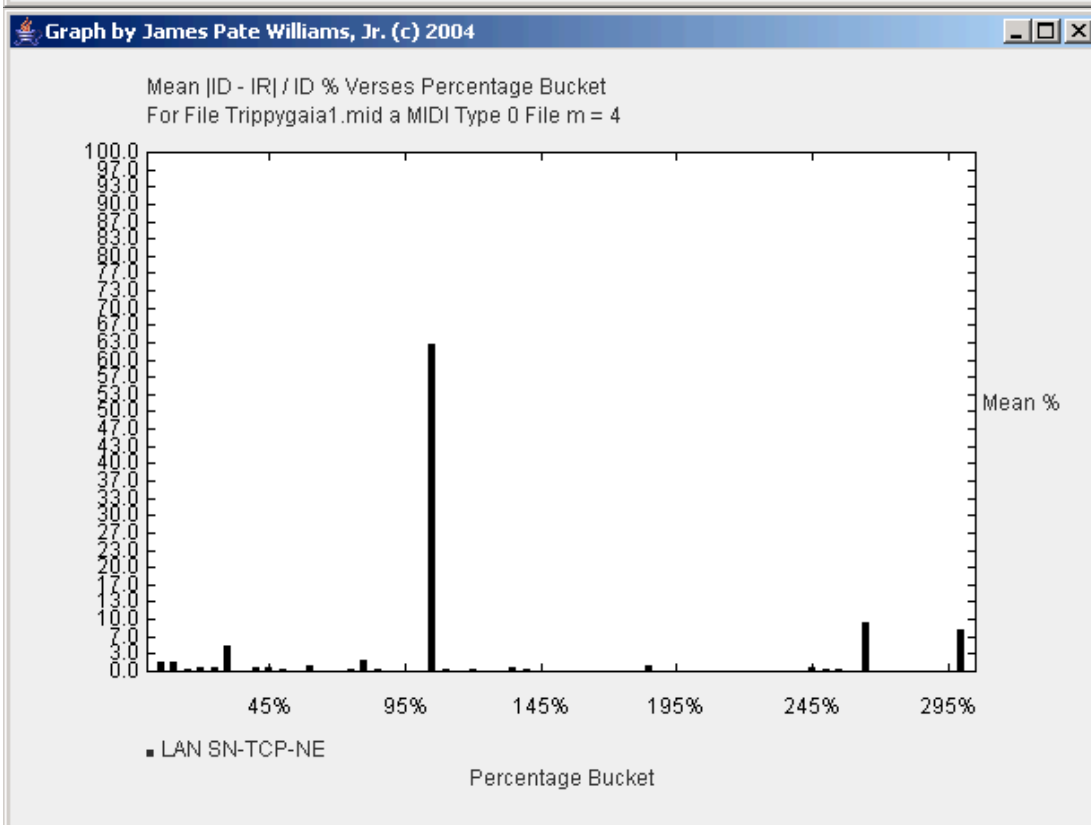
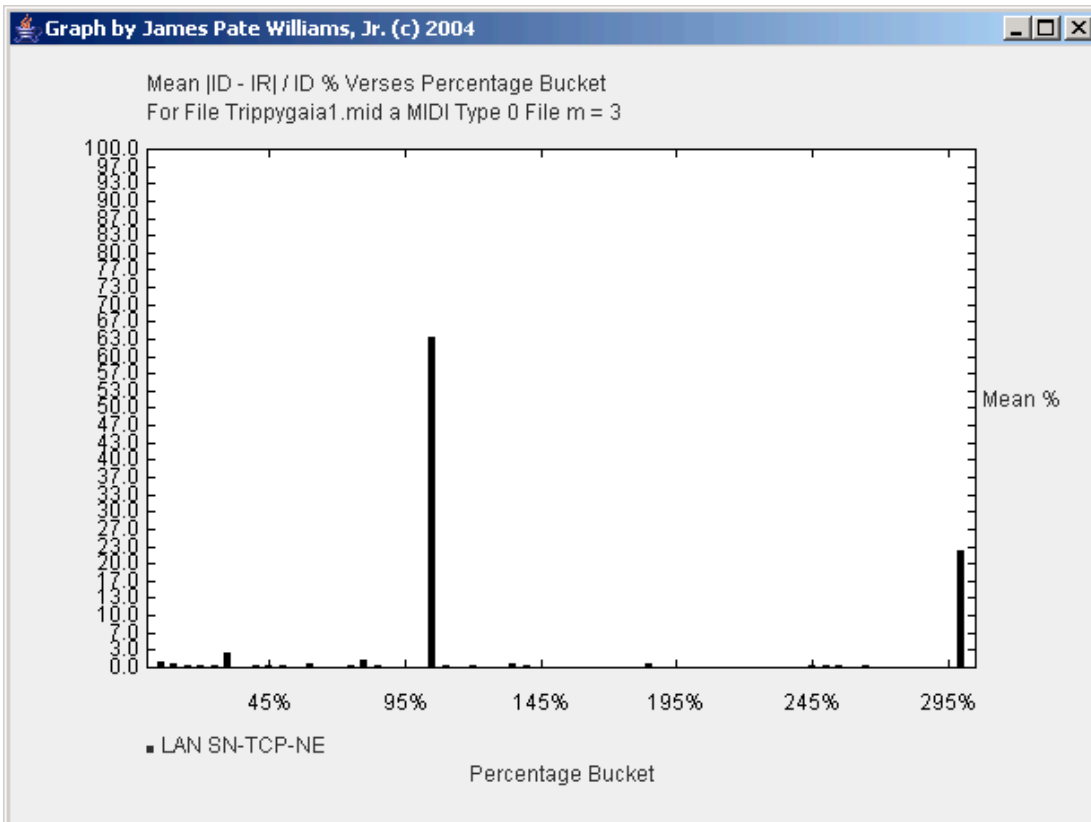






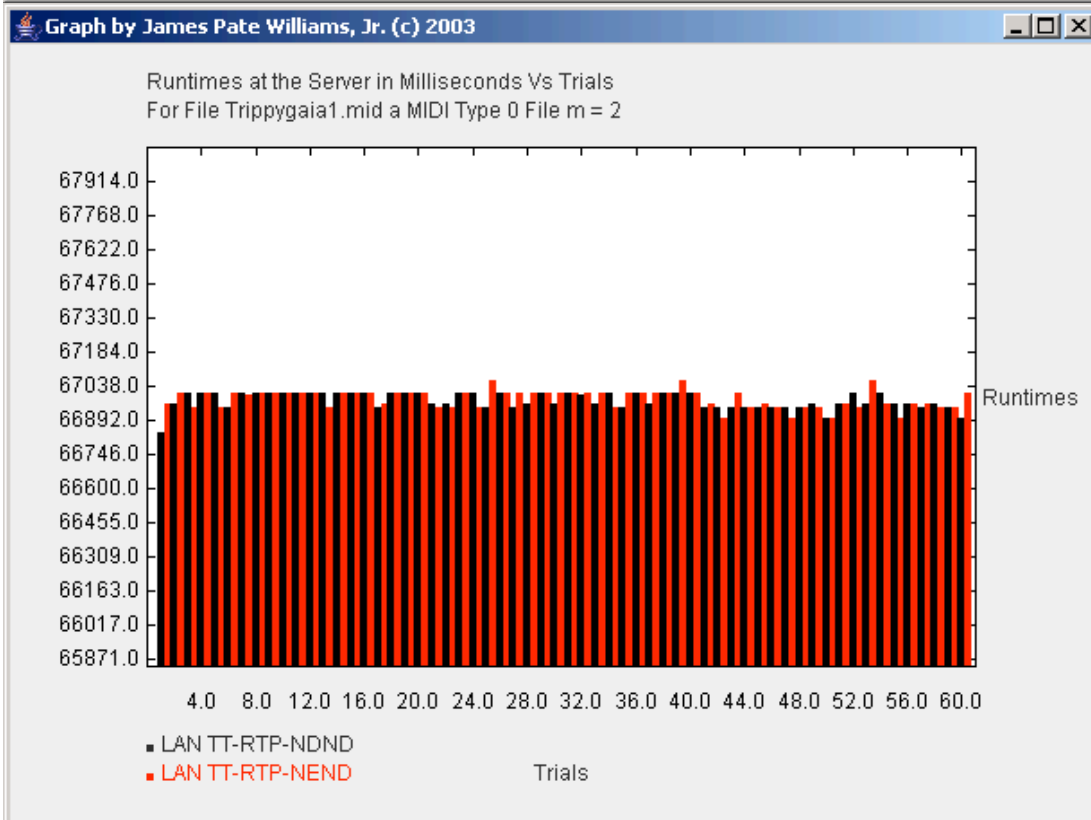
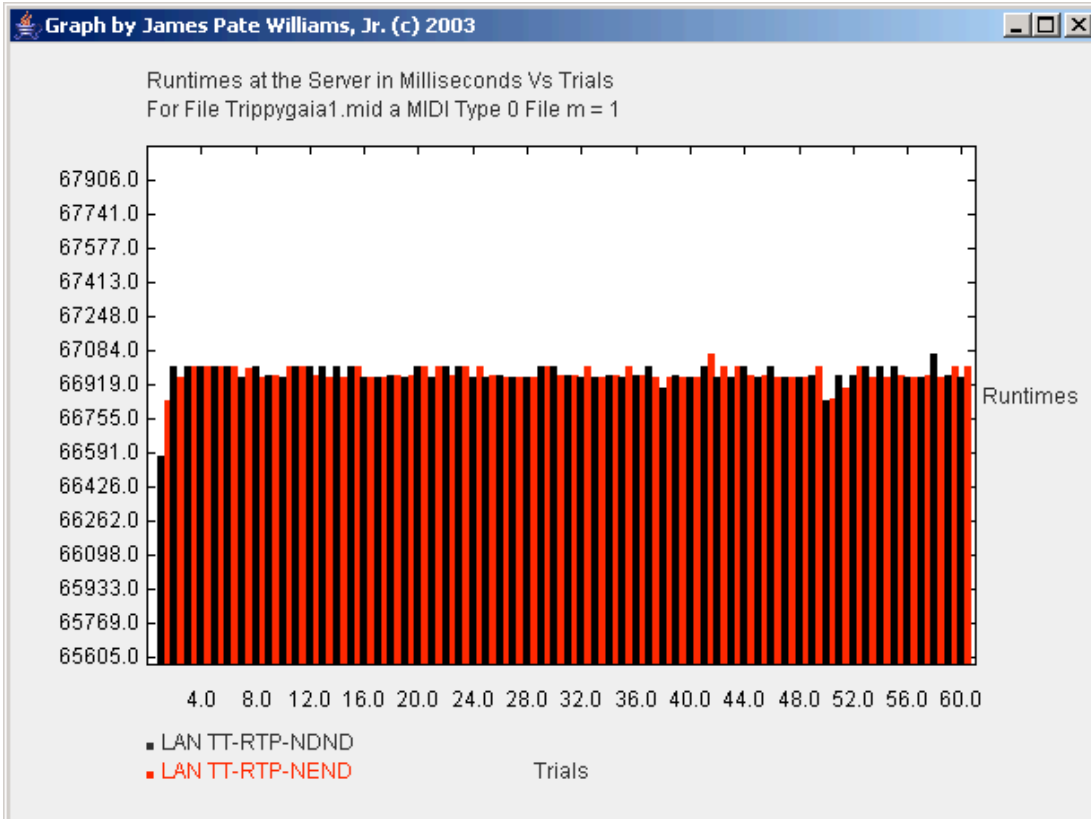


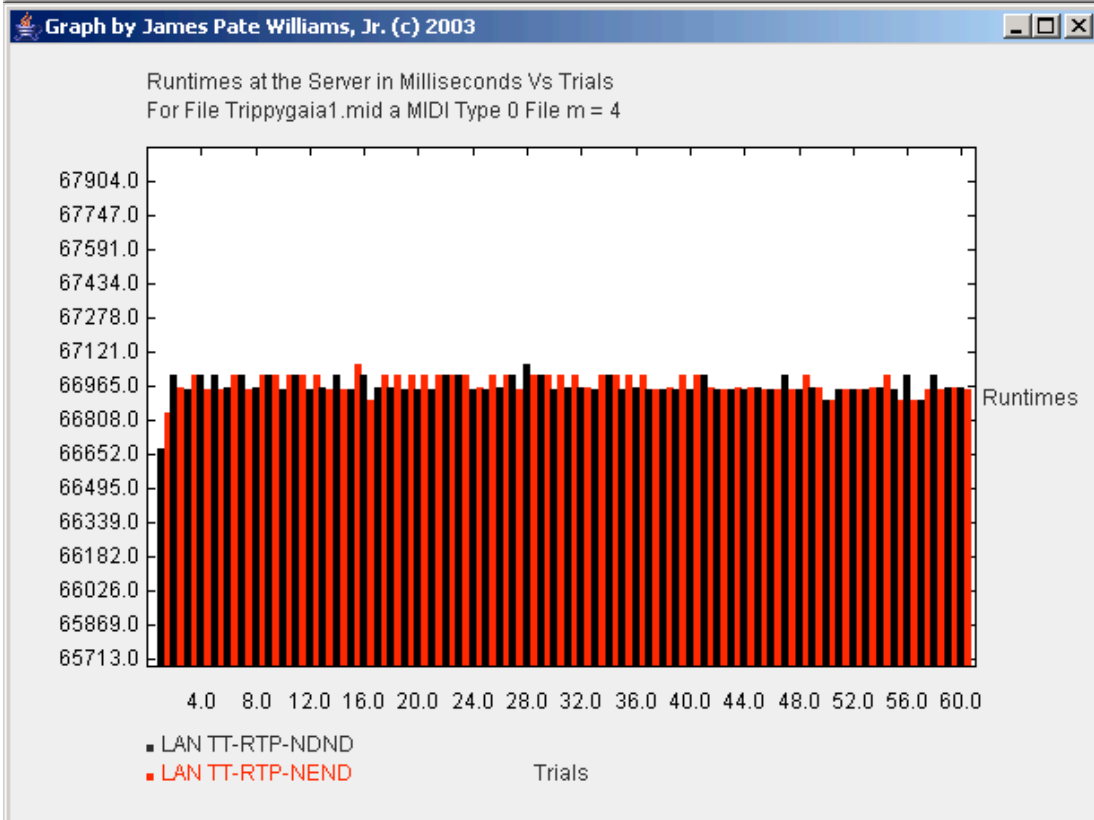
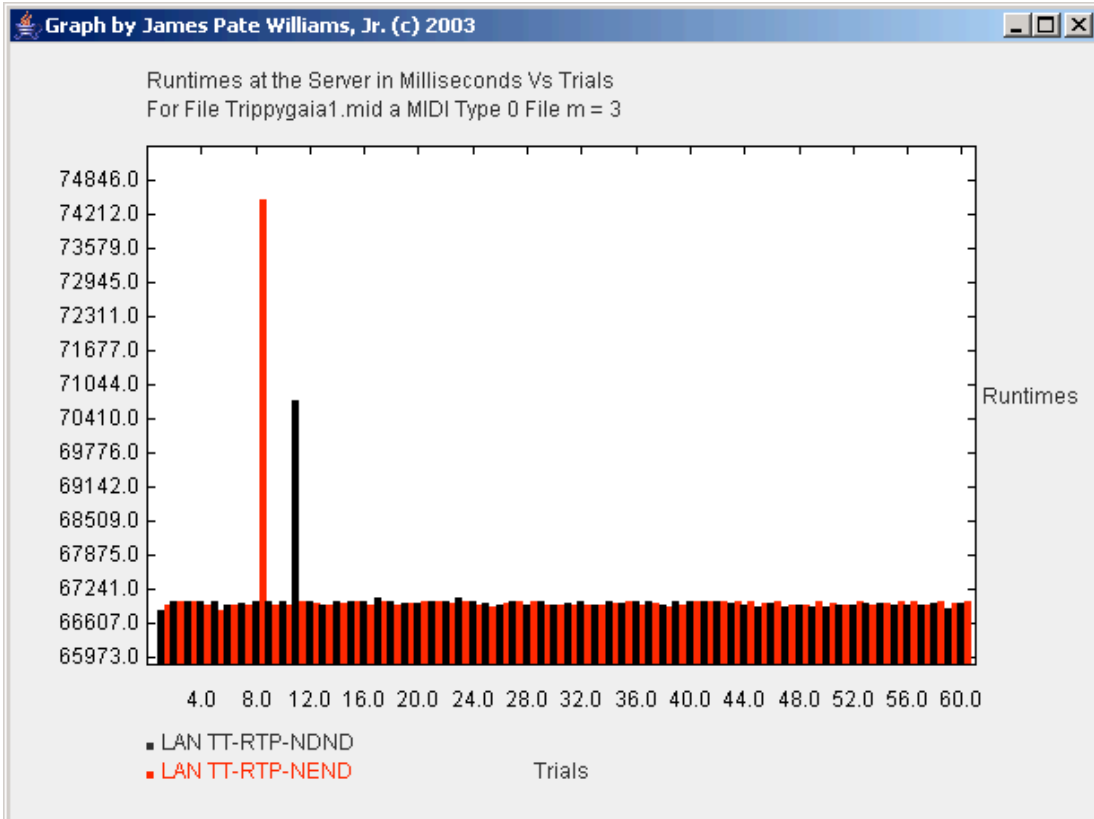




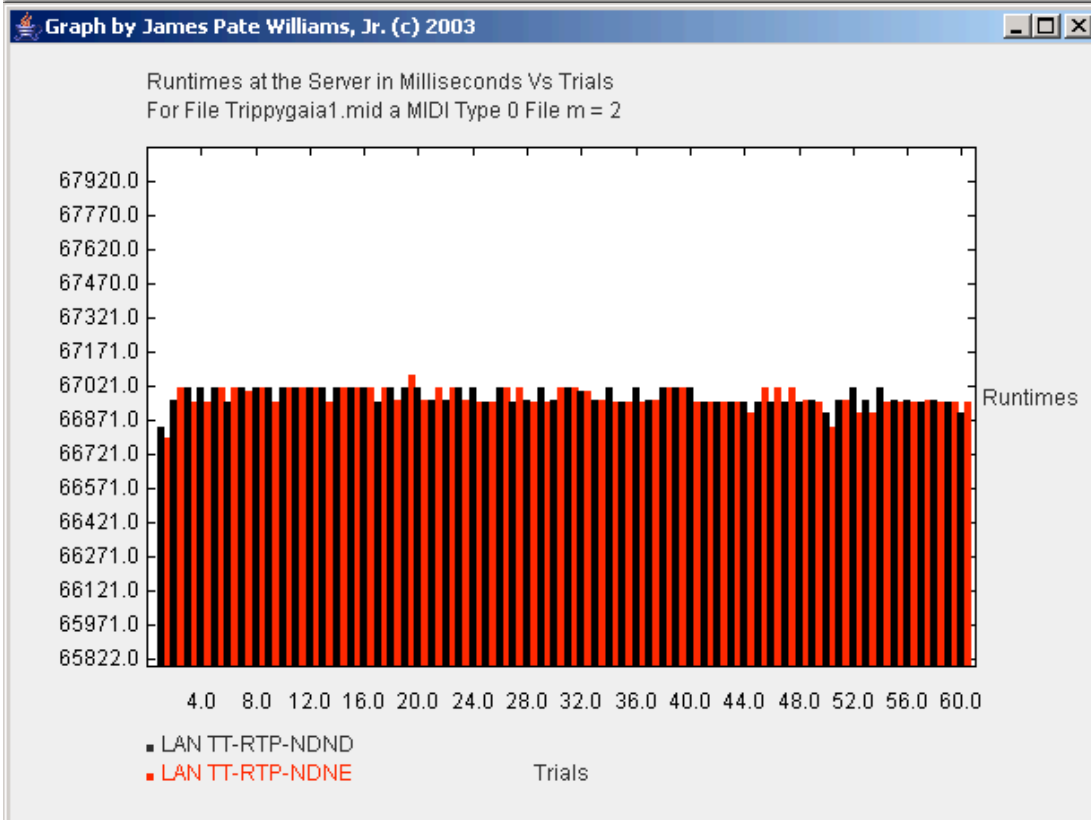
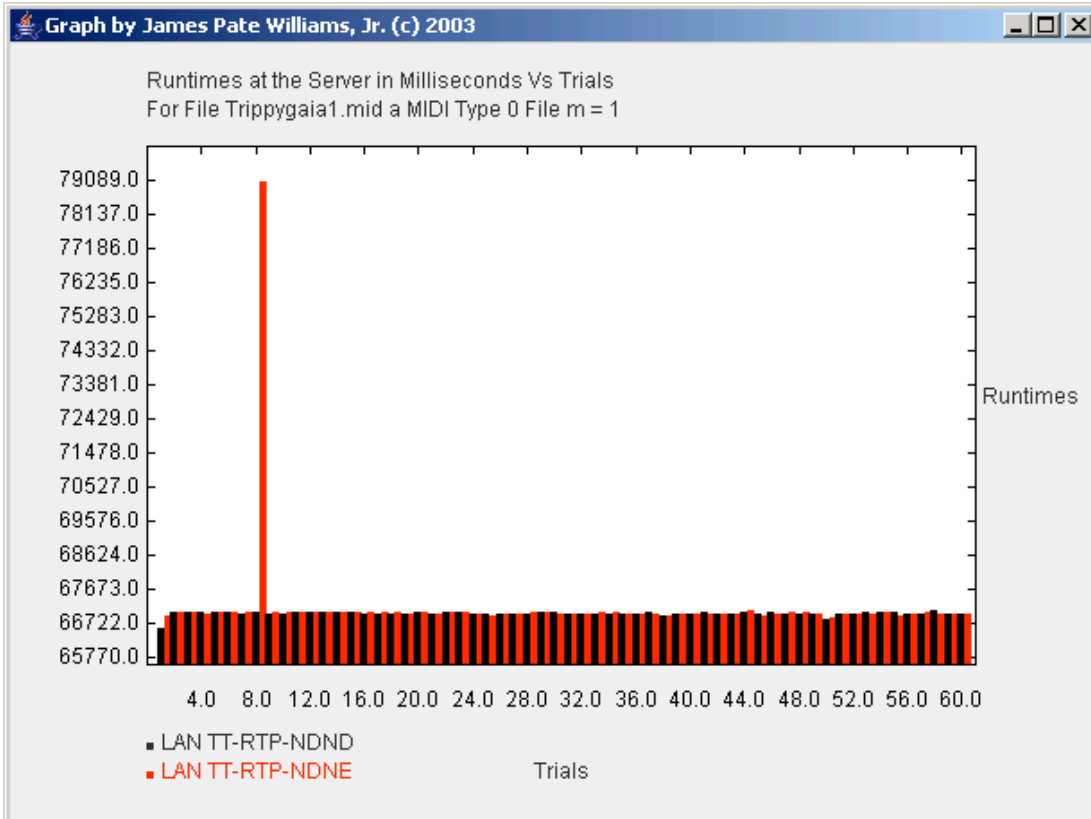
## APPENDIX B LAN PAIRED MEANS COMPARISON GRAPHS

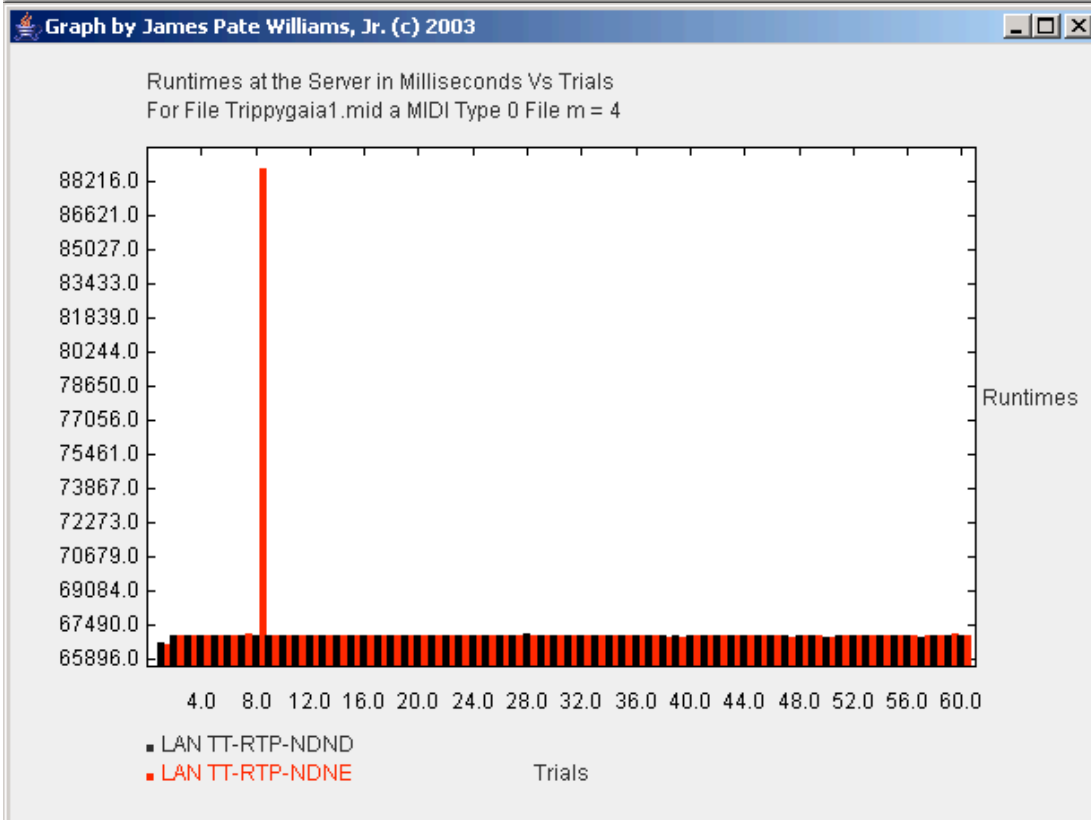
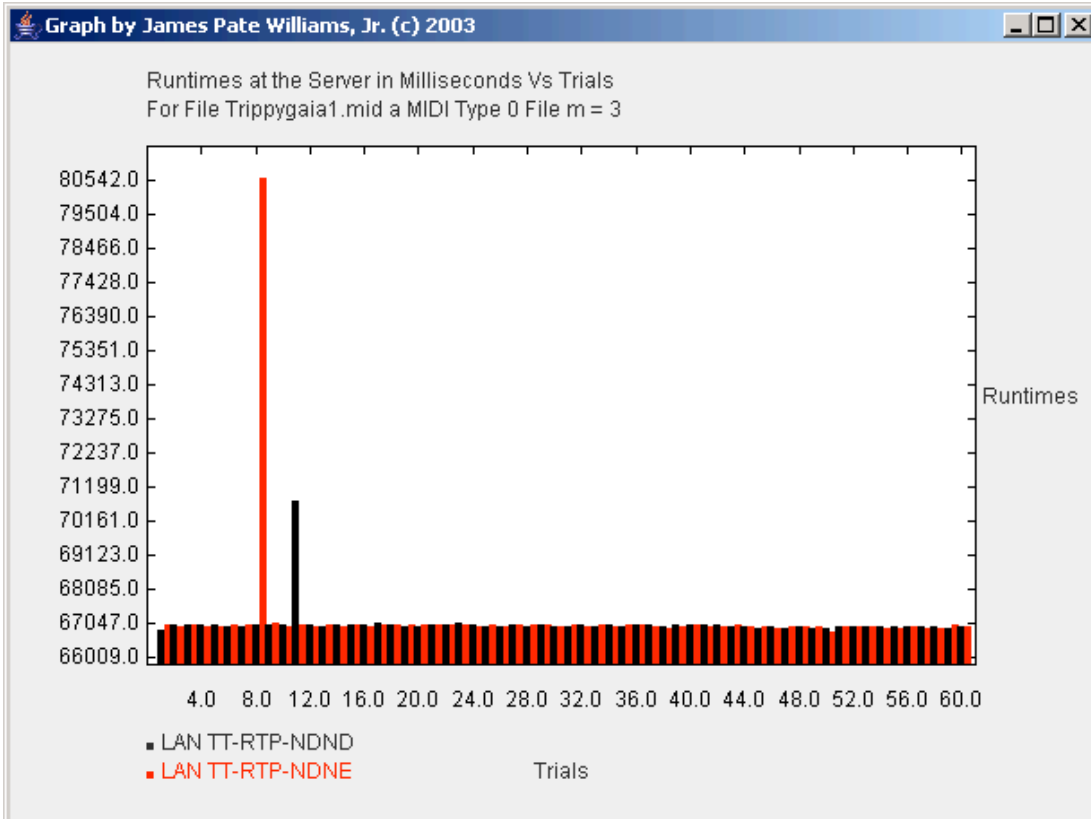
This appendix consists of 180 graphs of the mean runtime of a MIDI sequence on the destination host on a LAN. The number of graphs can be determined by calculating the total number of possible pairings of the protocols as  $(10 * 9) / 2 = 45$  and multiplying 45 by 4 to get 180, where 4 is the number of values of  $m$ , the number of MIDI short messages per TCP packet.

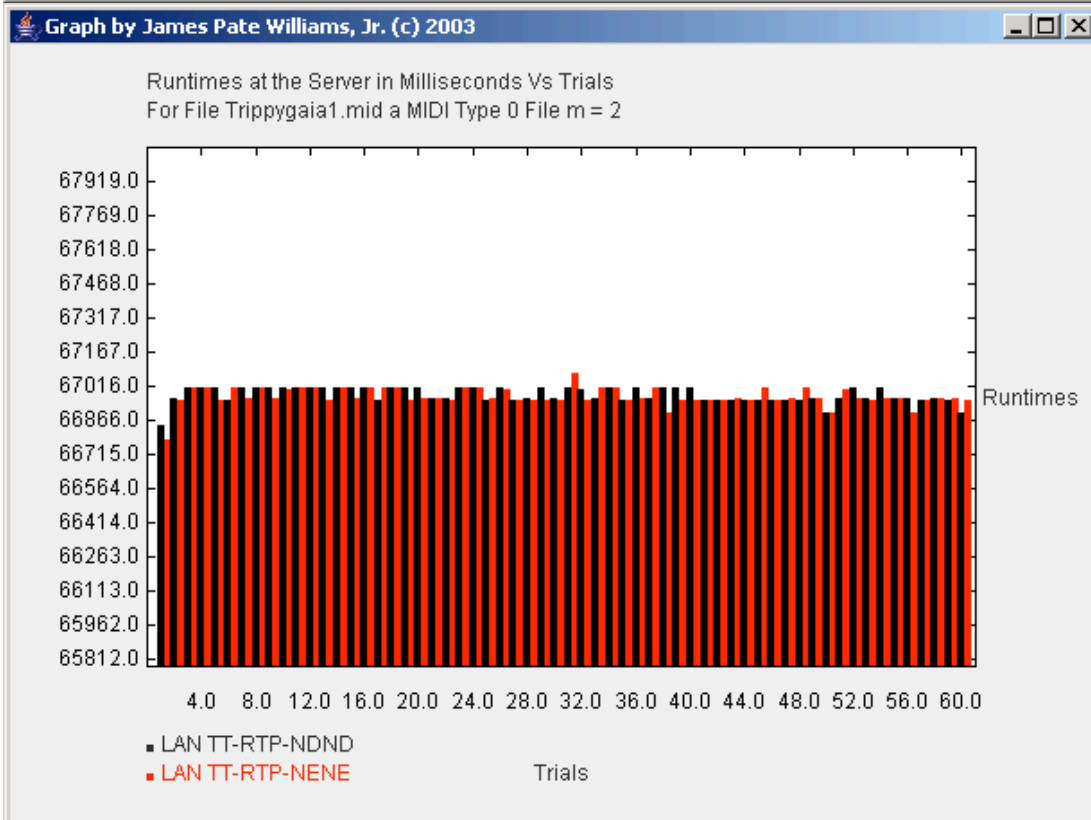
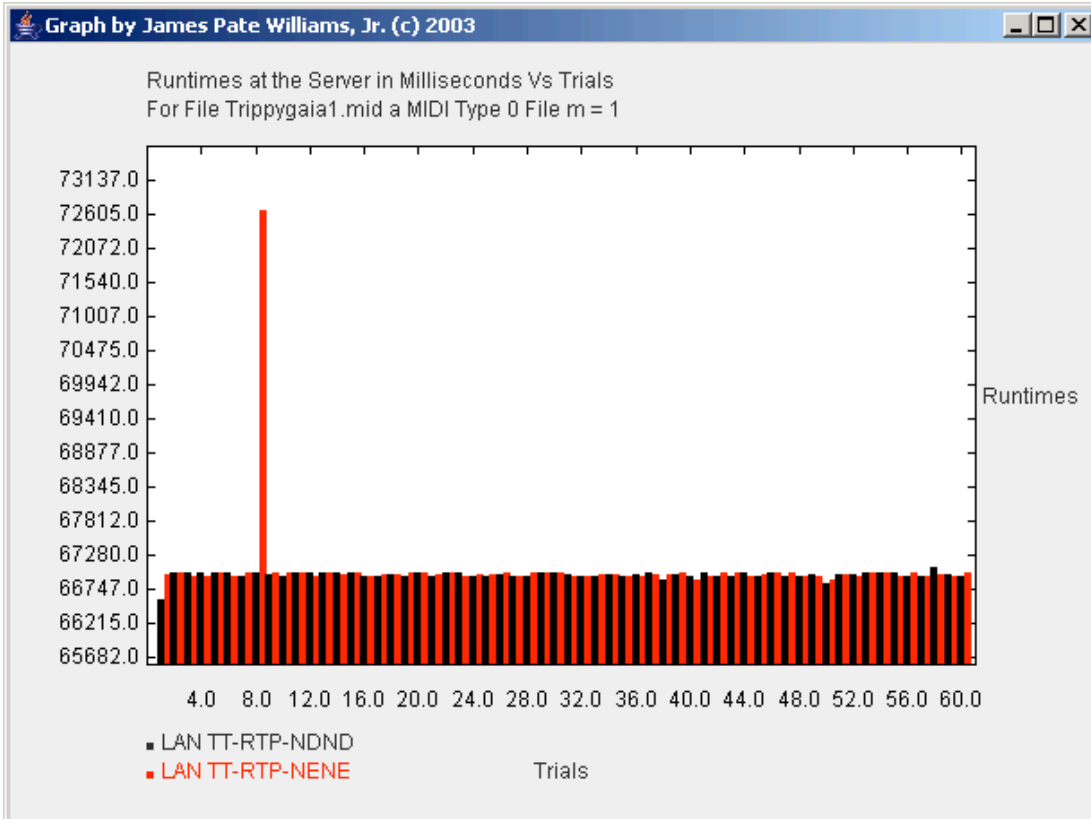


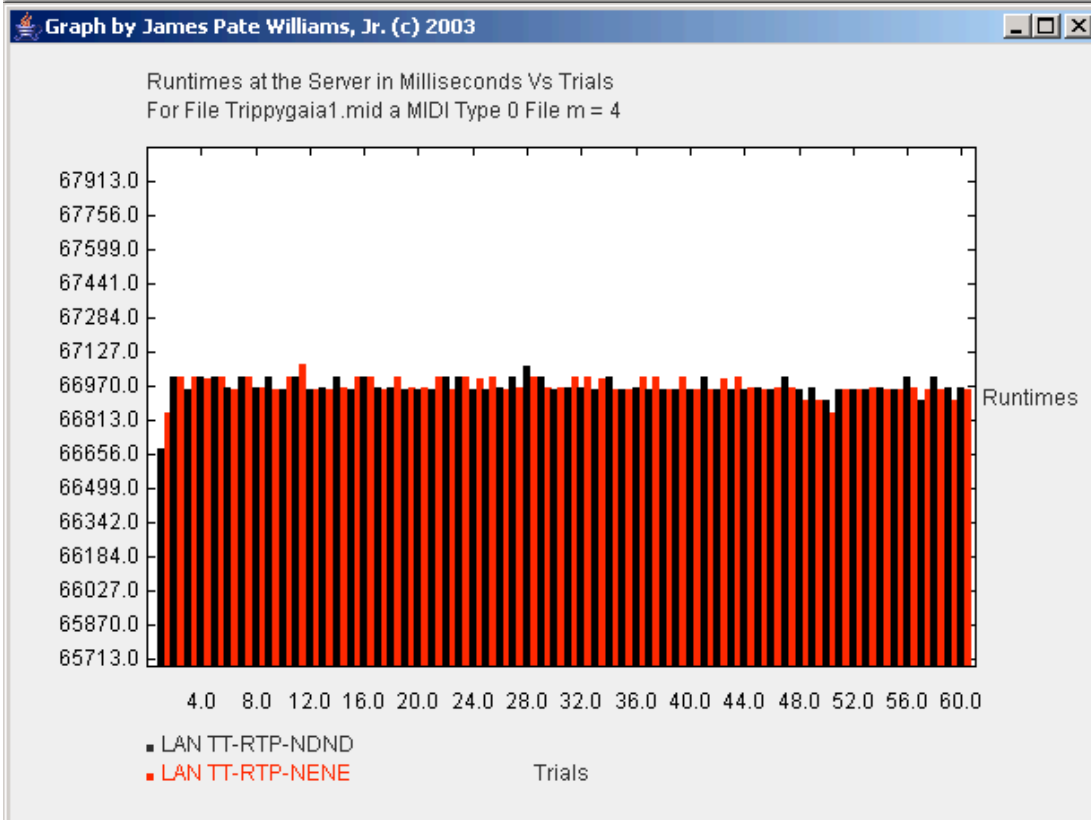
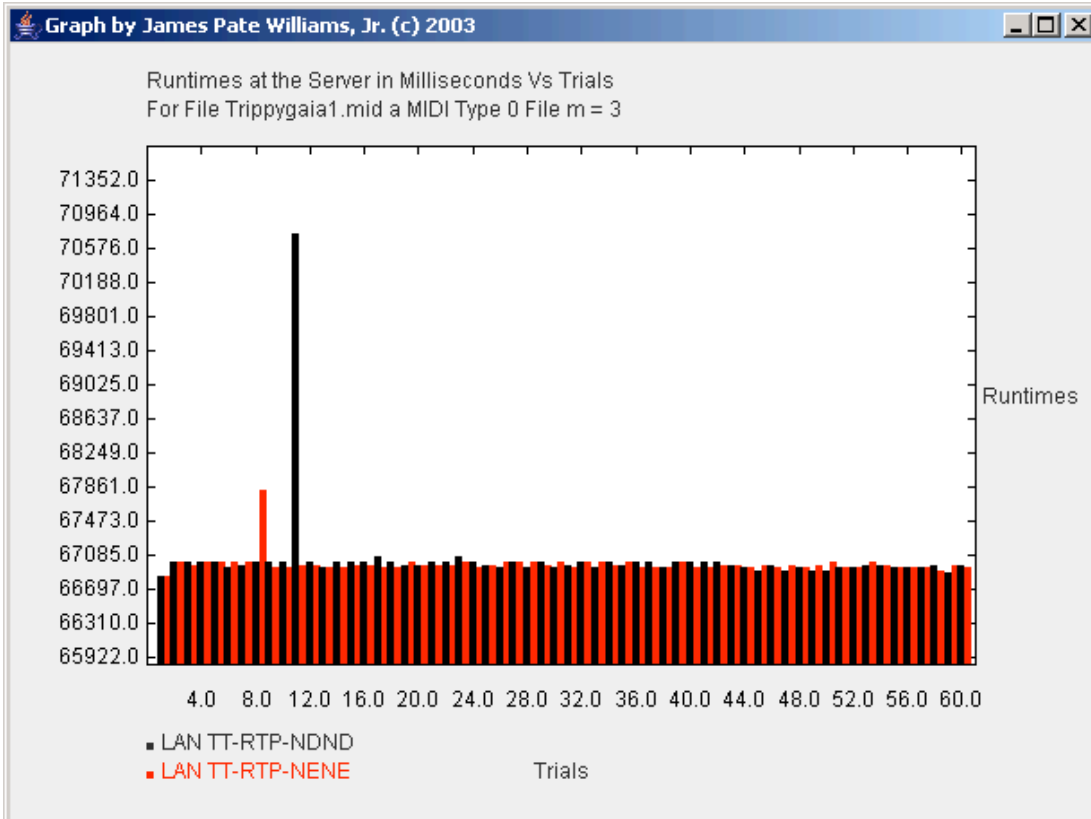


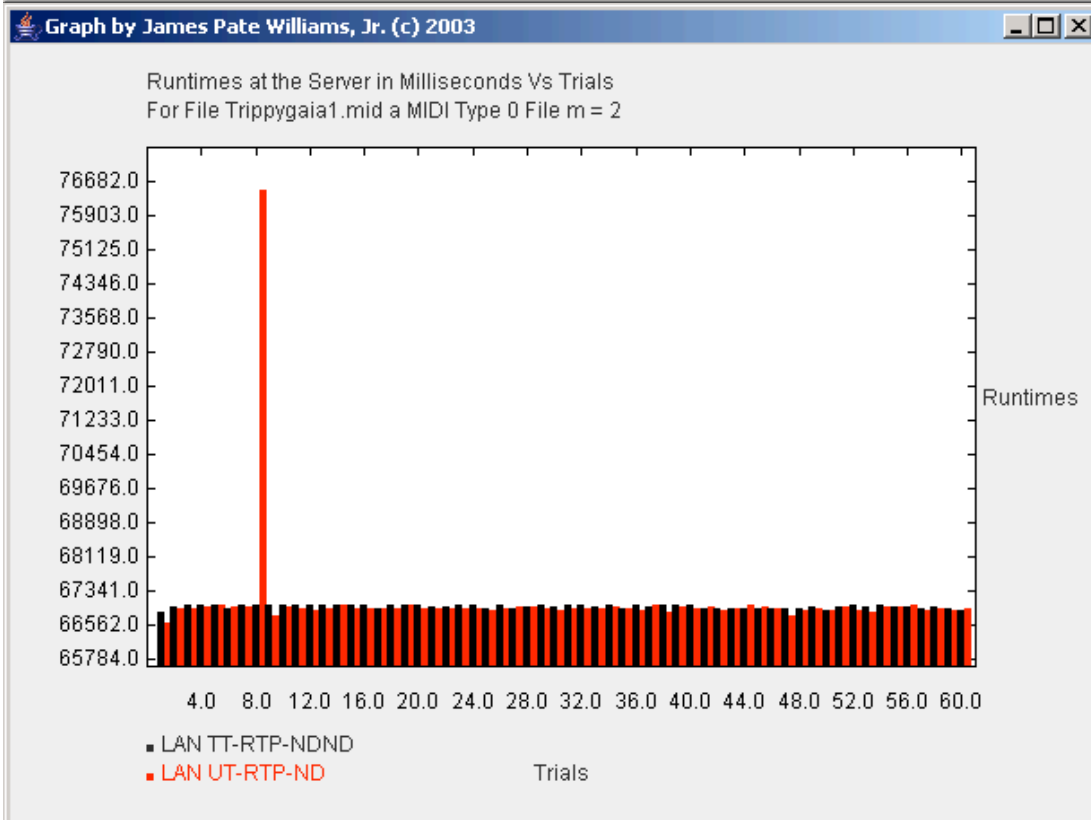
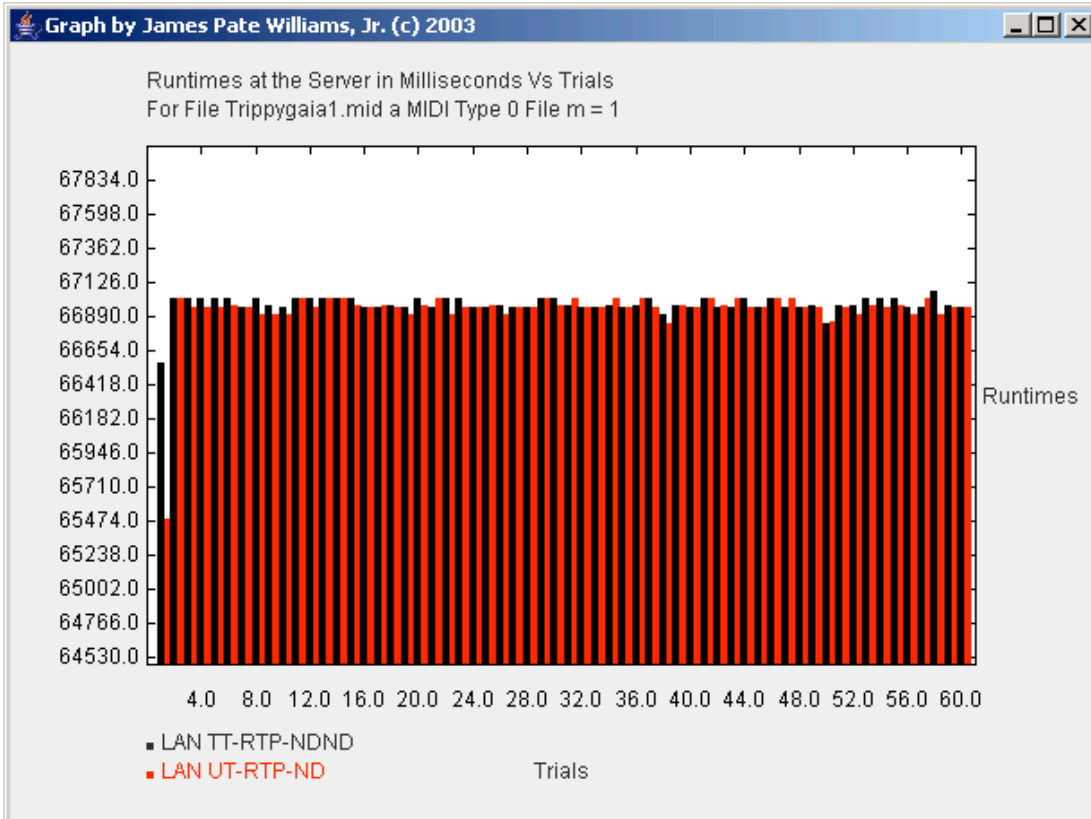


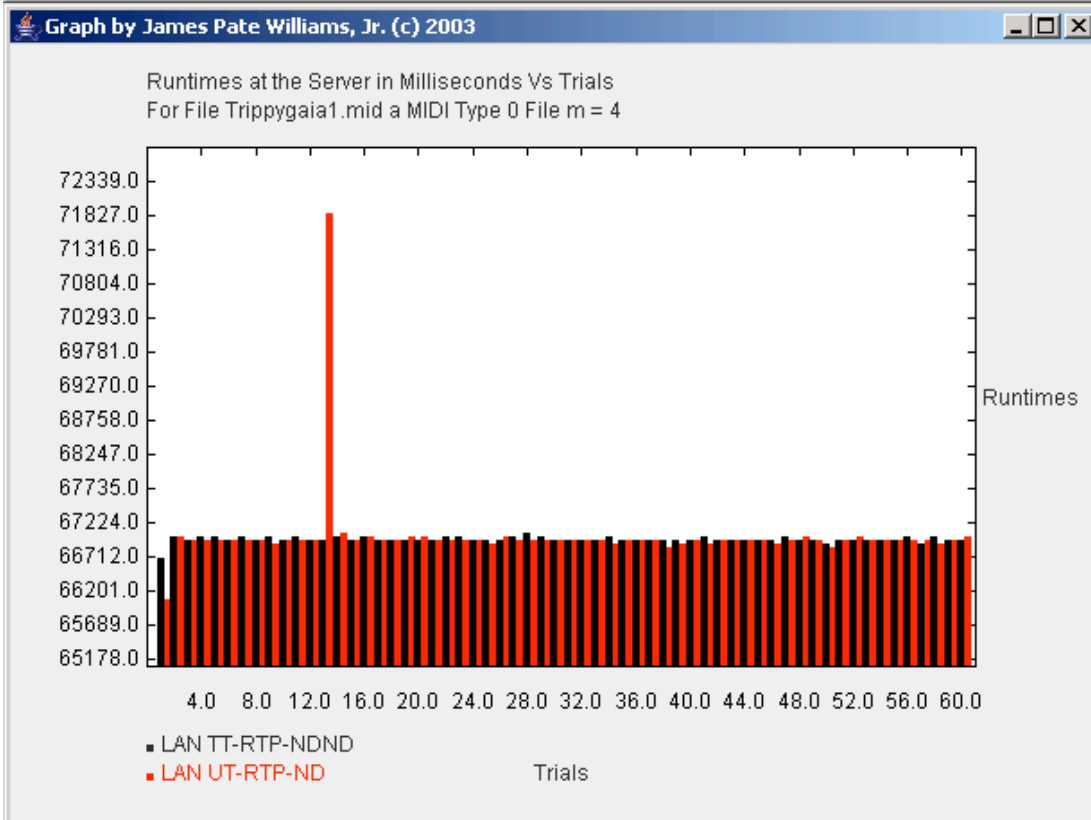
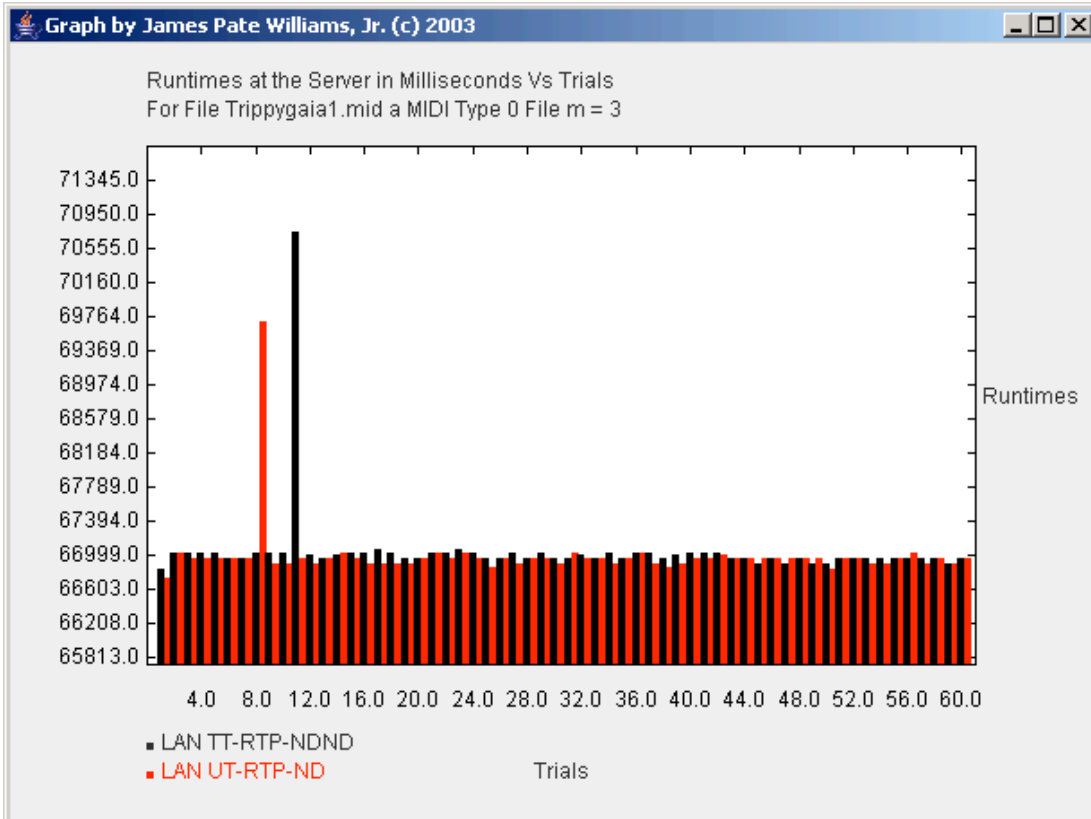


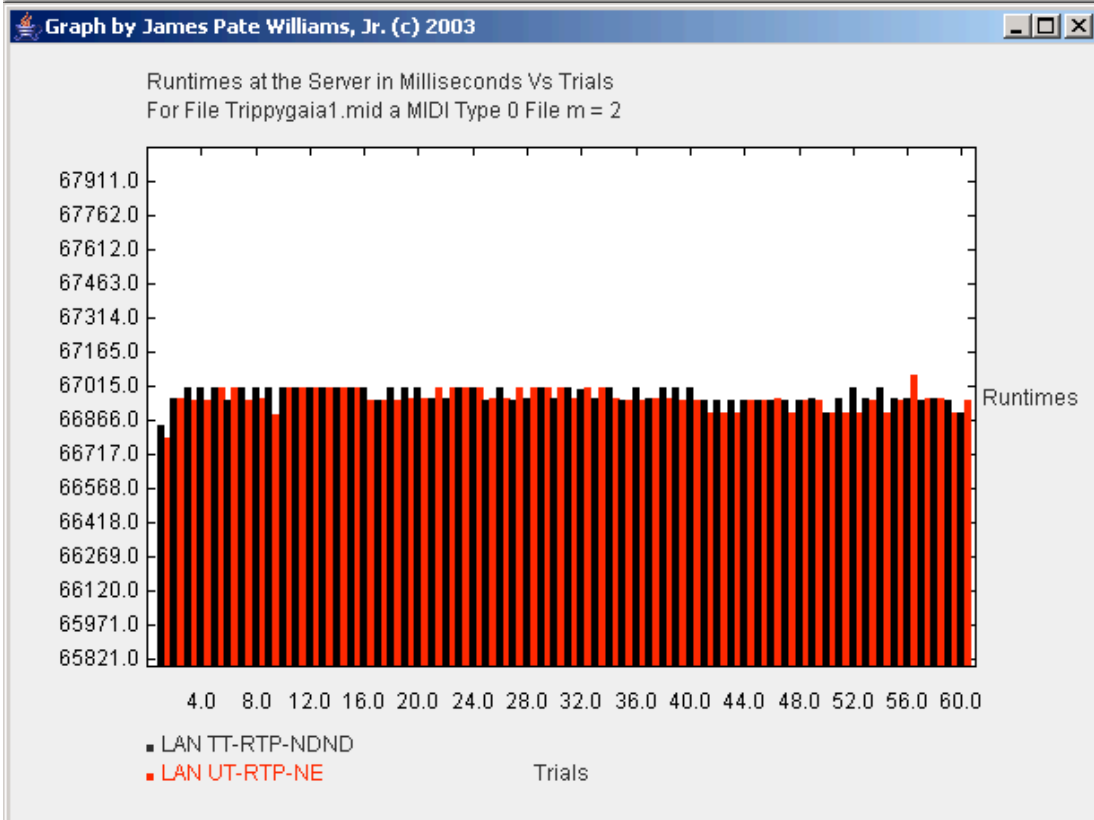
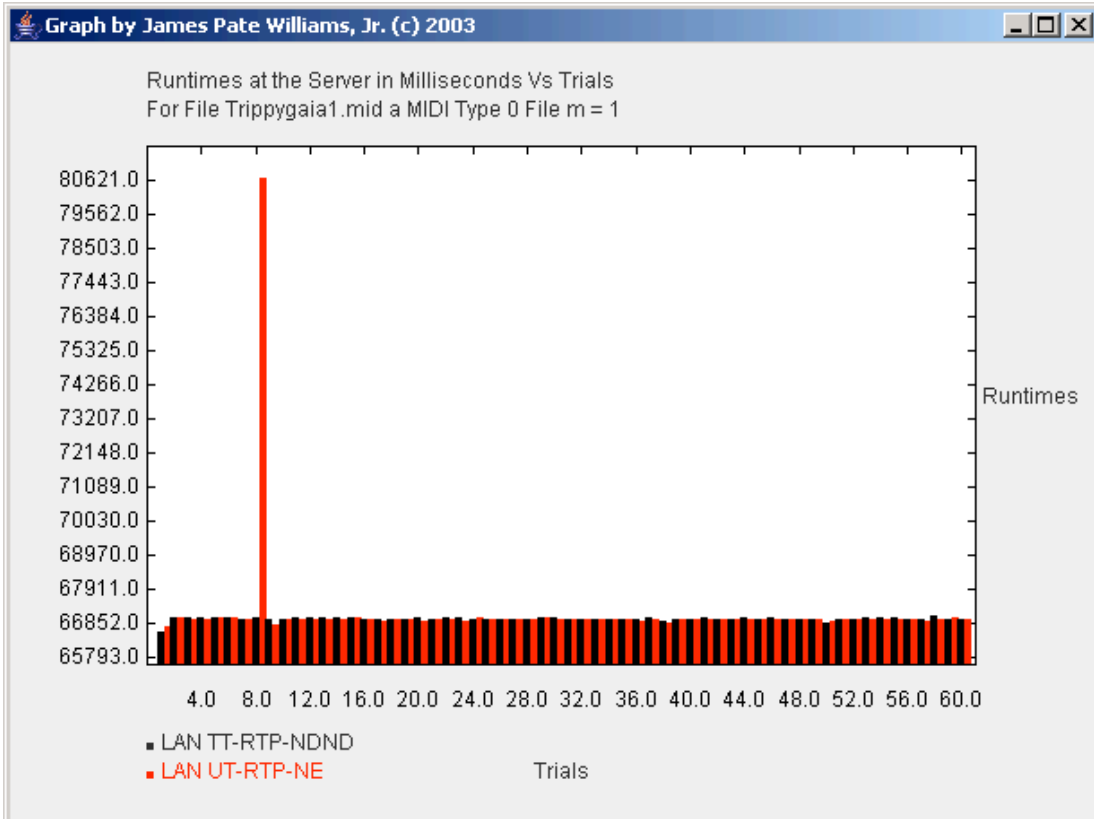


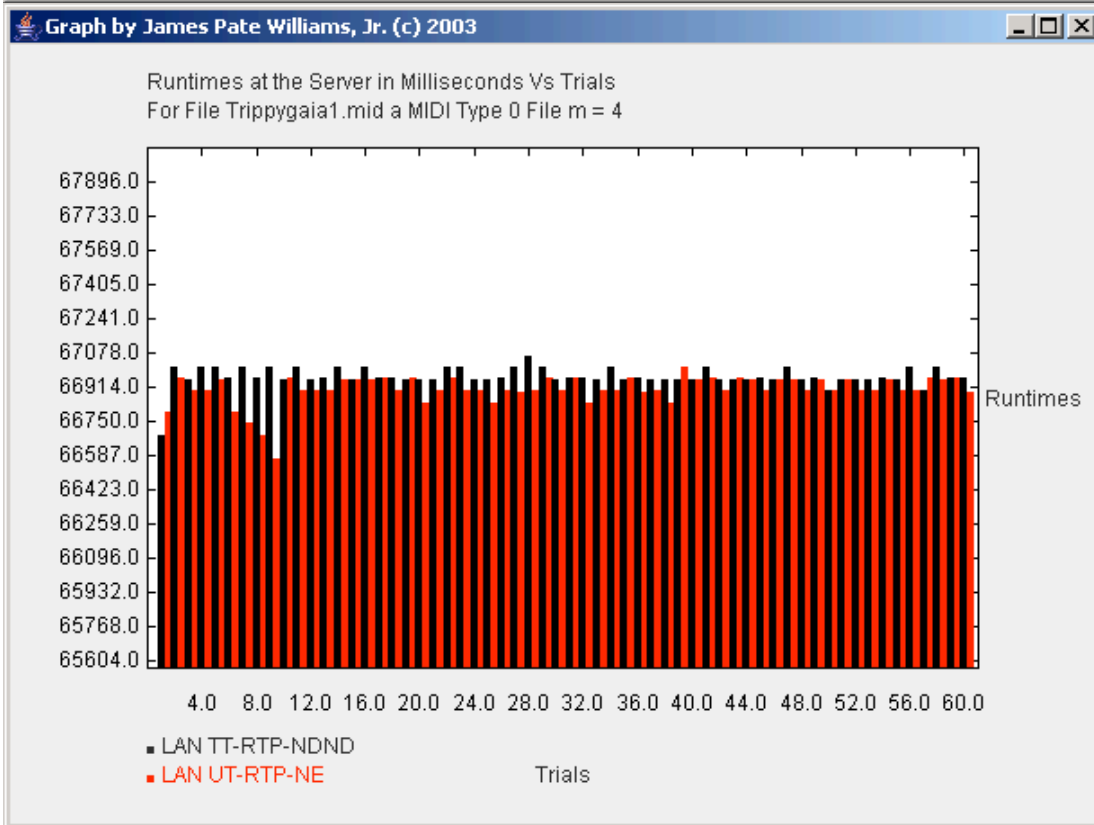
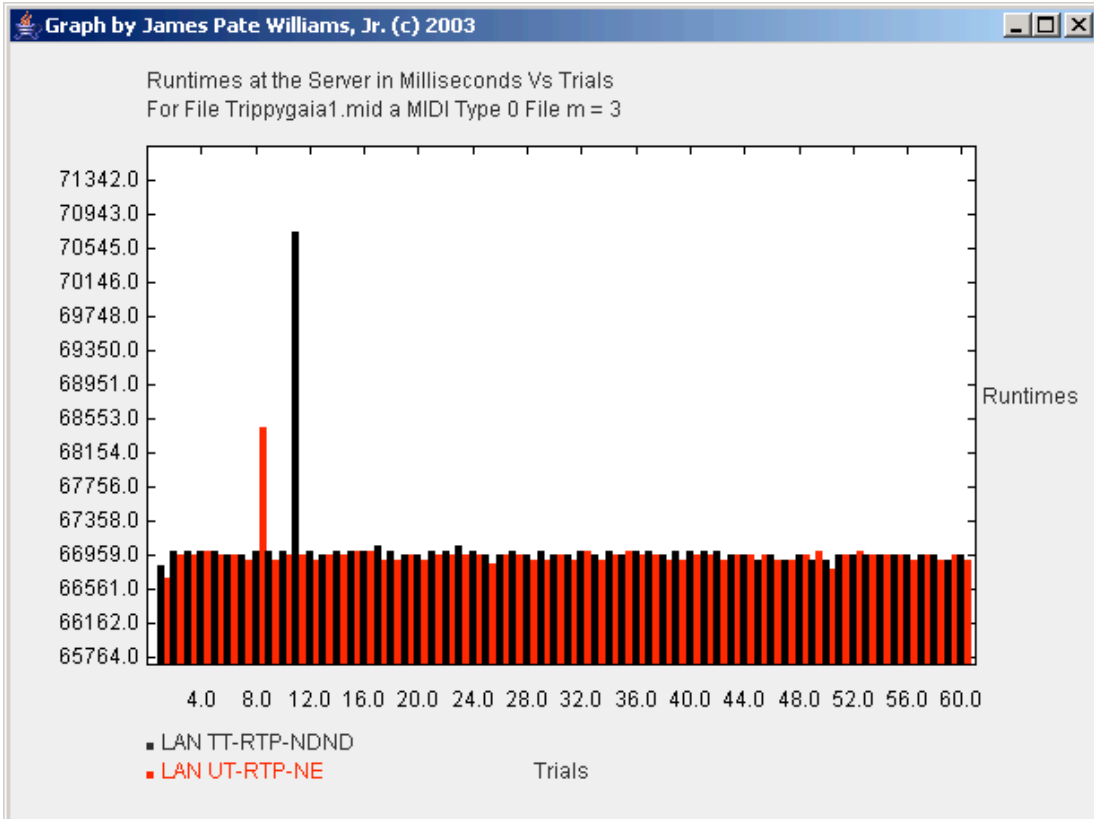




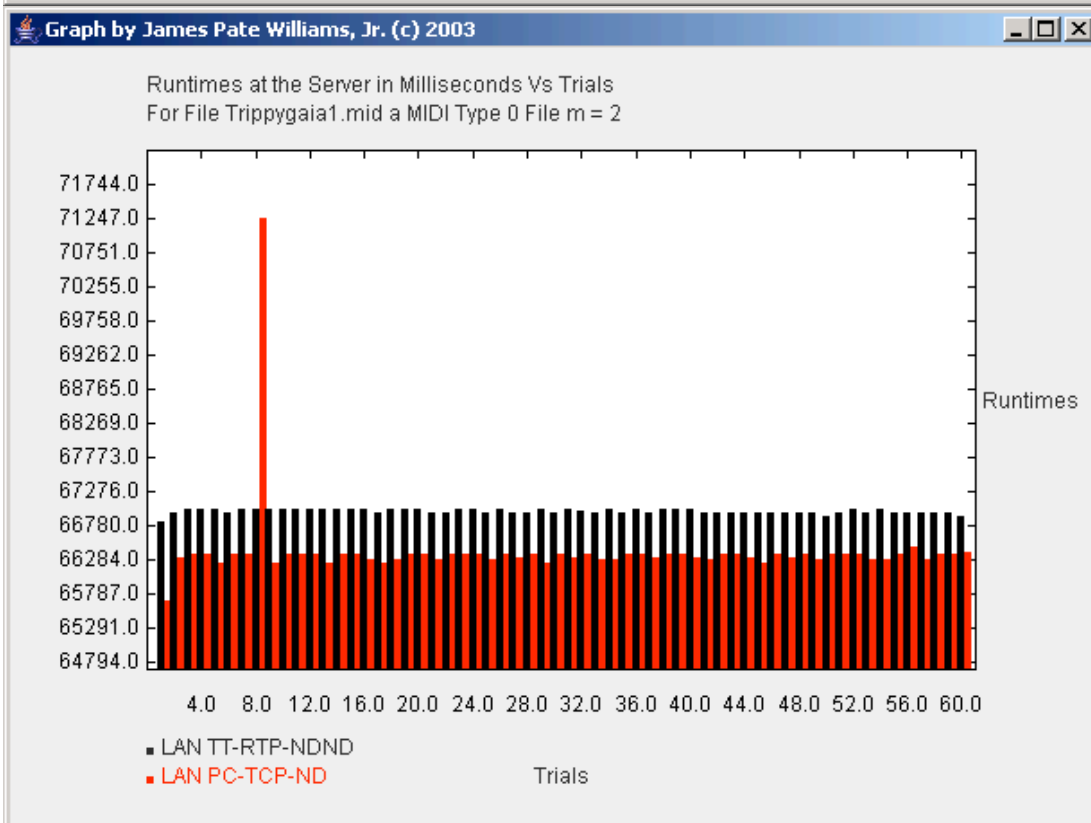
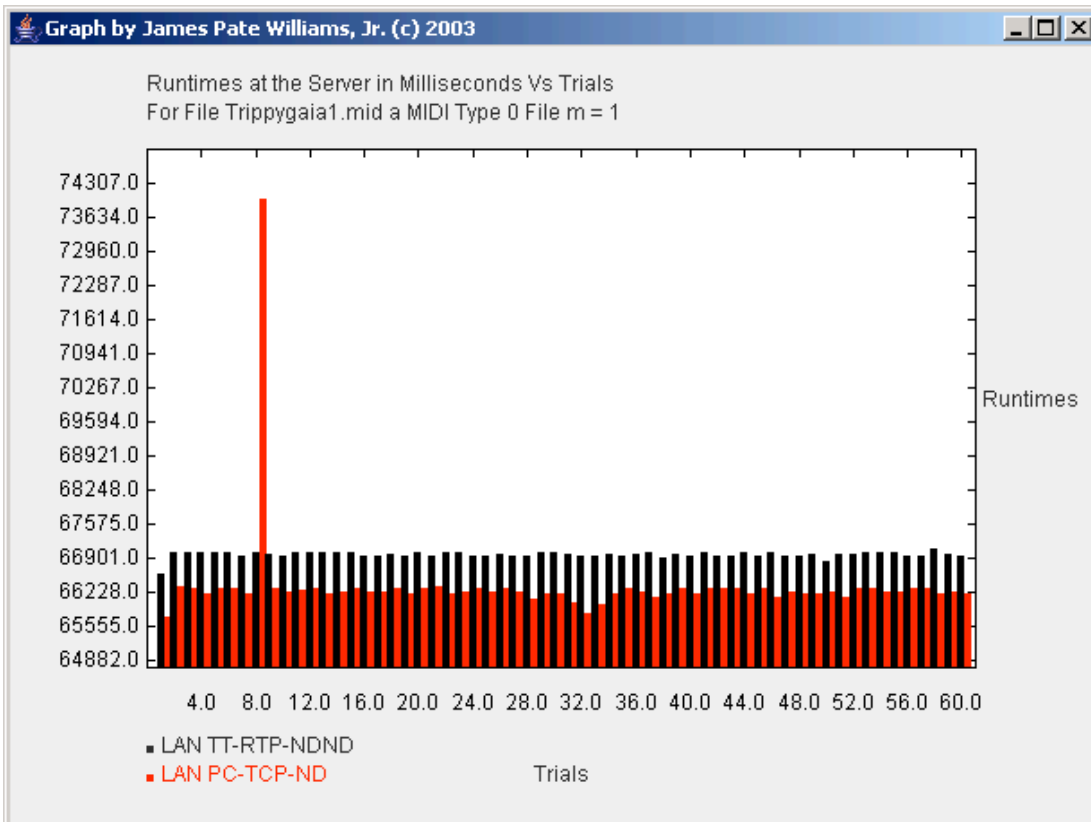


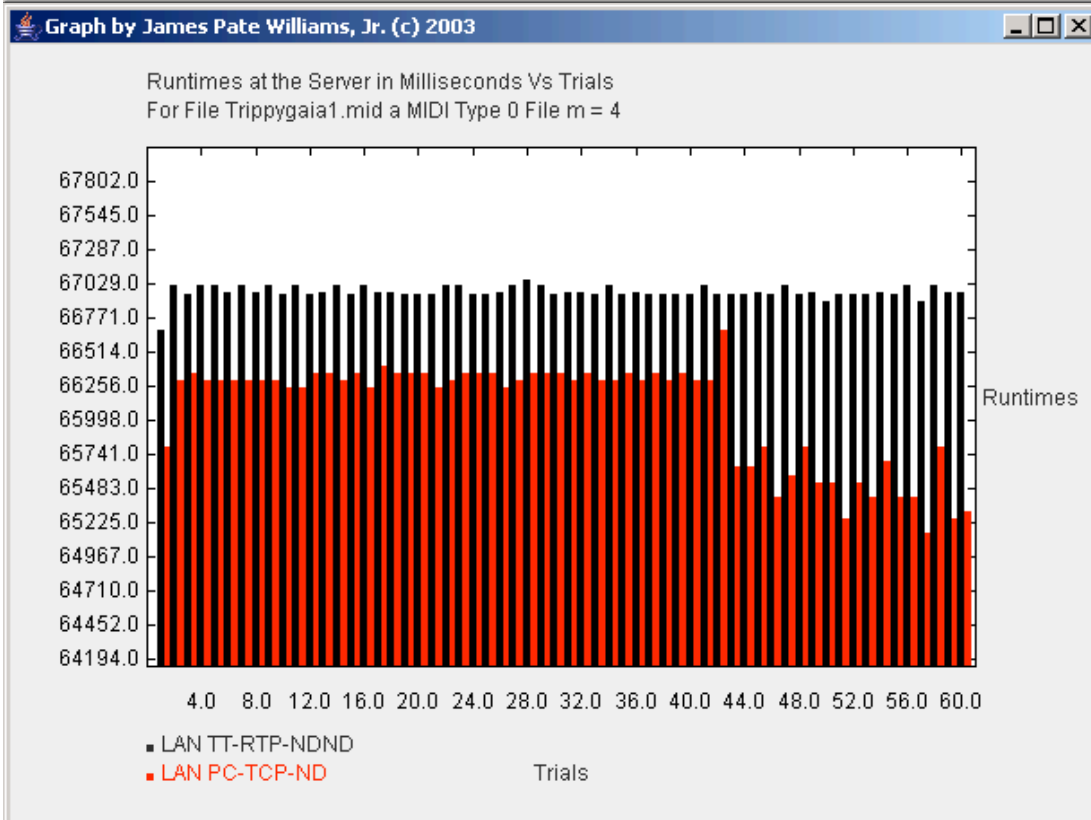
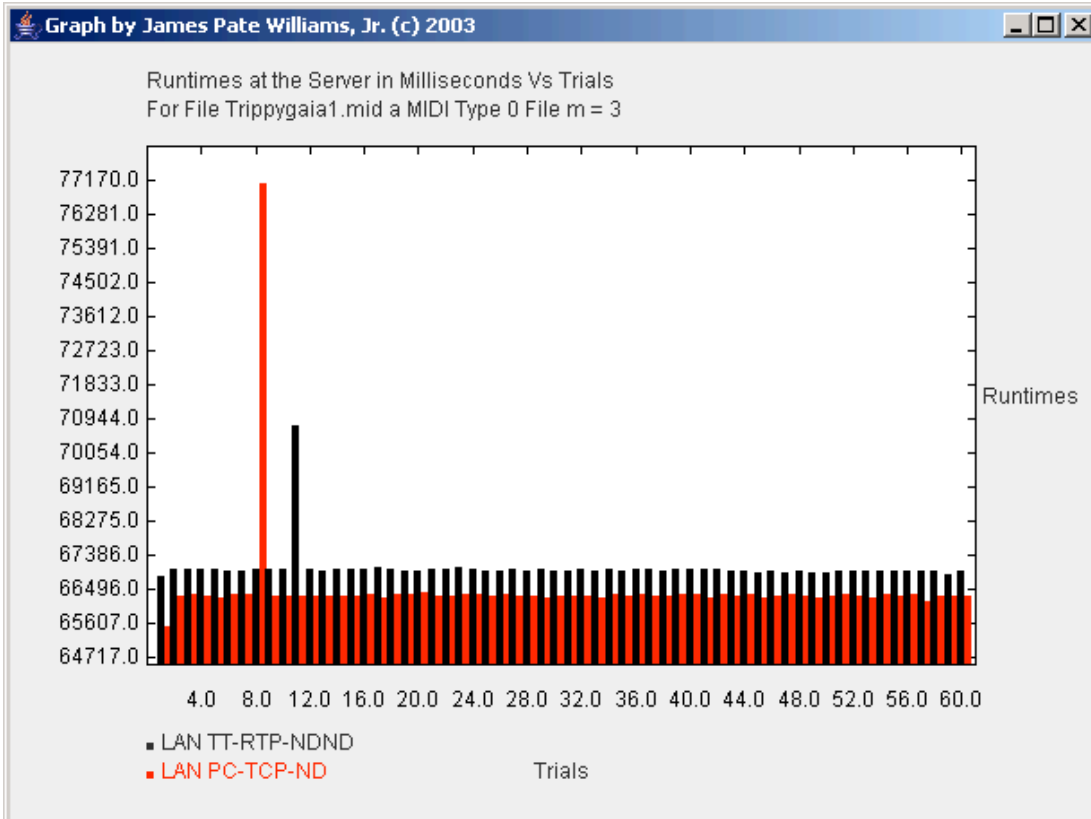


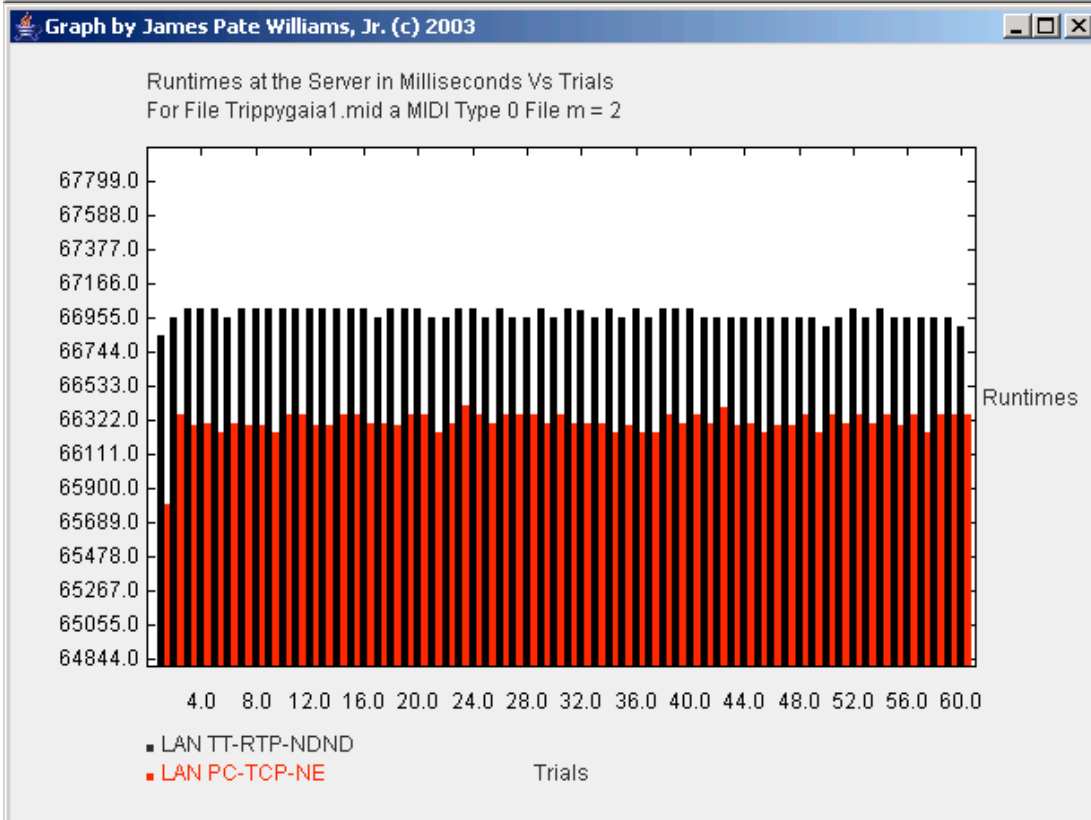
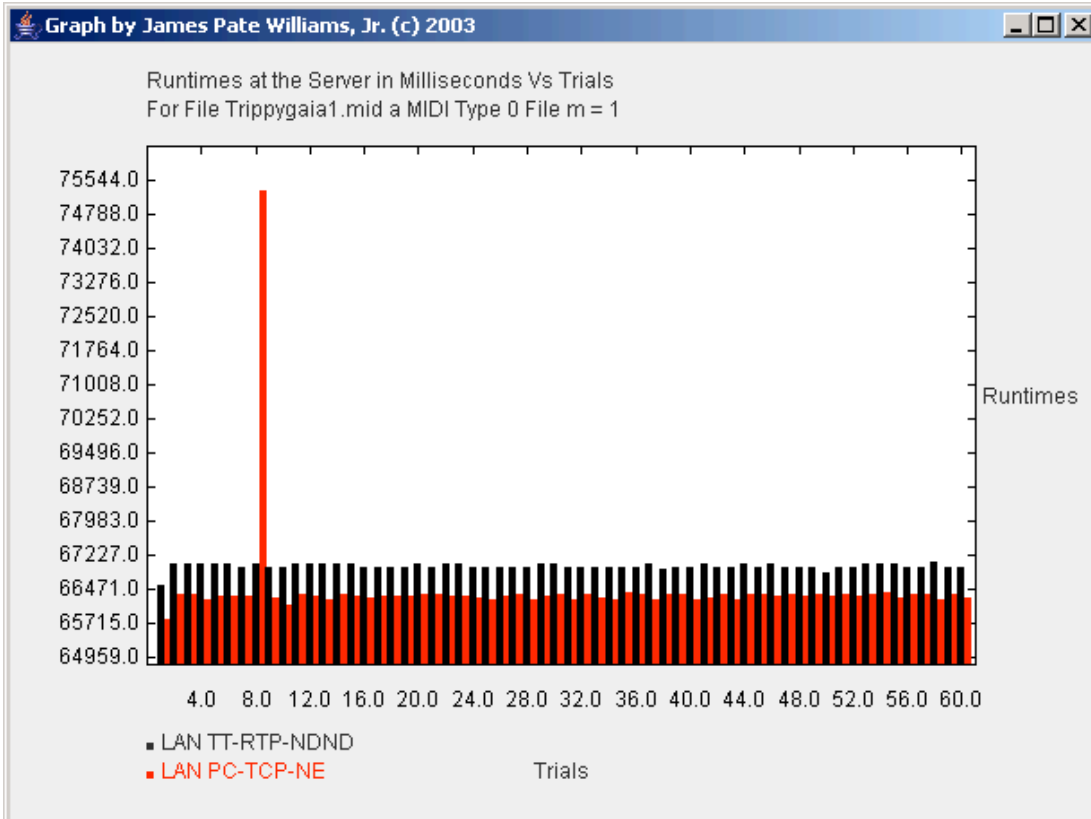


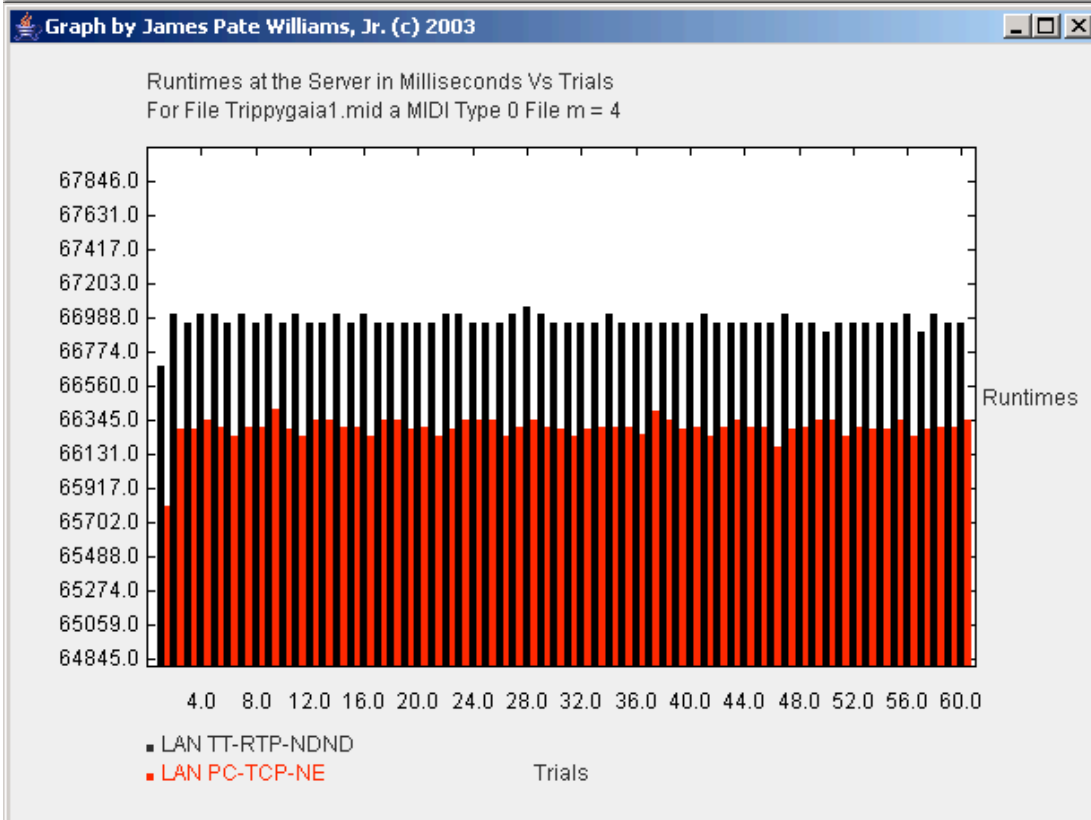
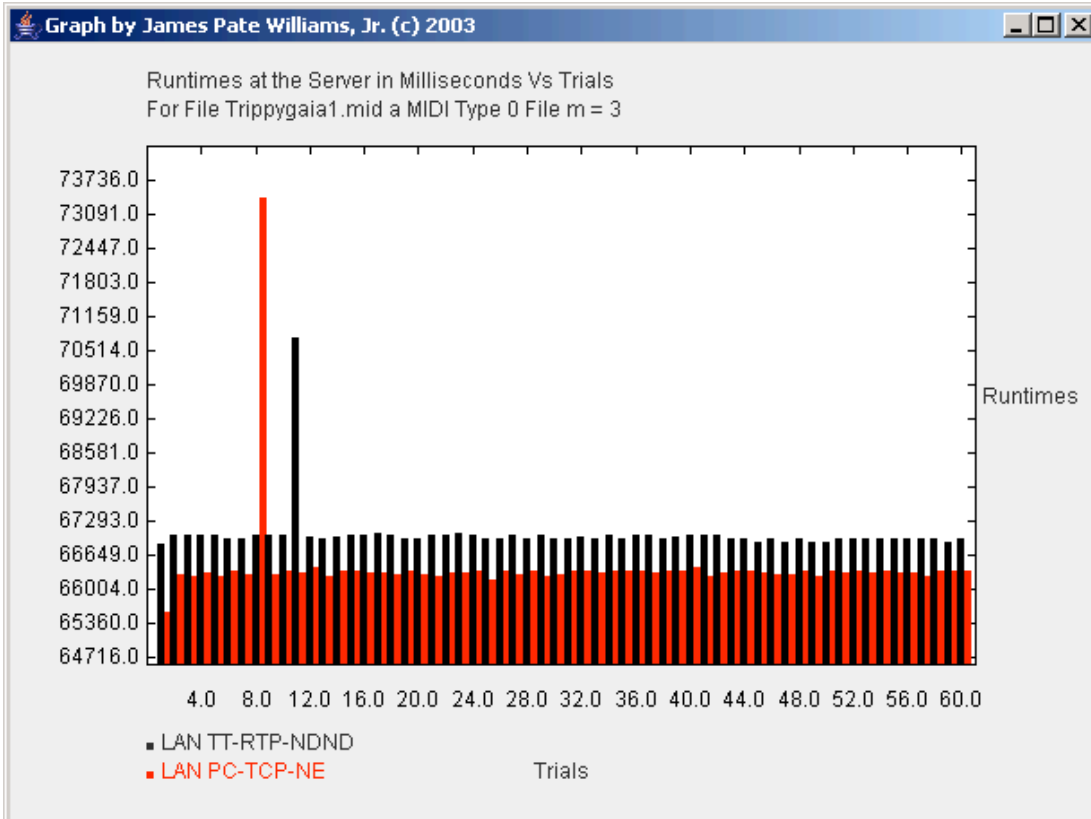


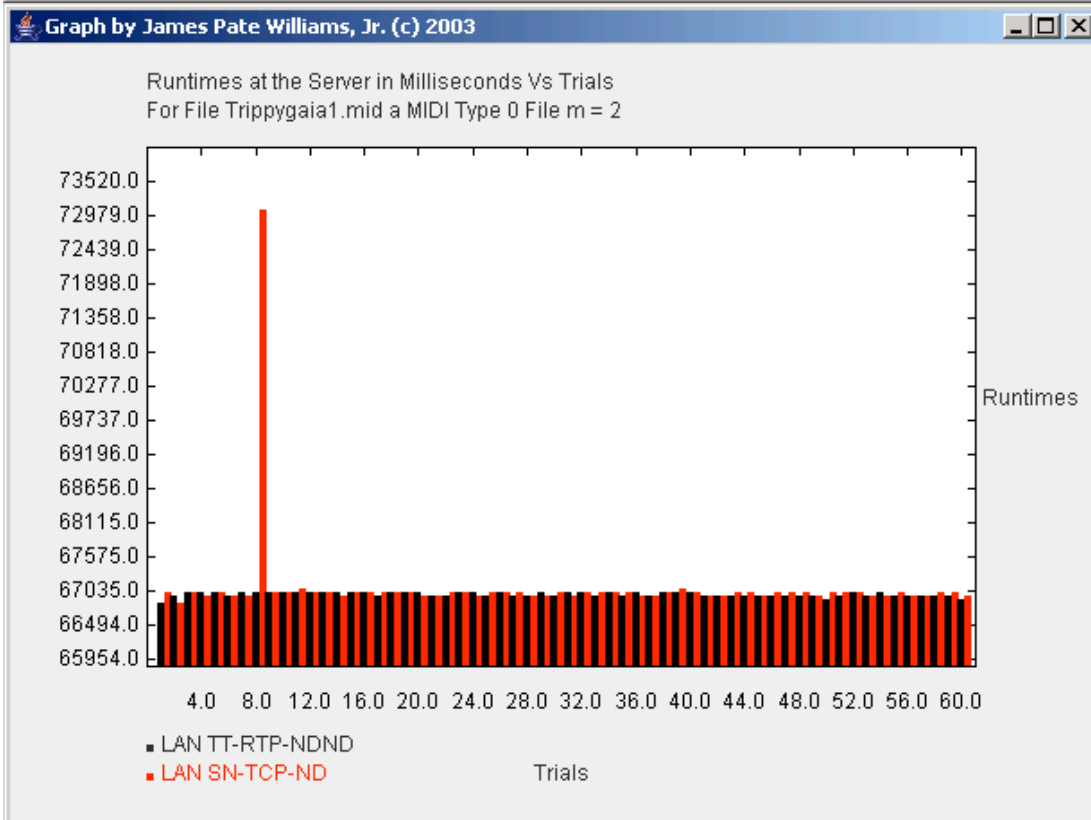
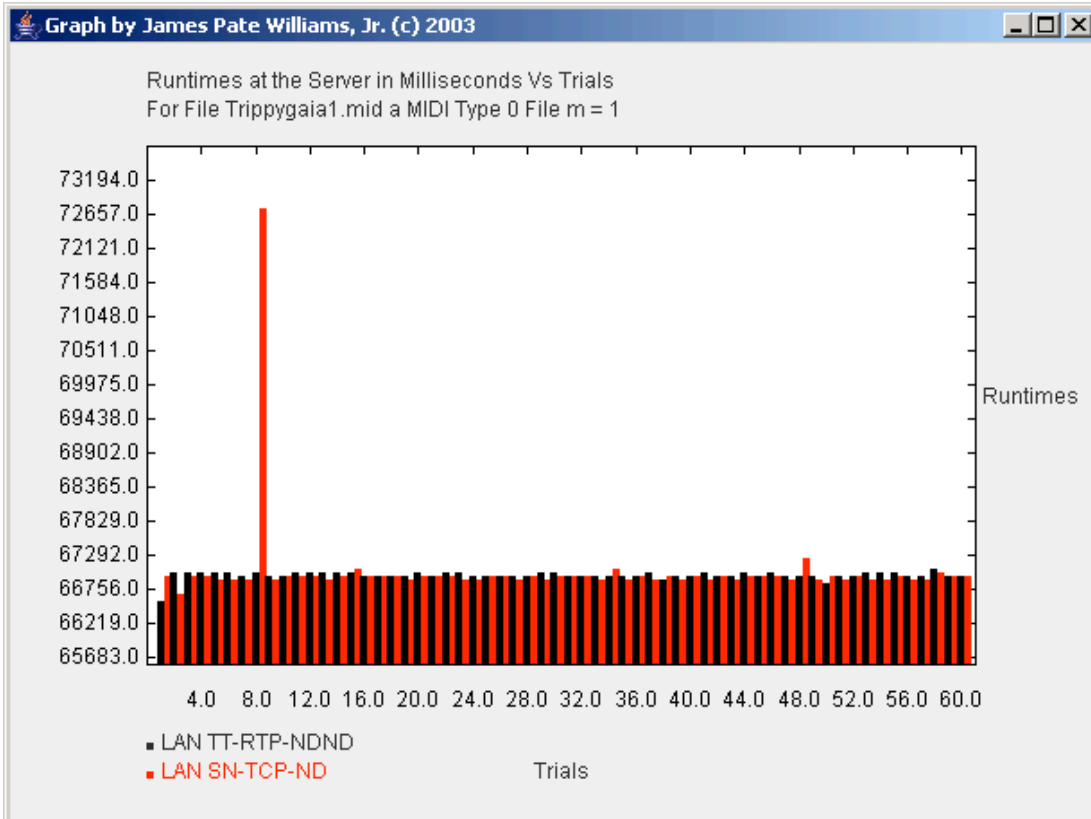


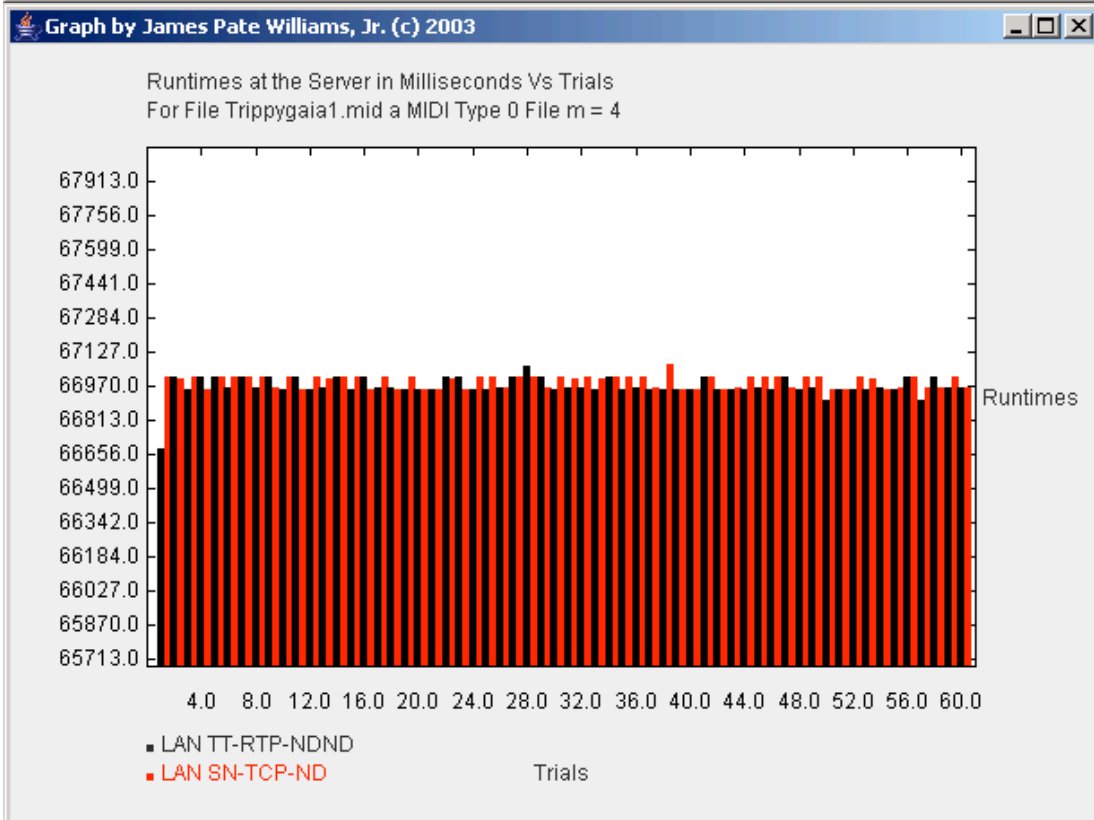
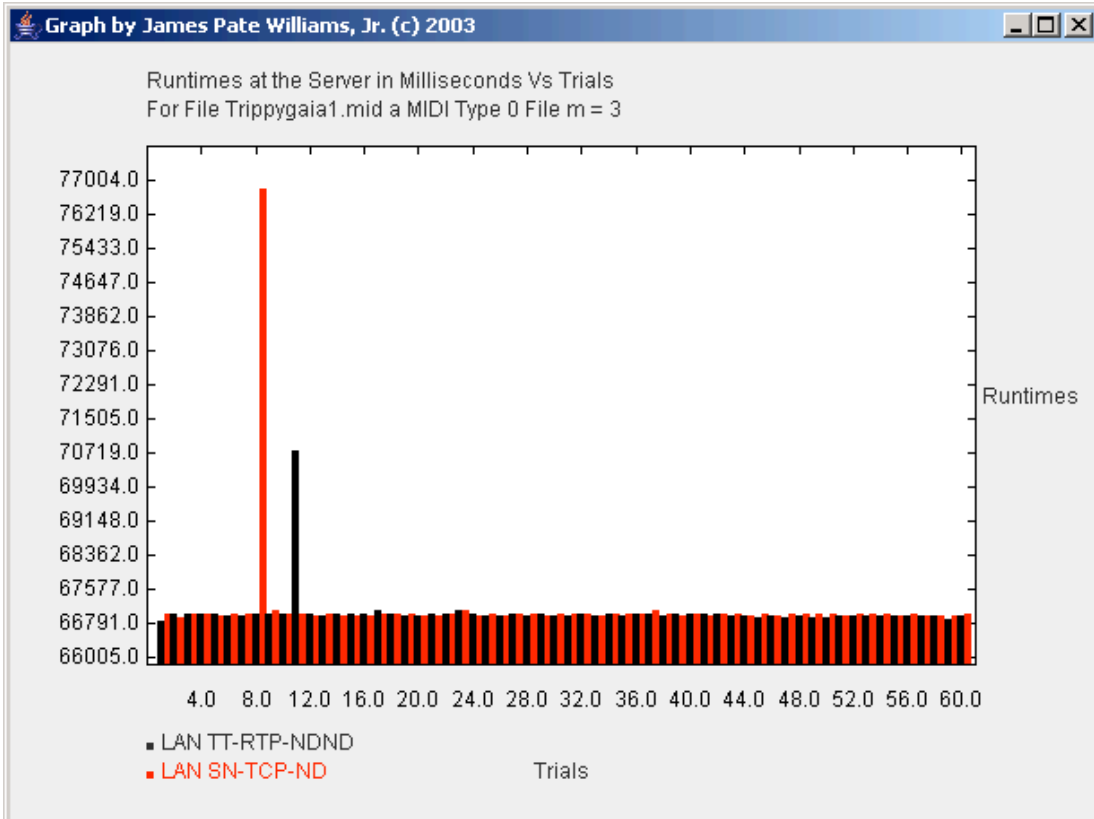


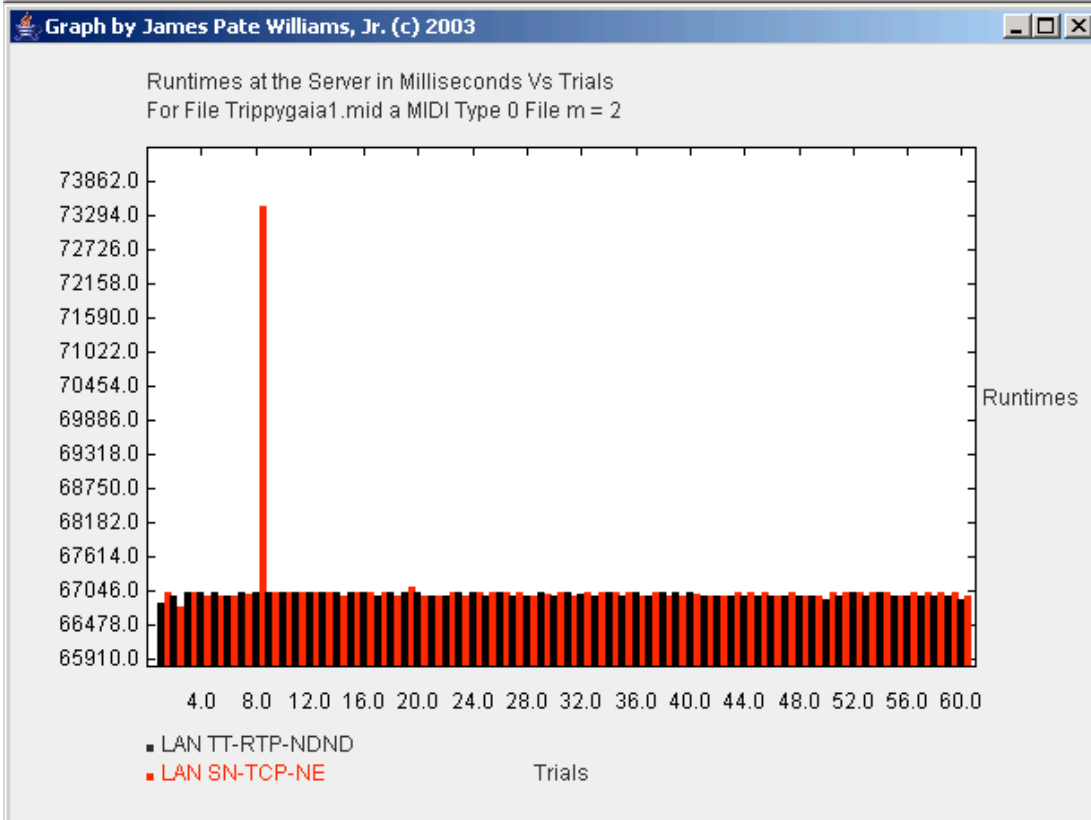
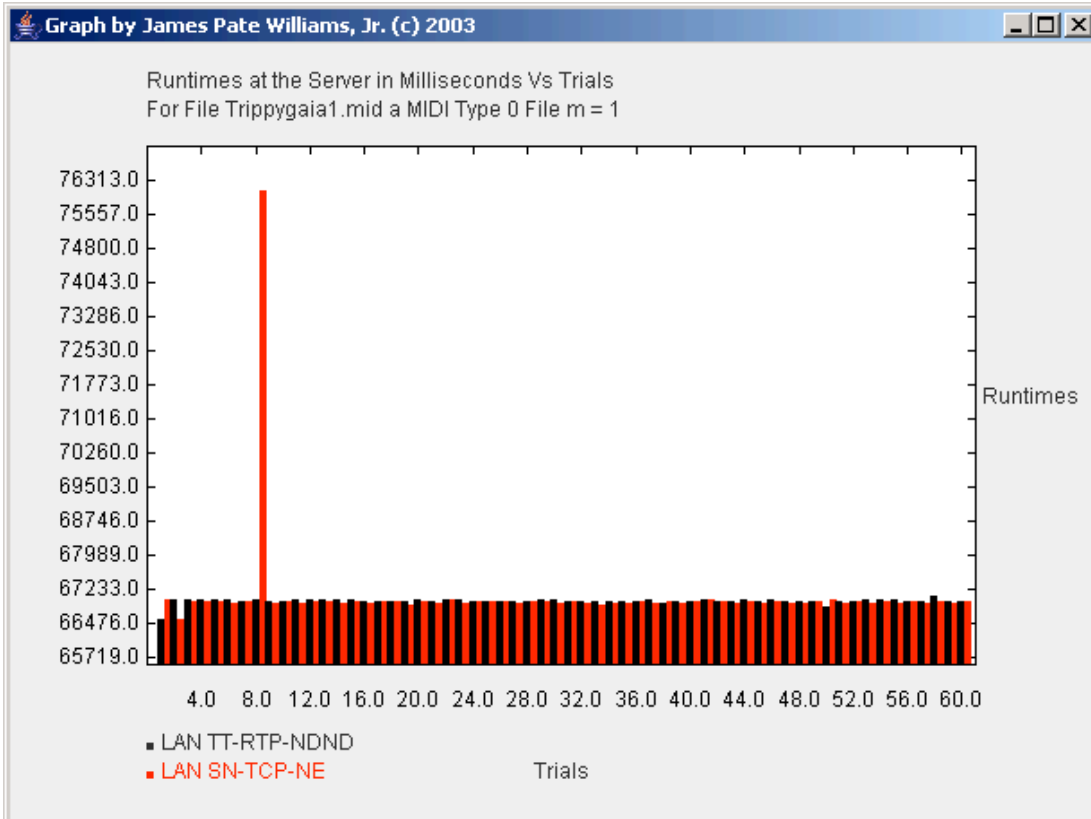


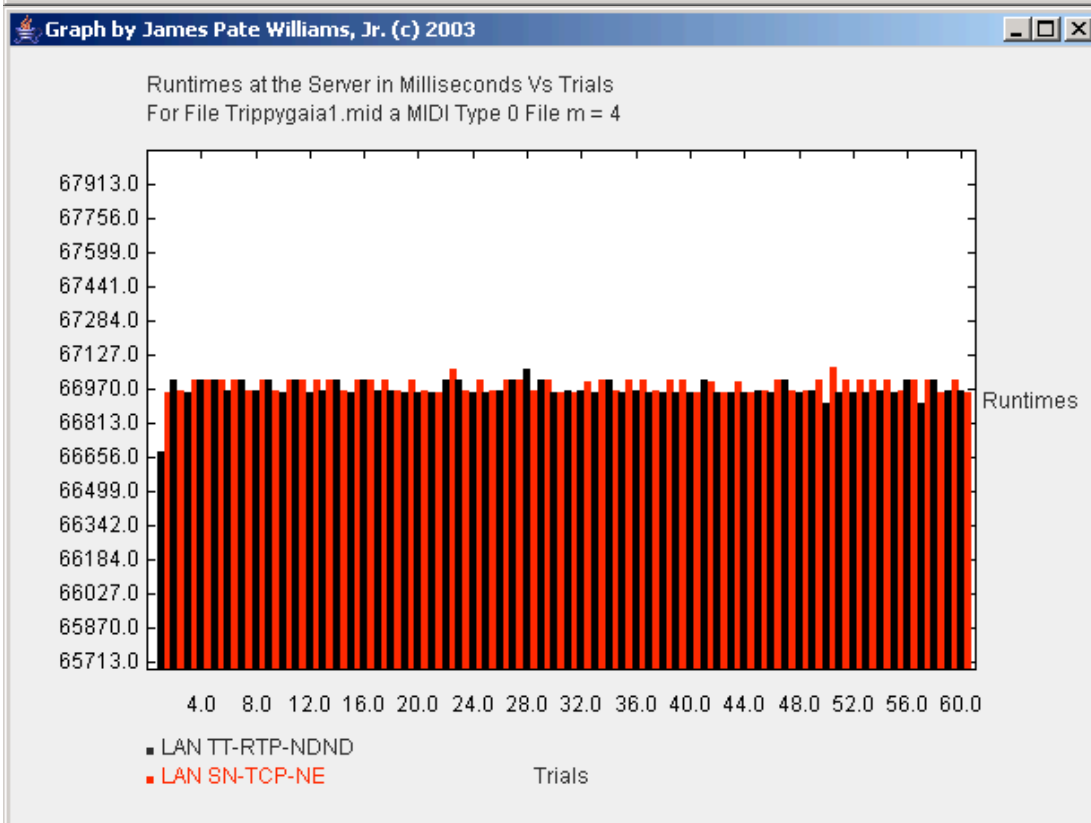
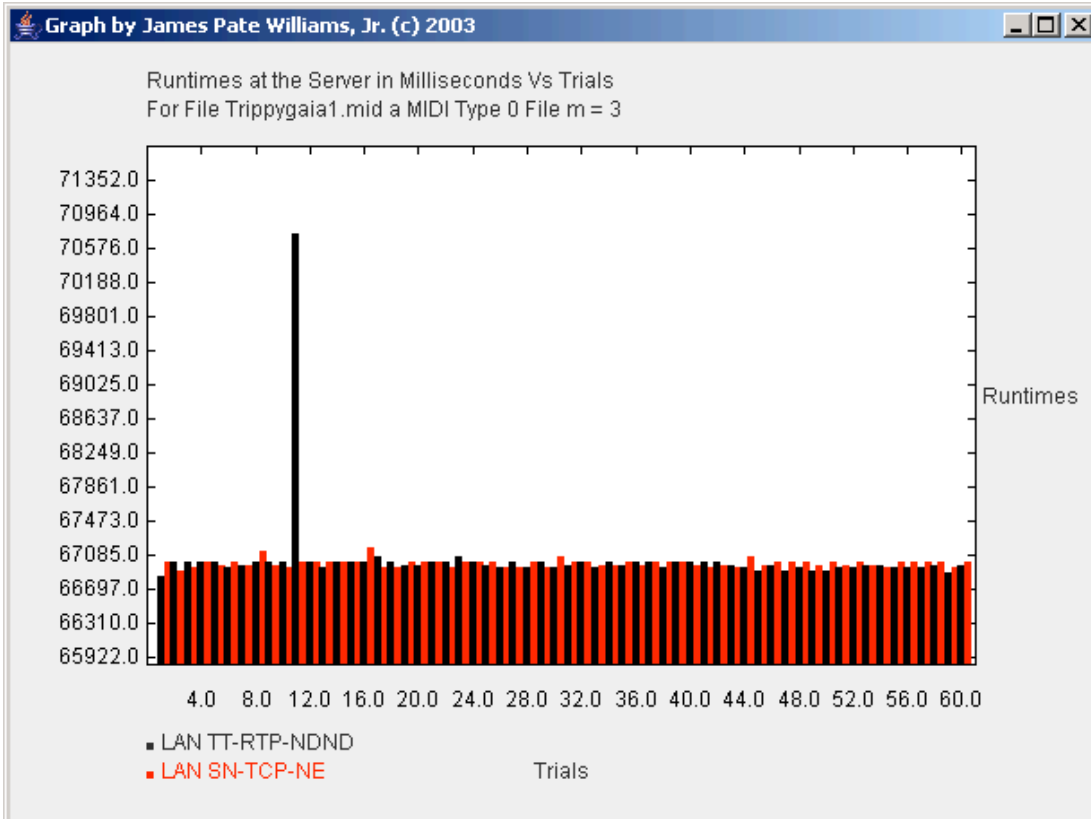




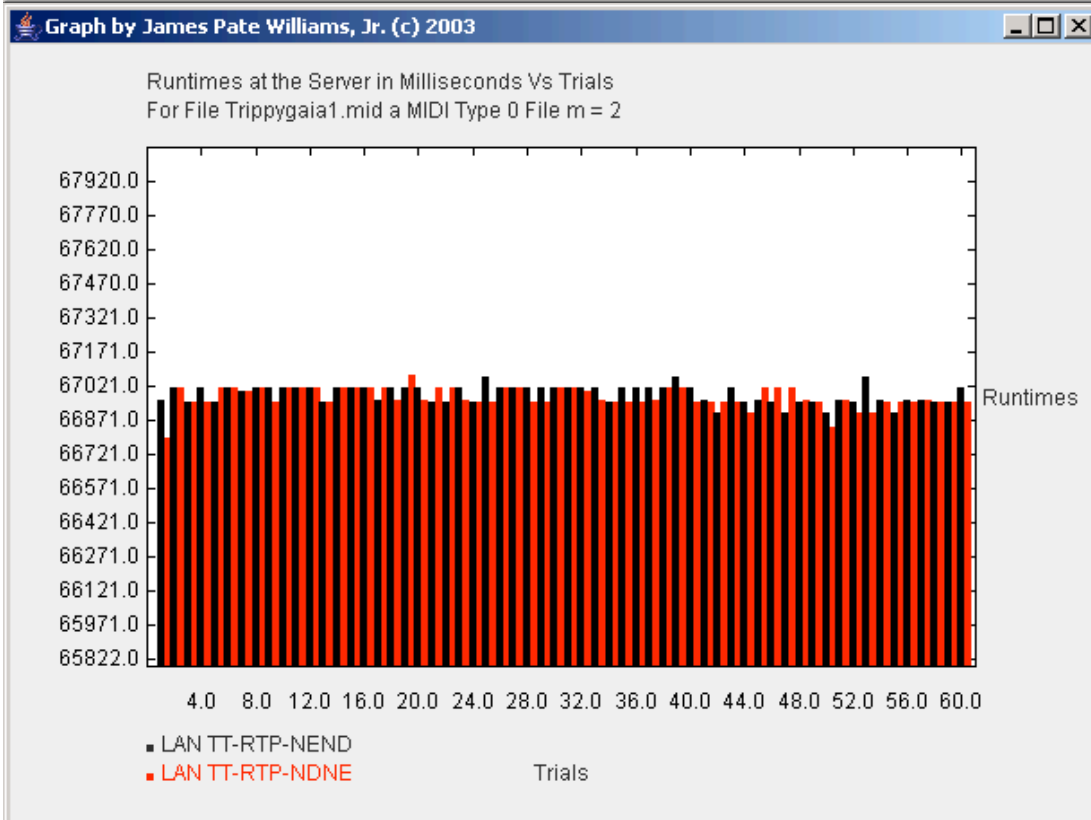
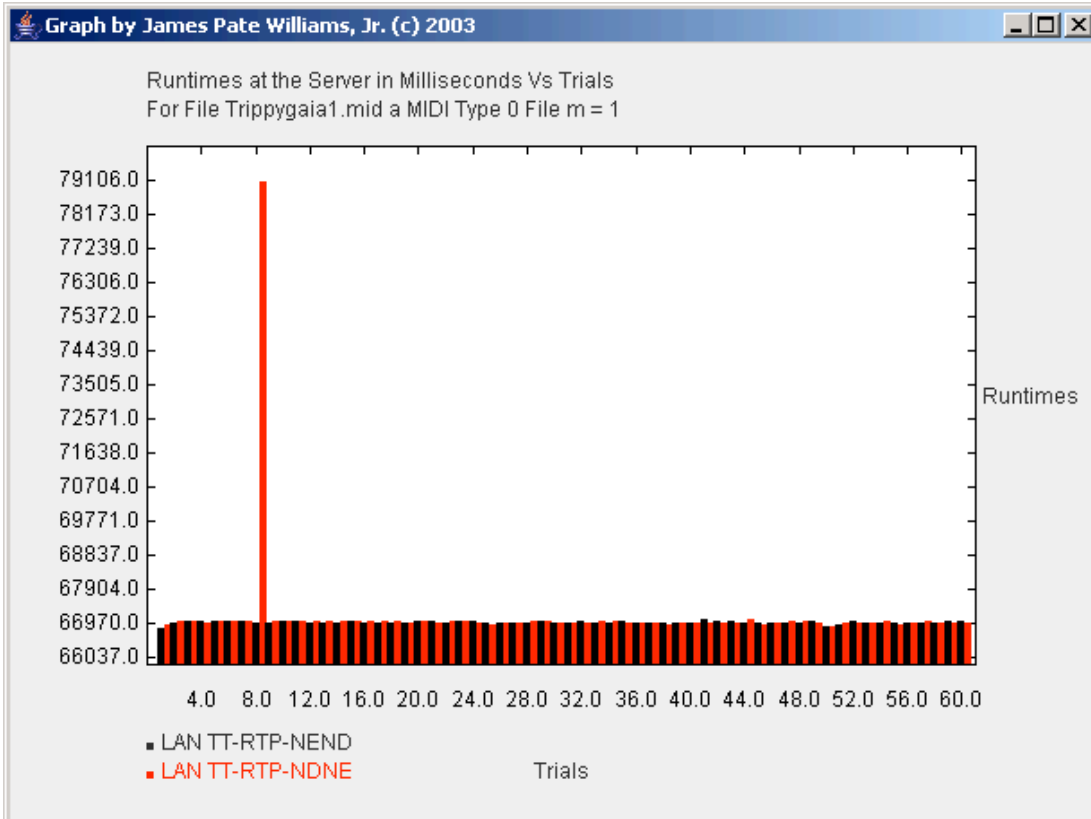


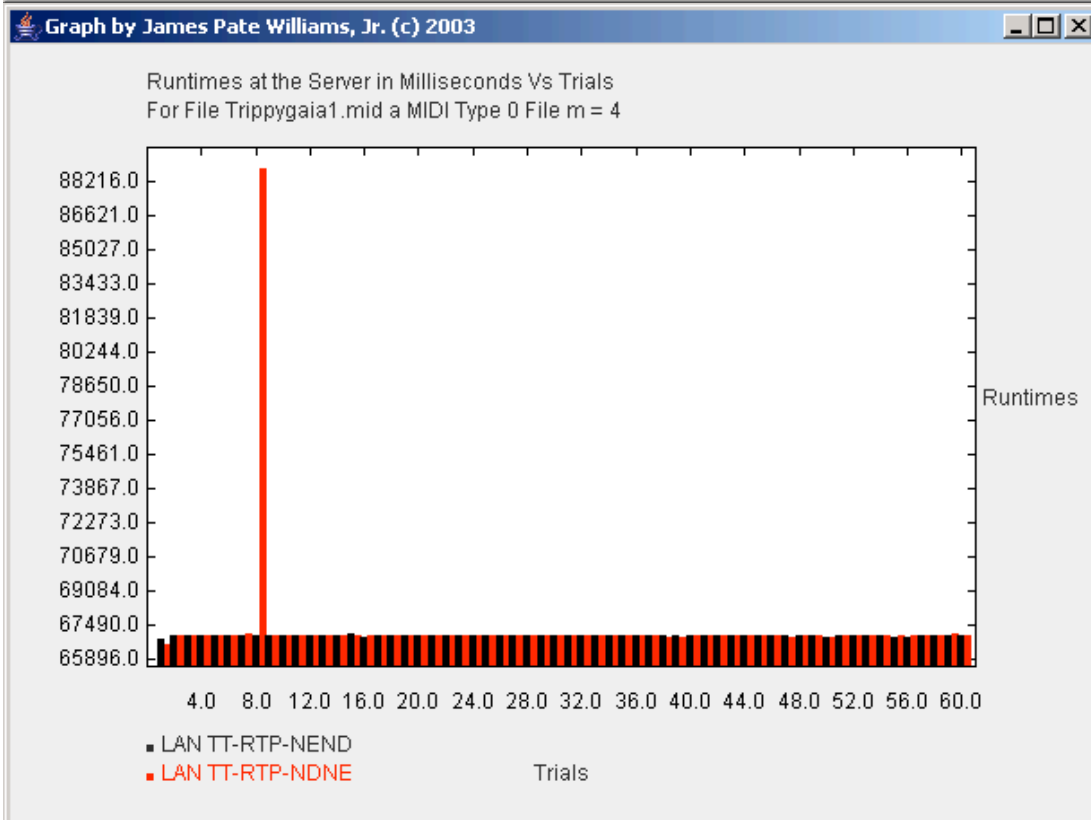
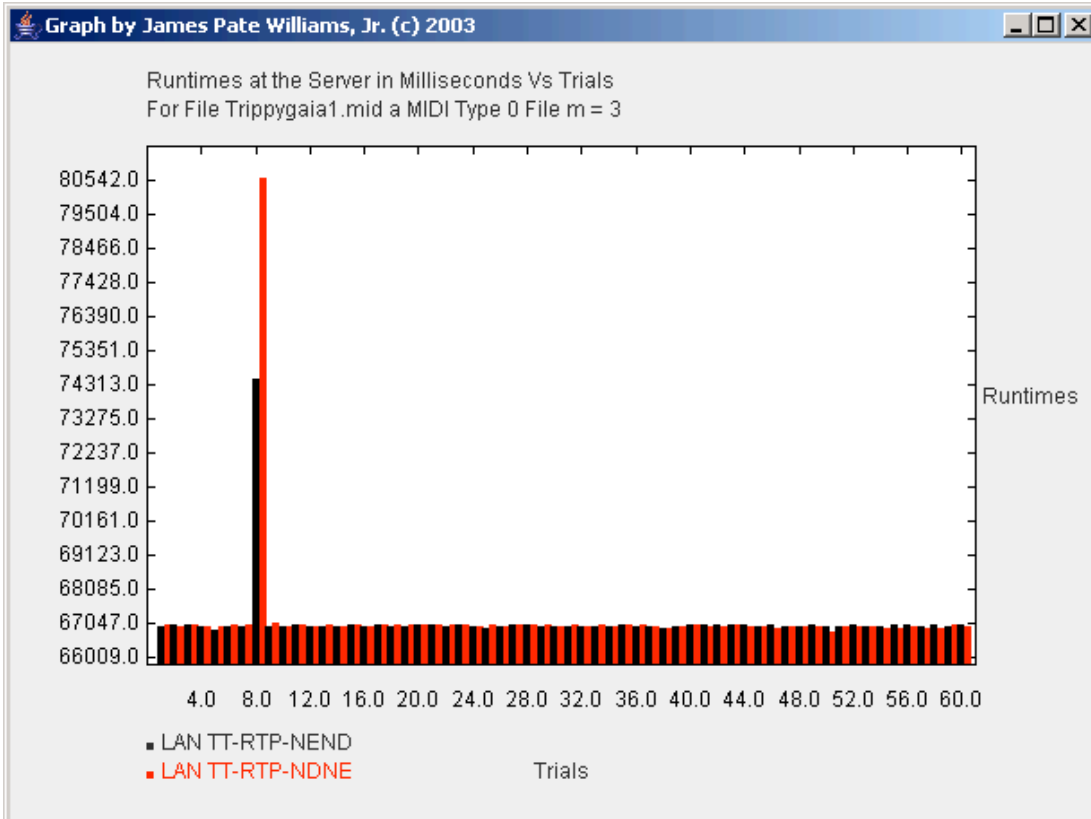


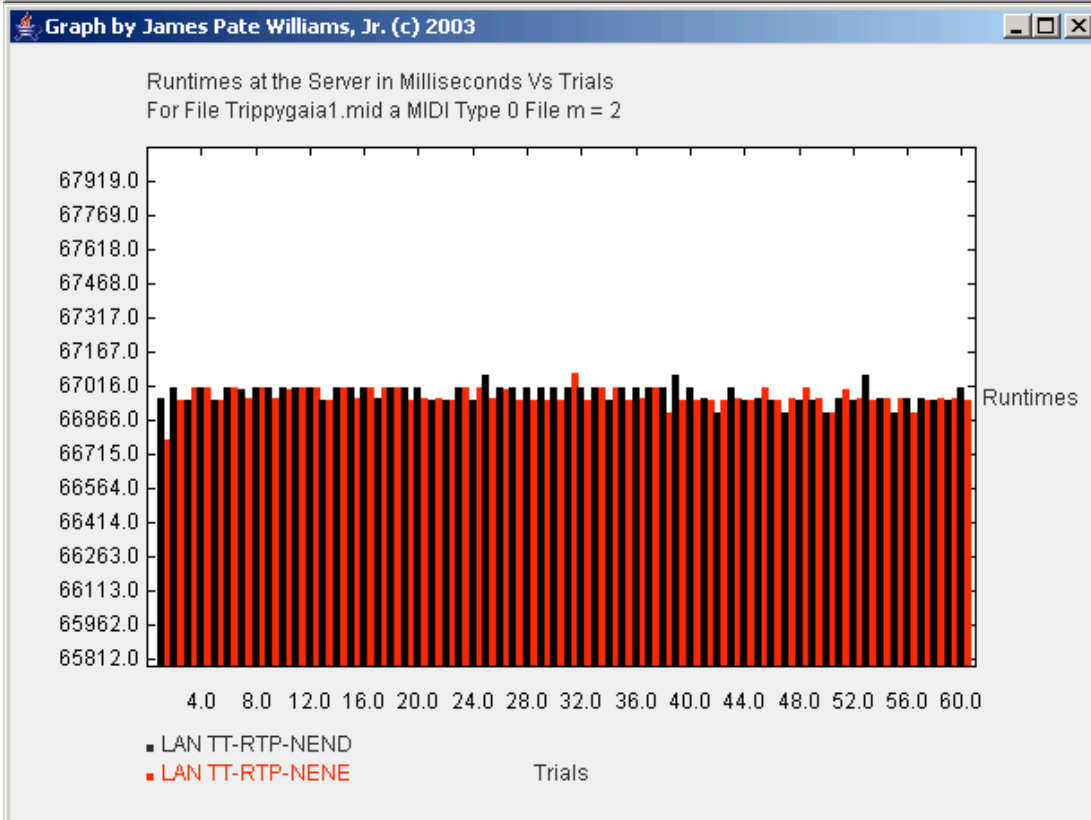
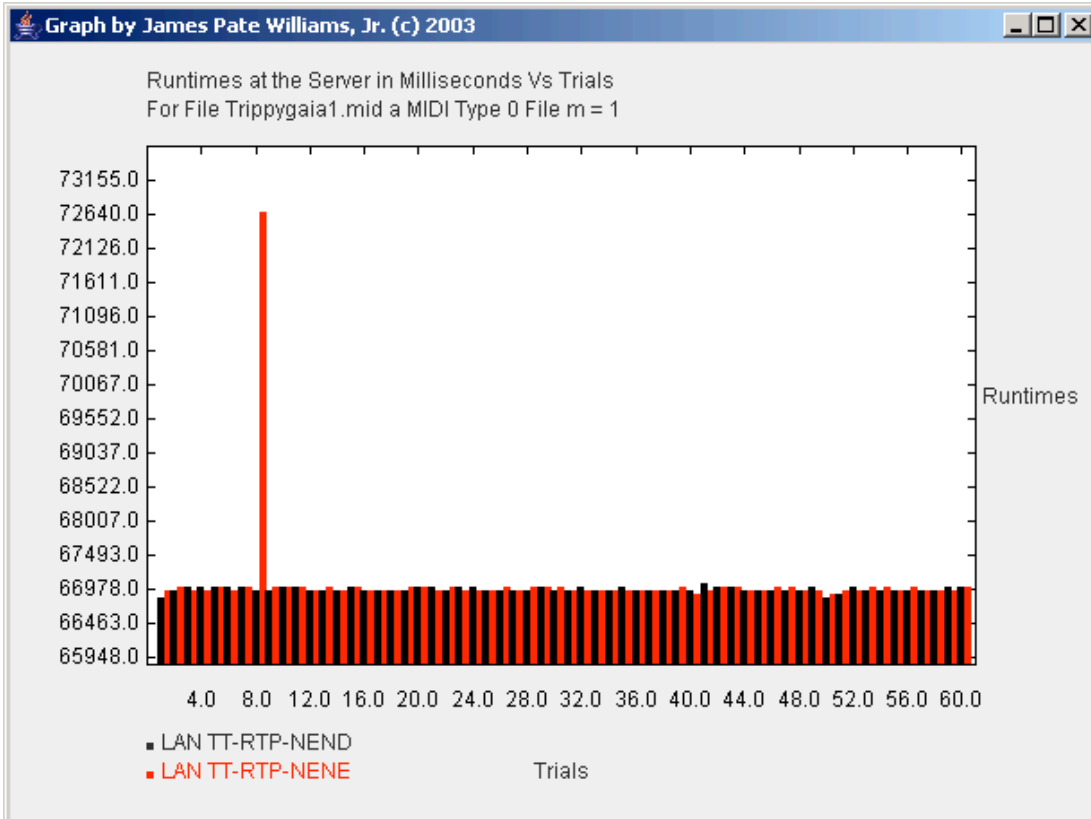


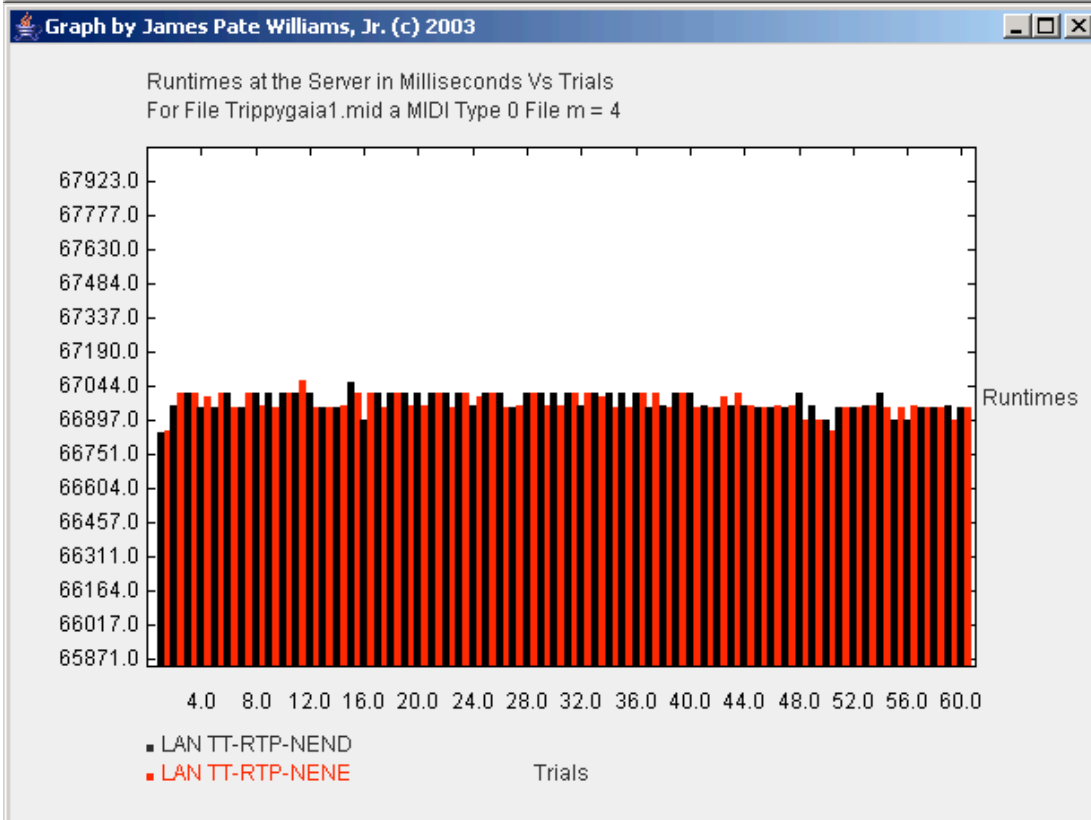
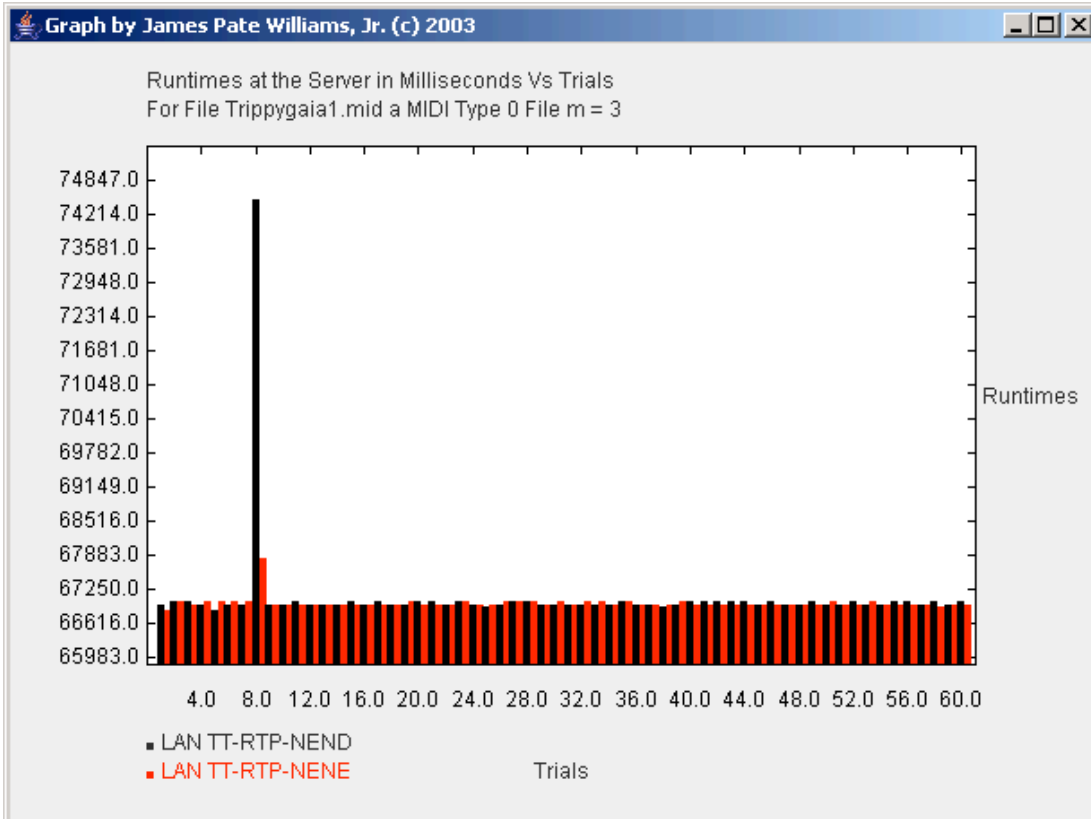


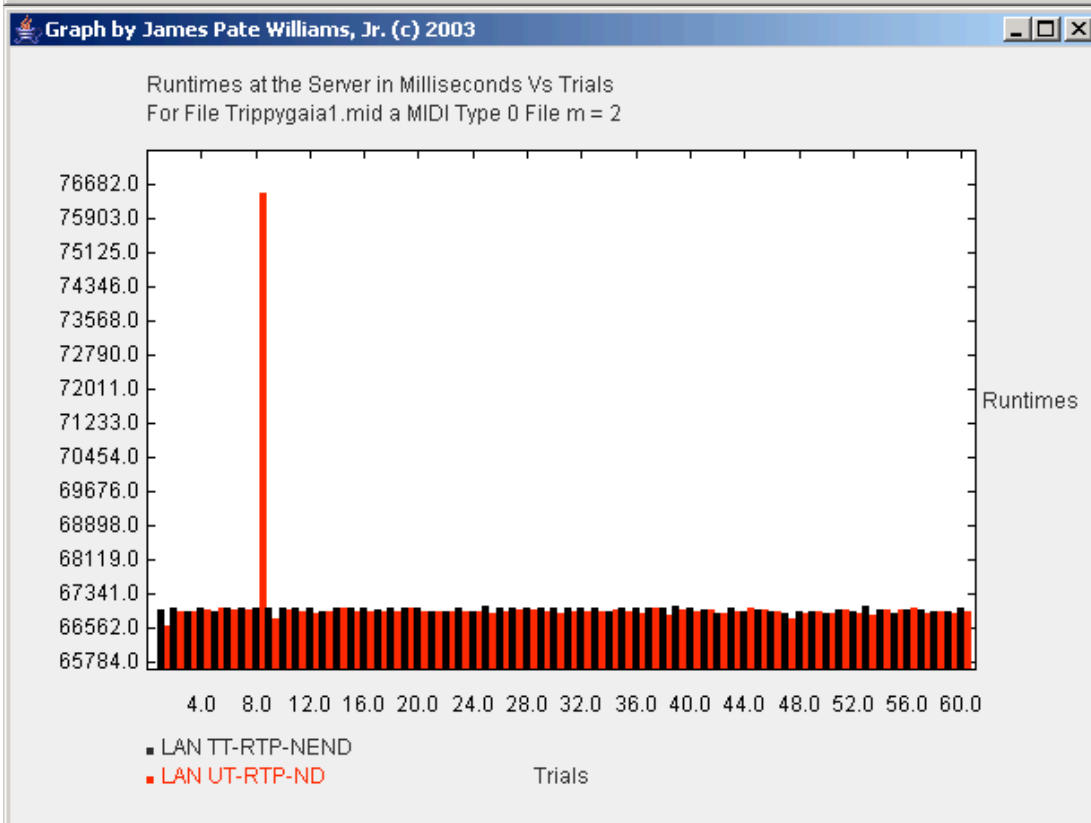
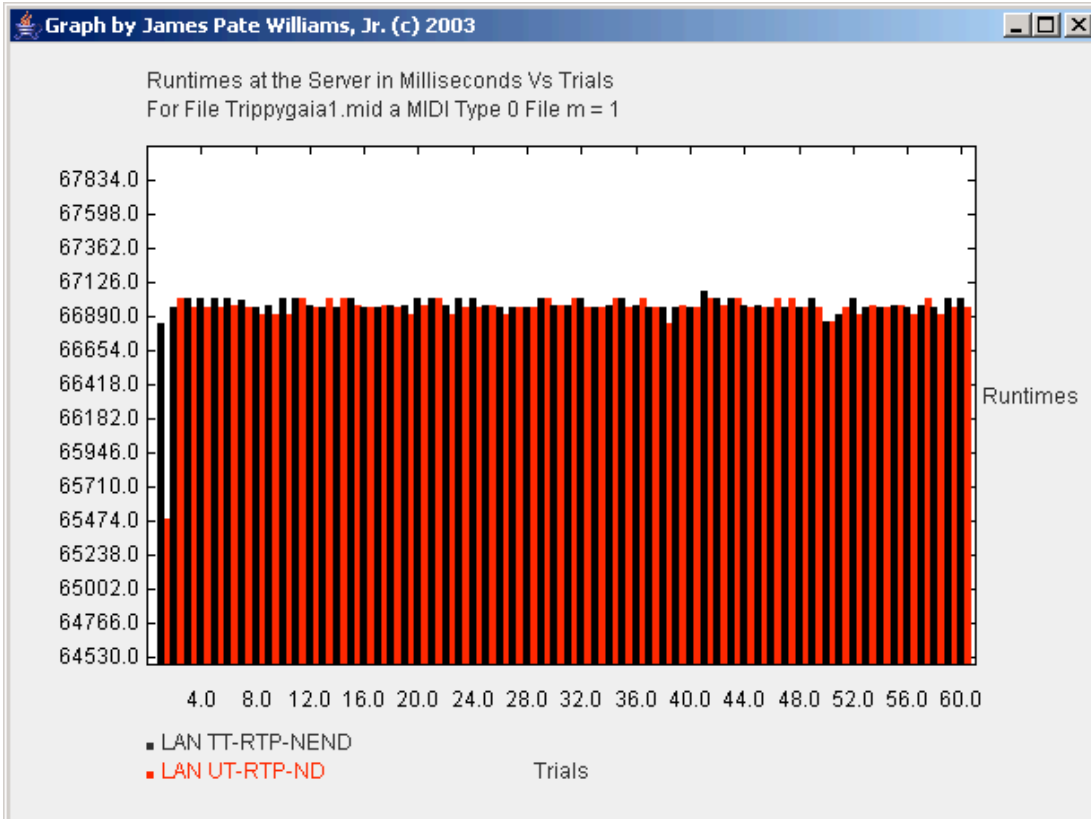


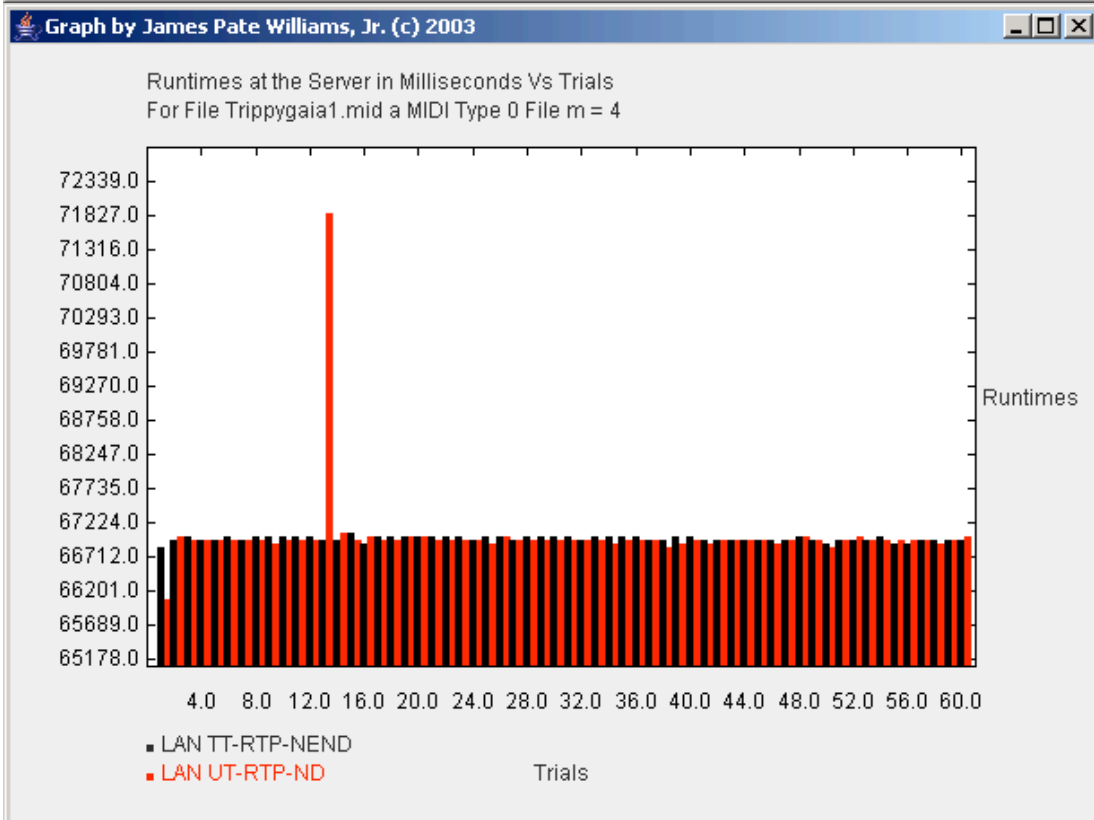
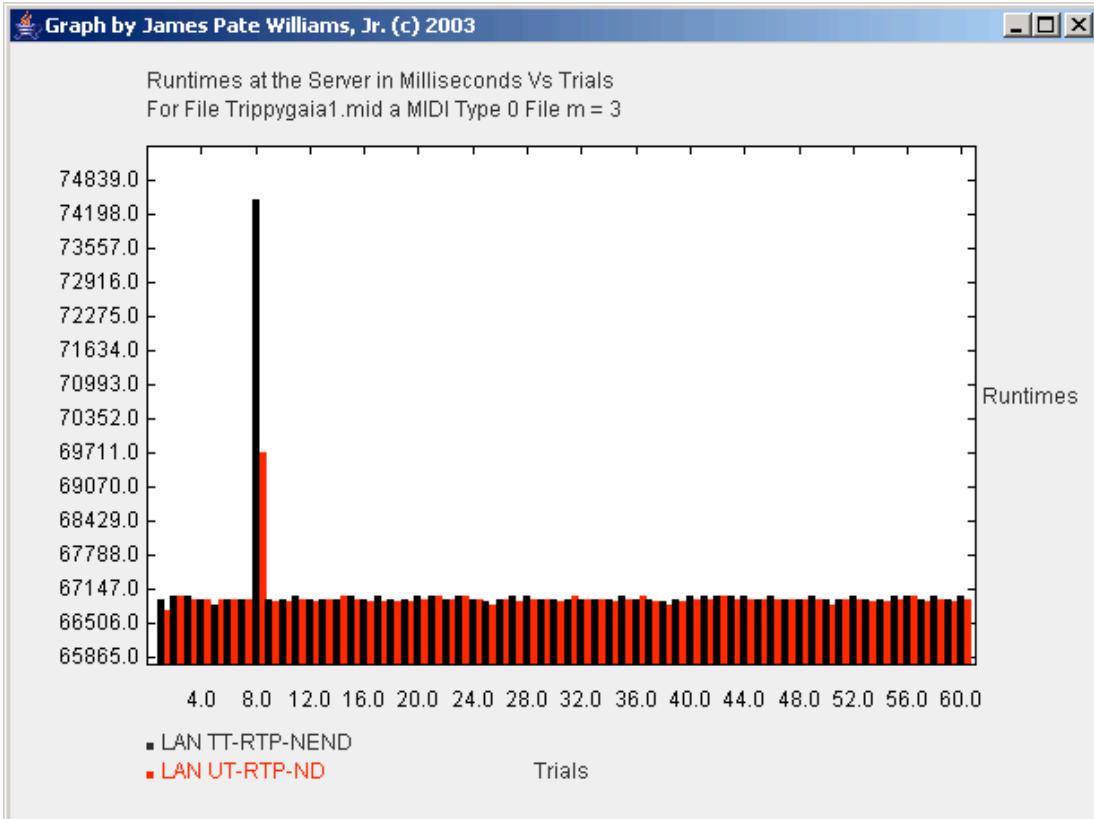


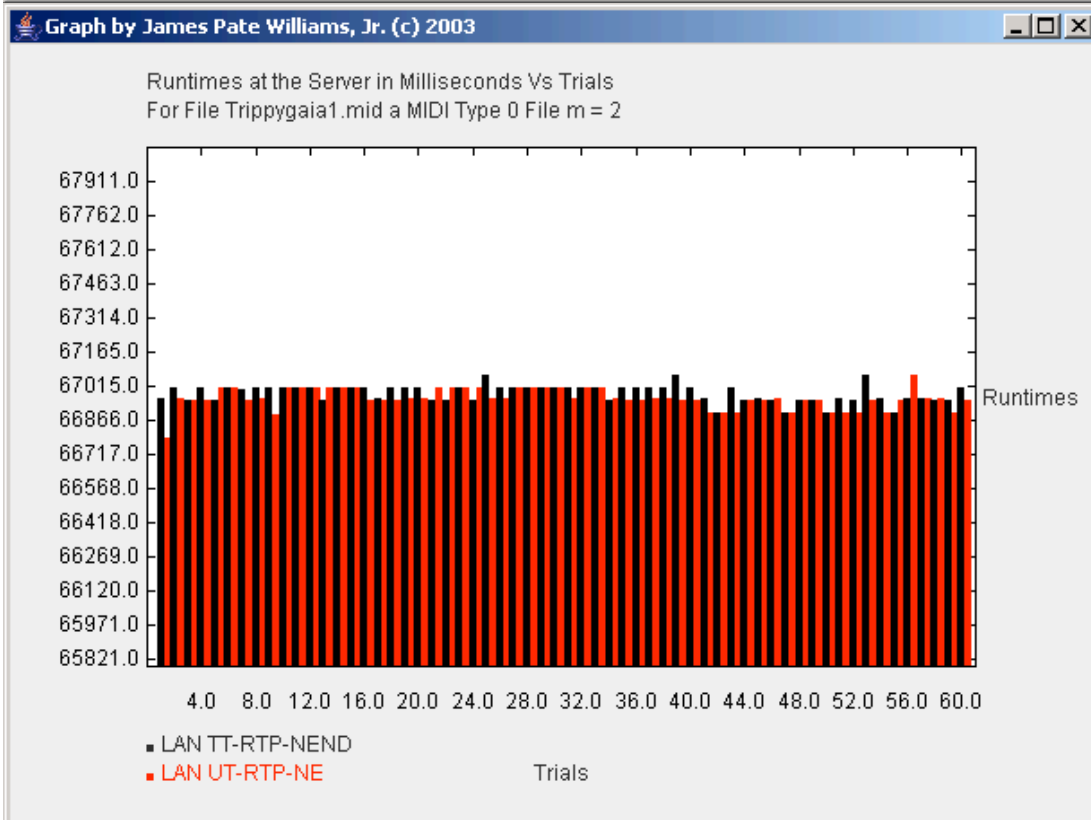
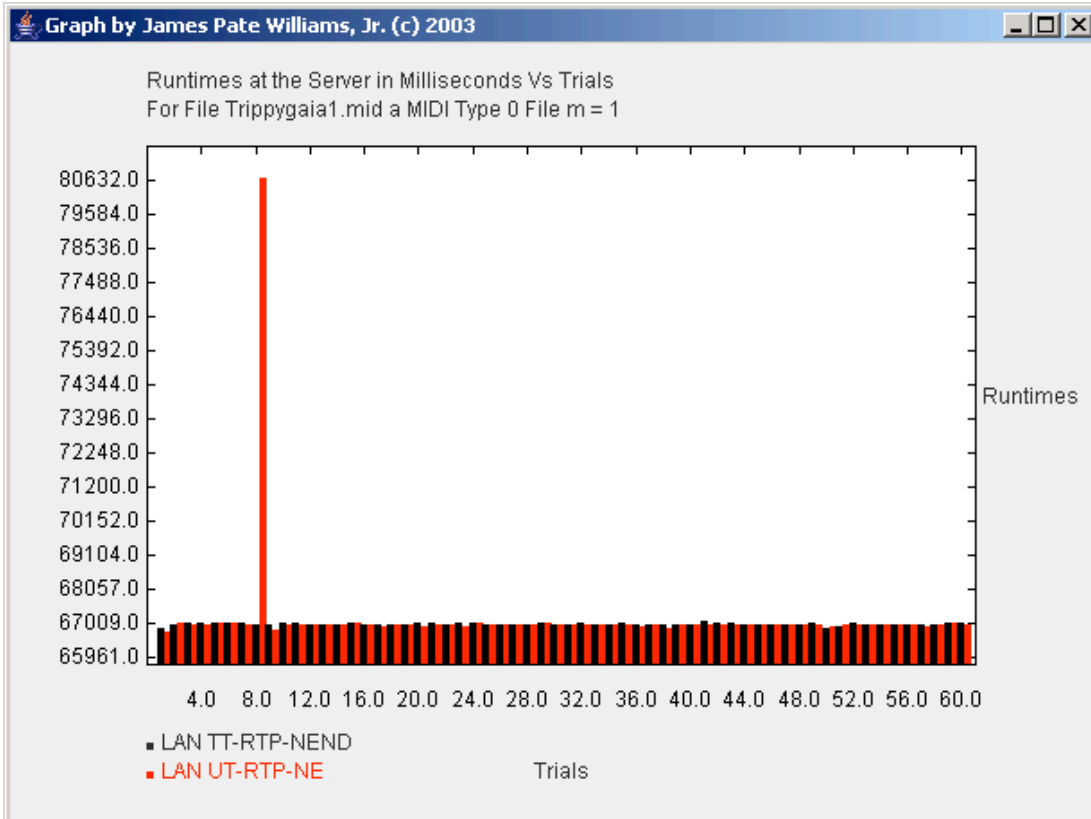


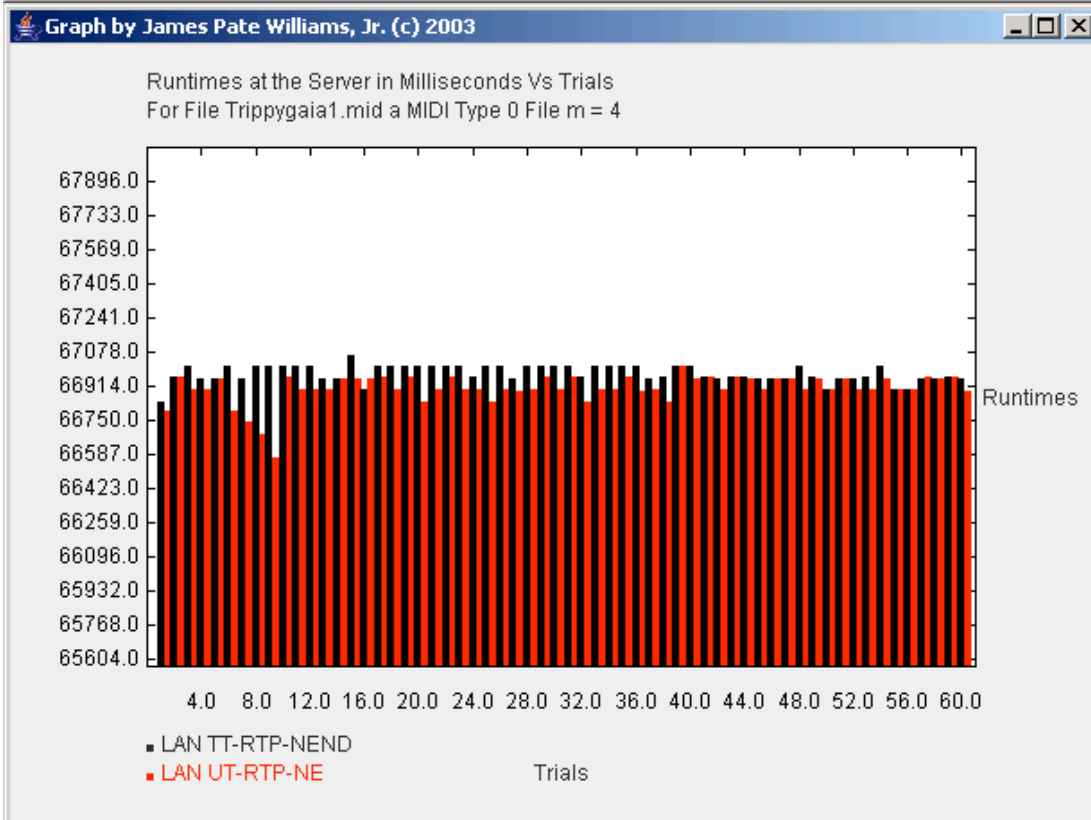
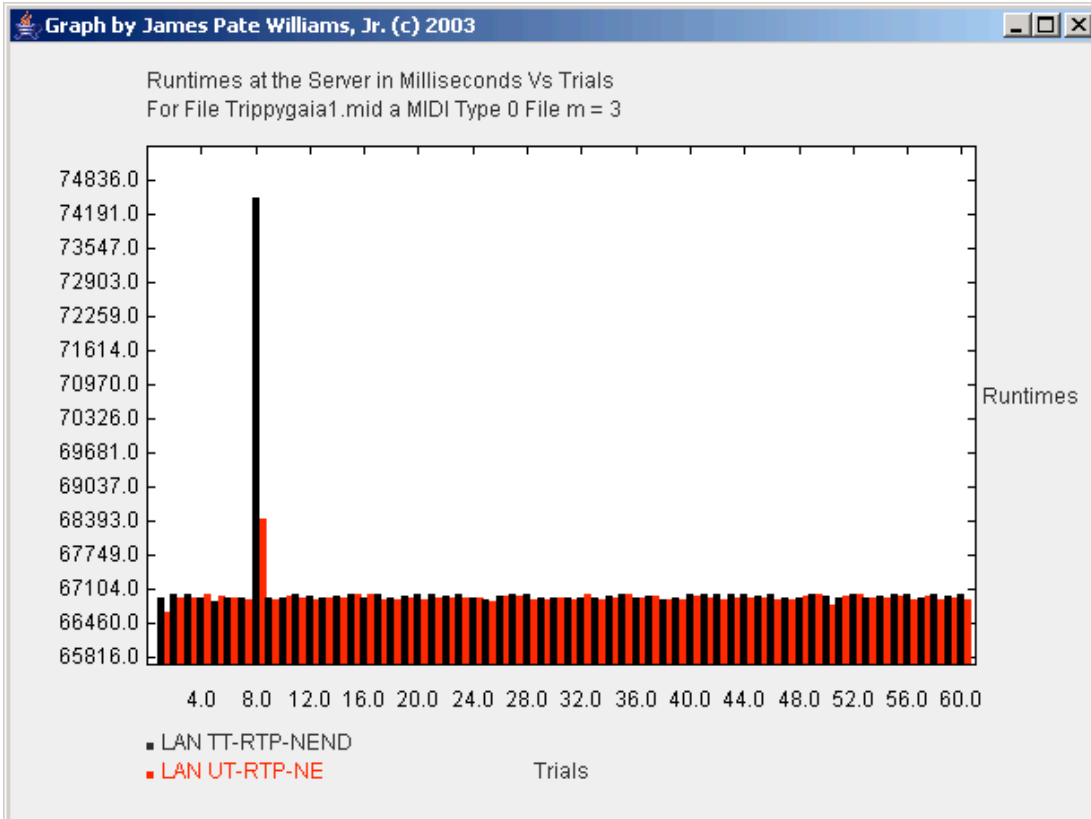




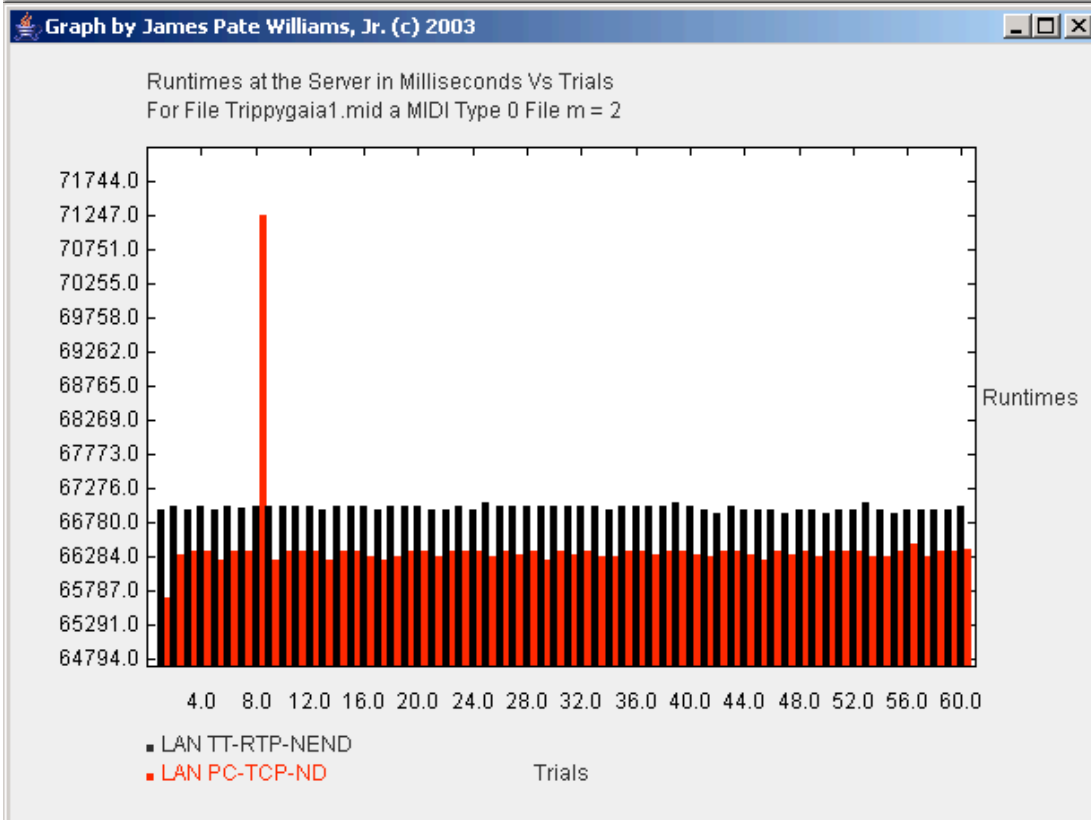
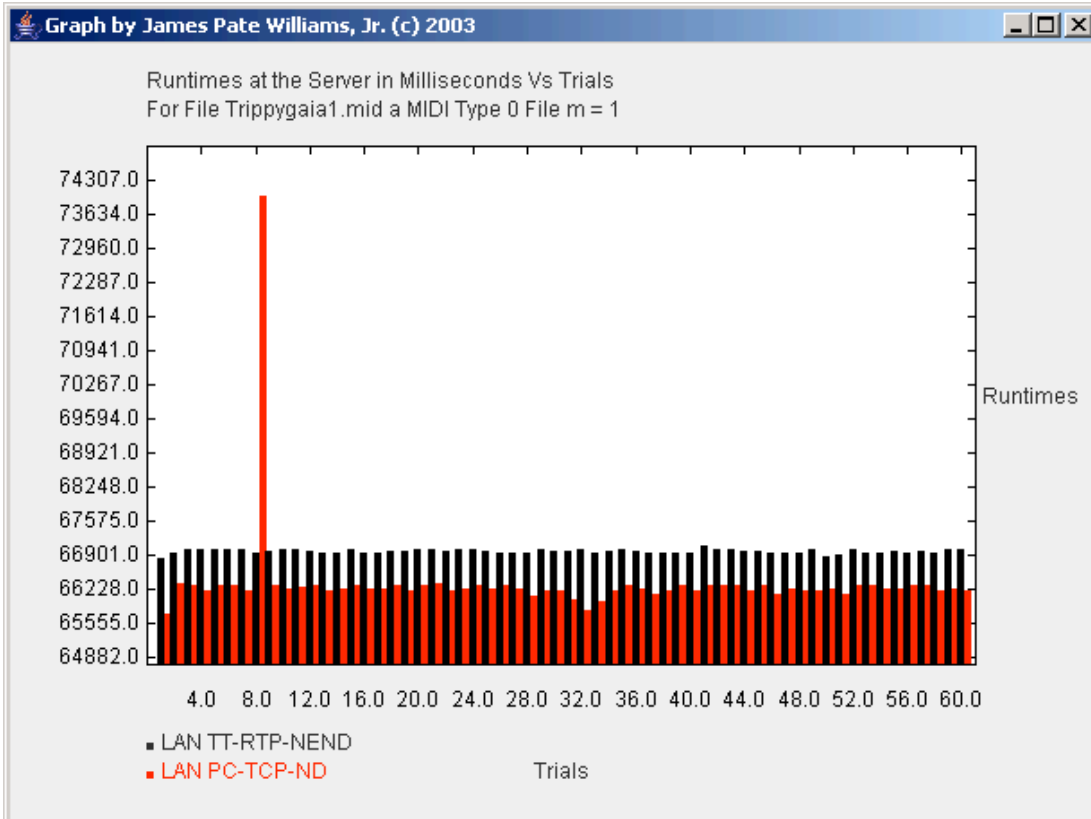


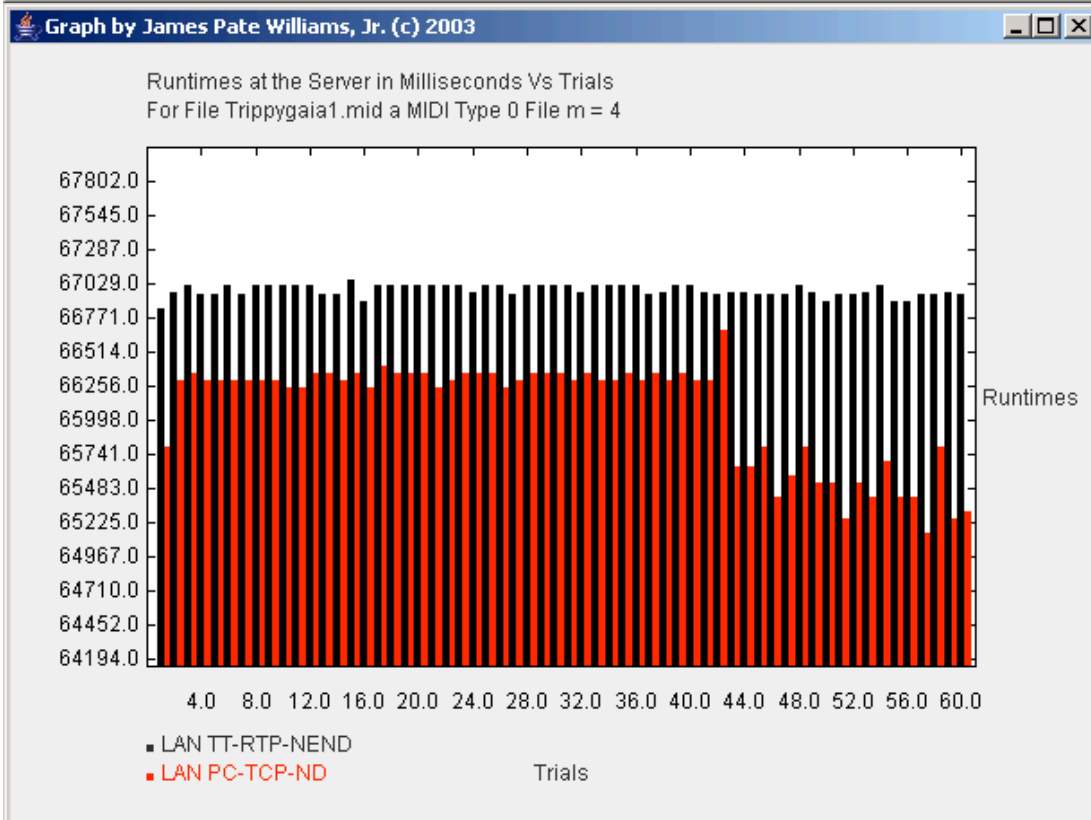
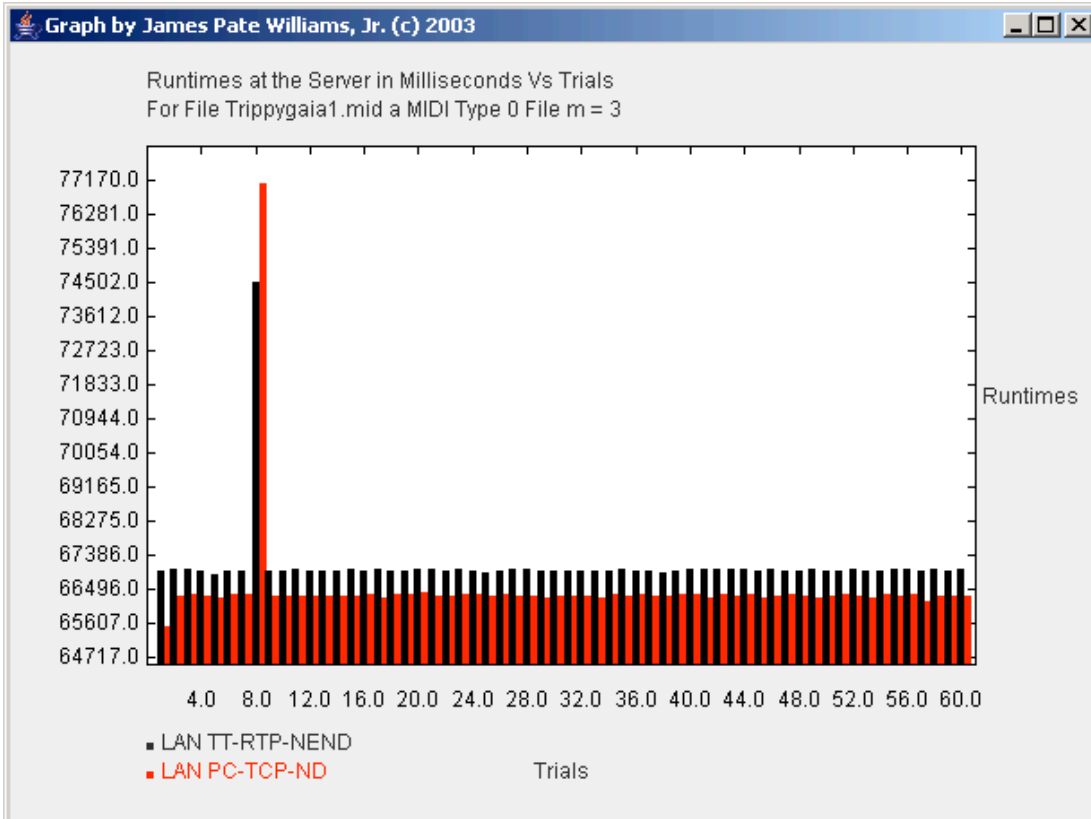


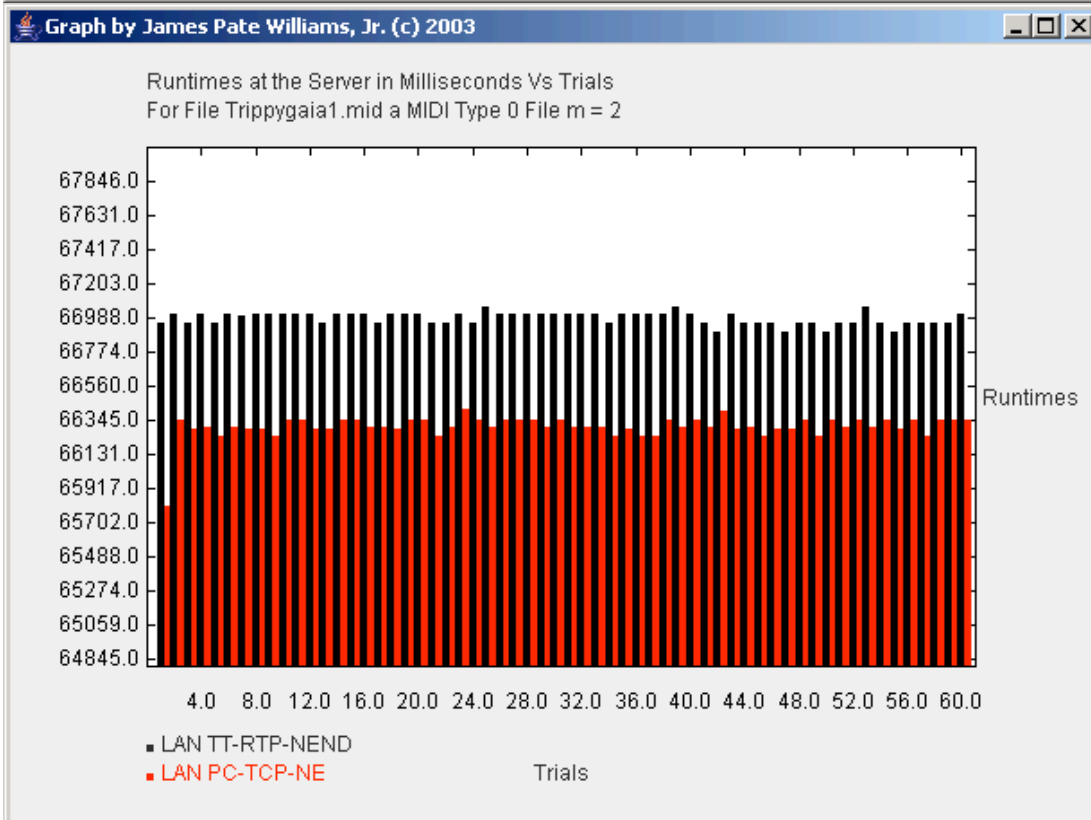
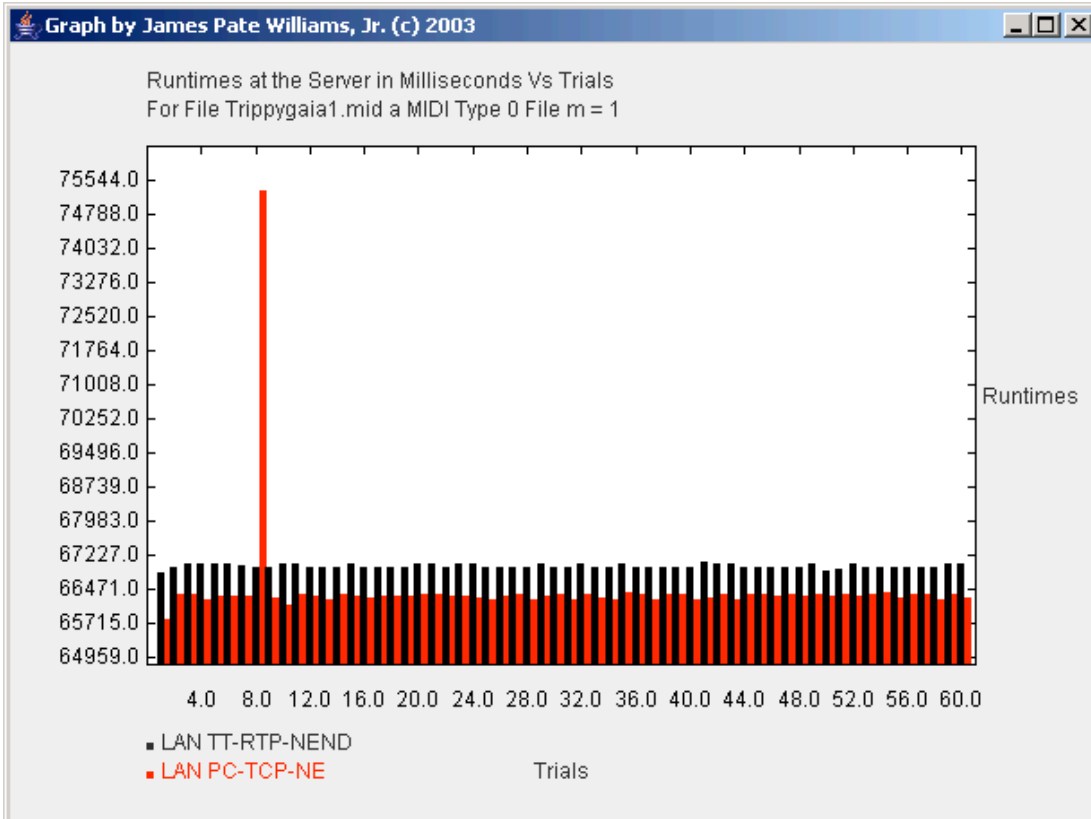


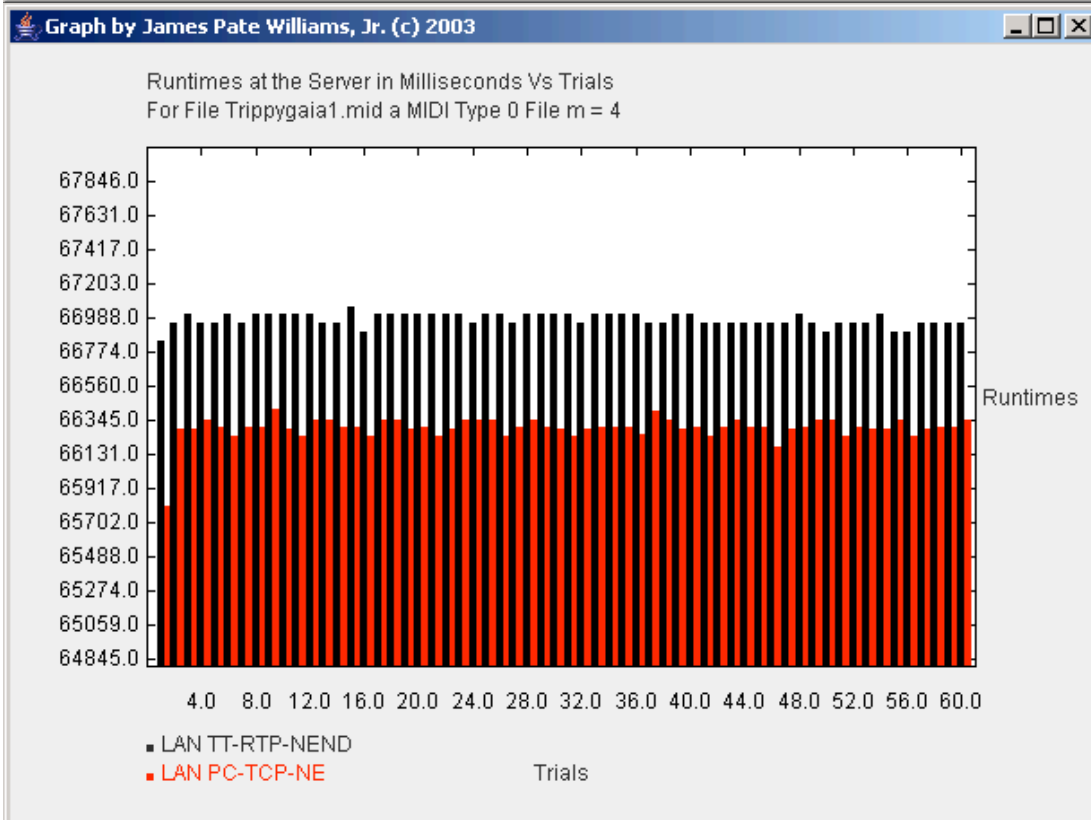
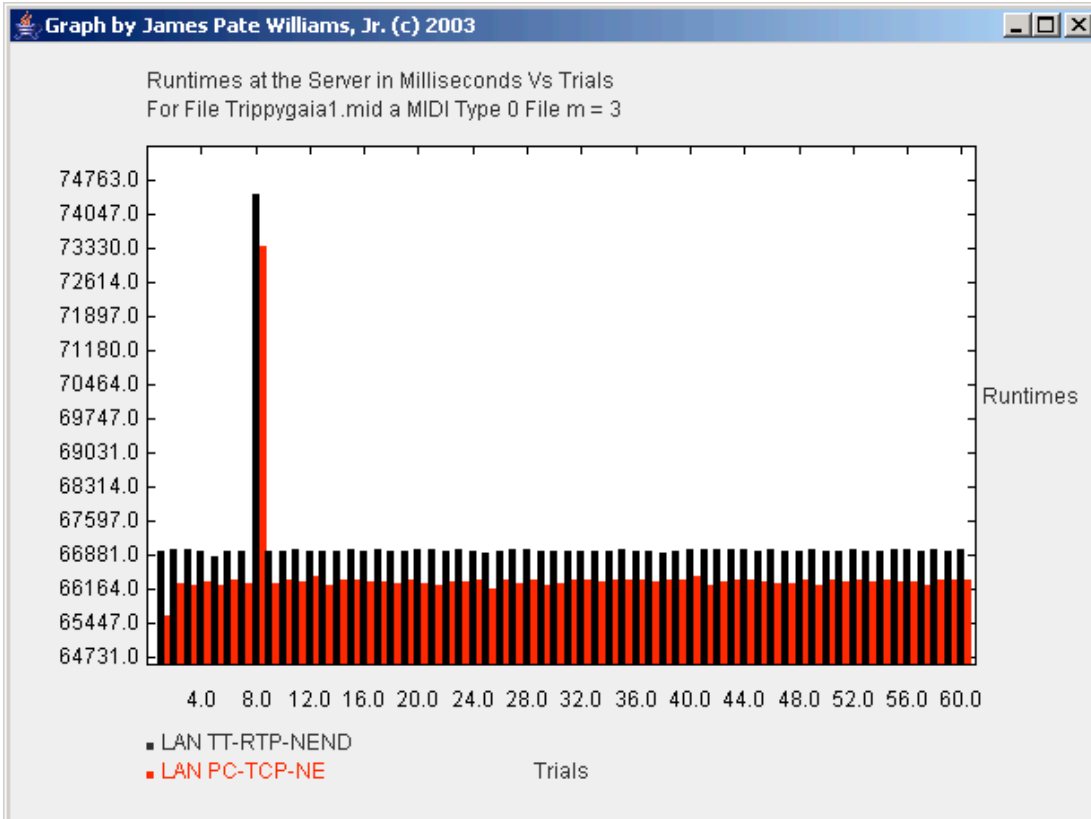


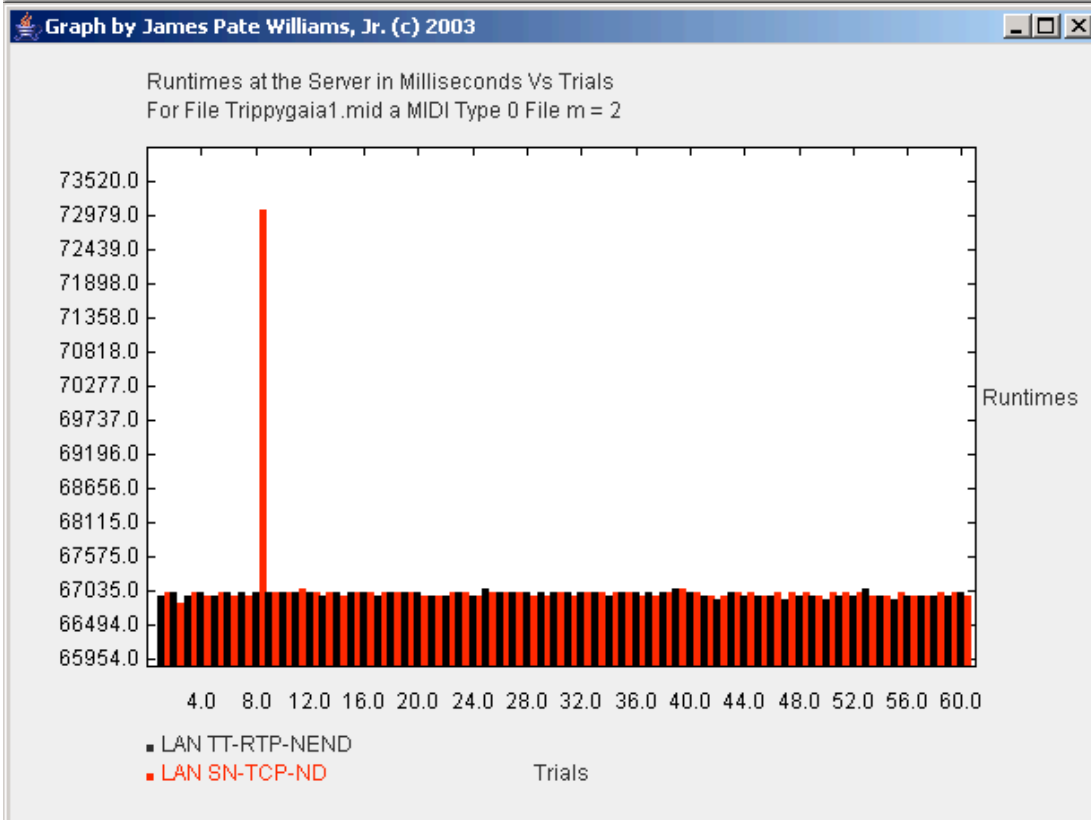
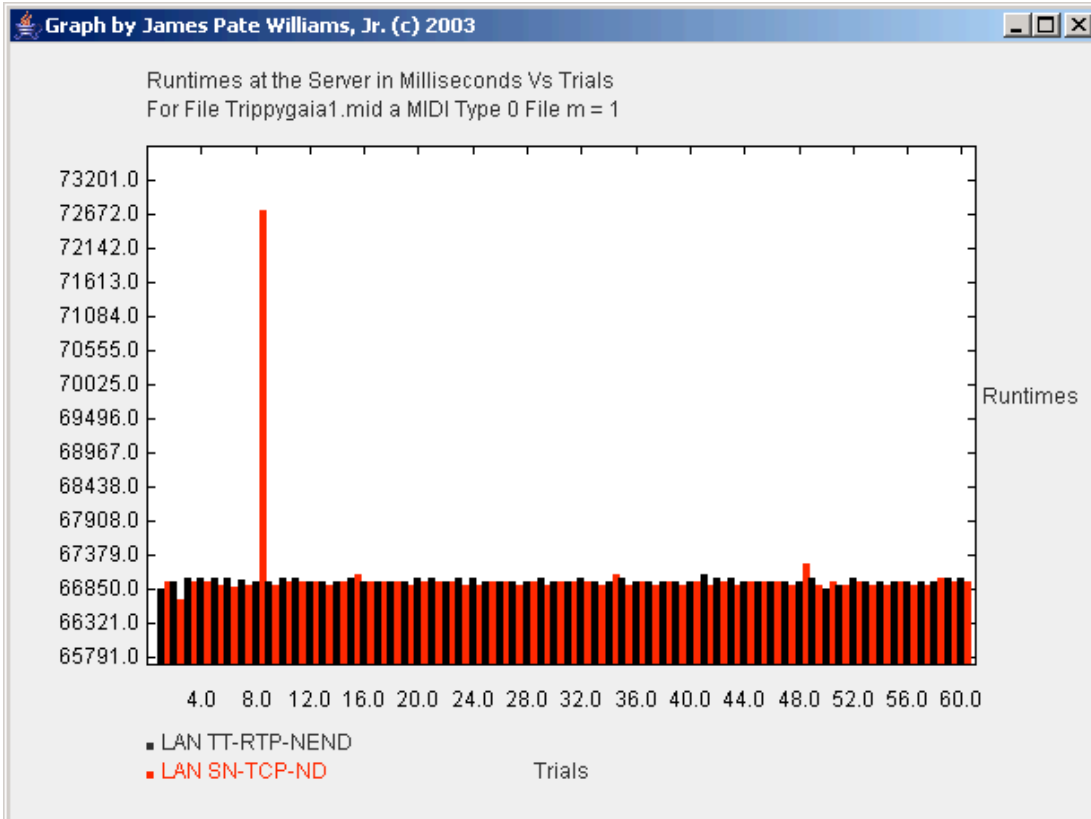


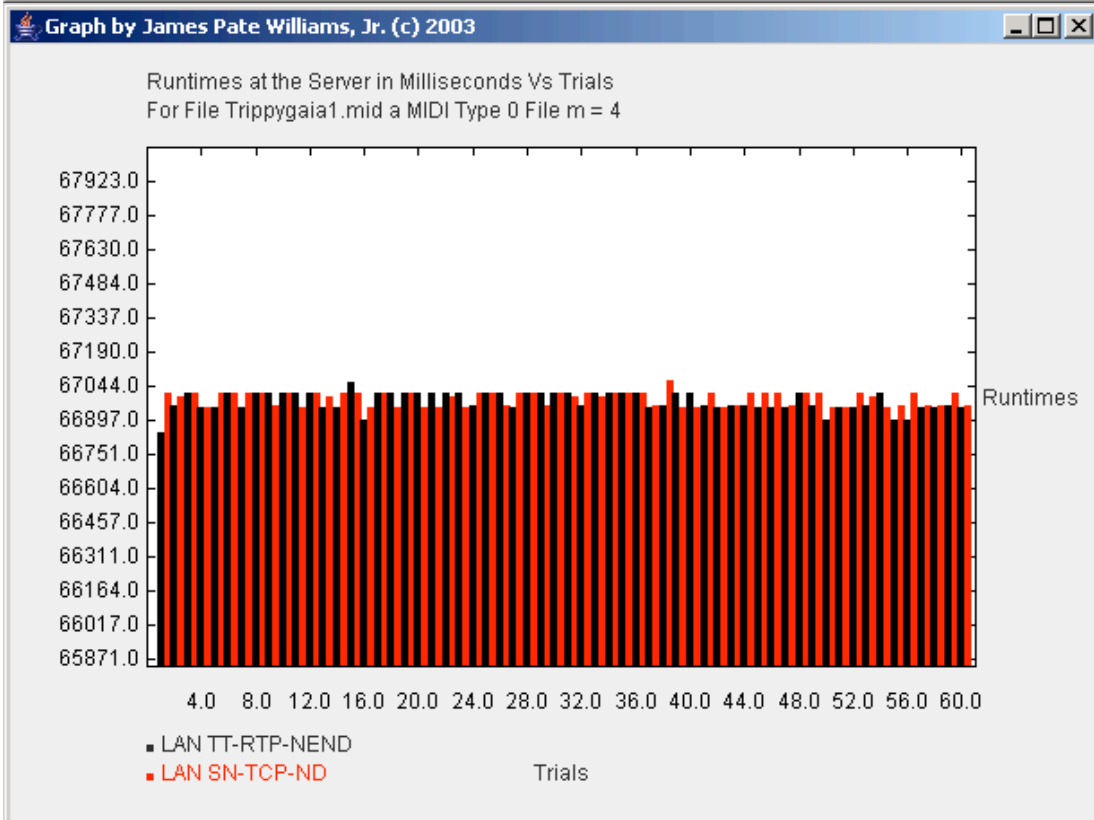
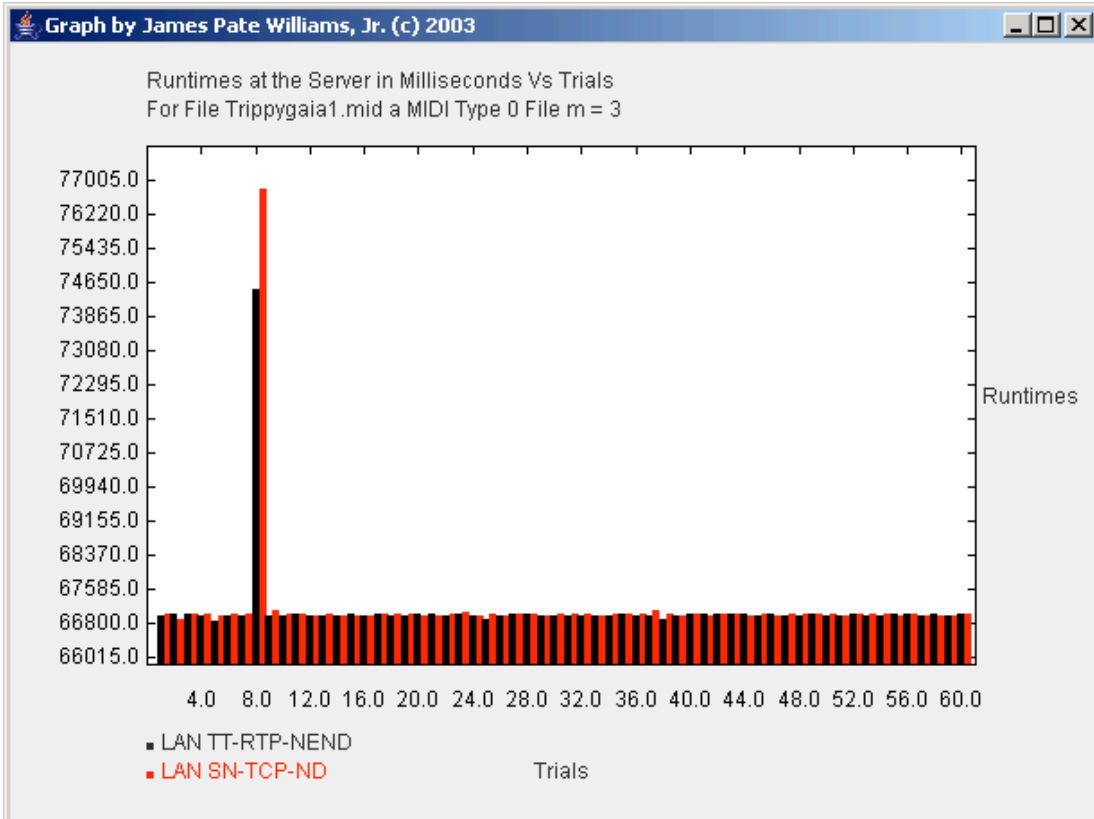


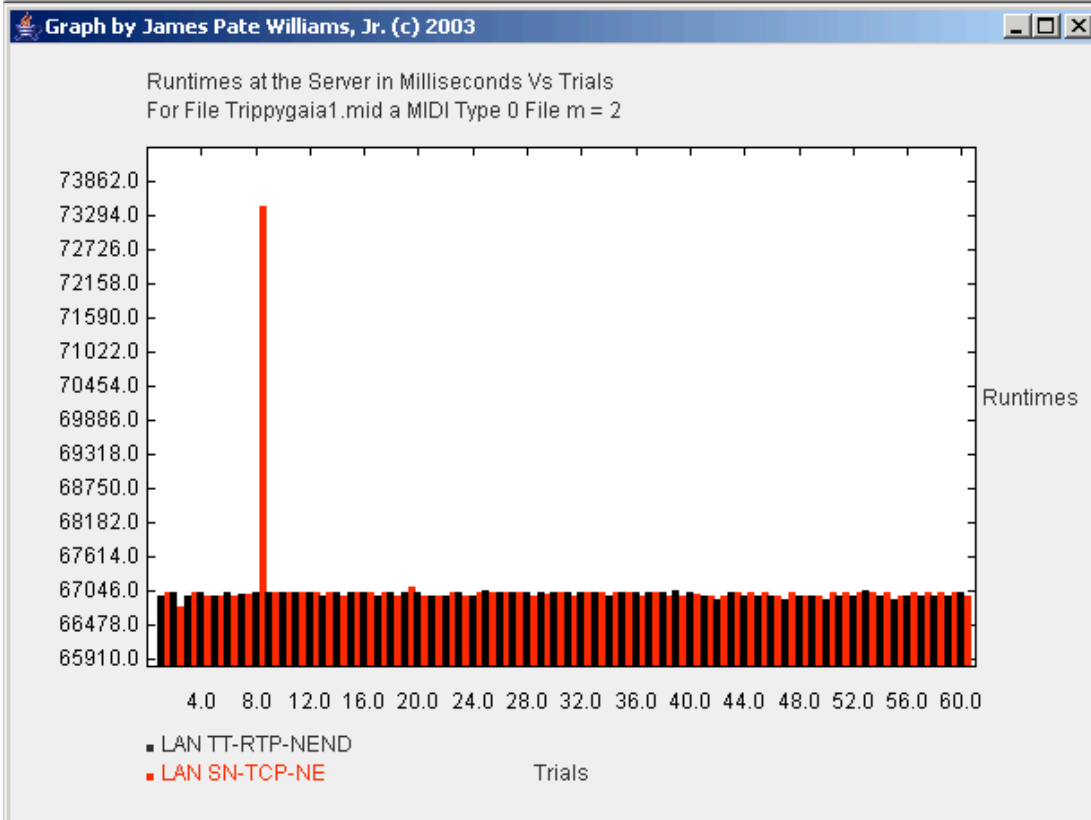
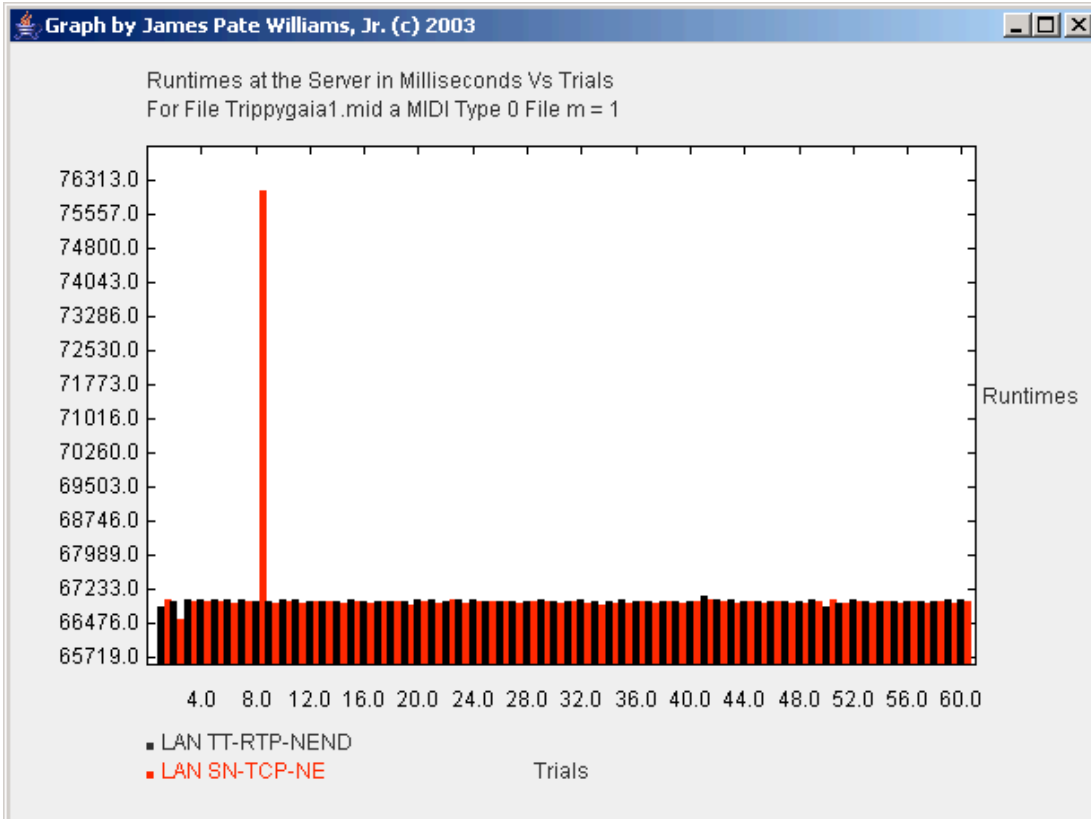


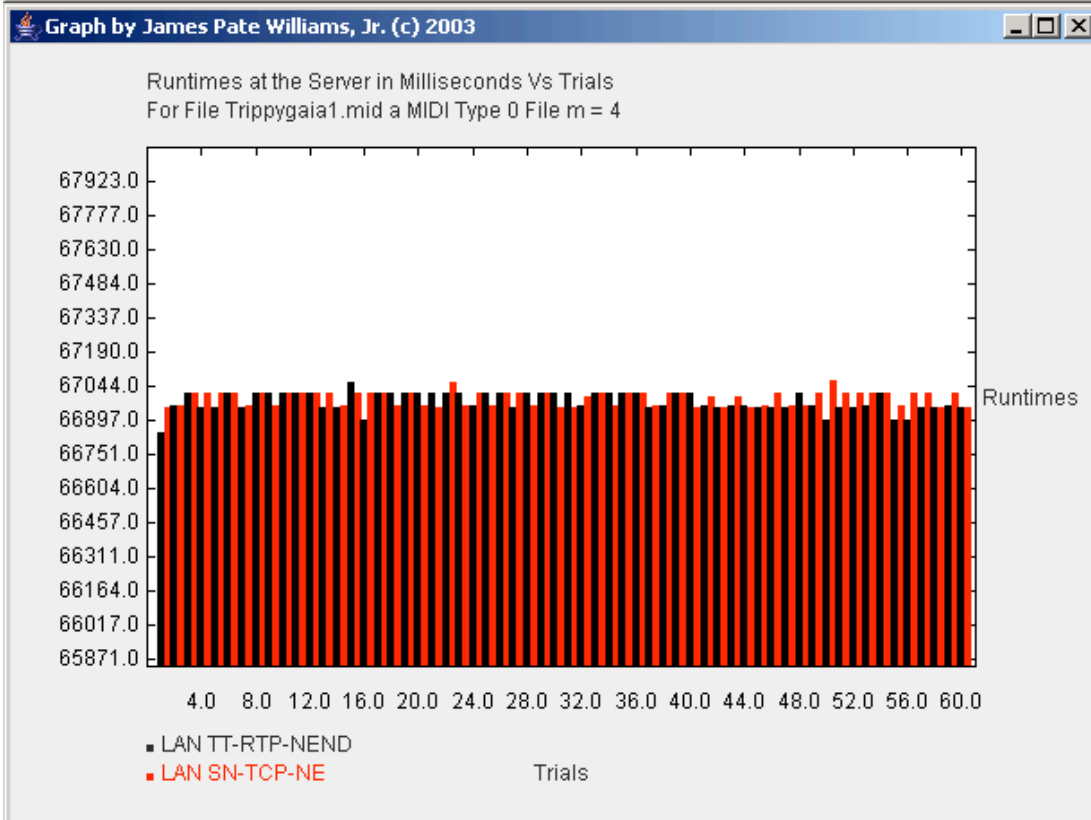
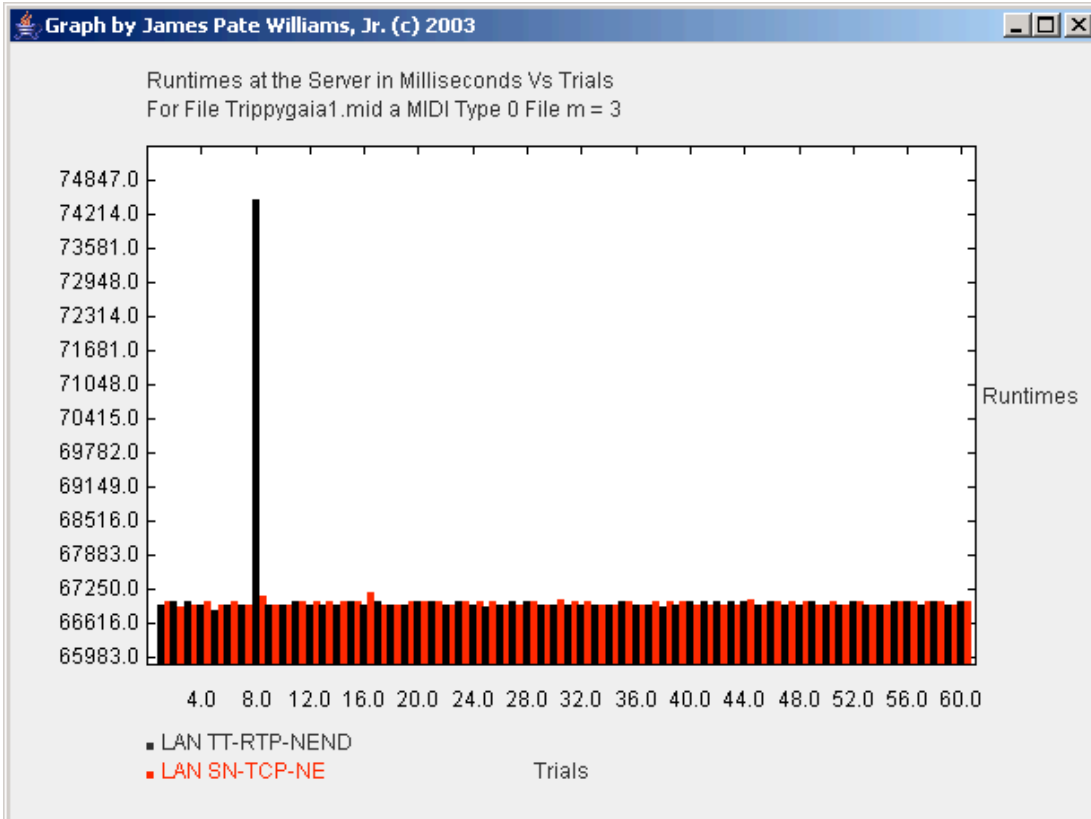




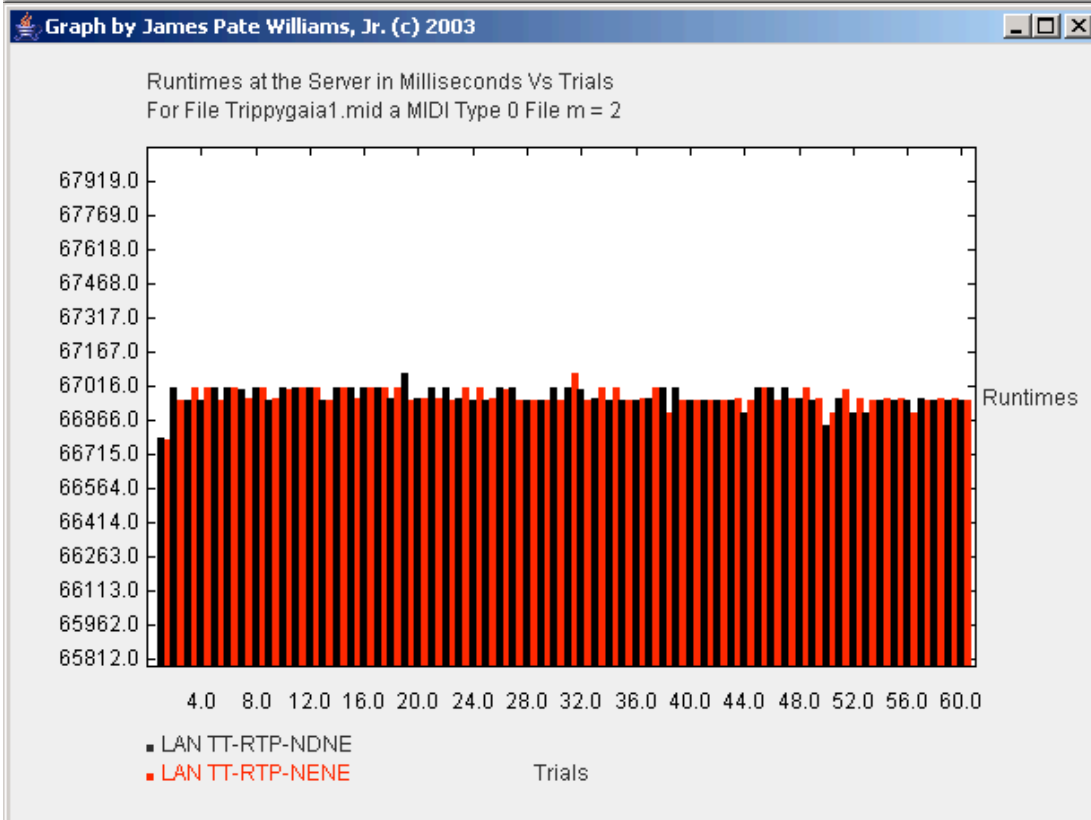
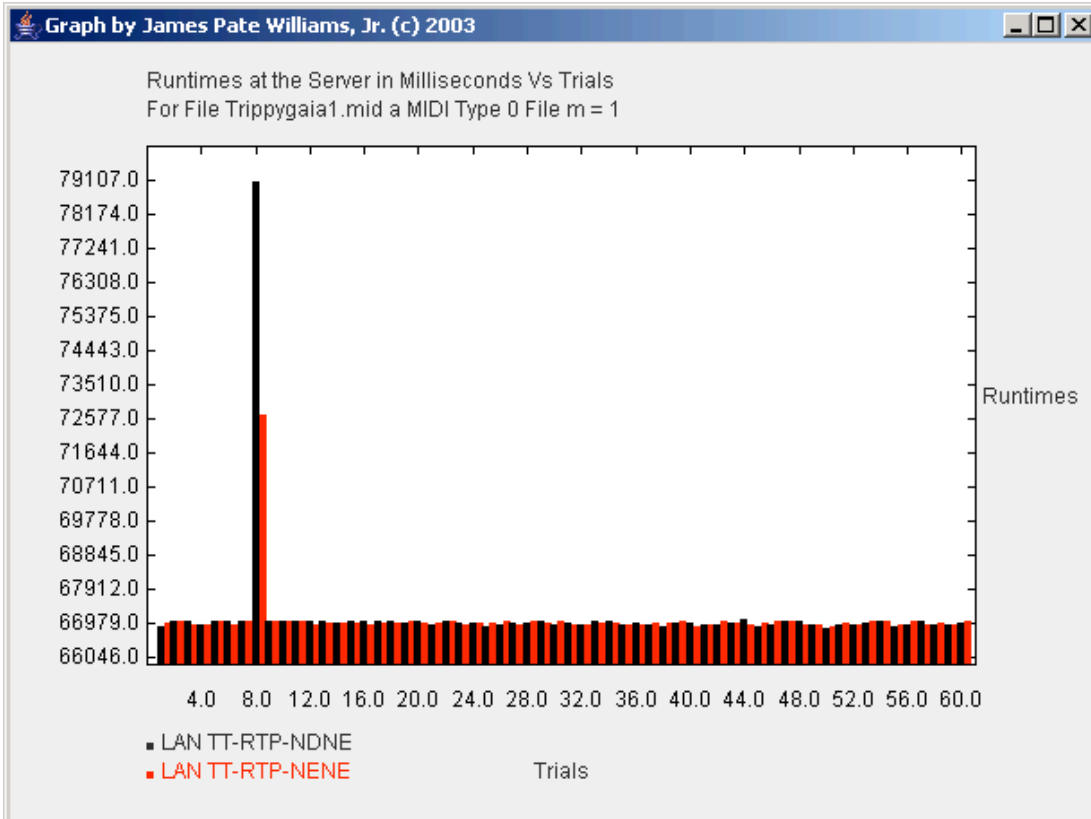


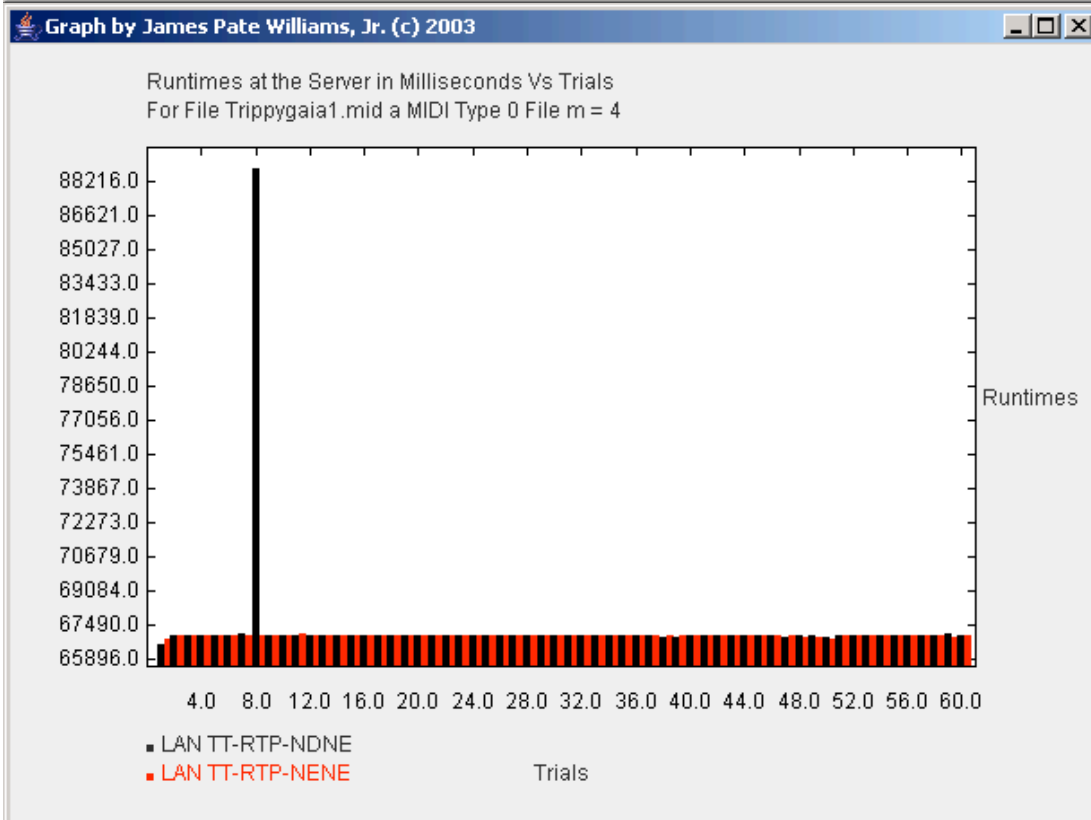
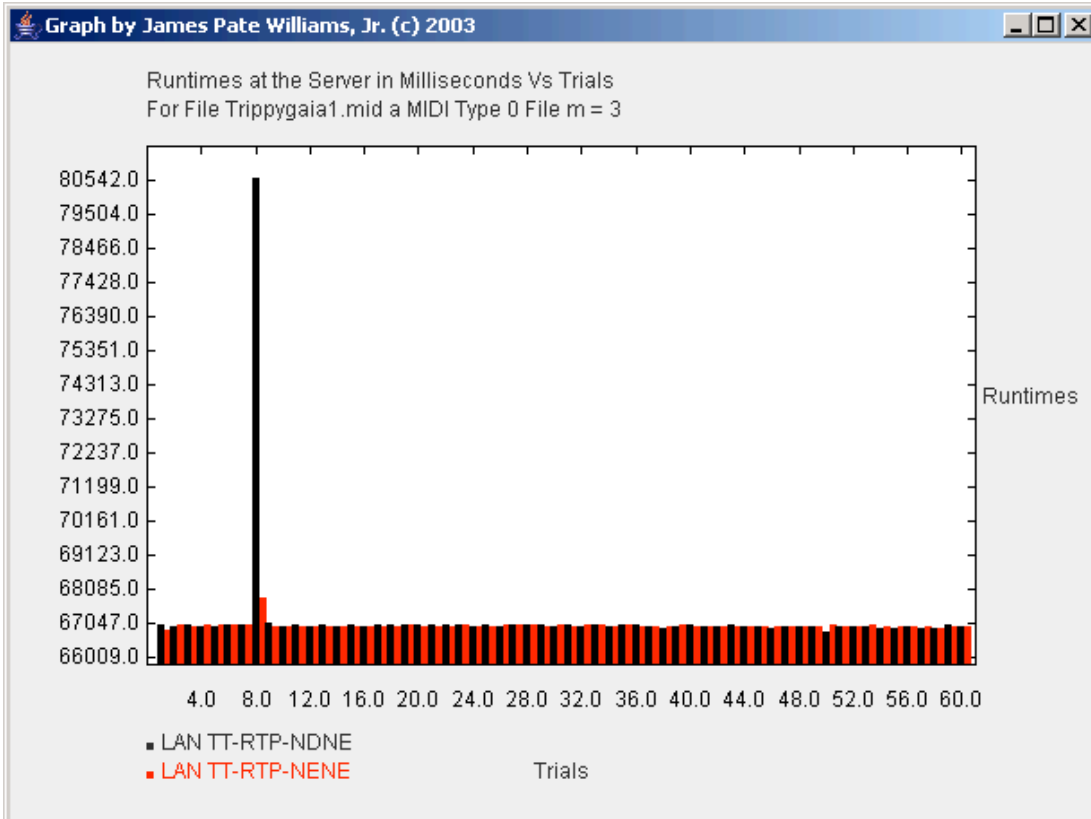


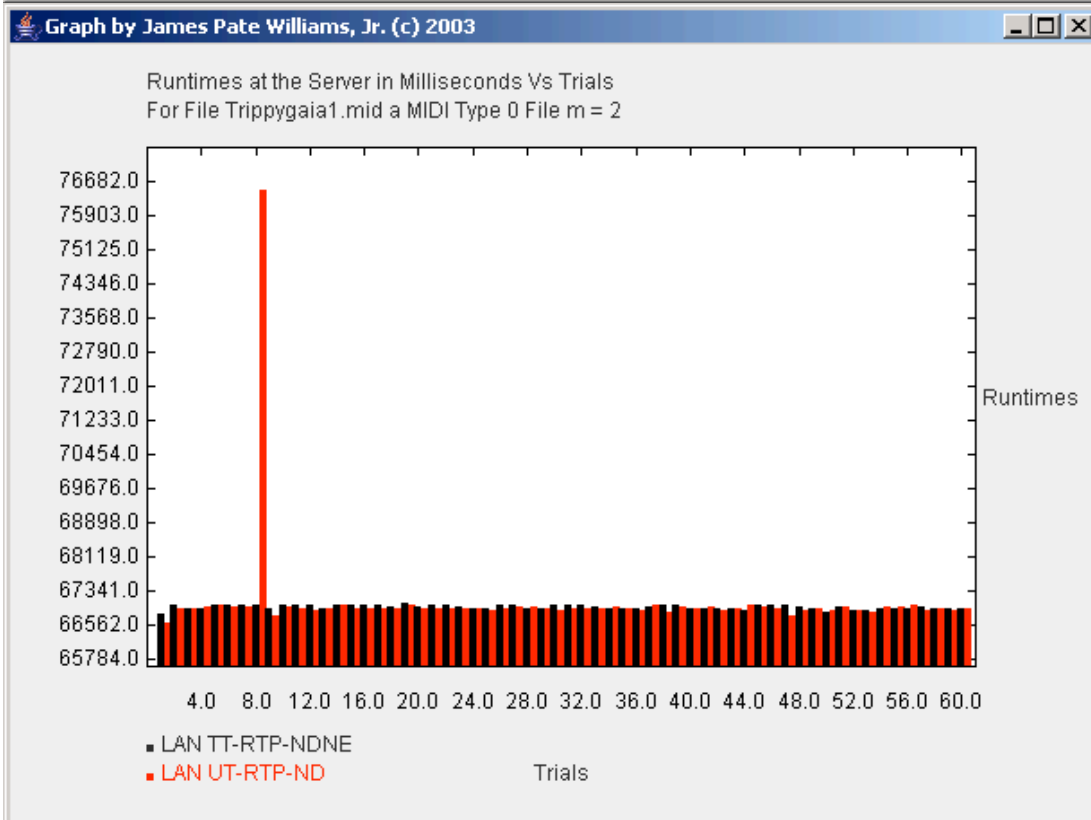
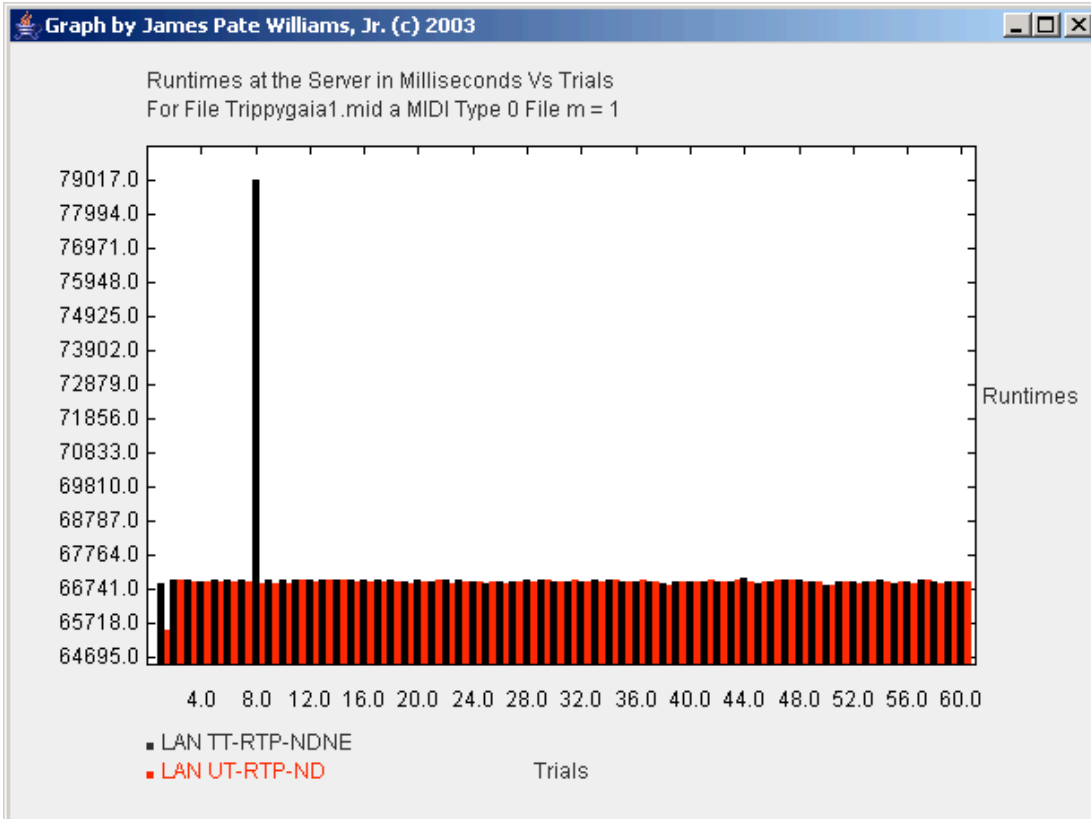


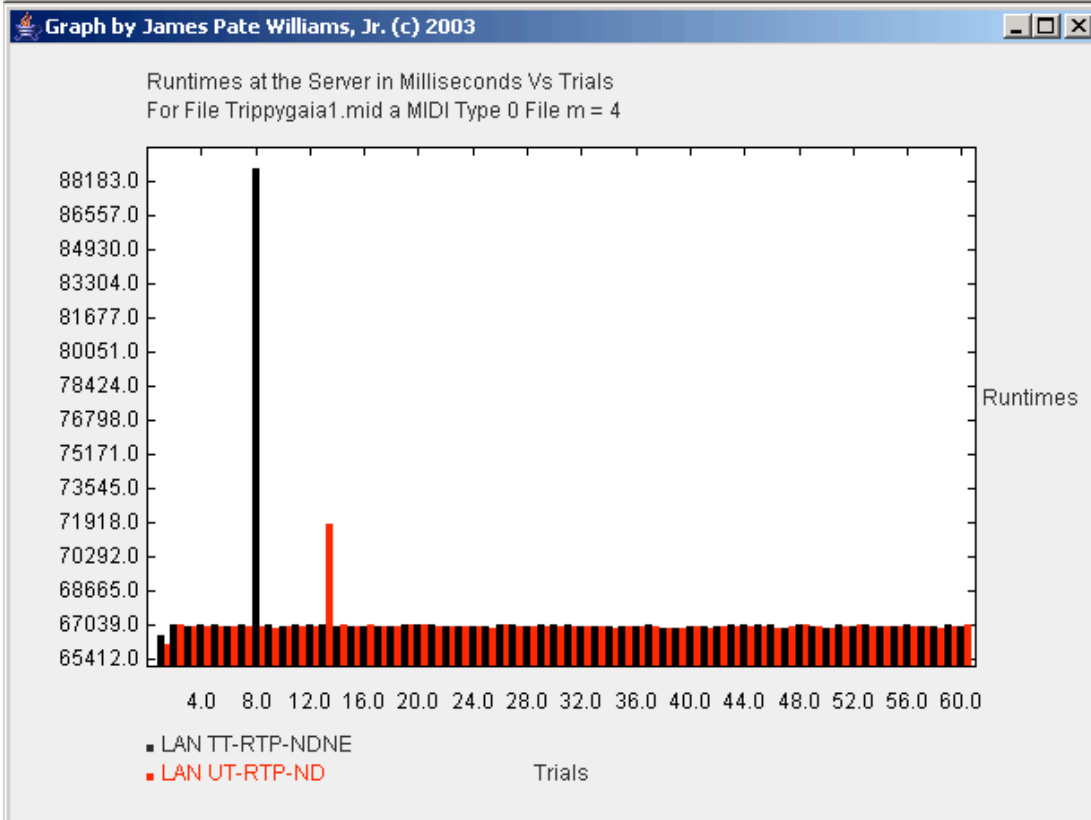
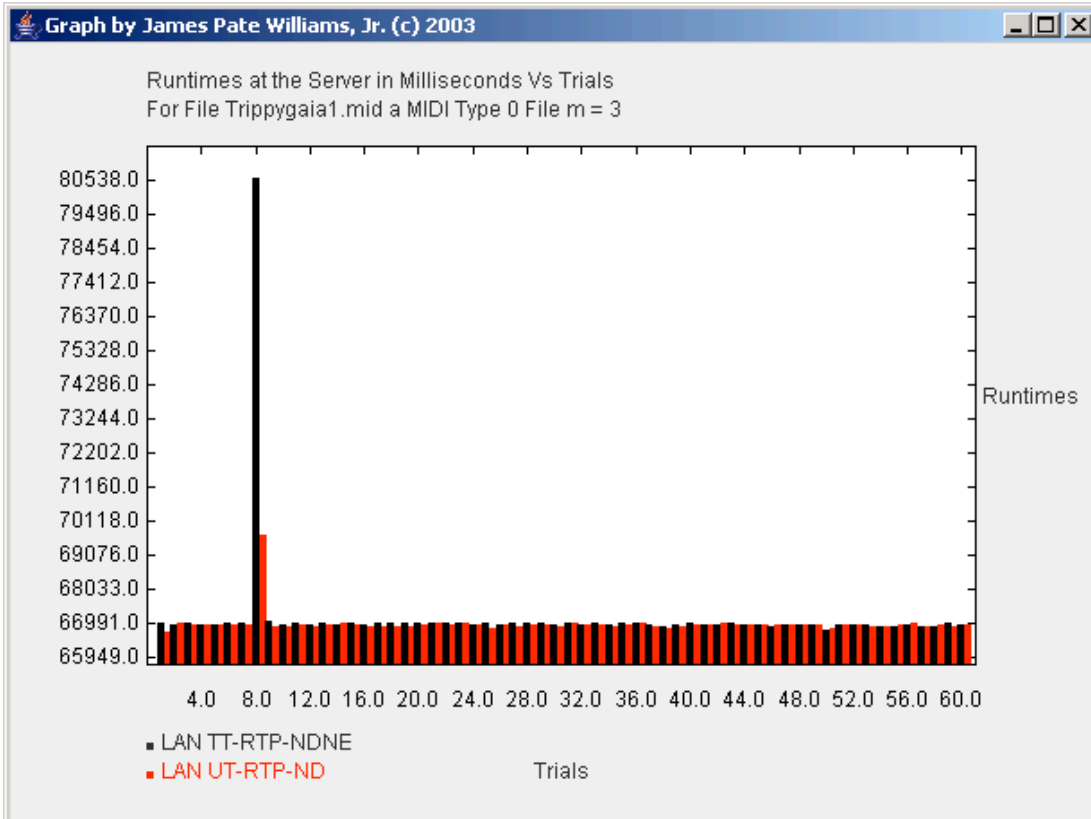


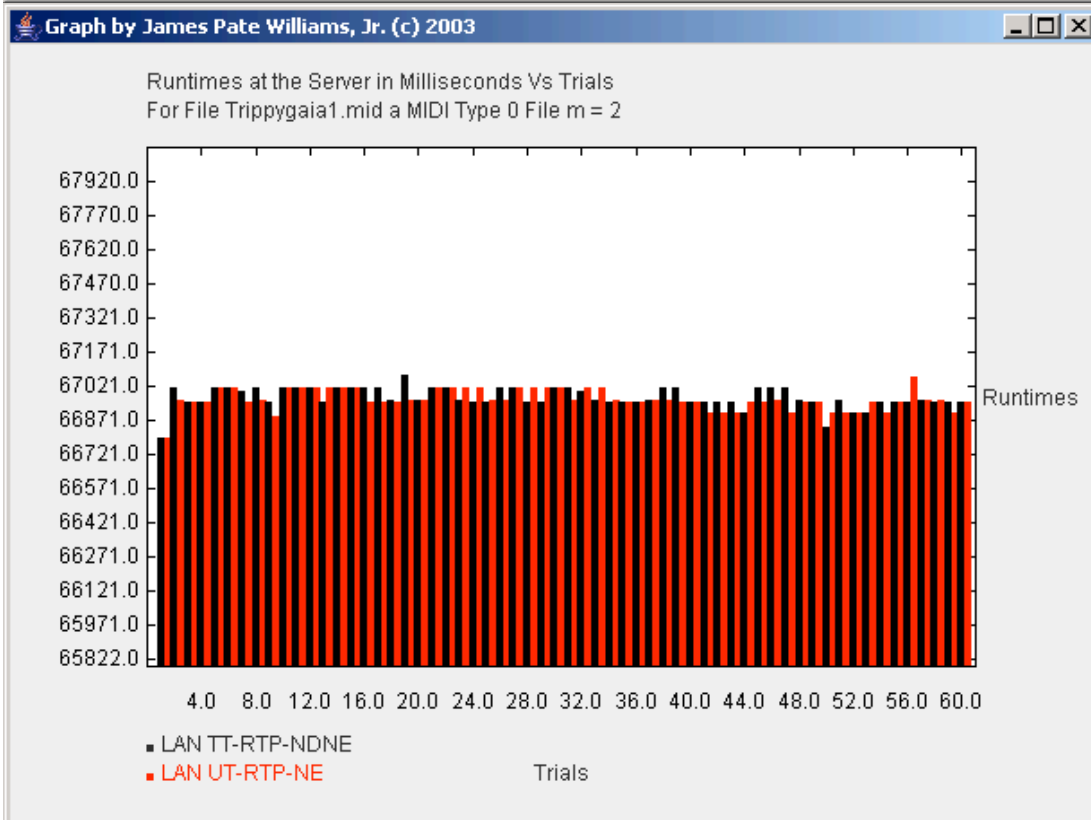
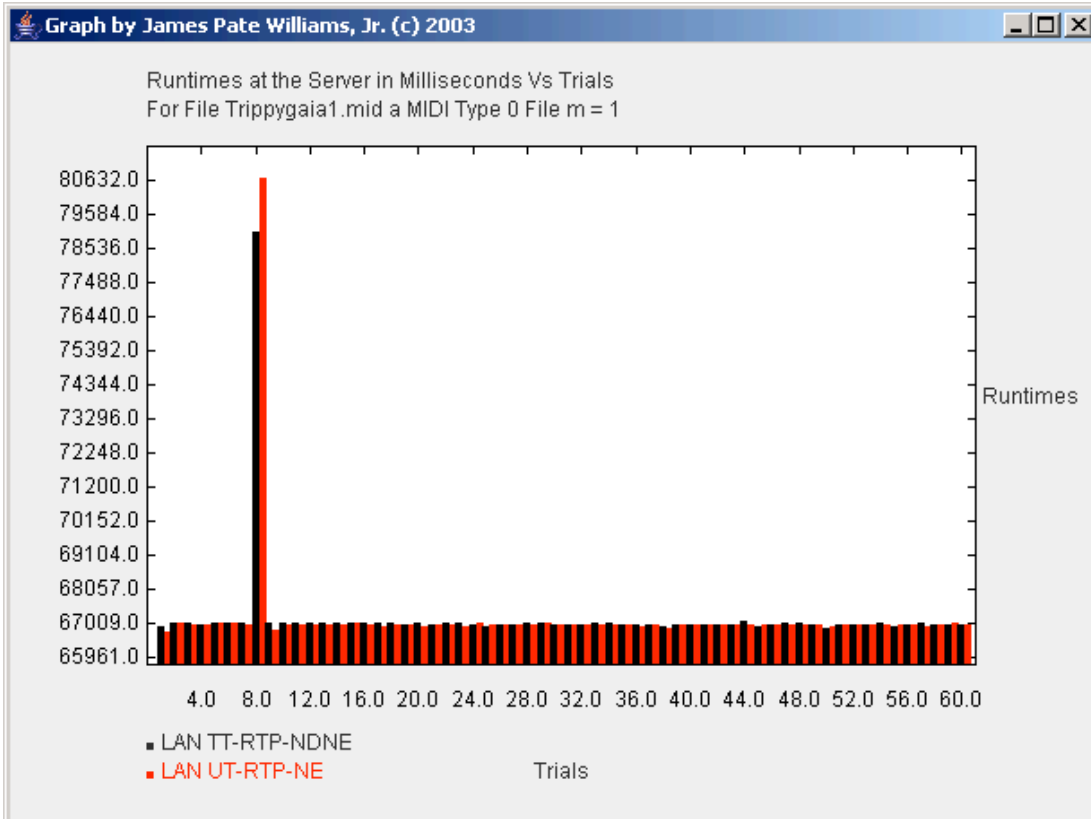


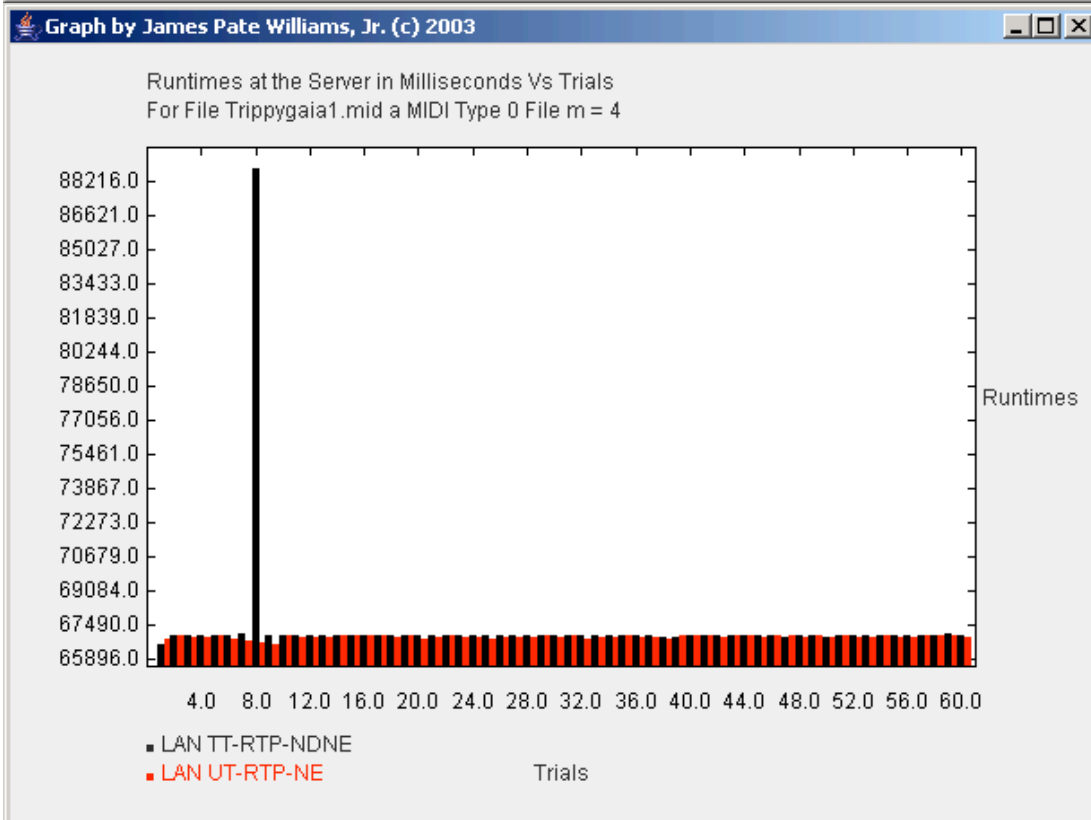
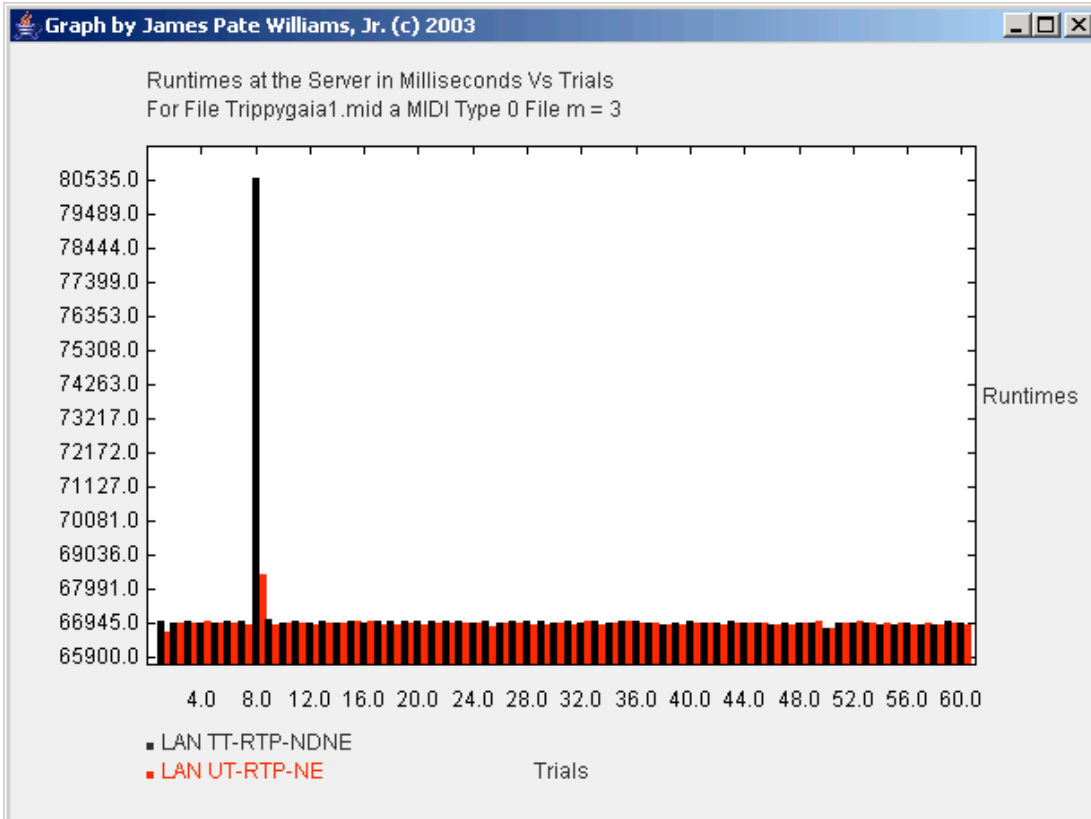


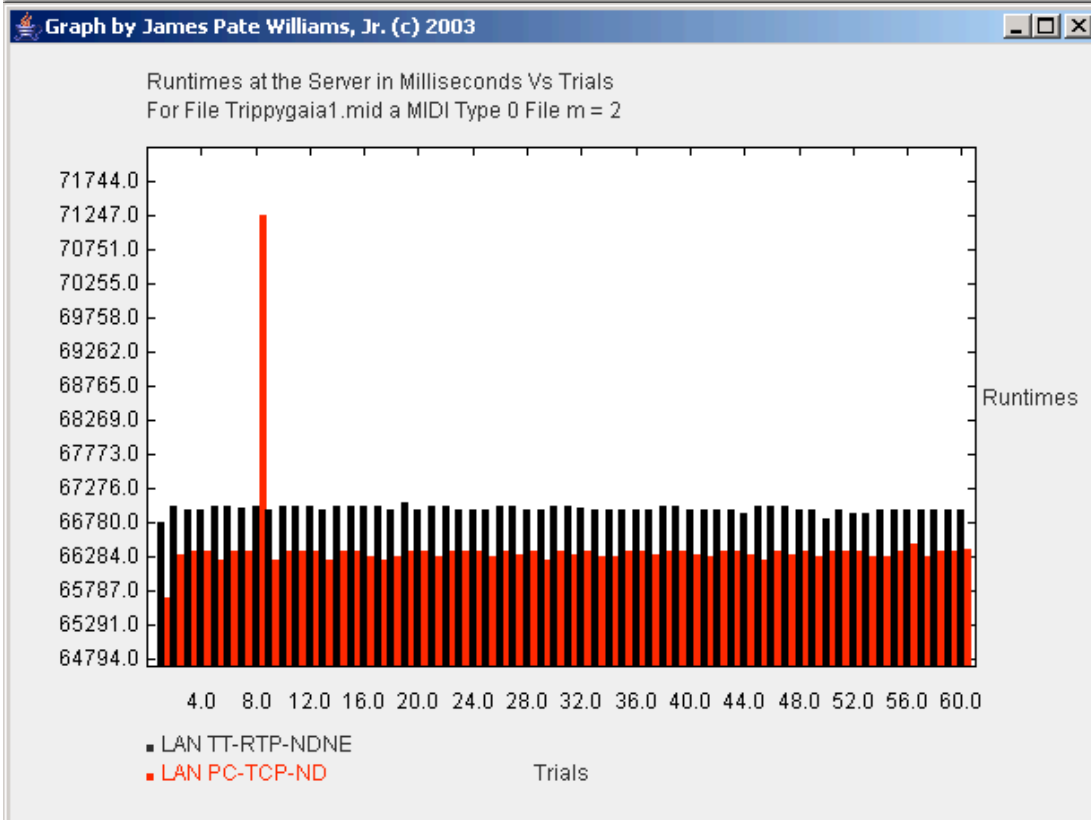
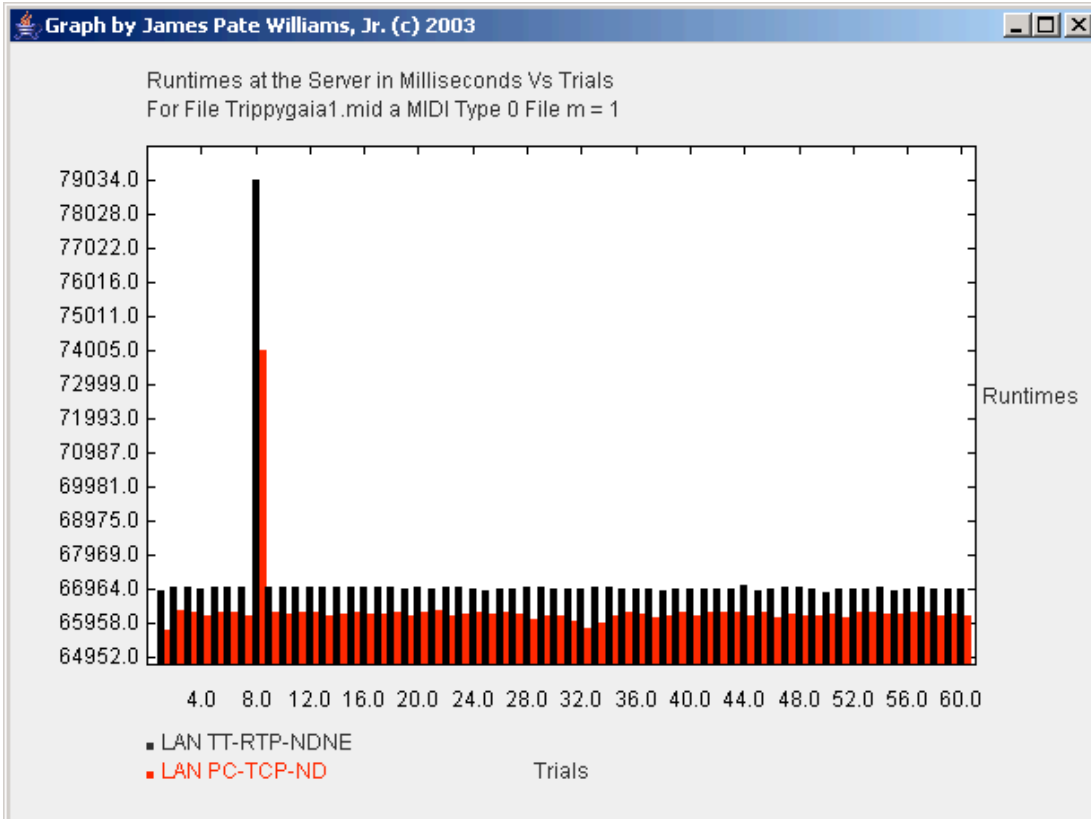


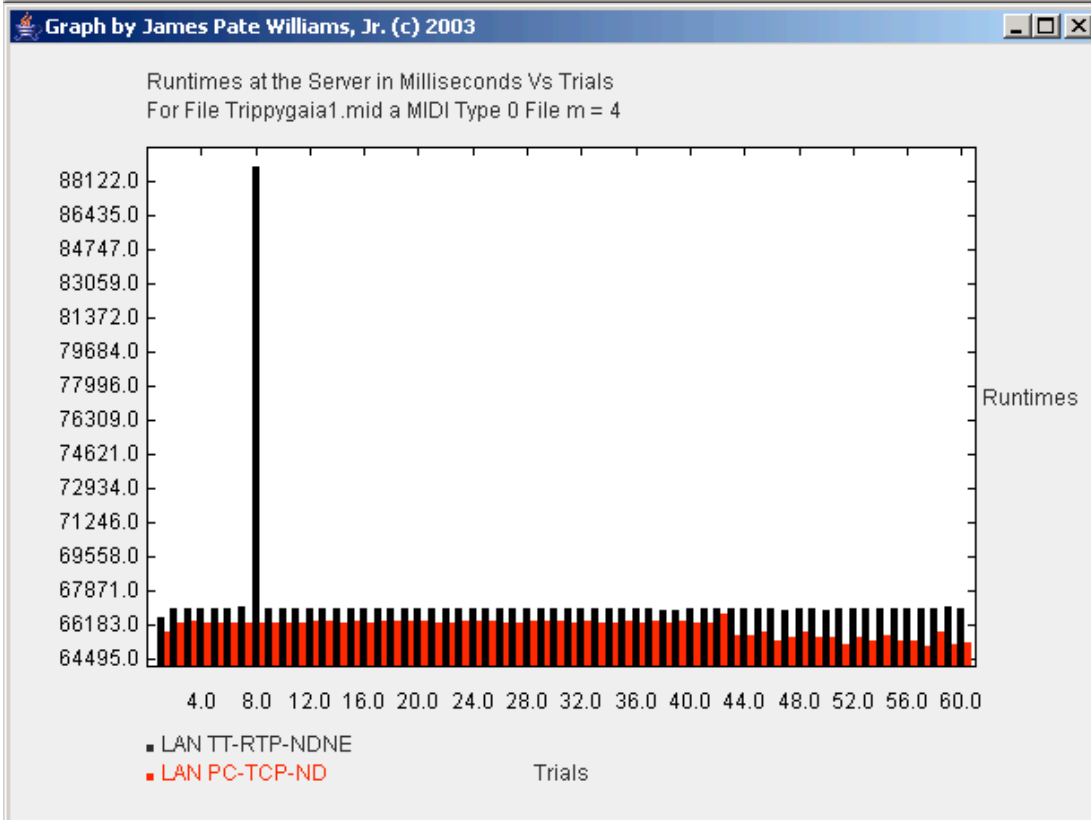
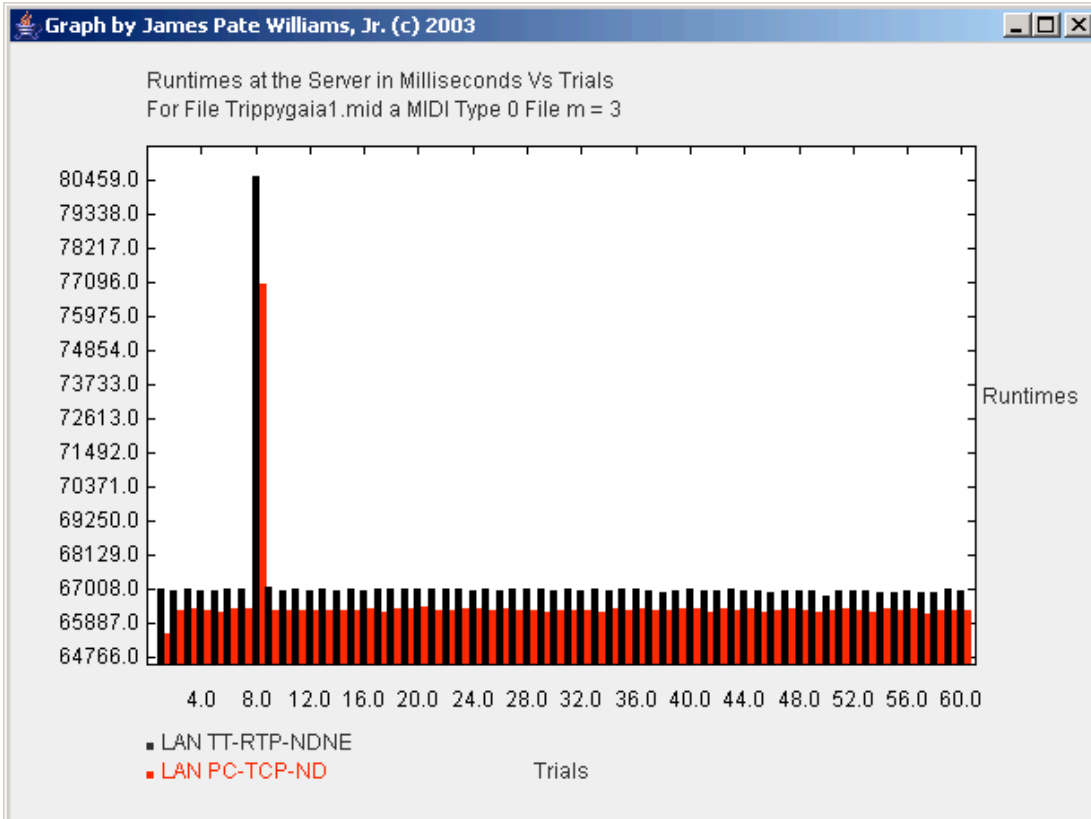




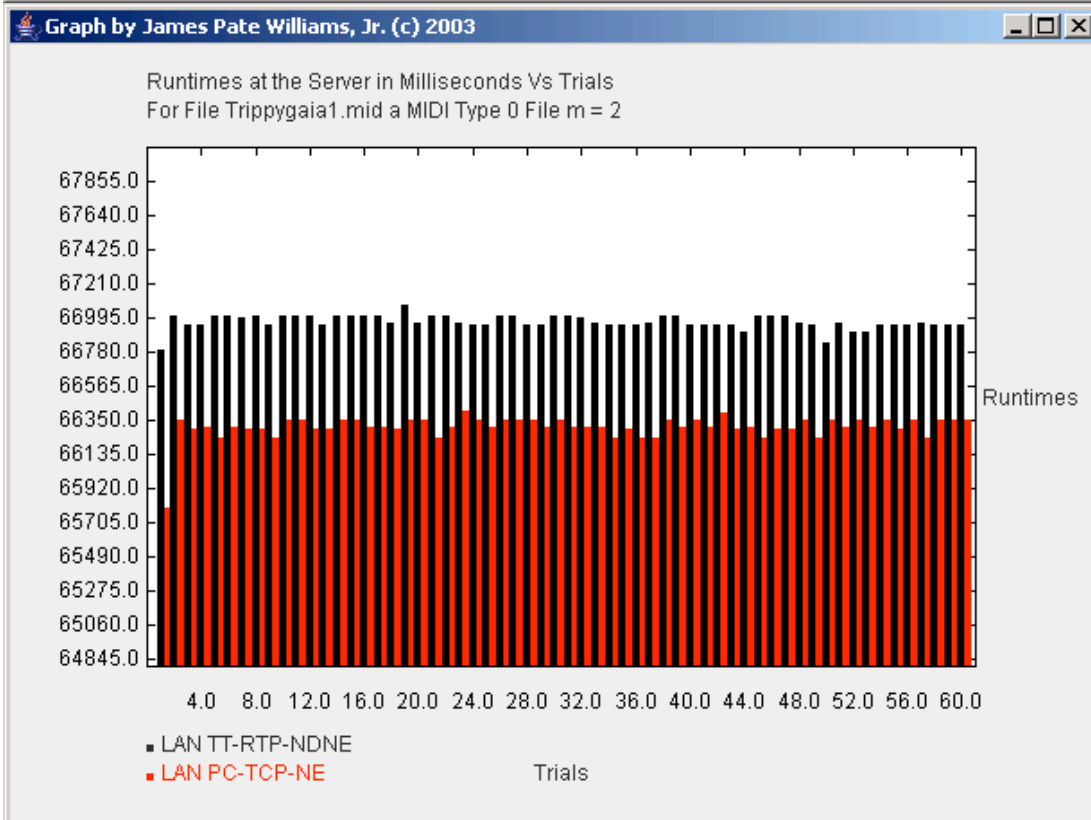
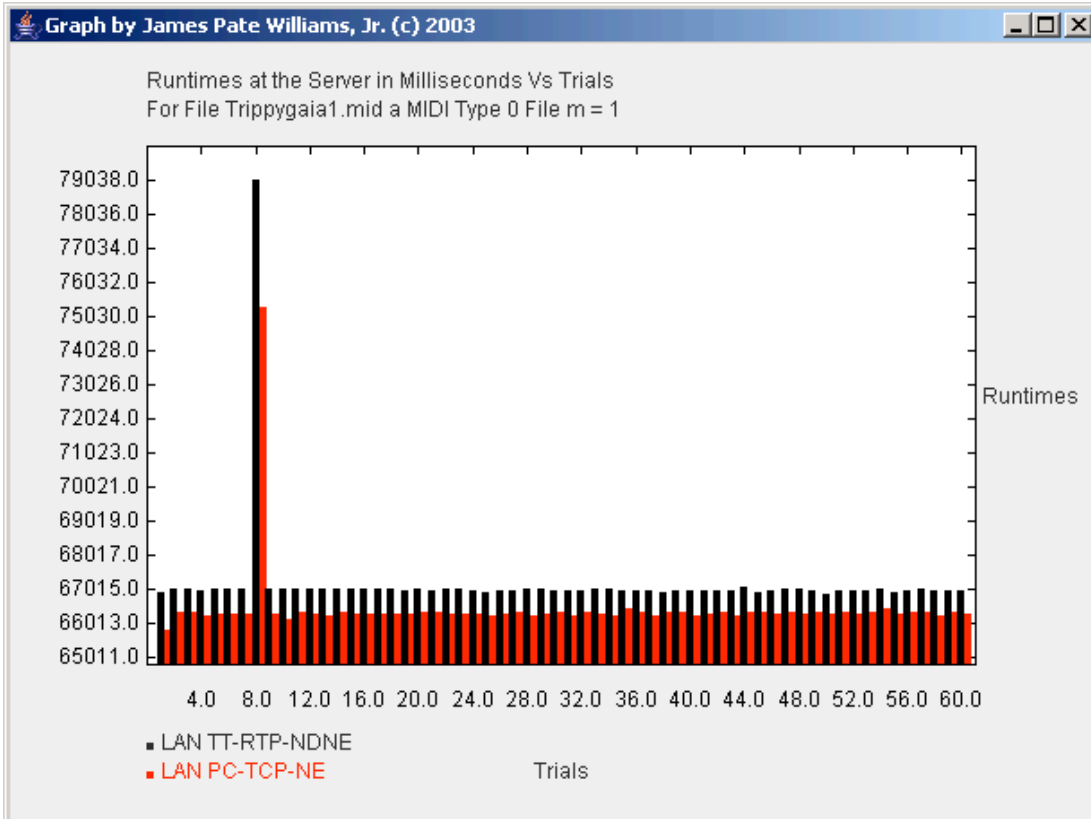


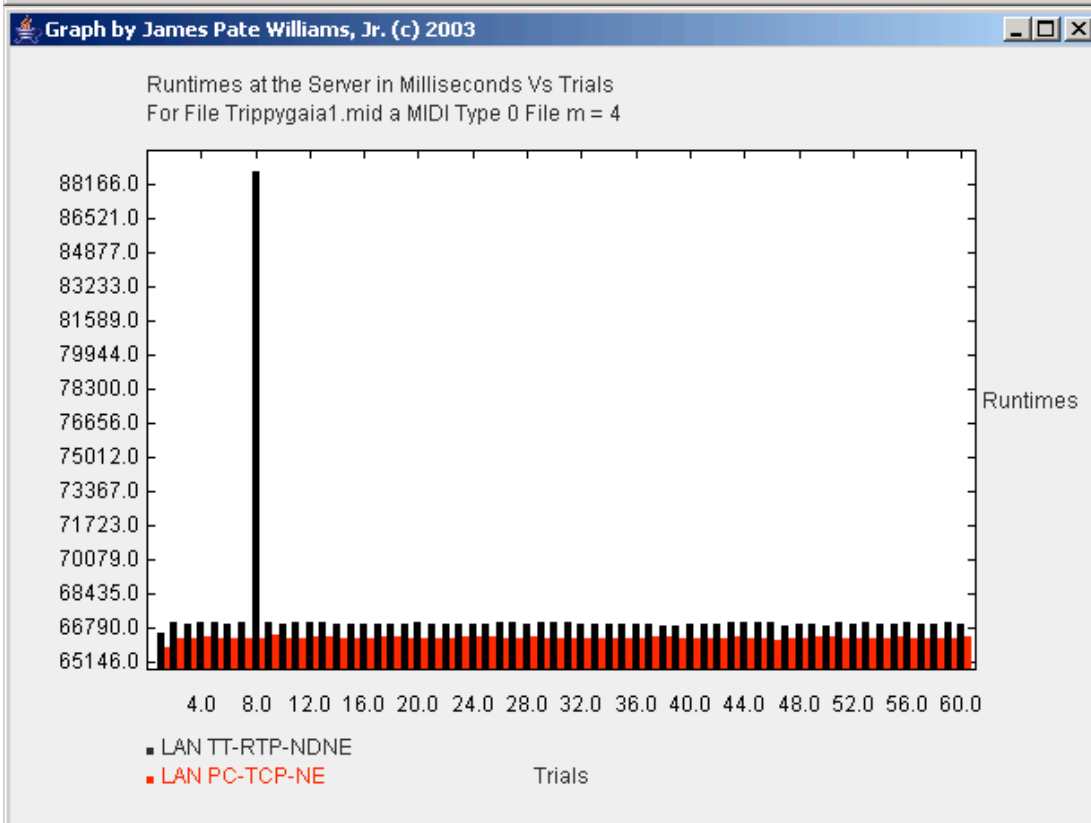
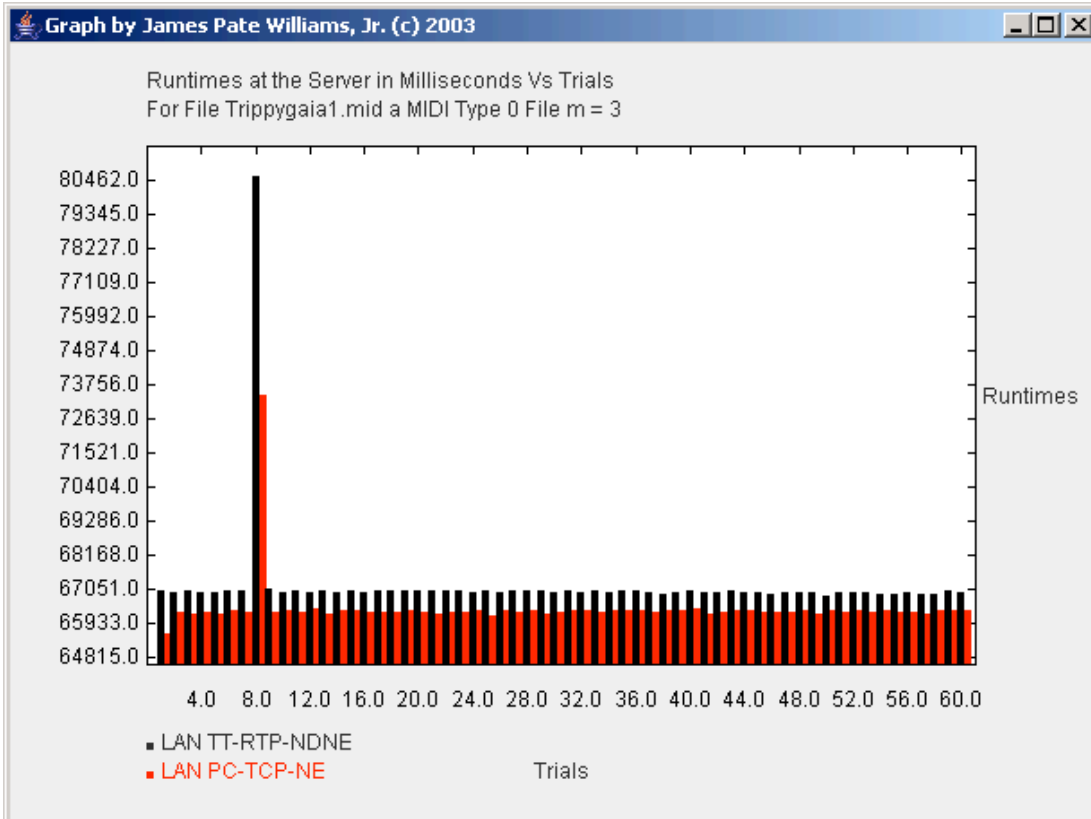


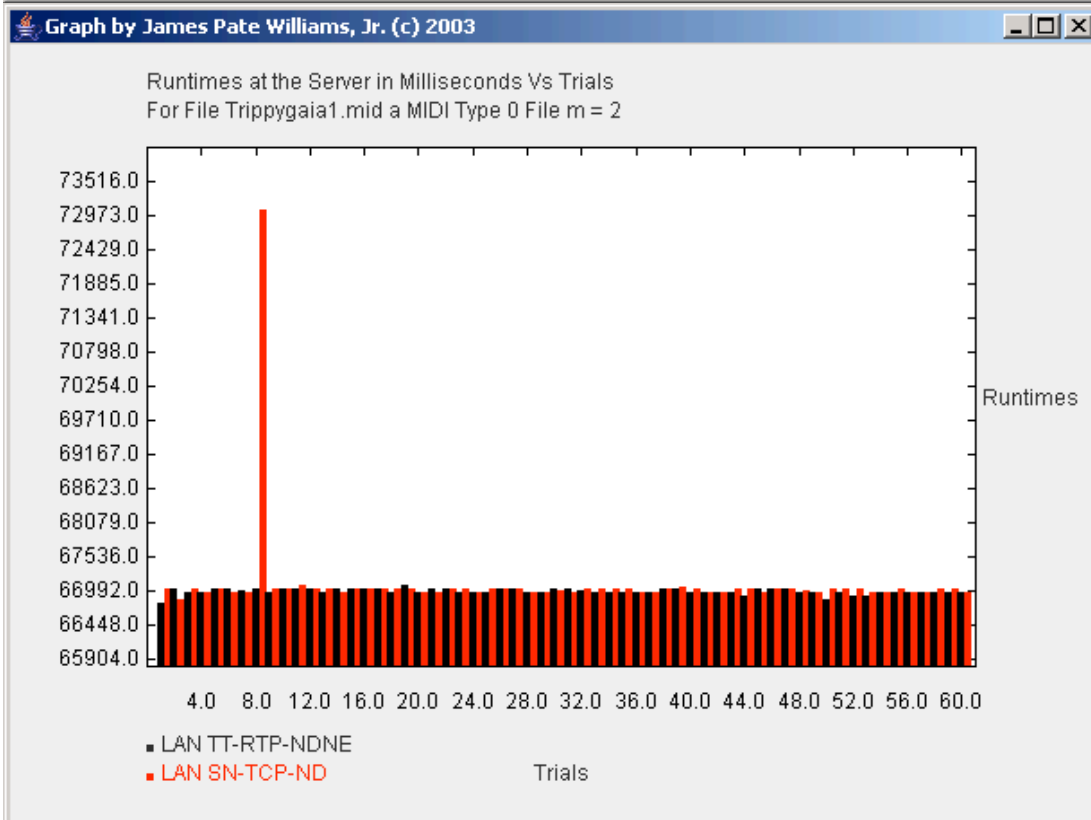
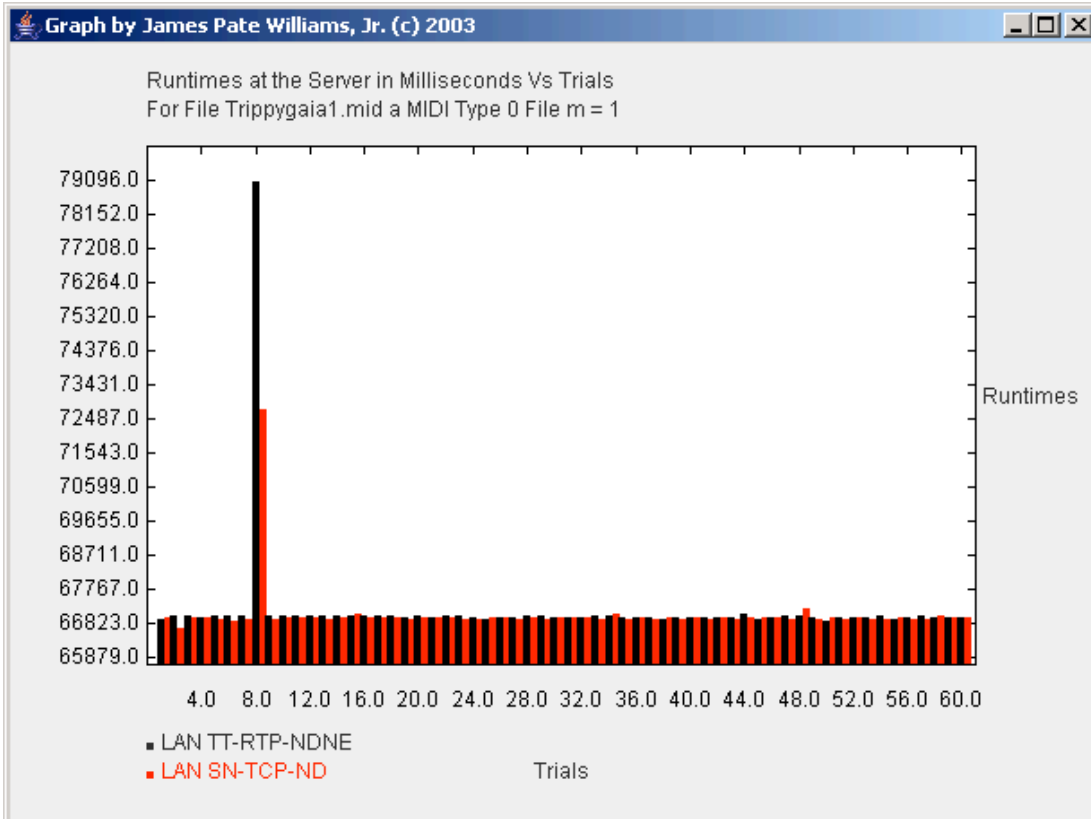


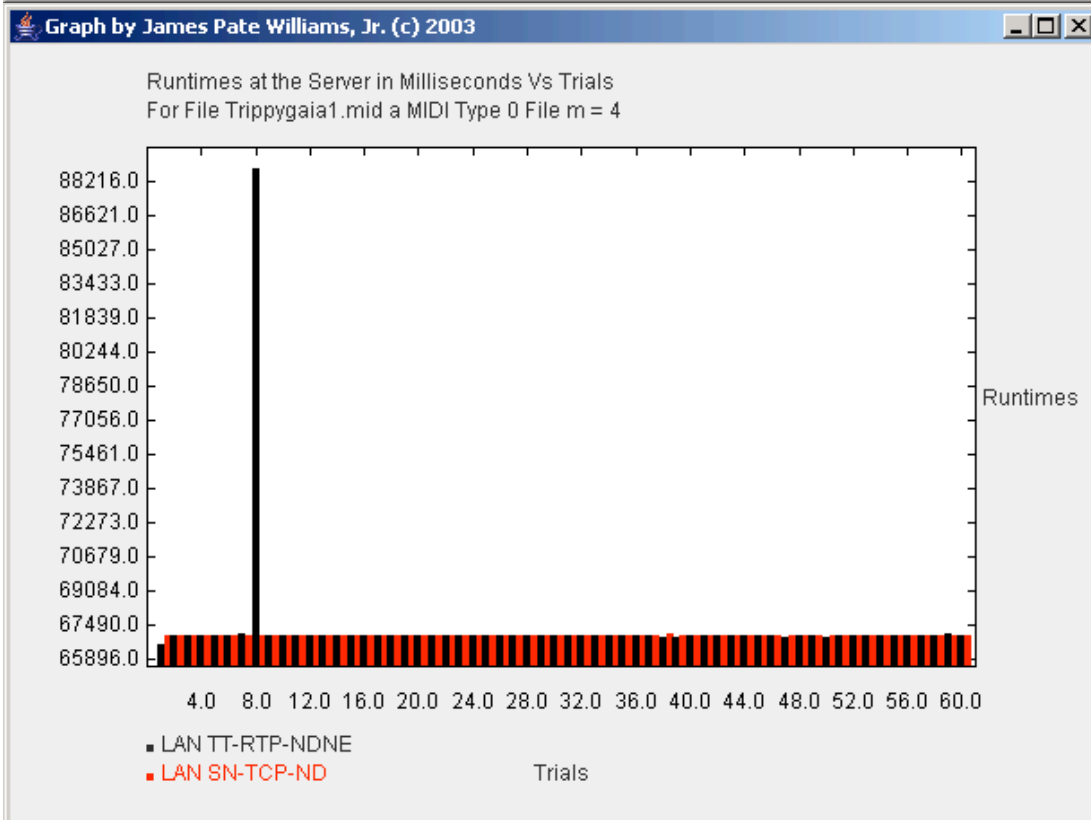
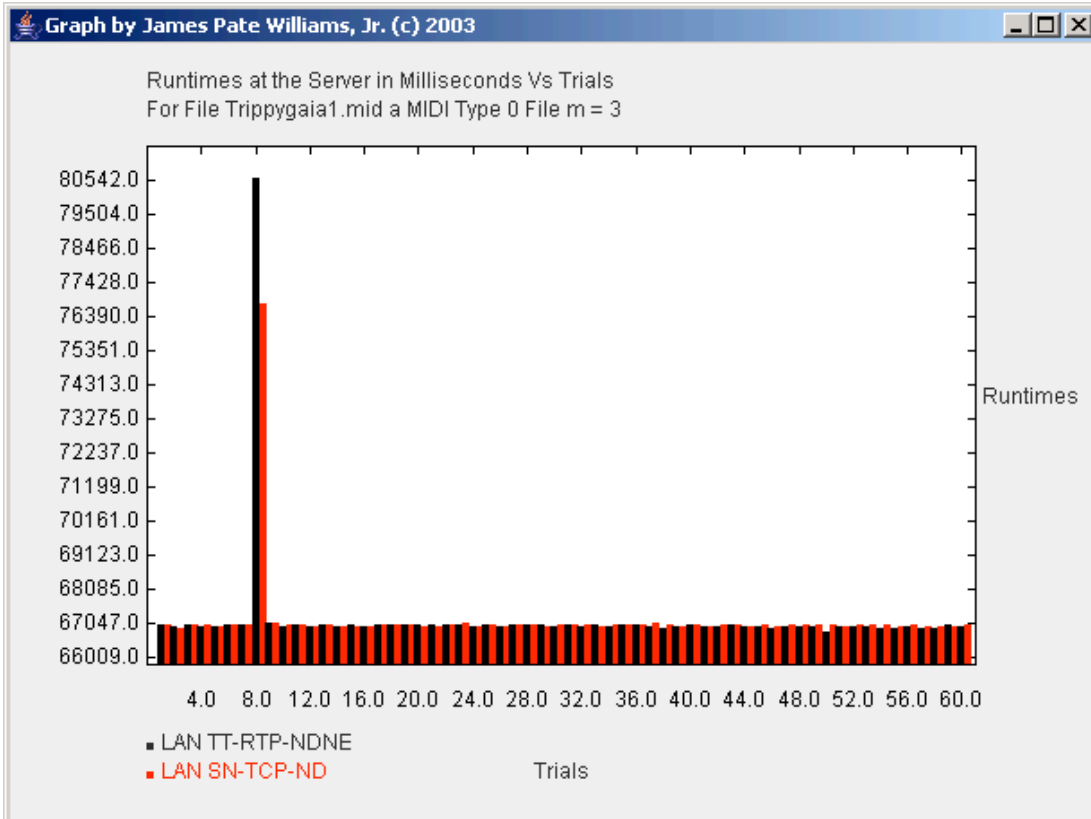


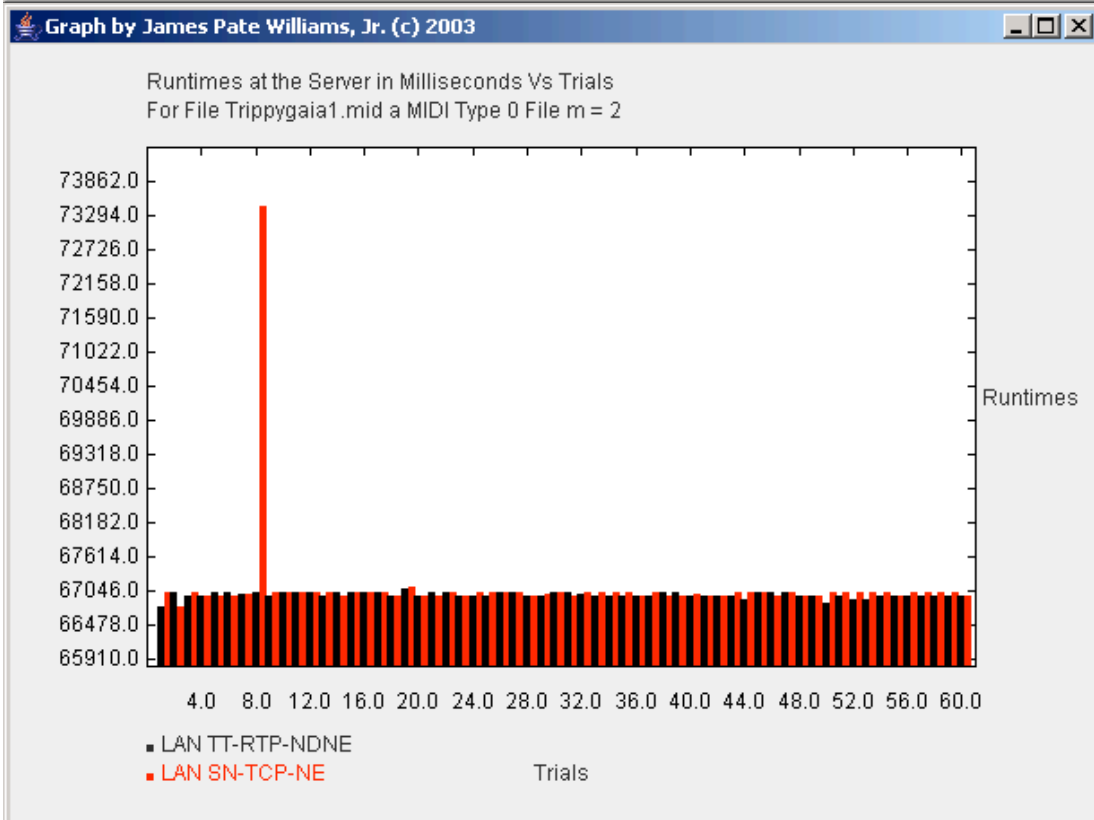
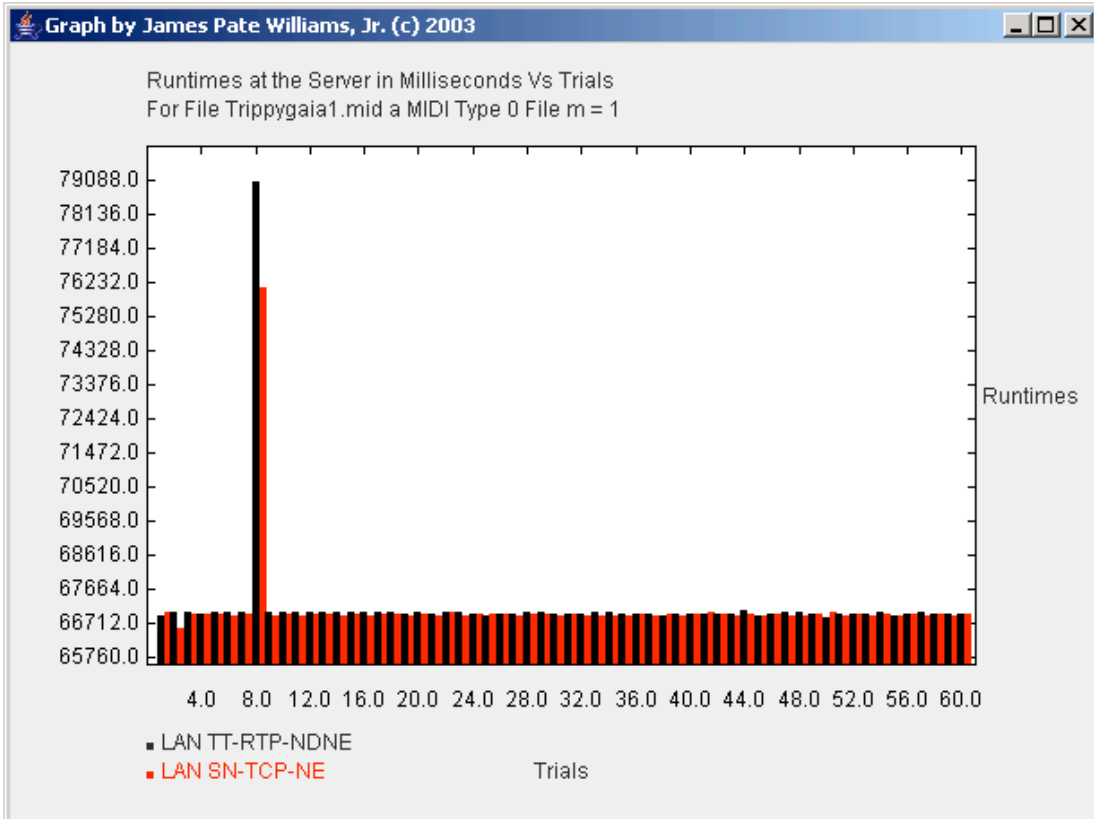


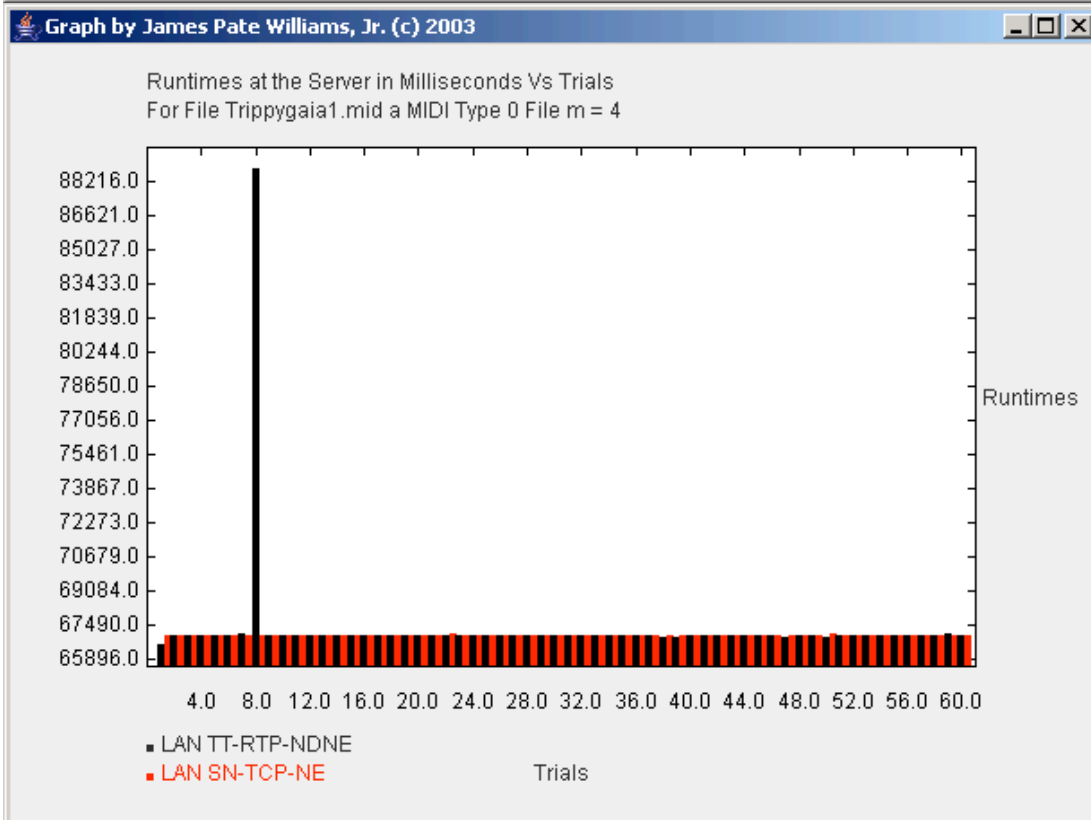
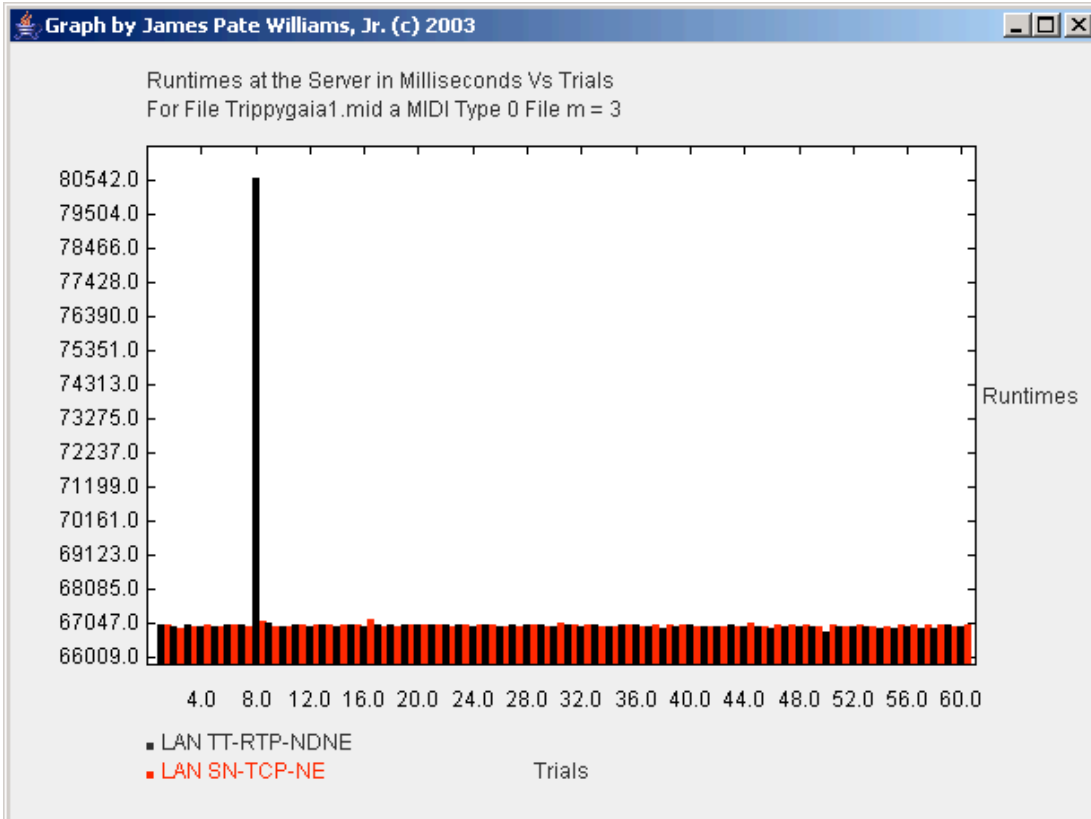


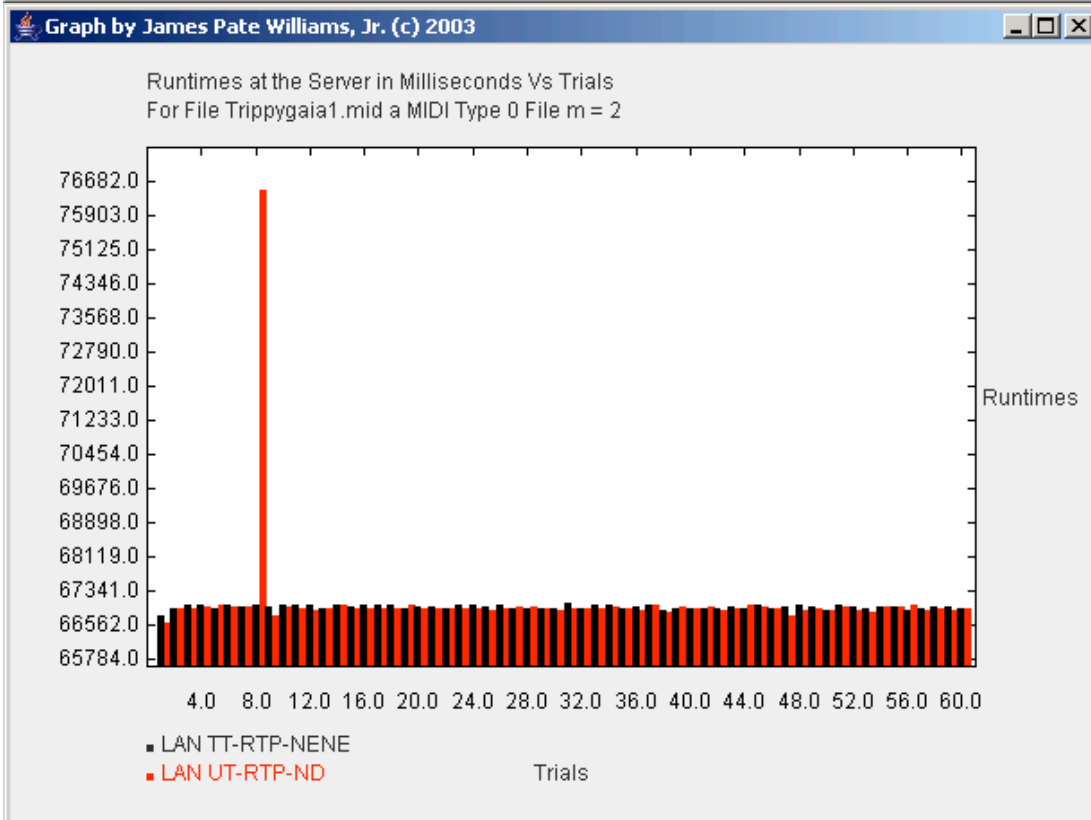
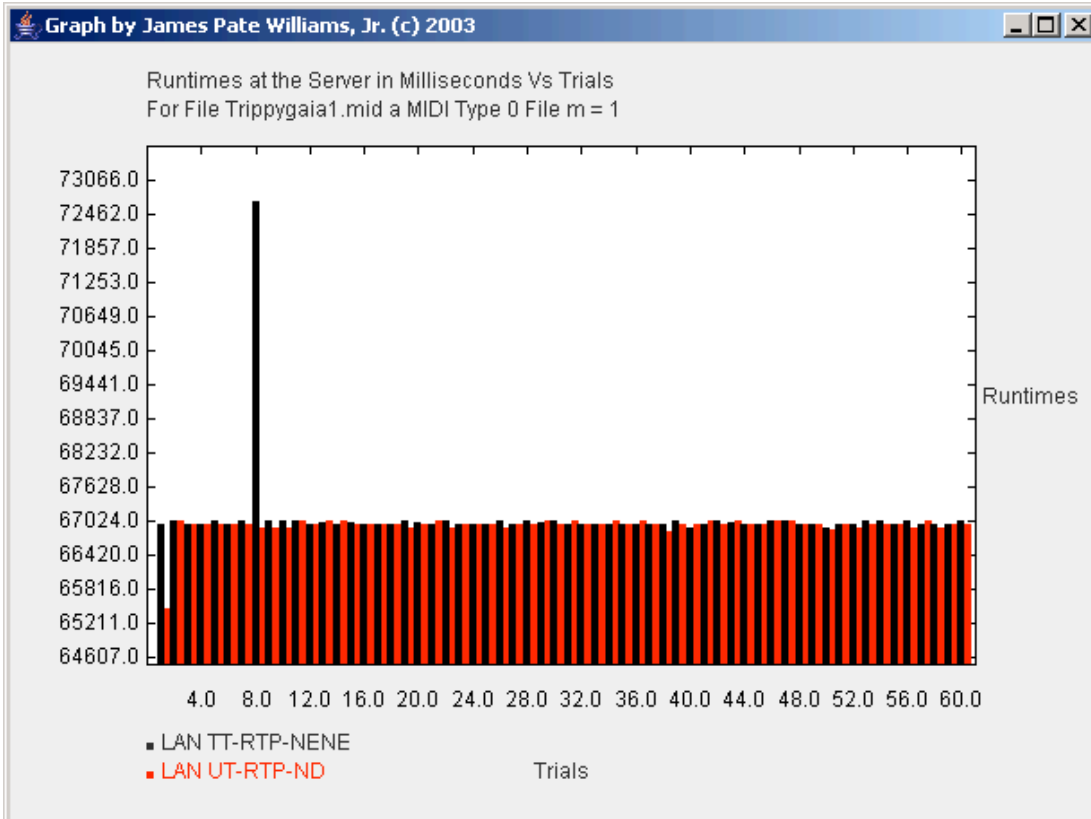


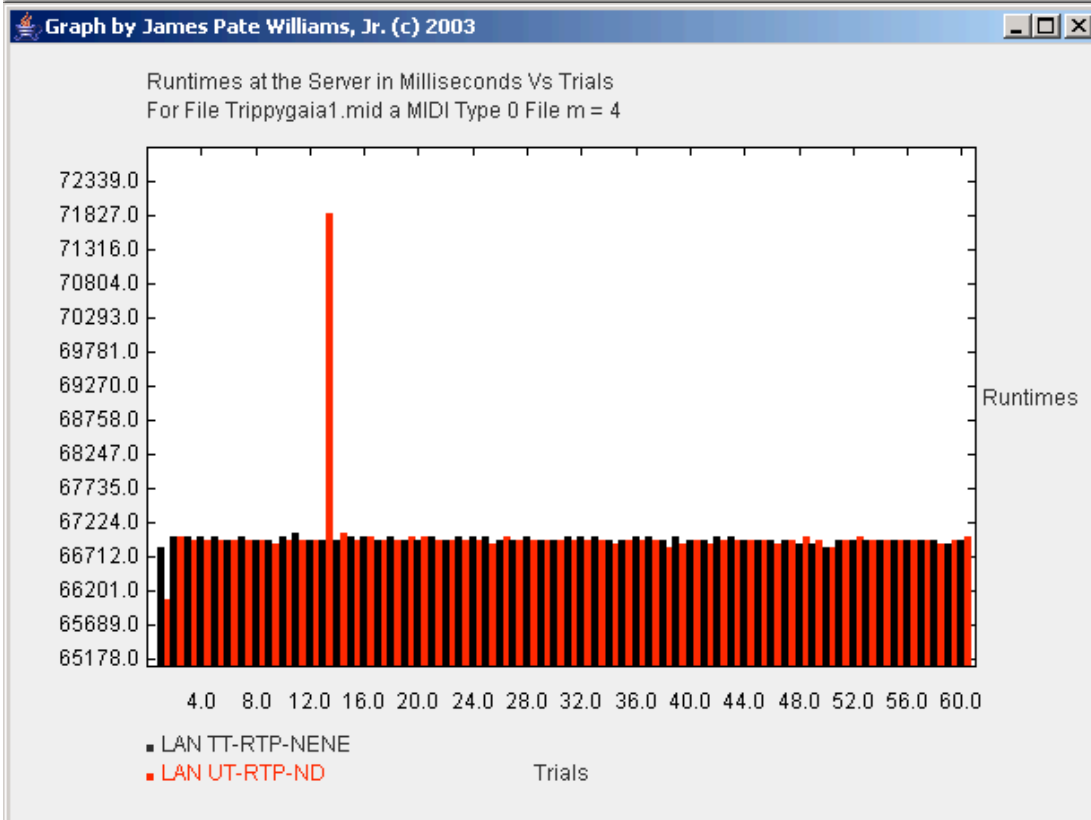
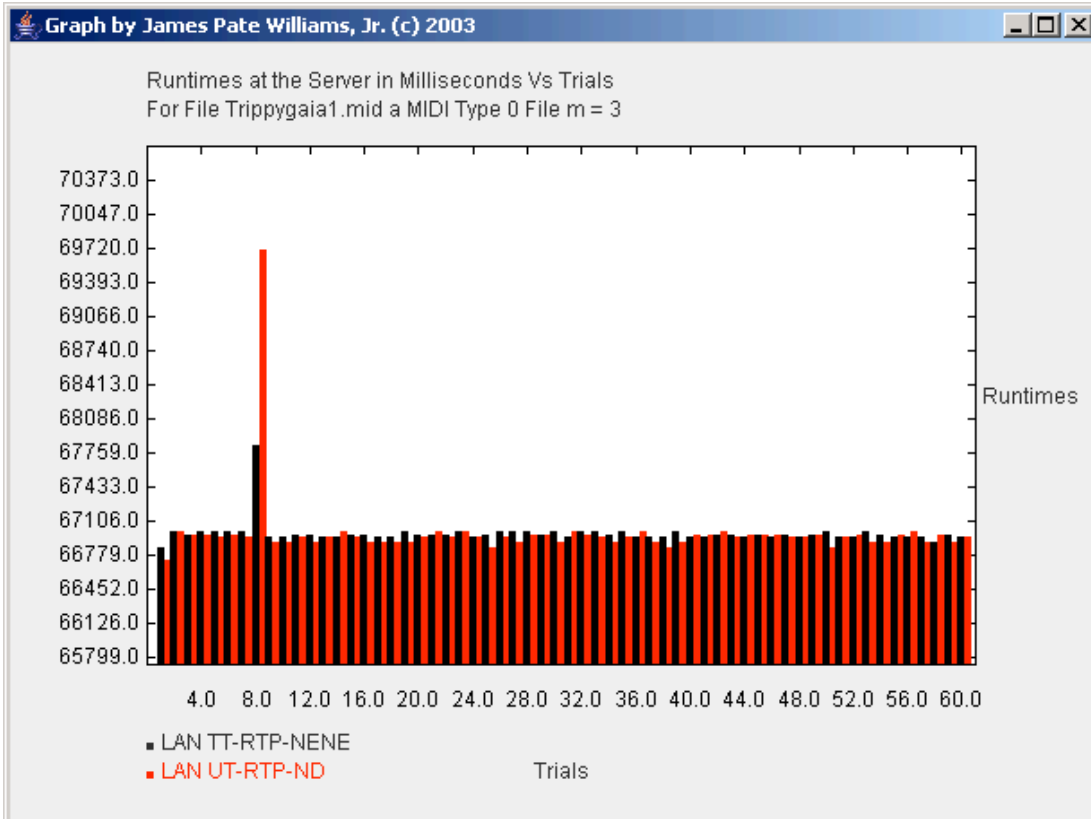




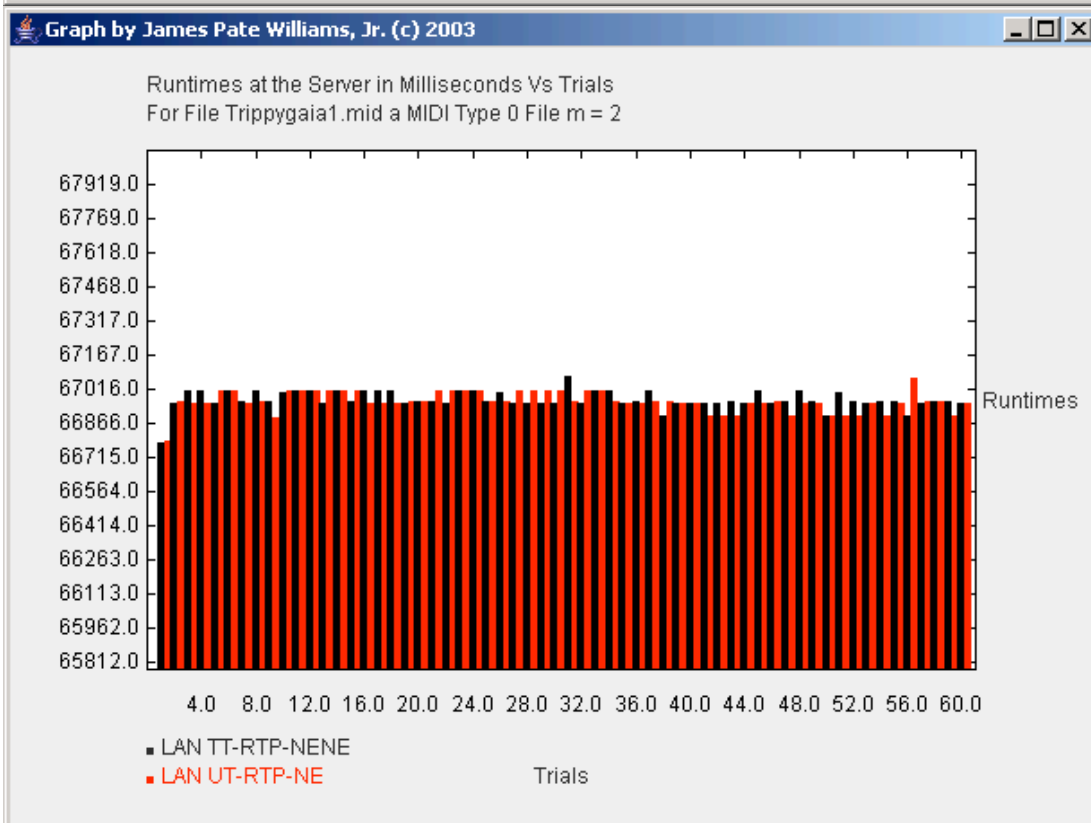
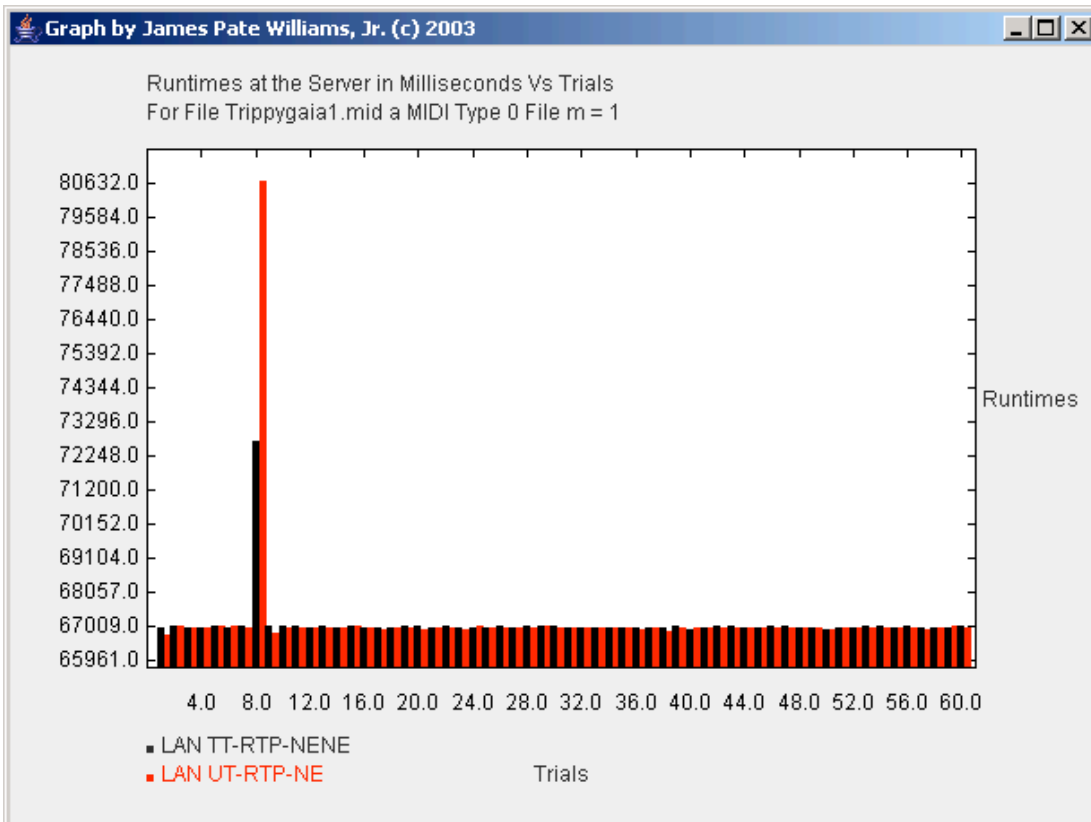


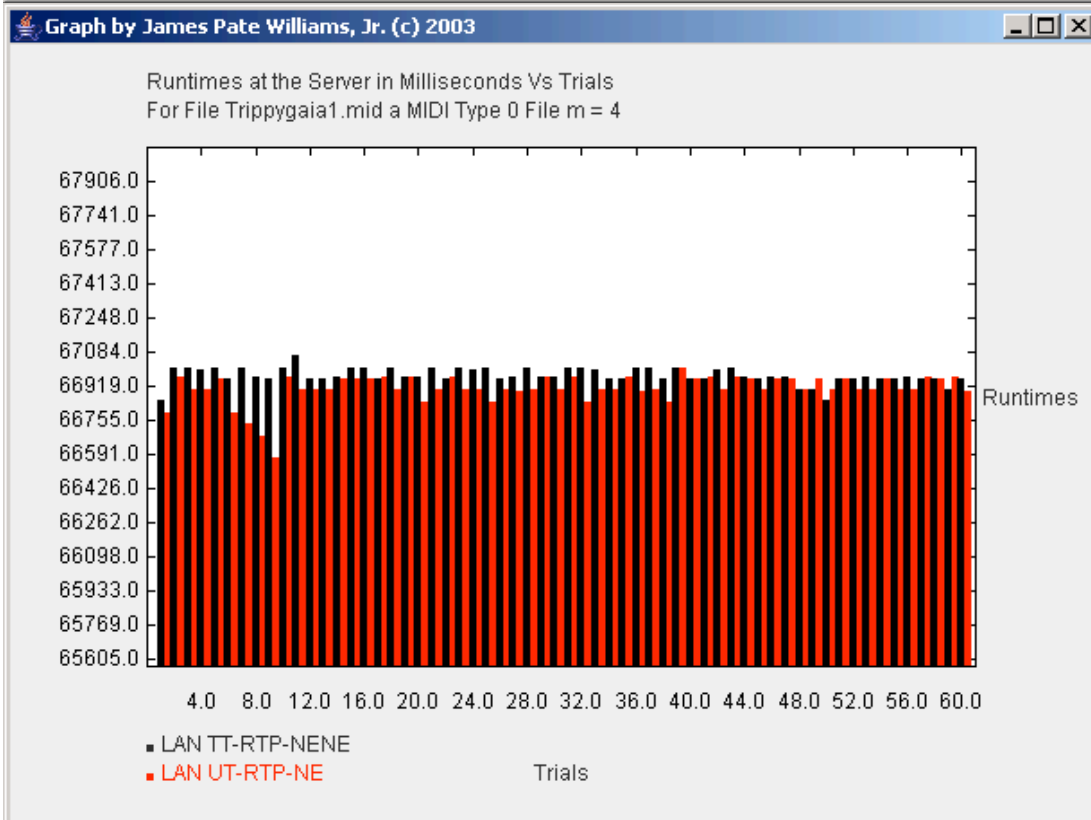
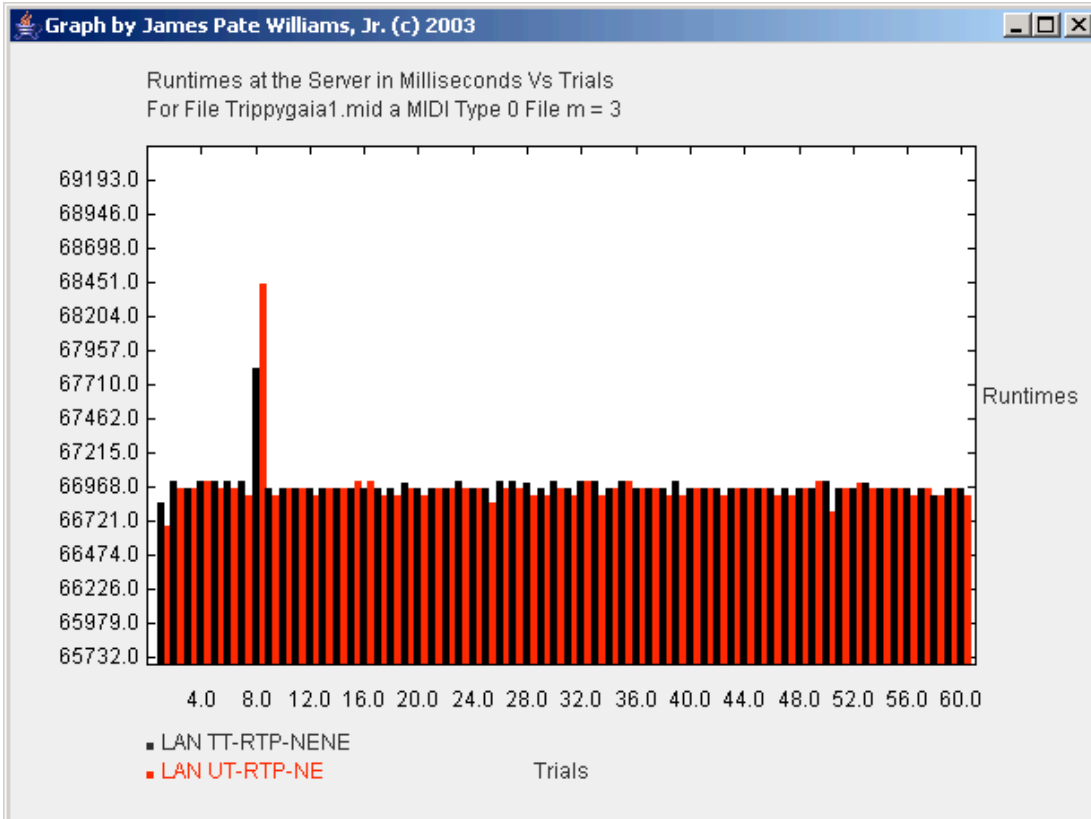


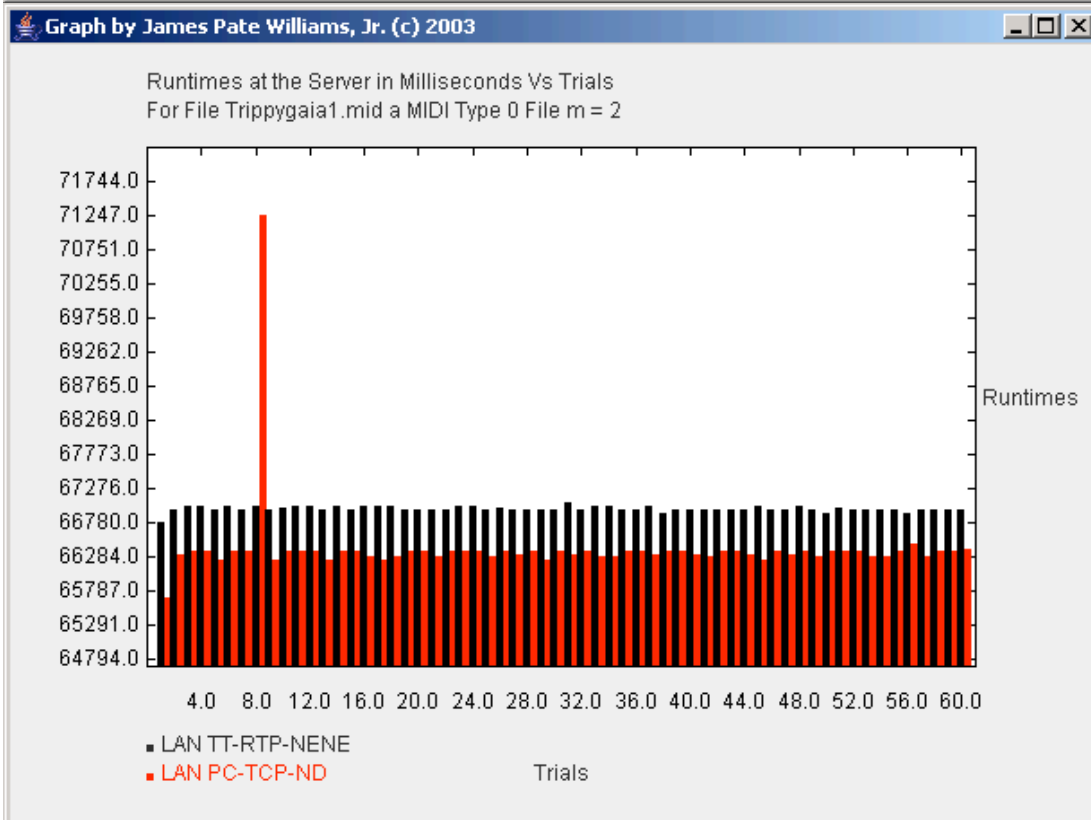
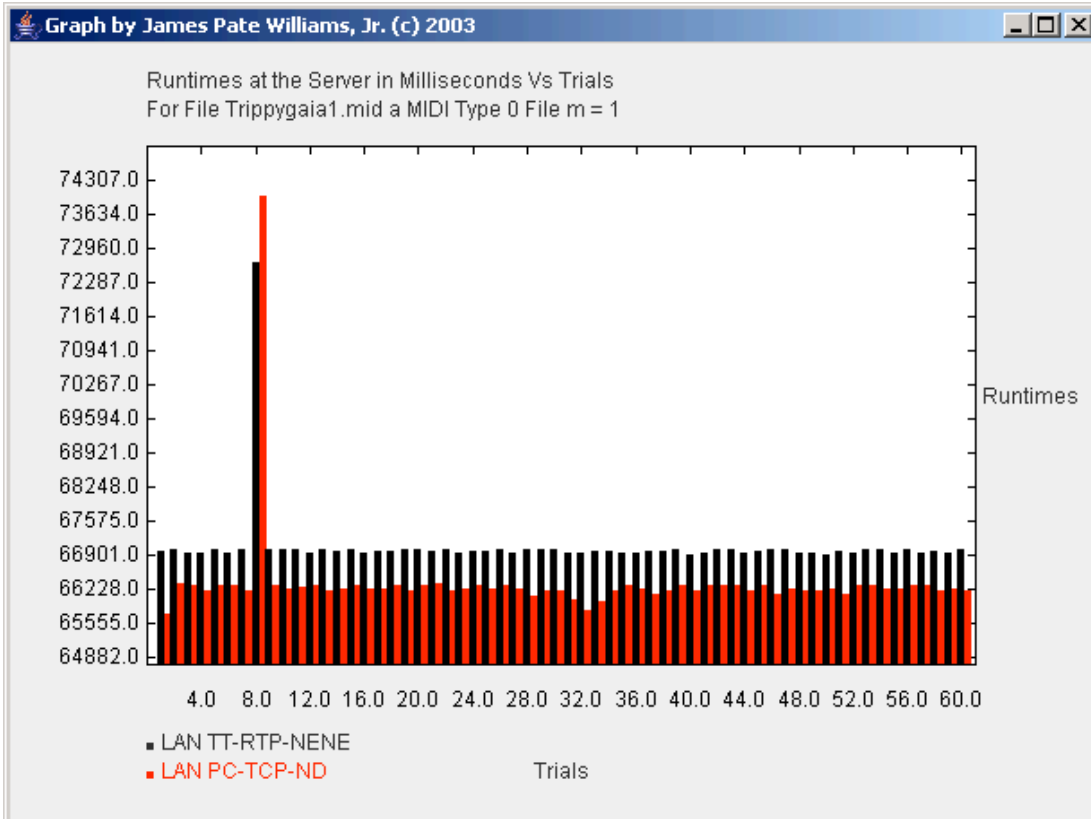


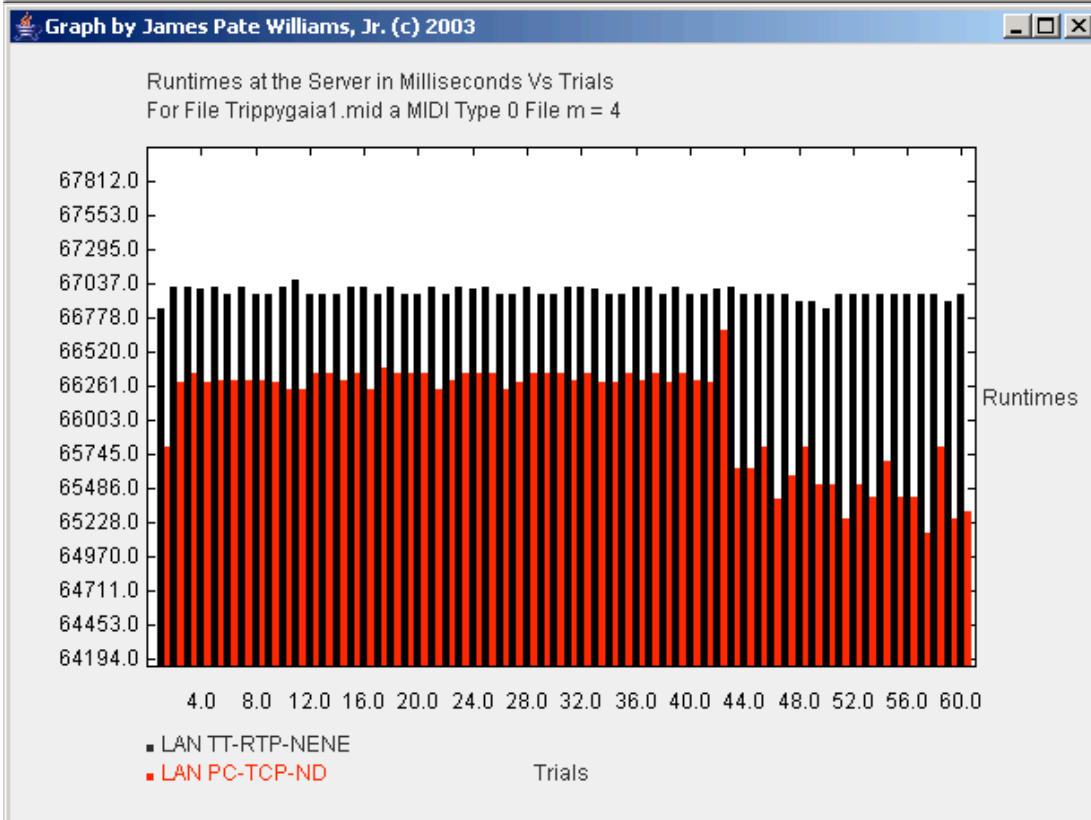
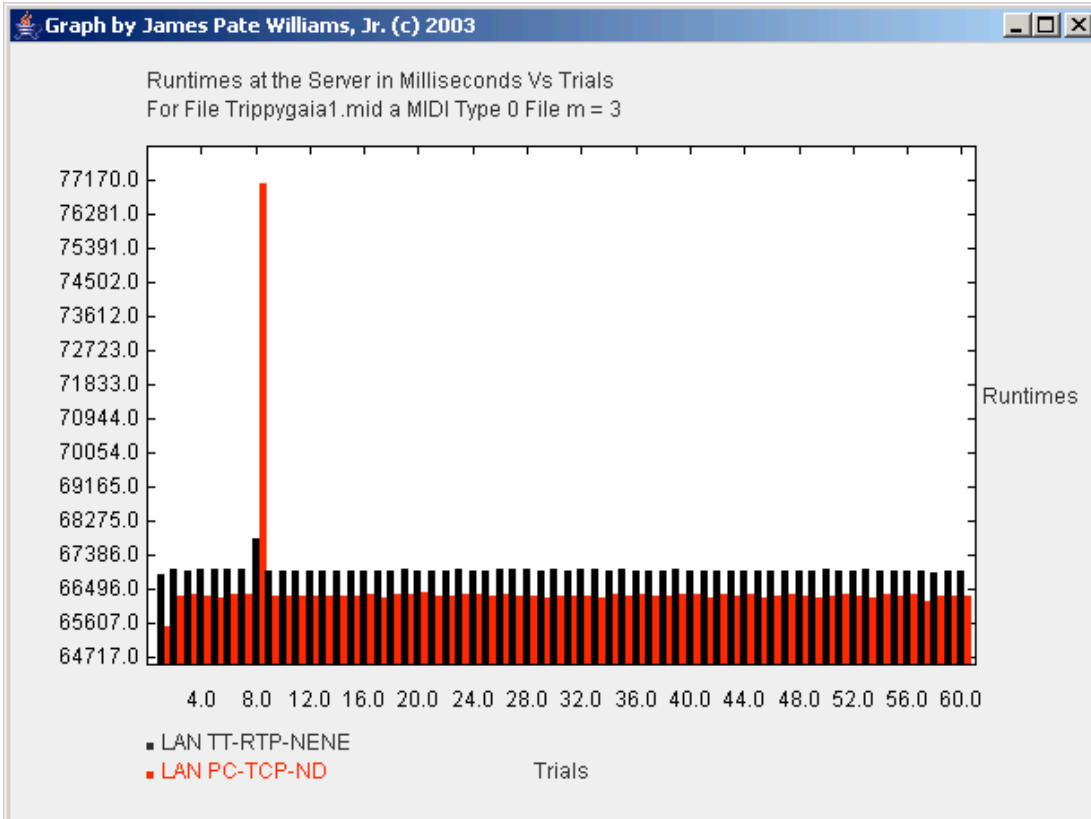


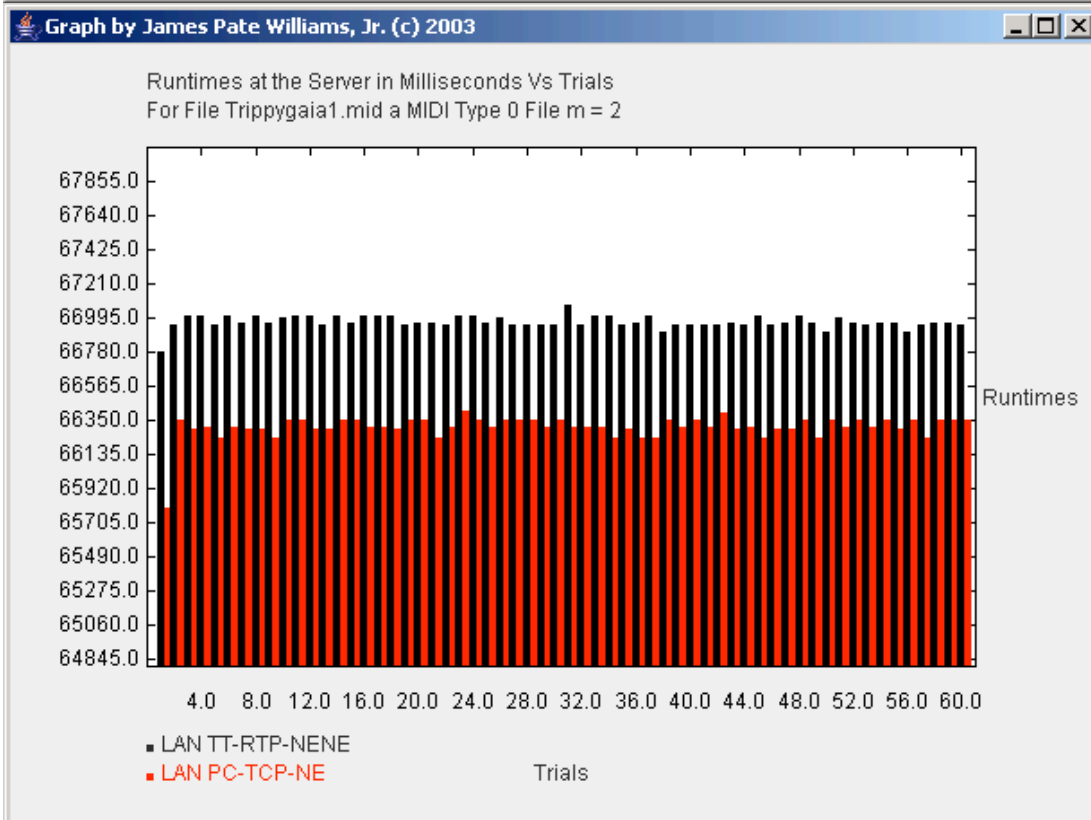
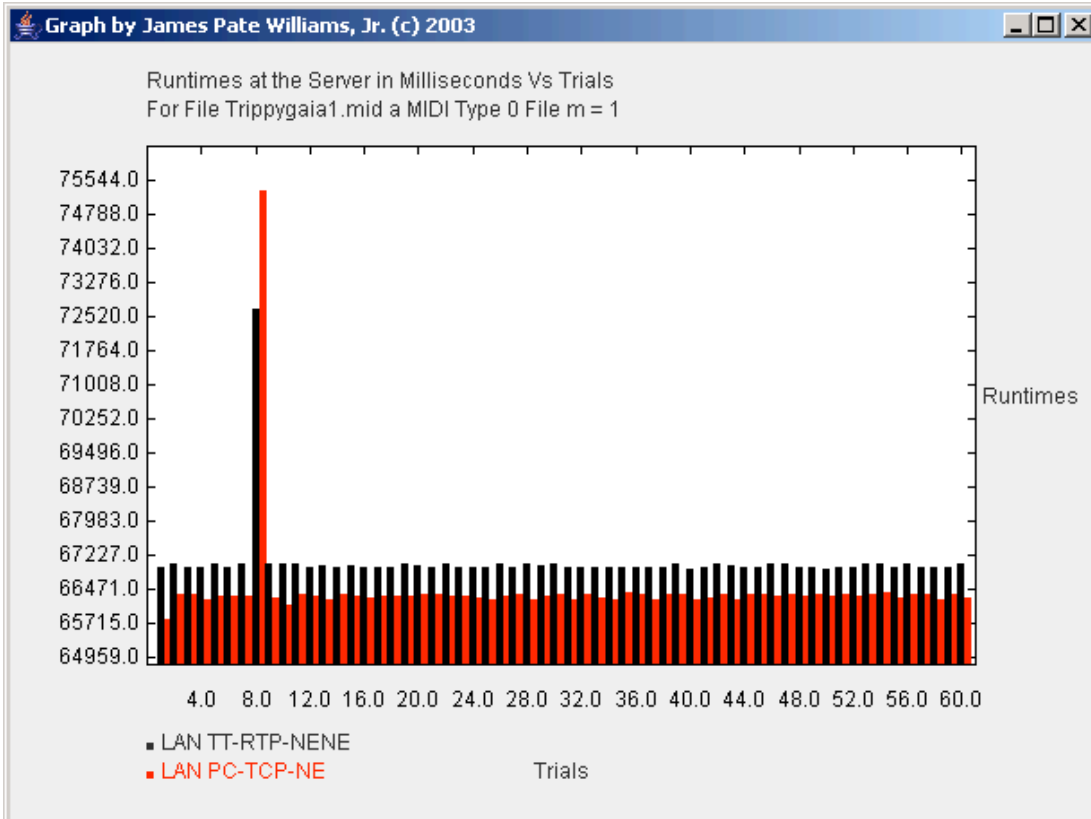


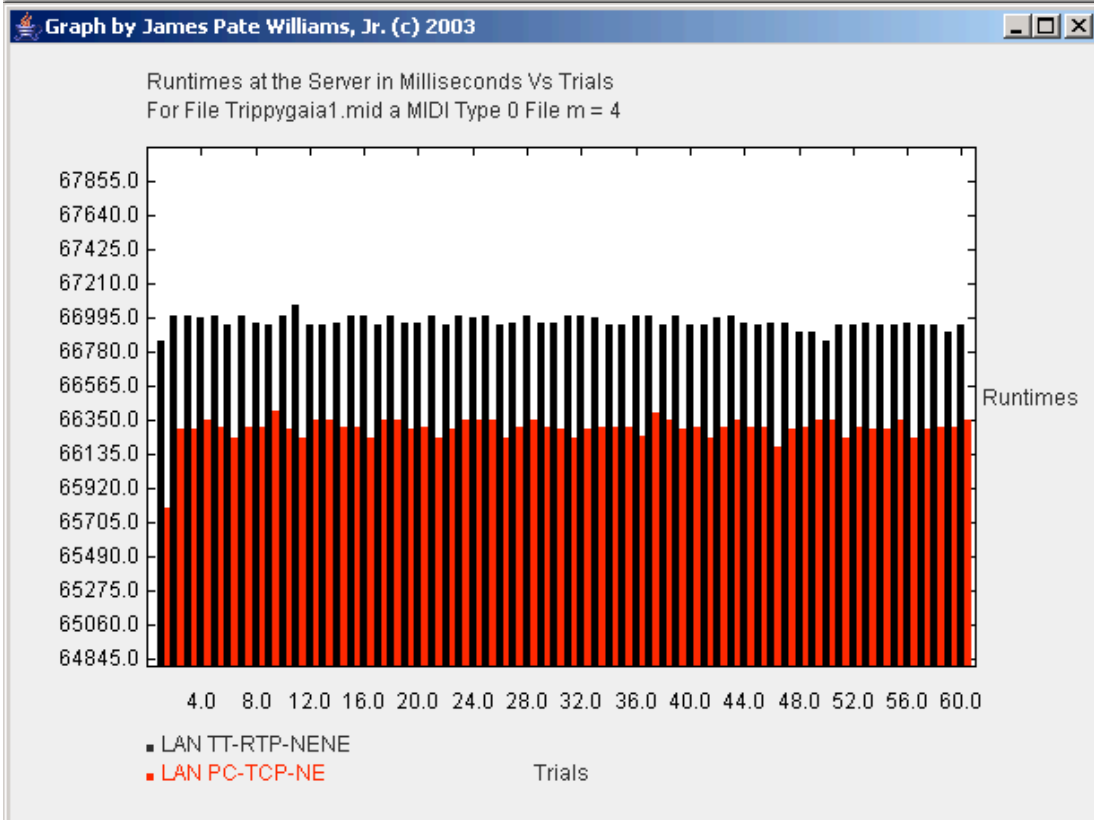
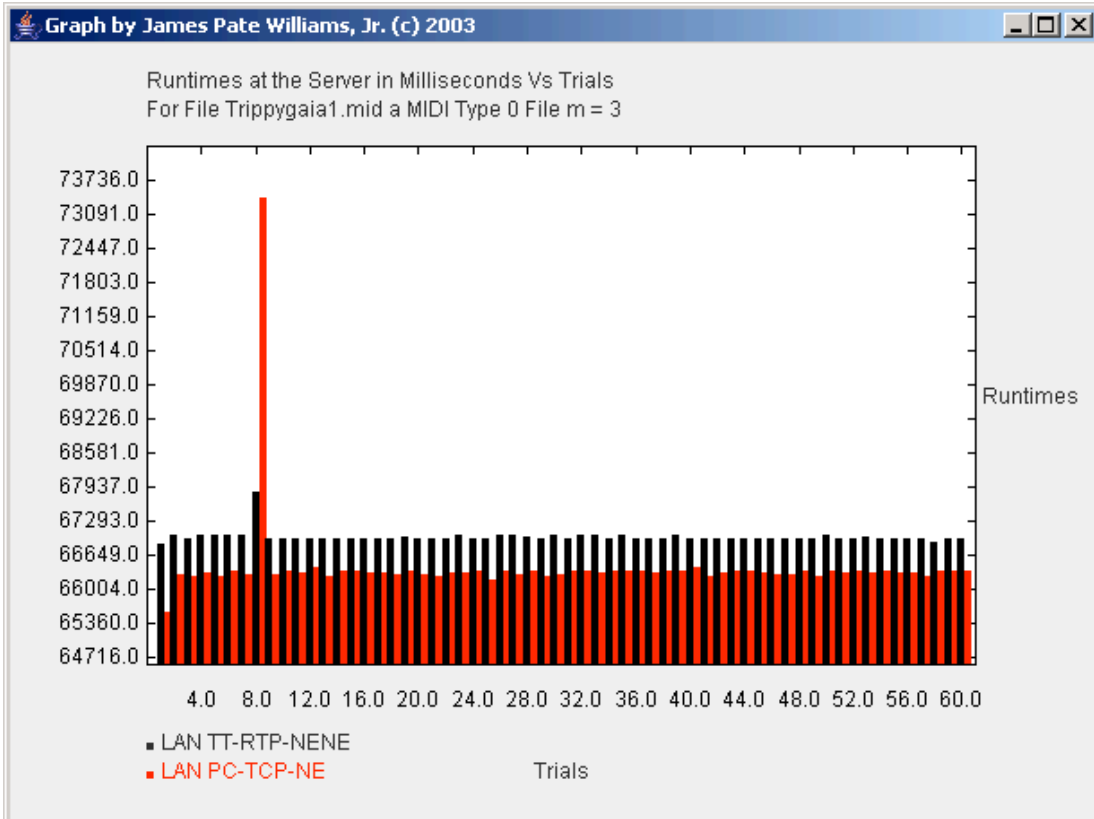


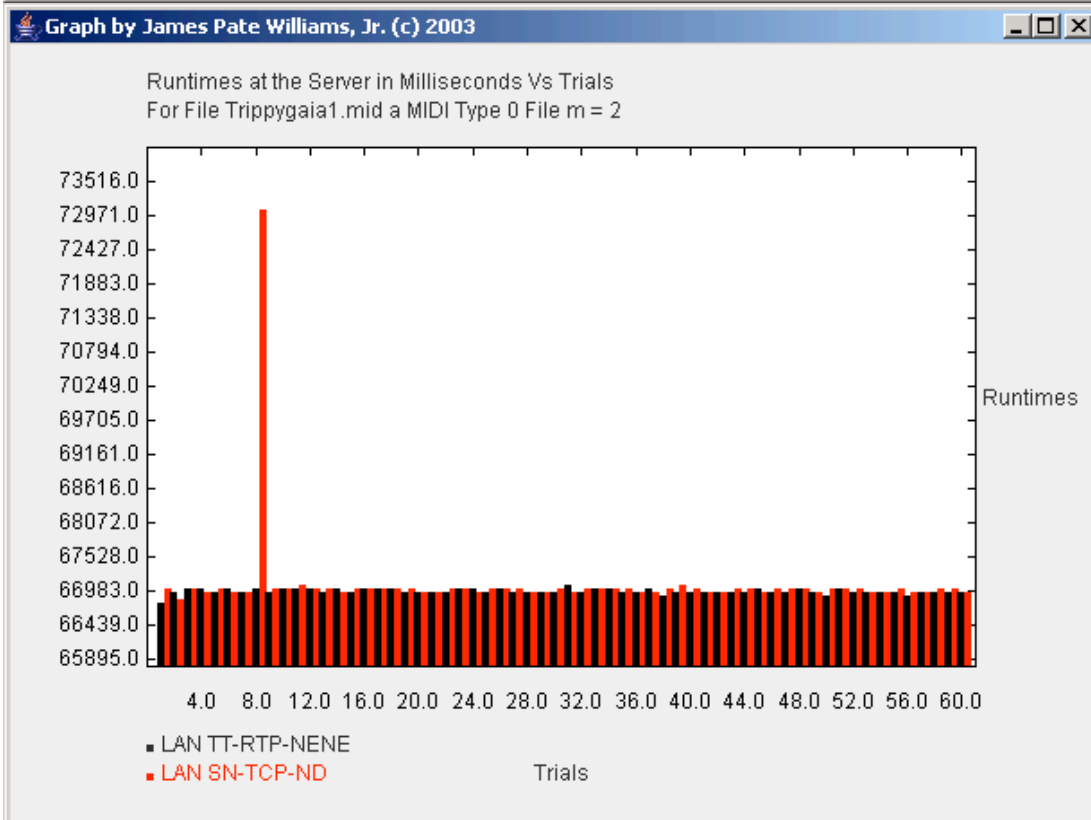
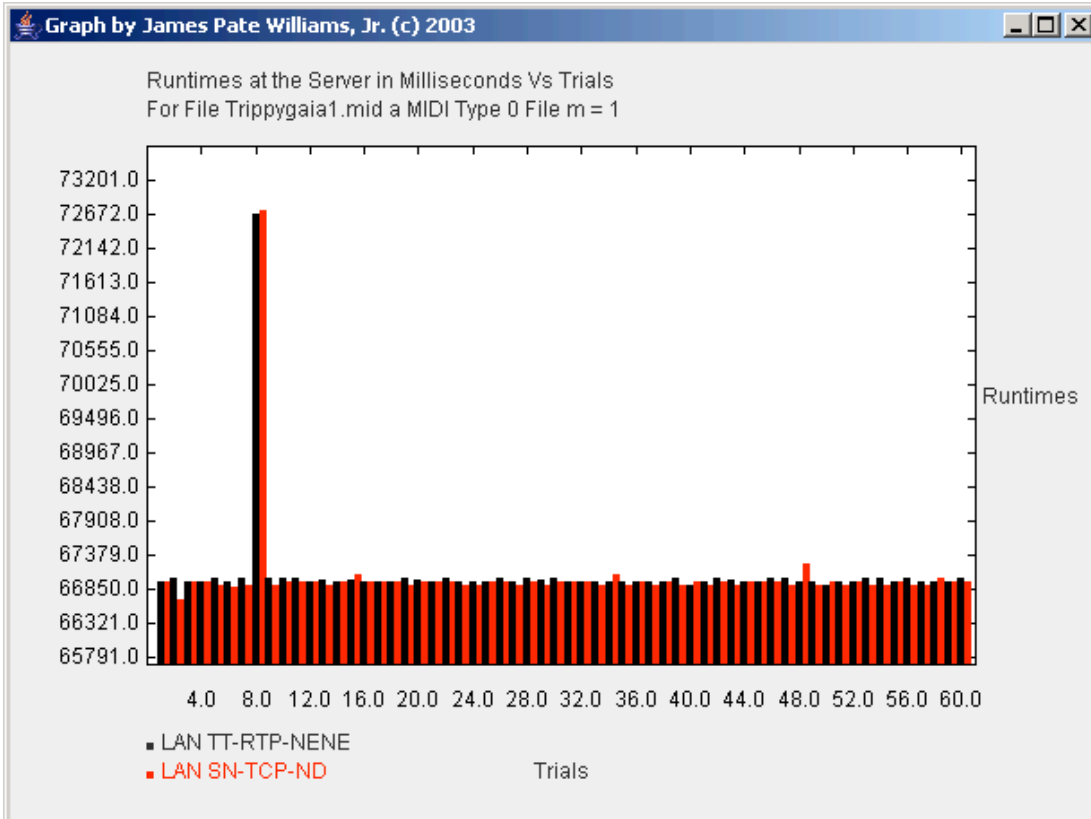


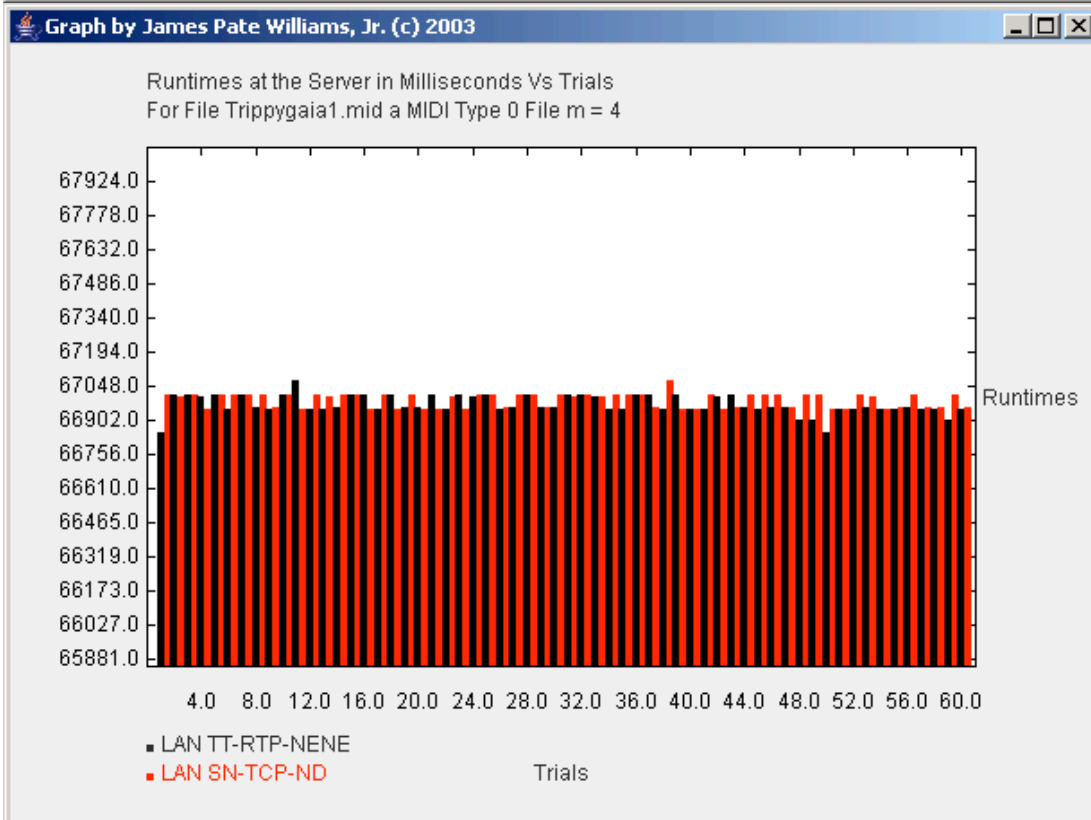
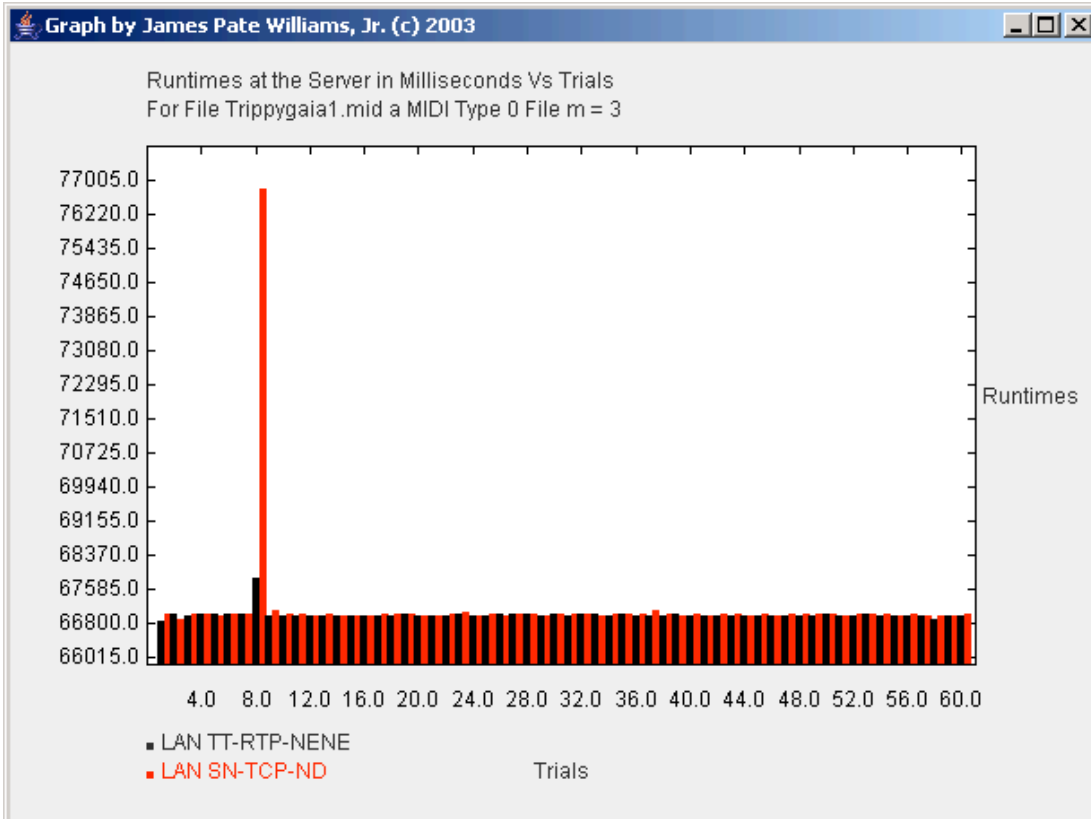




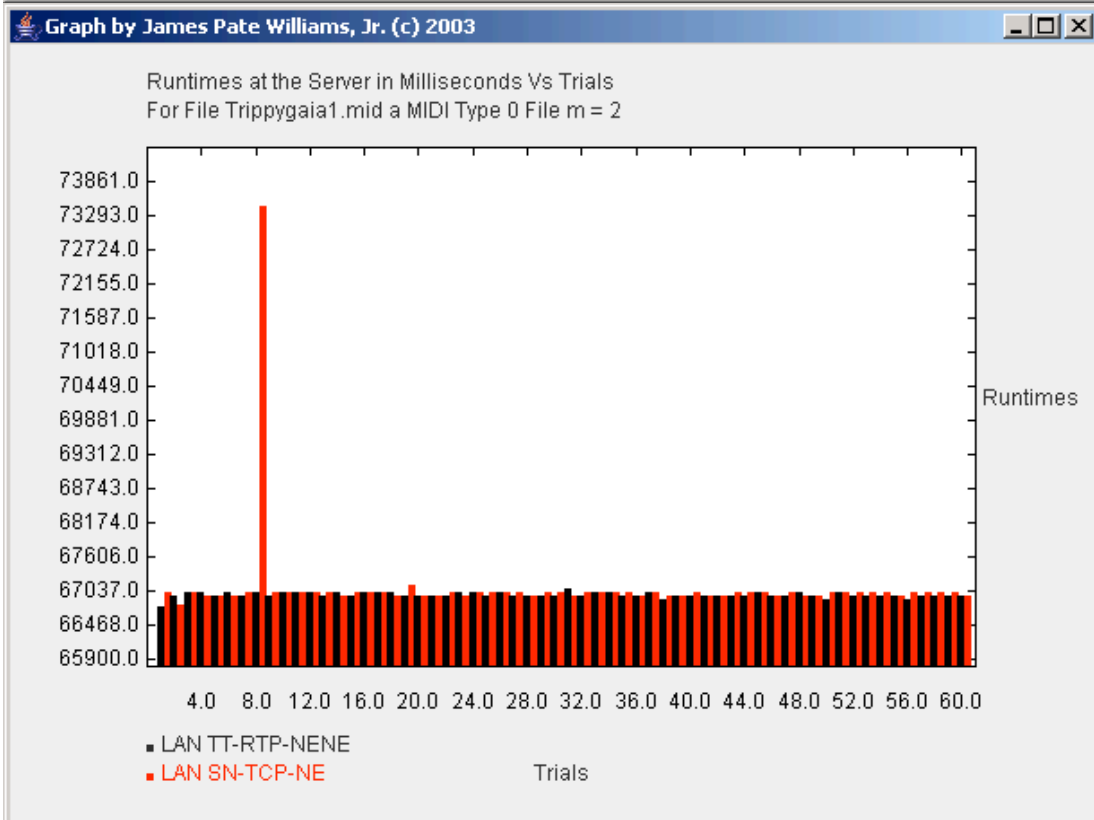
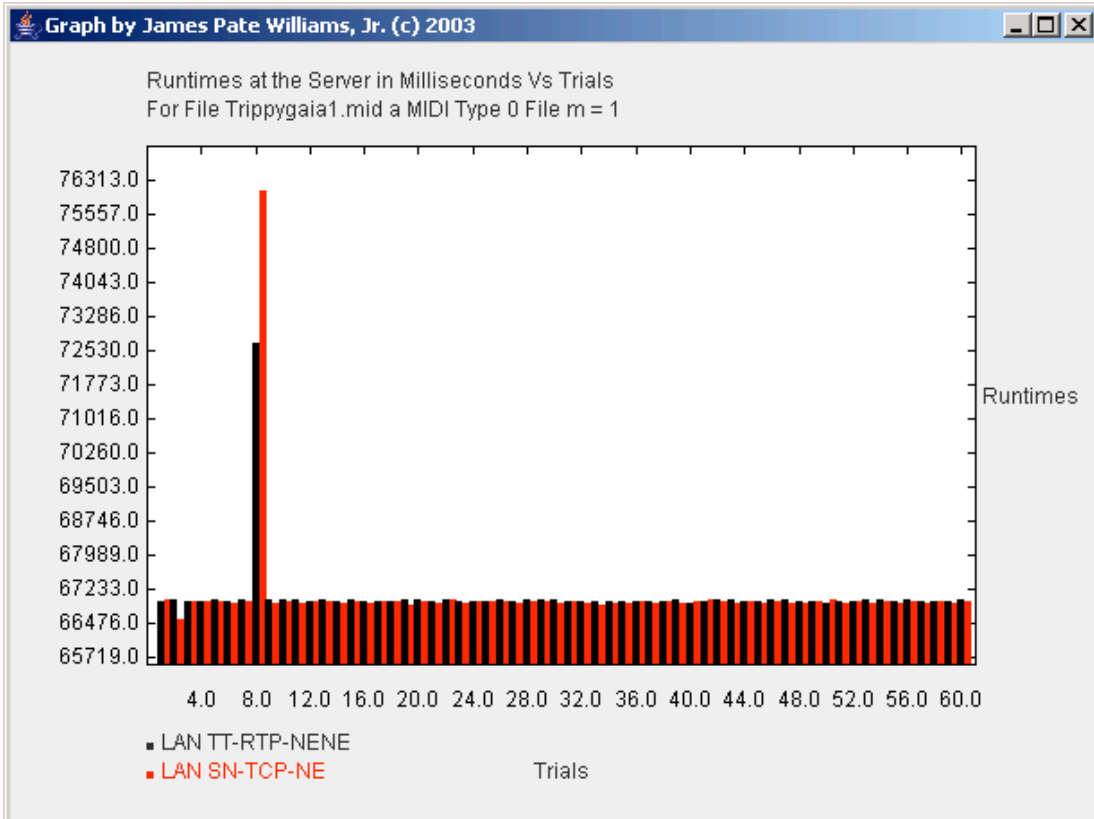


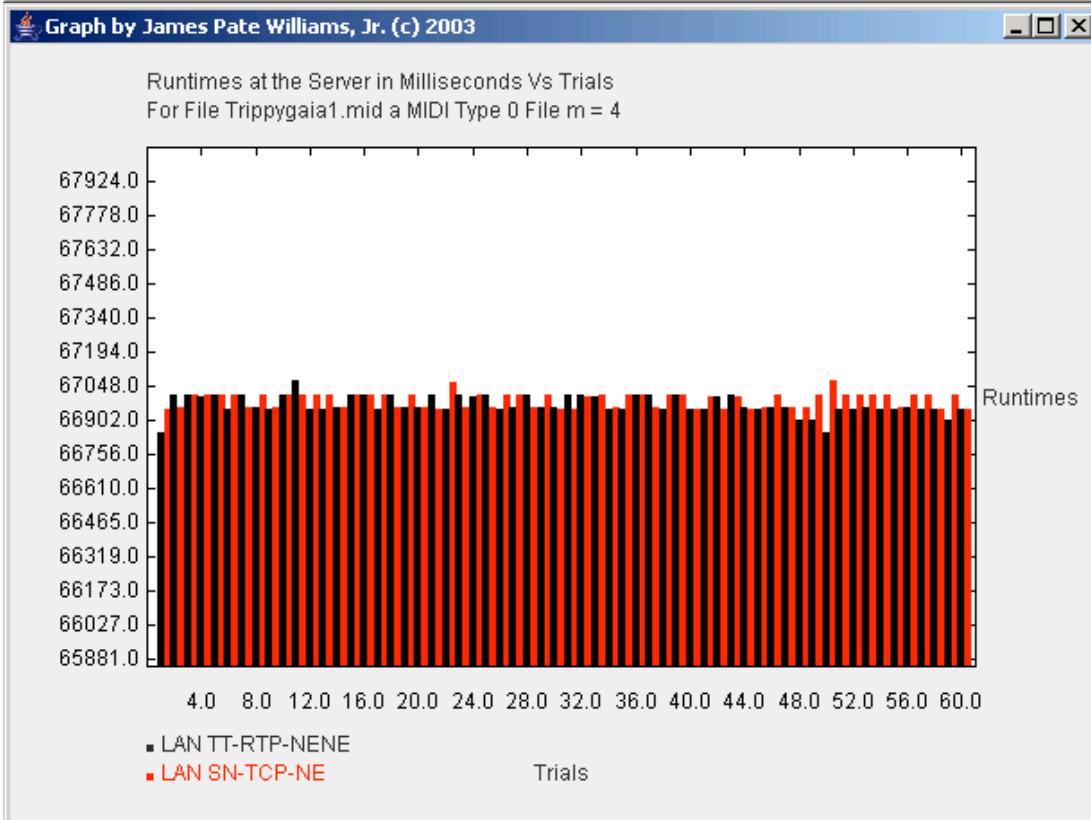
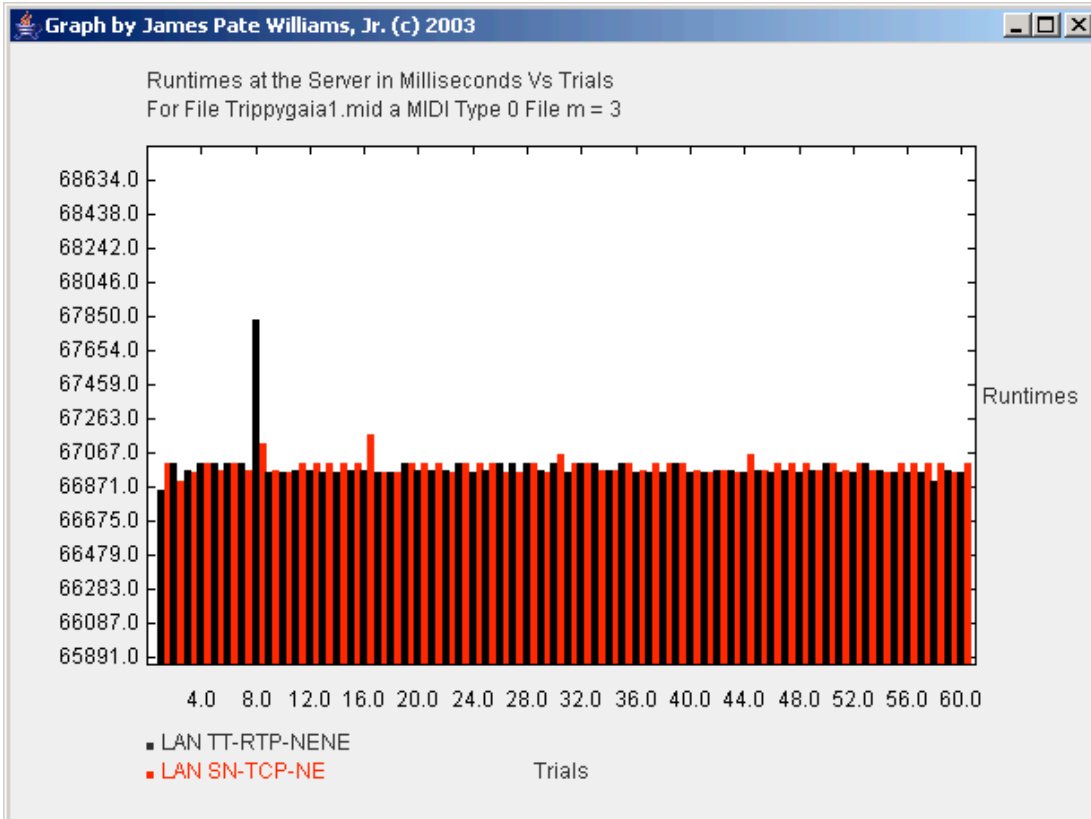


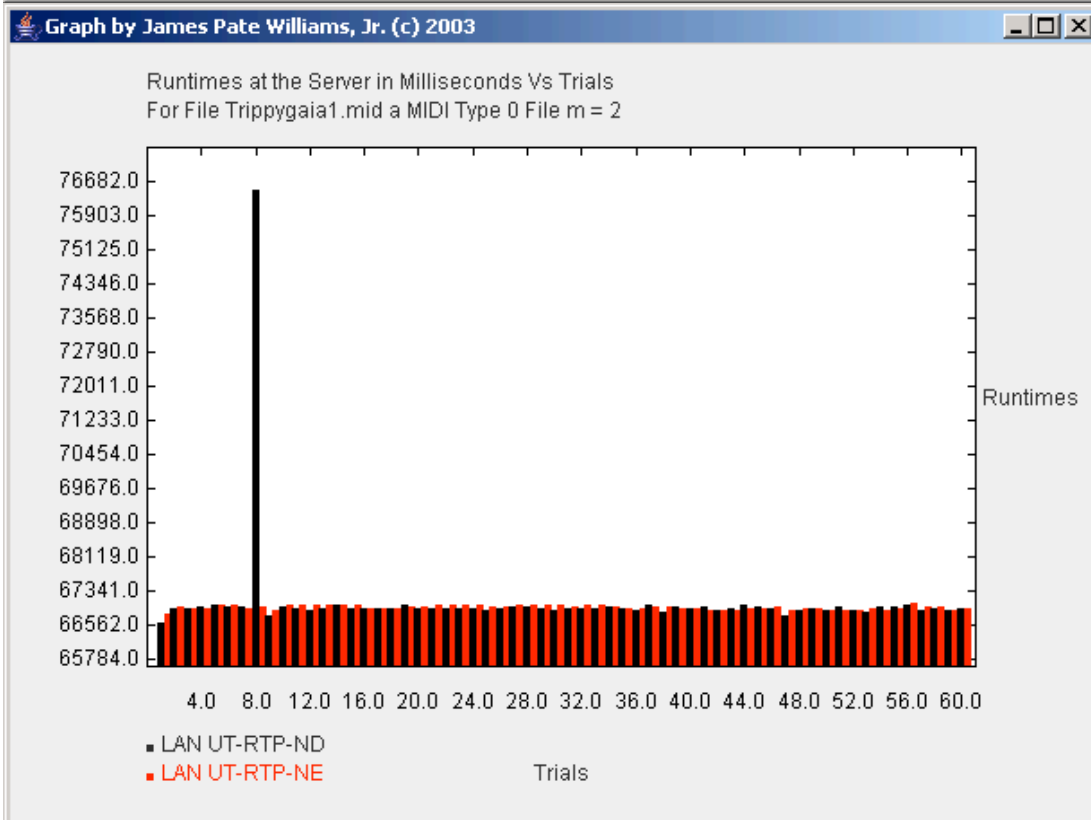
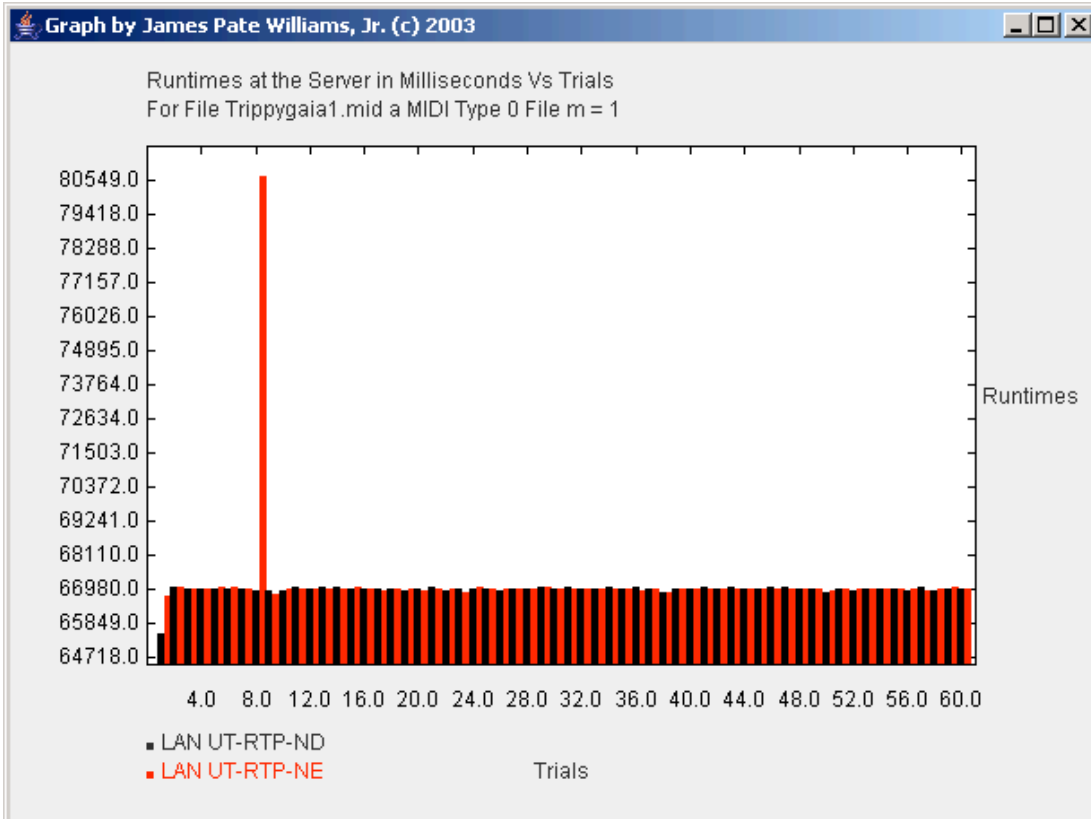


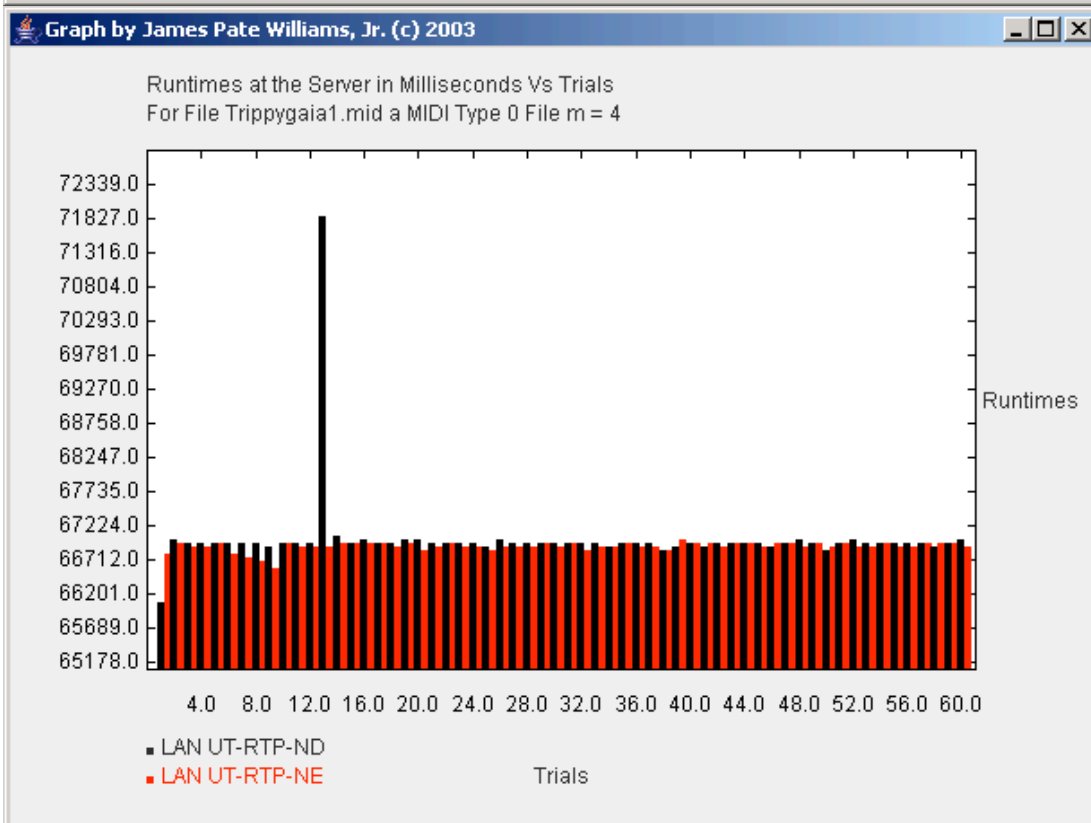
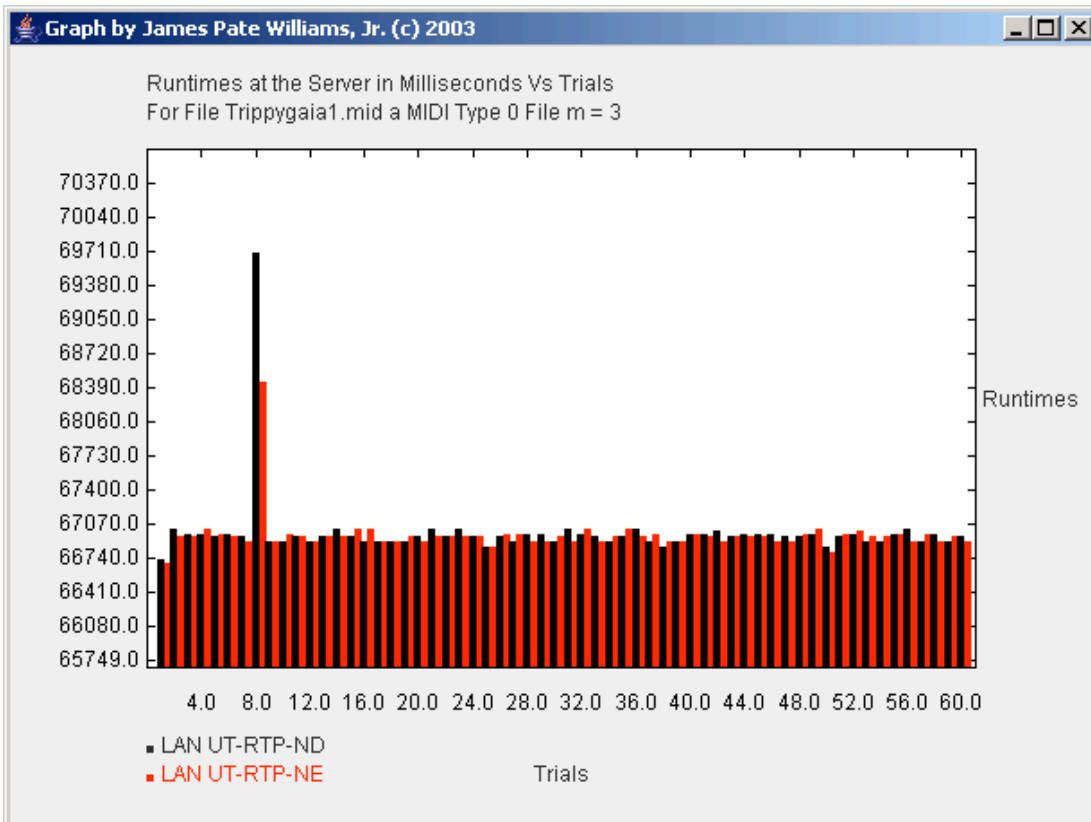


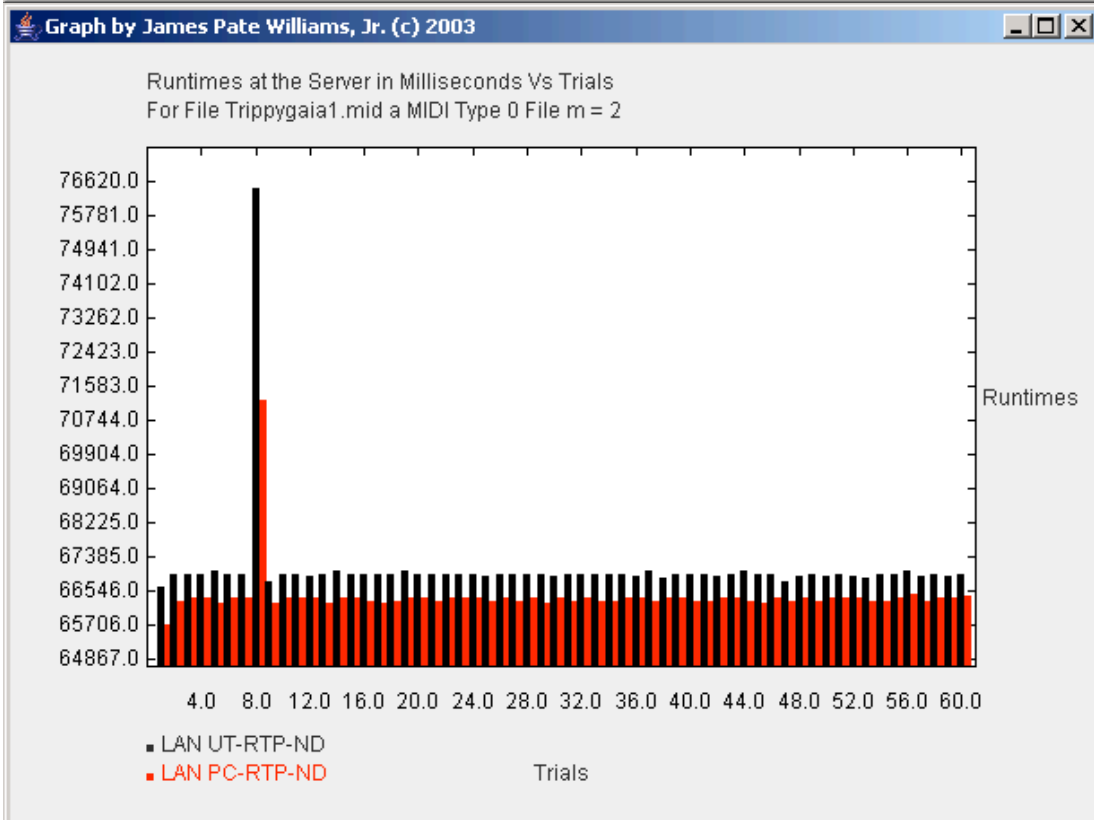
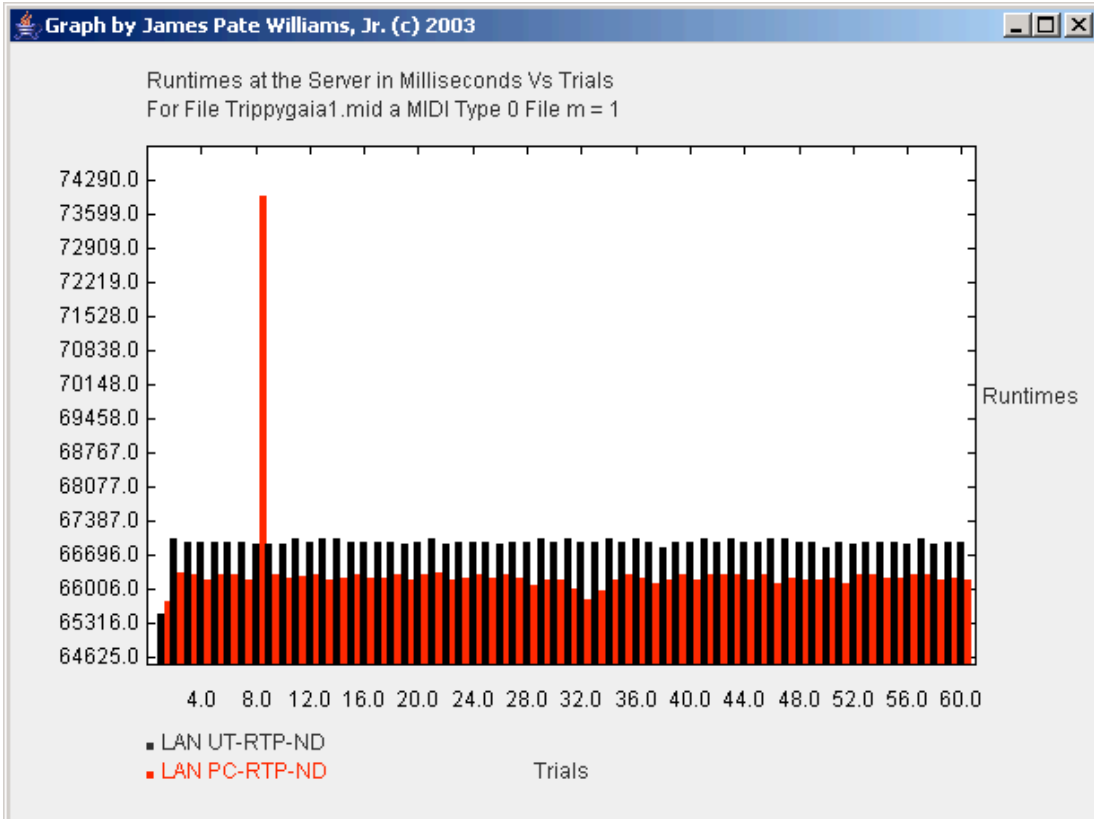


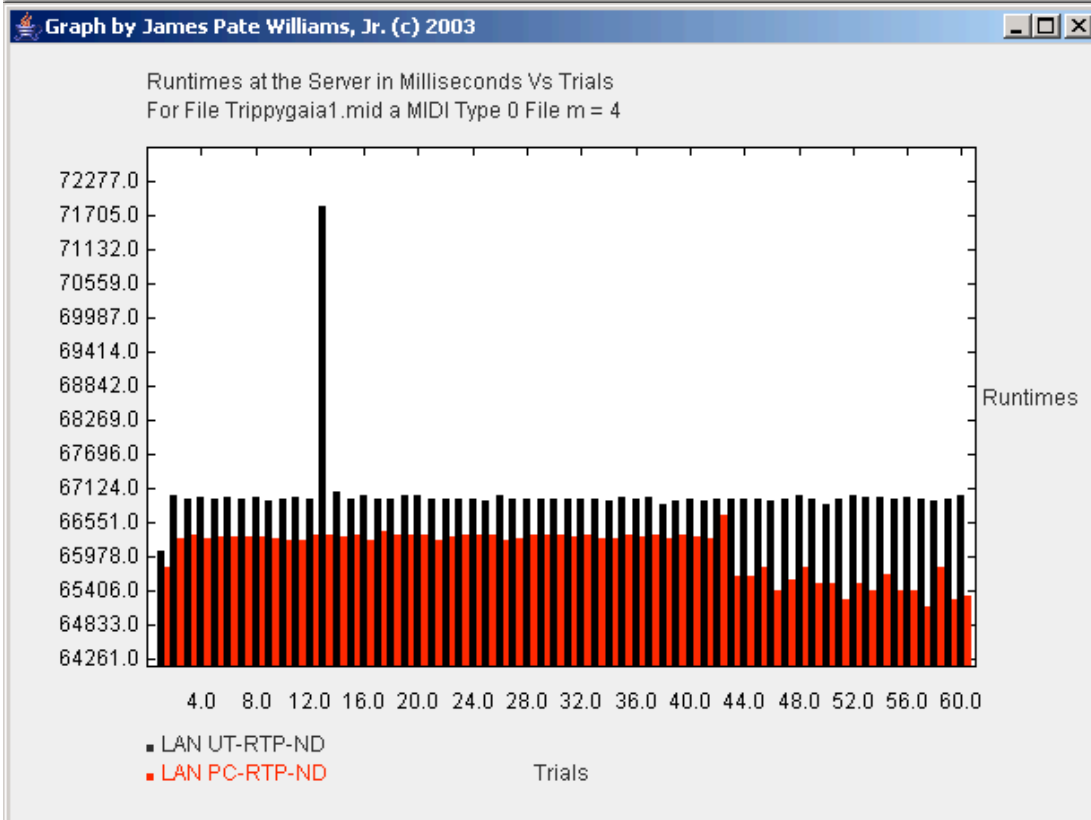
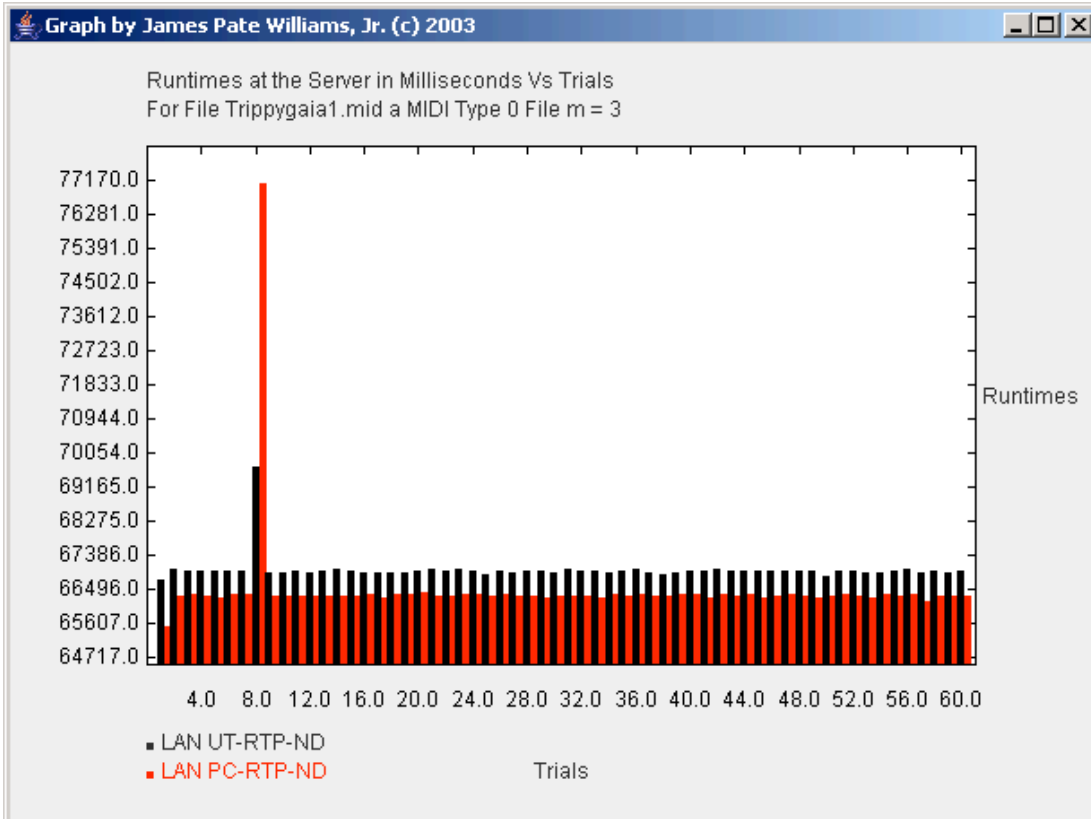


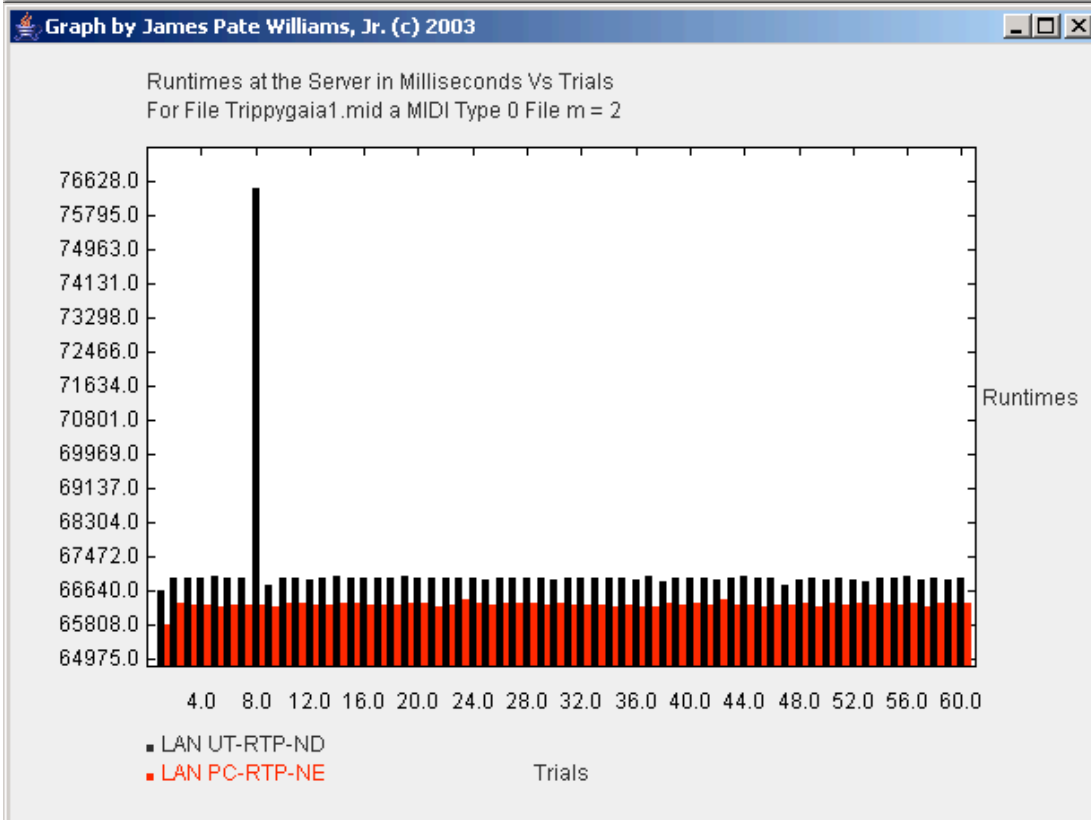
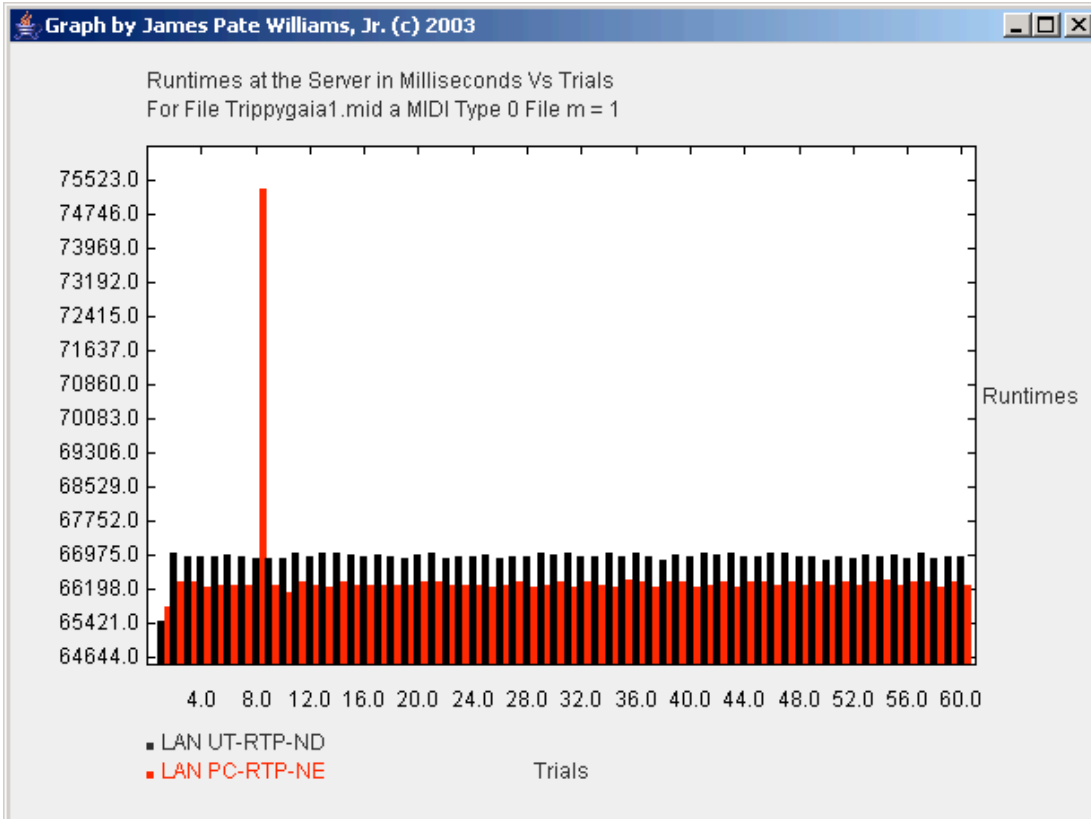


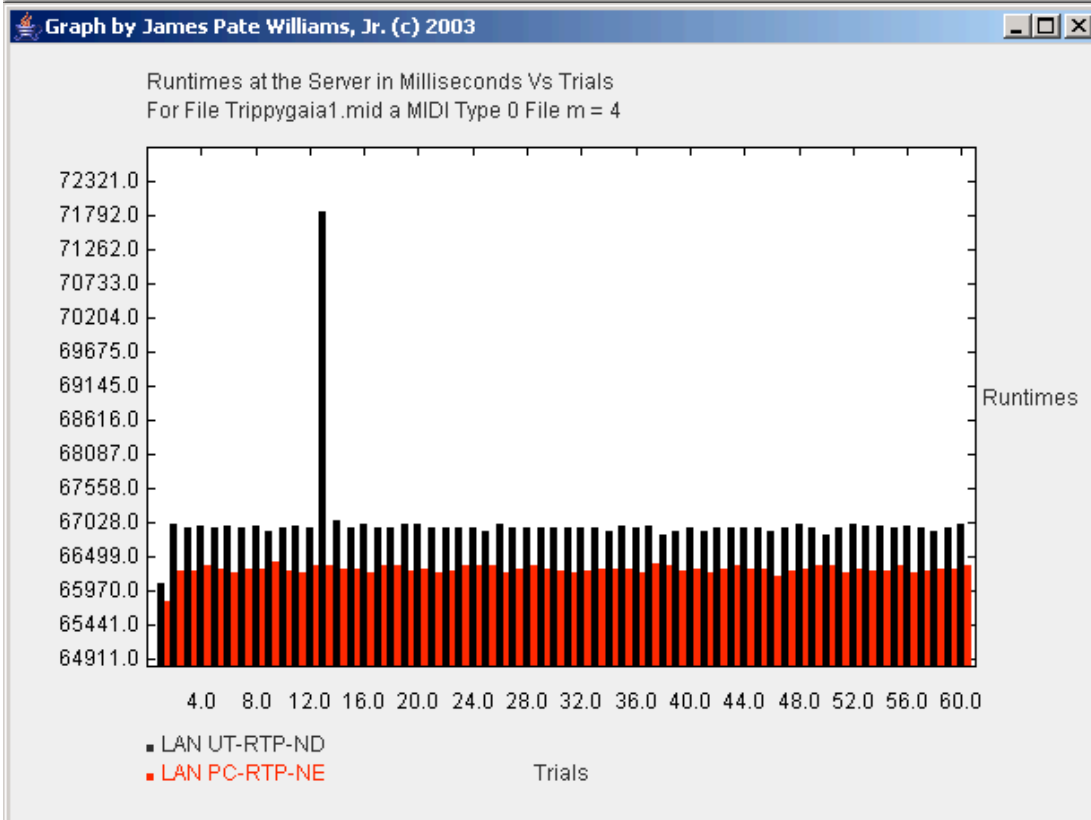
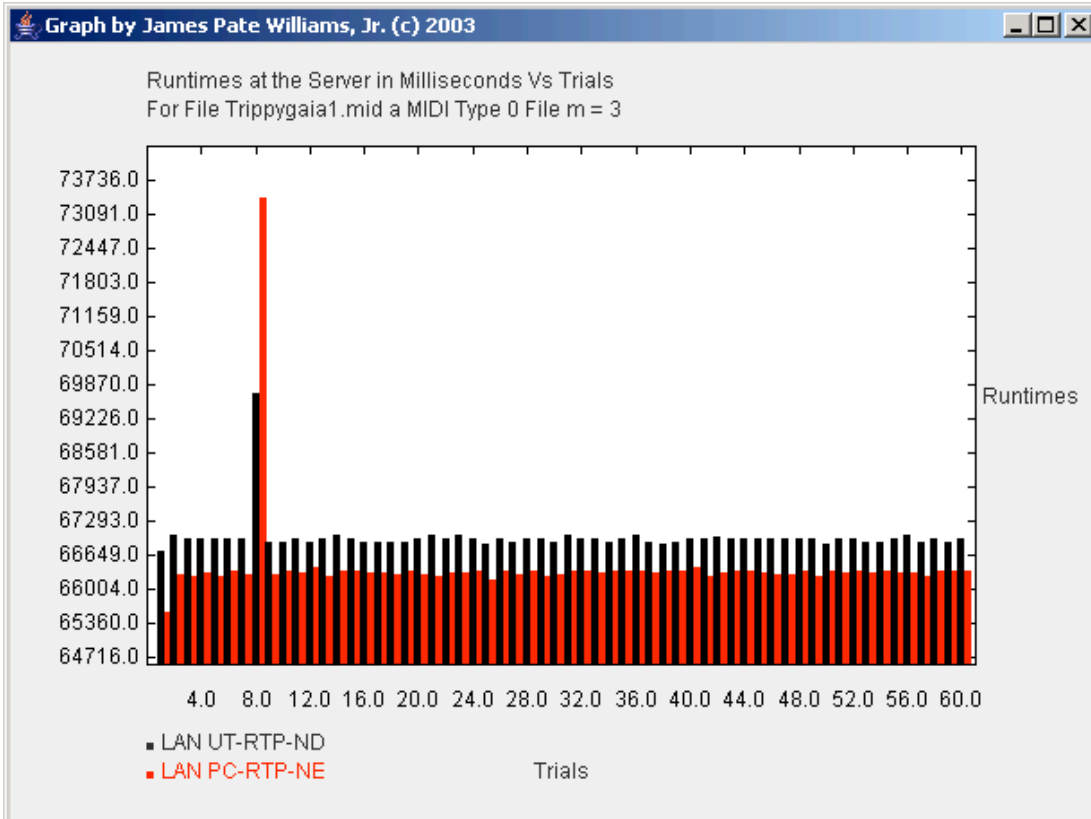




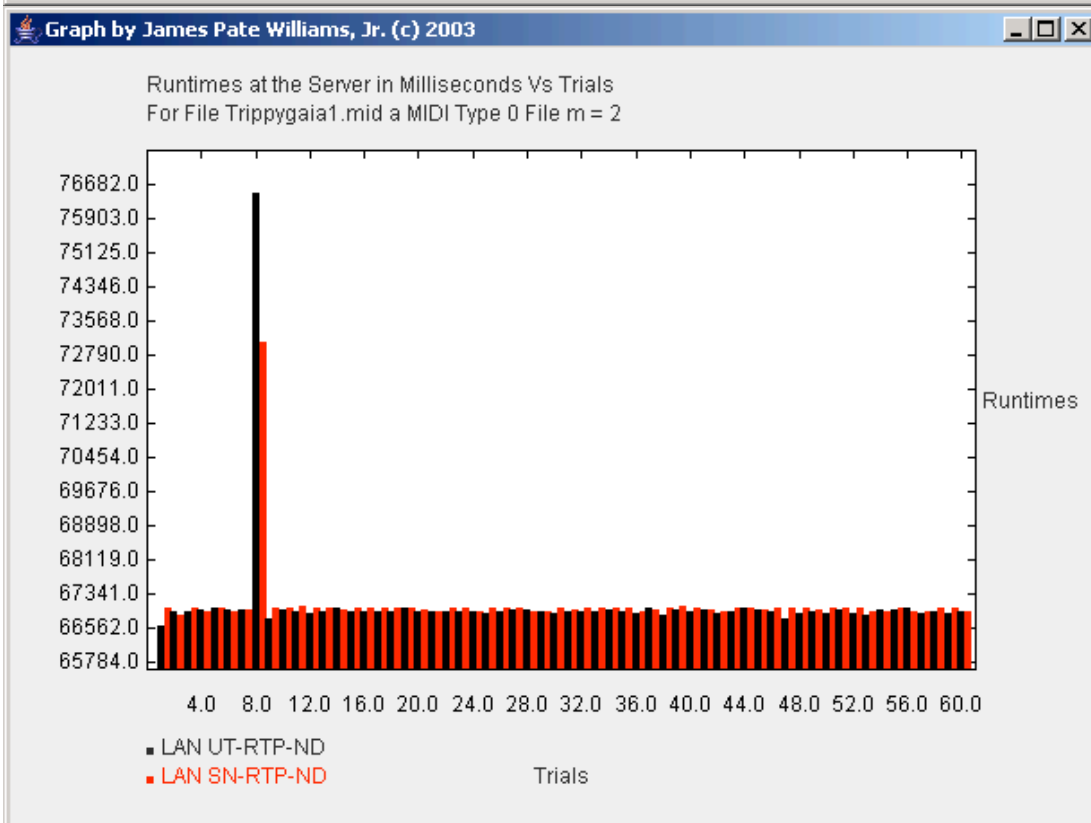
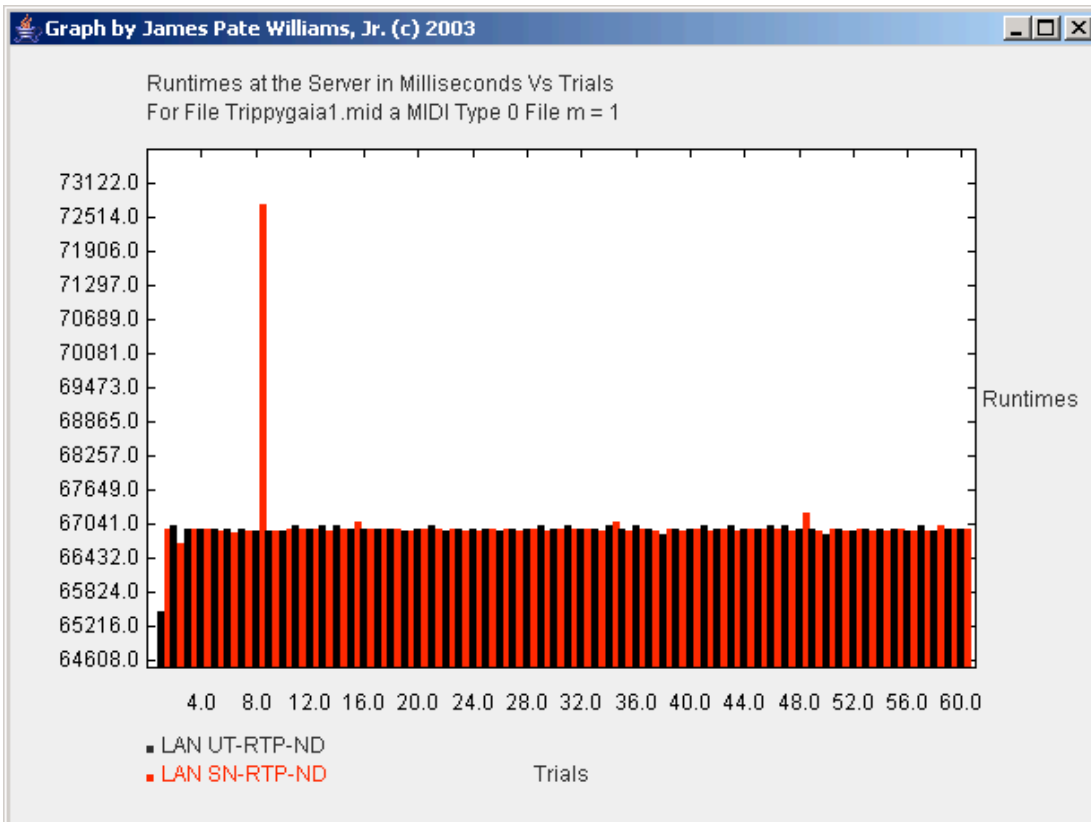


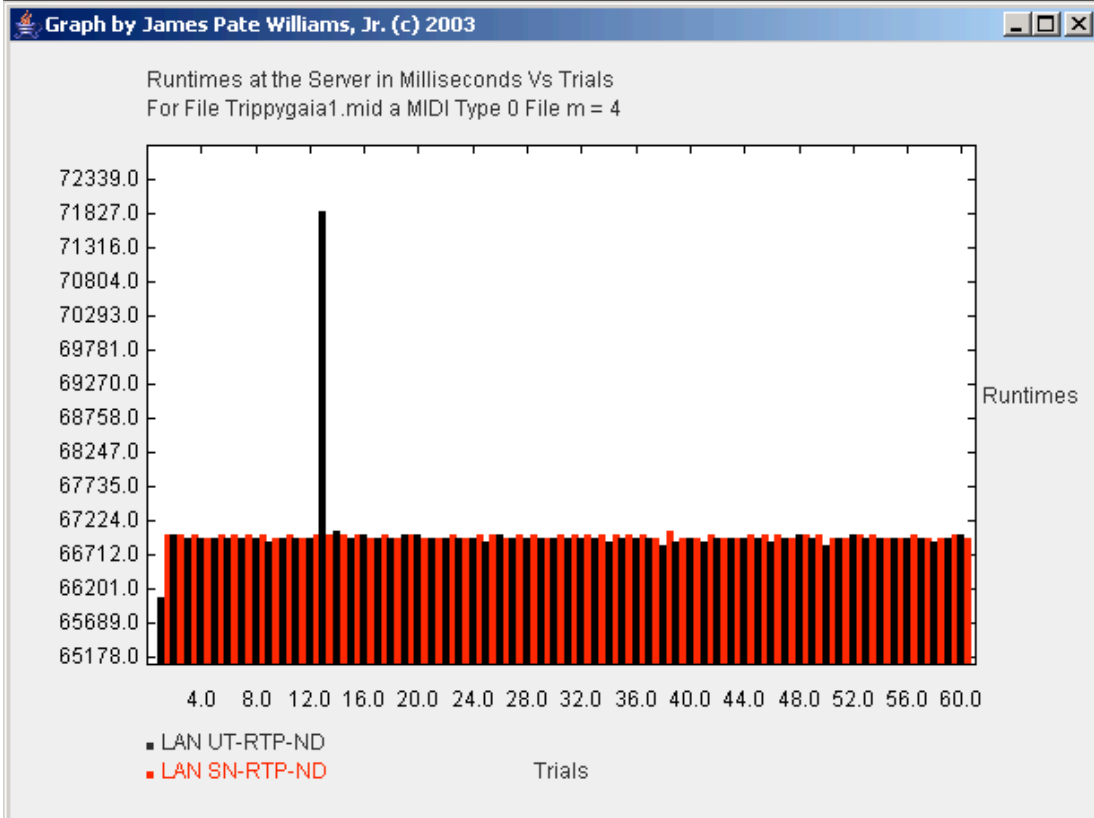
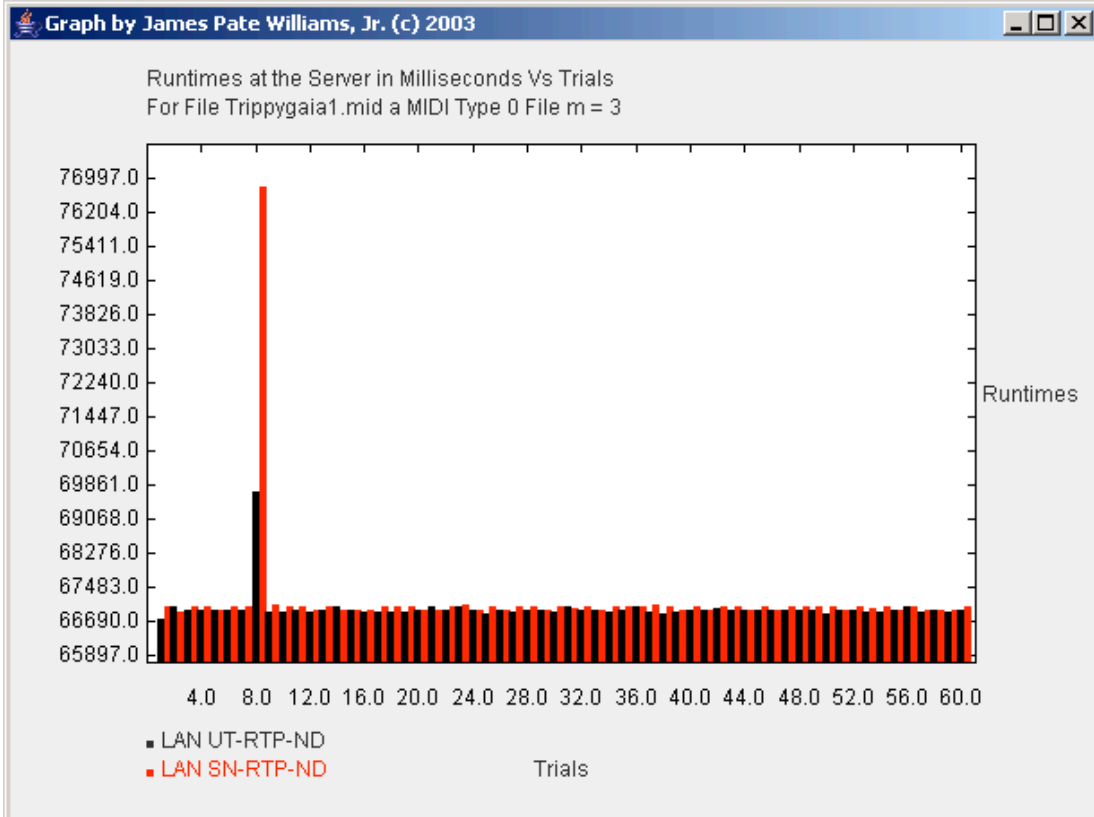


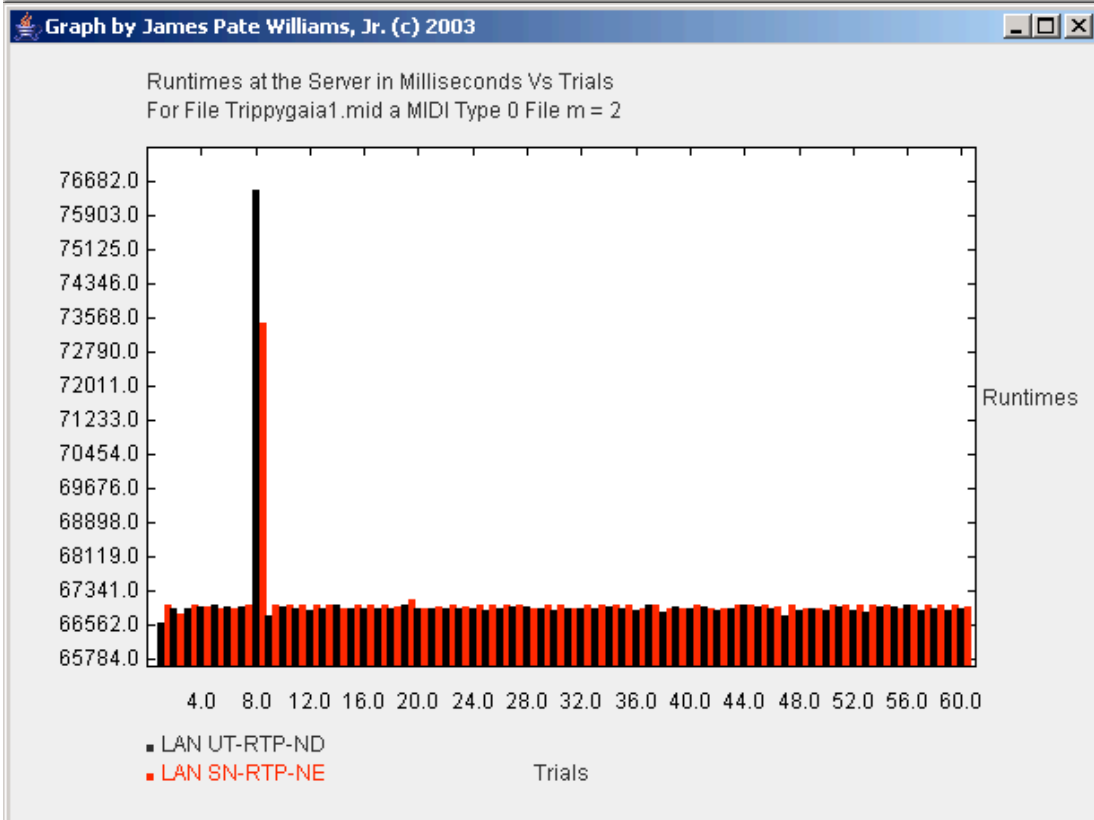
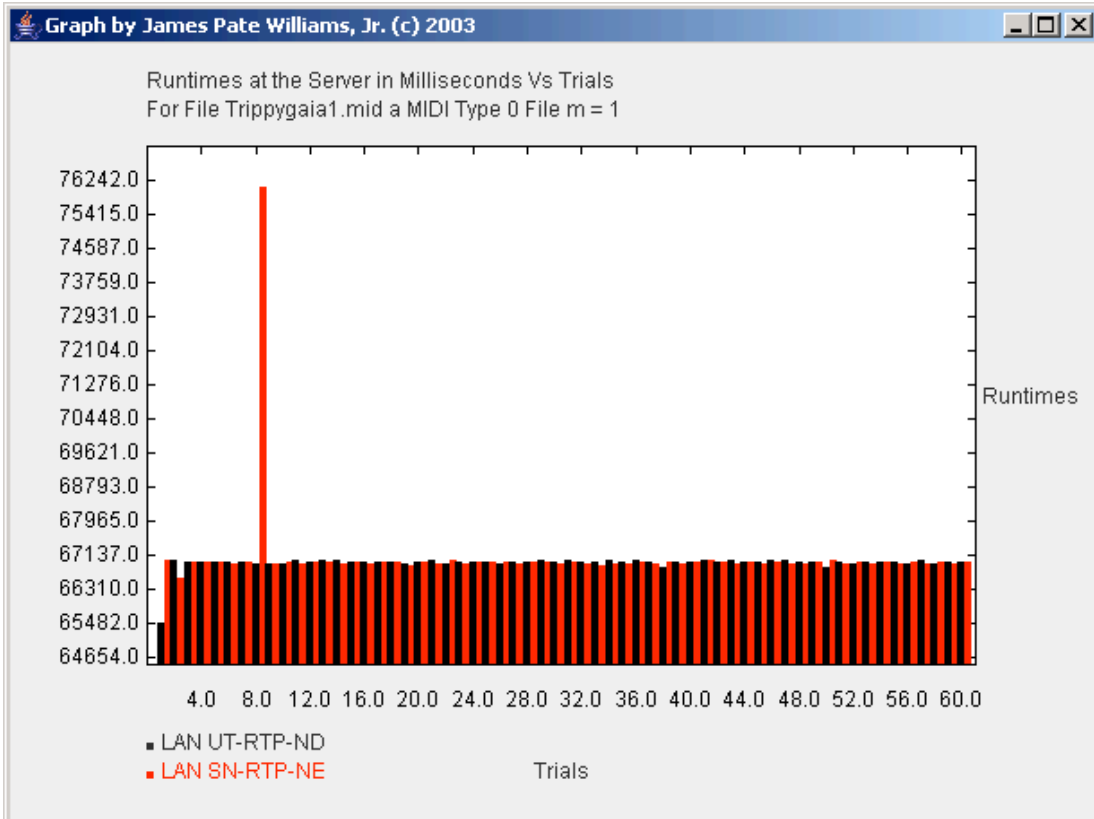


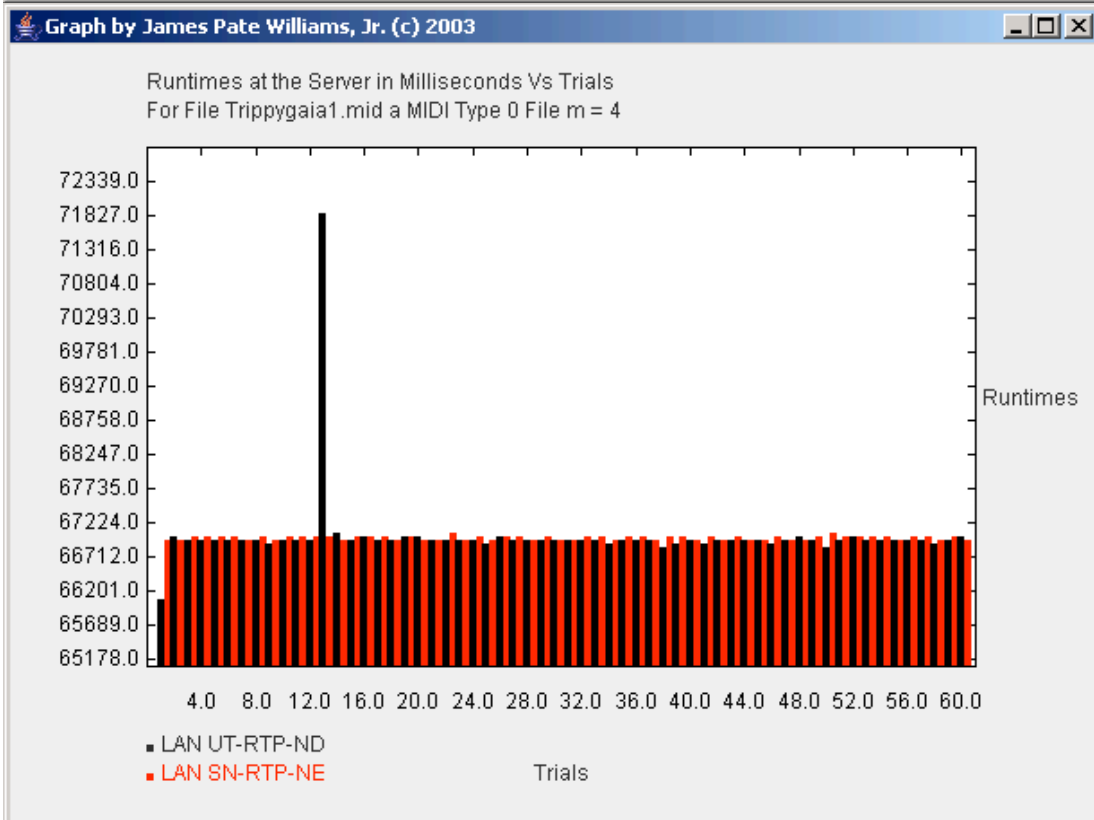
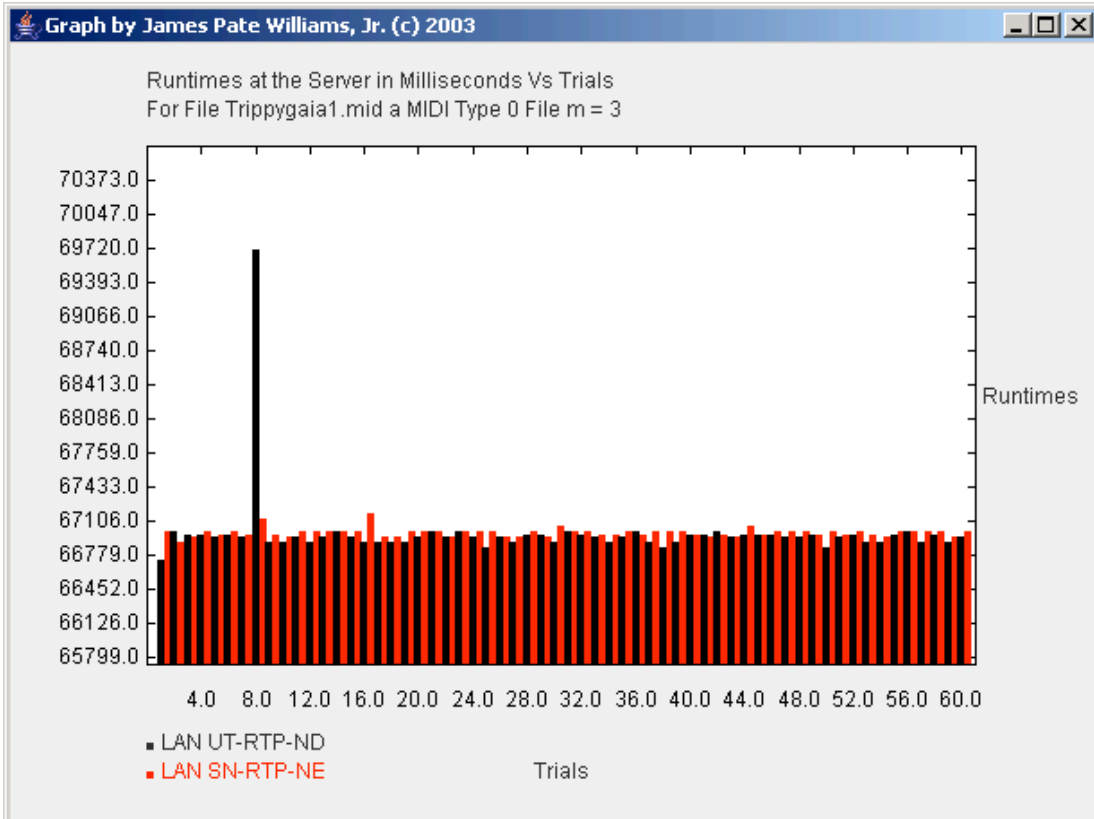


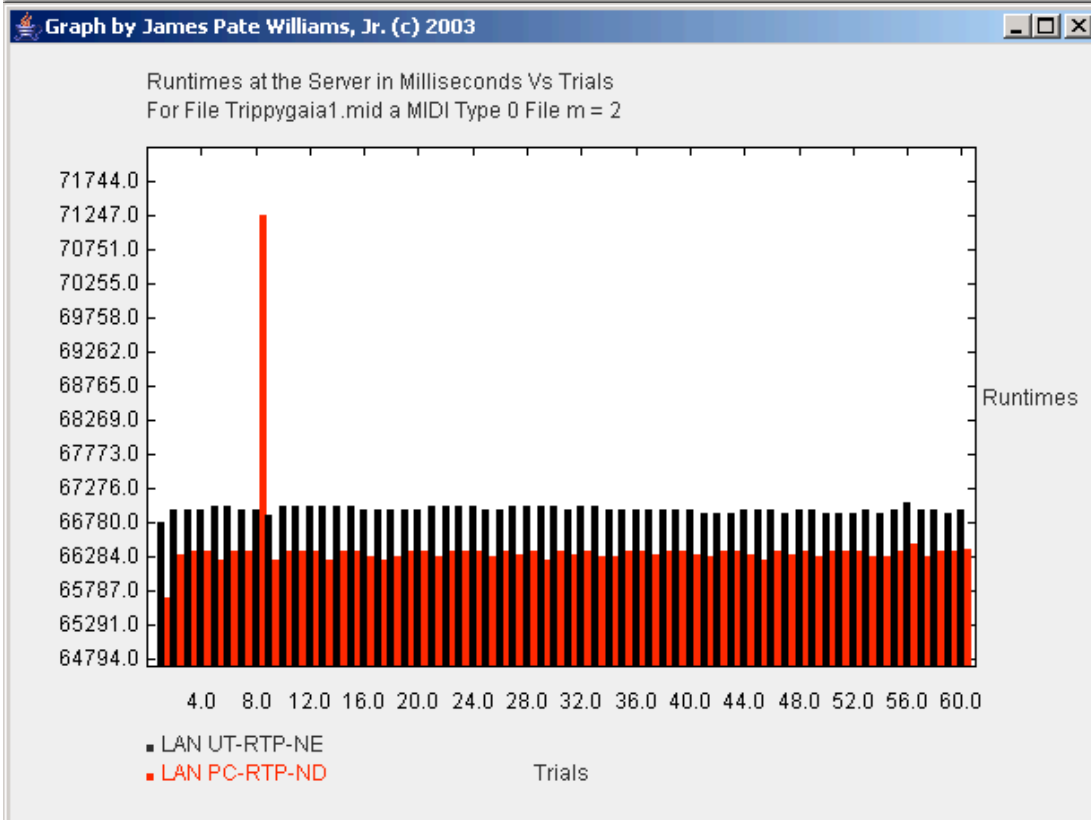
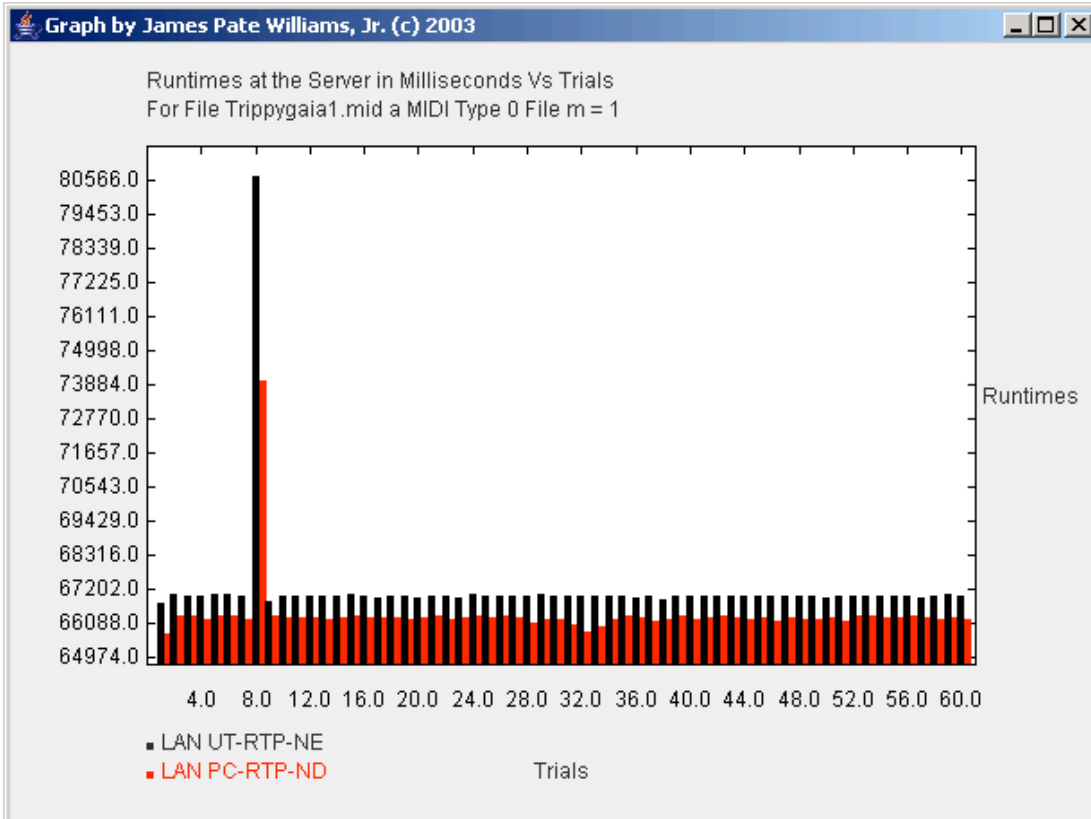


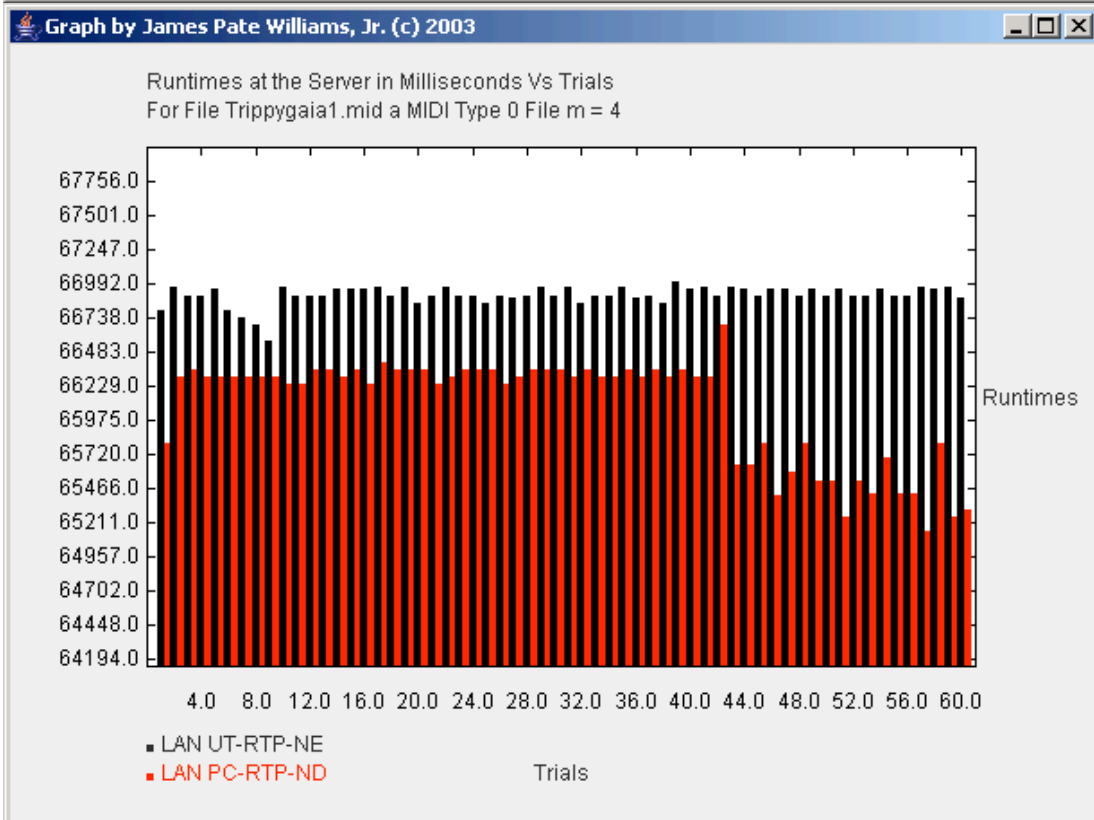
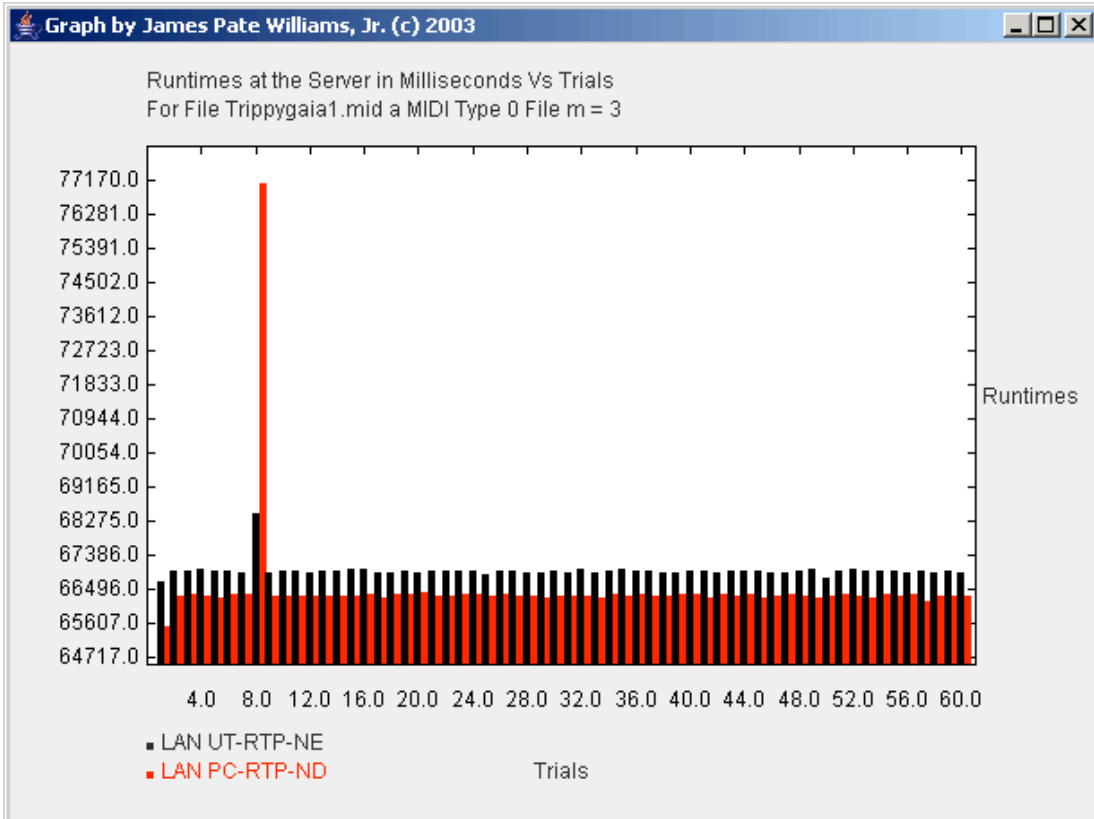


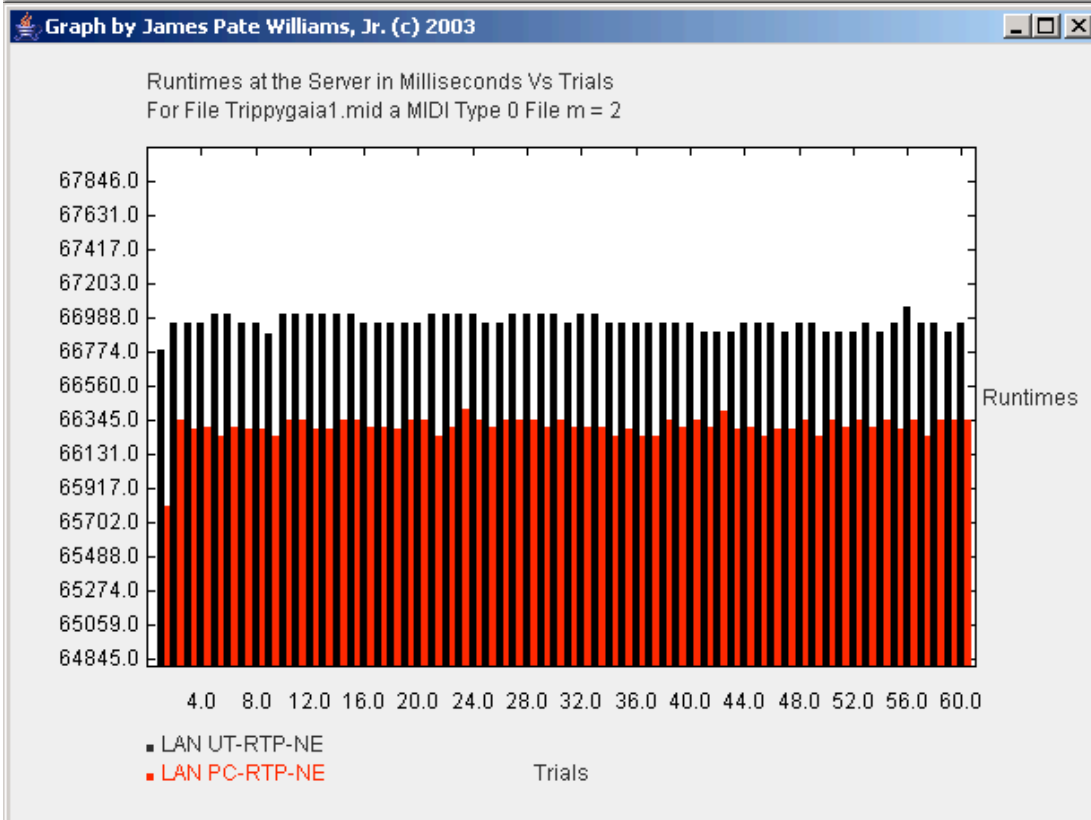
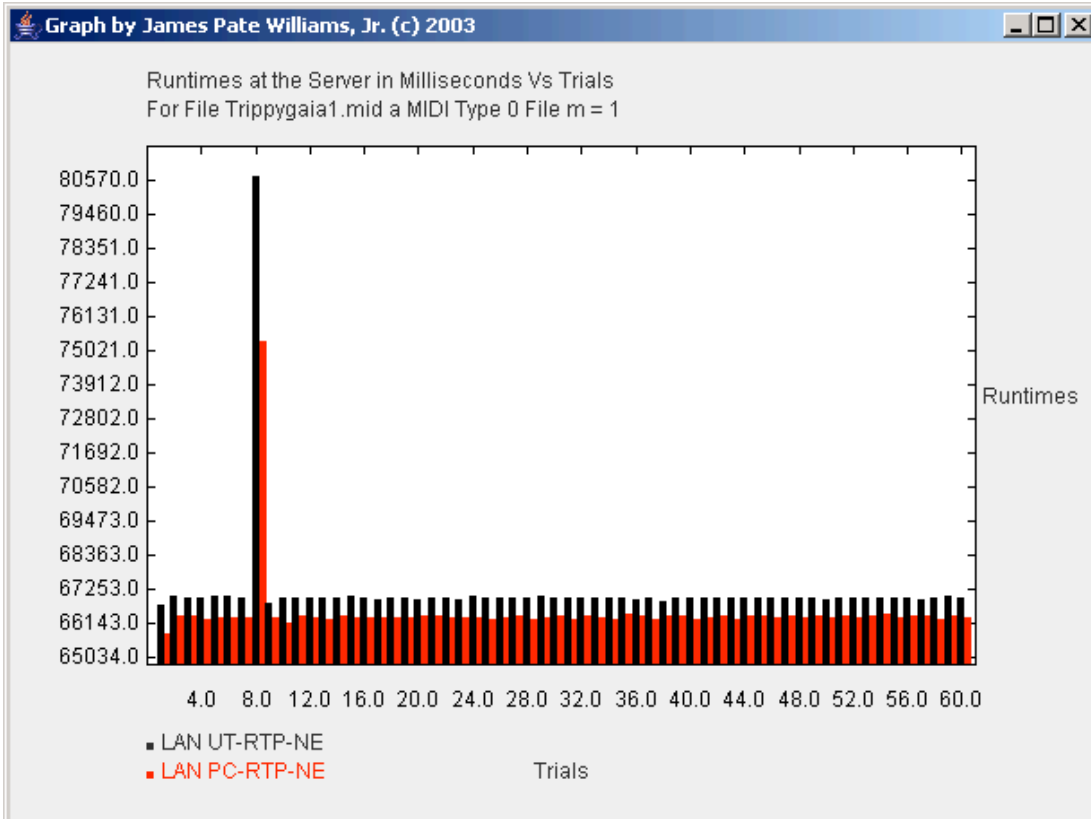


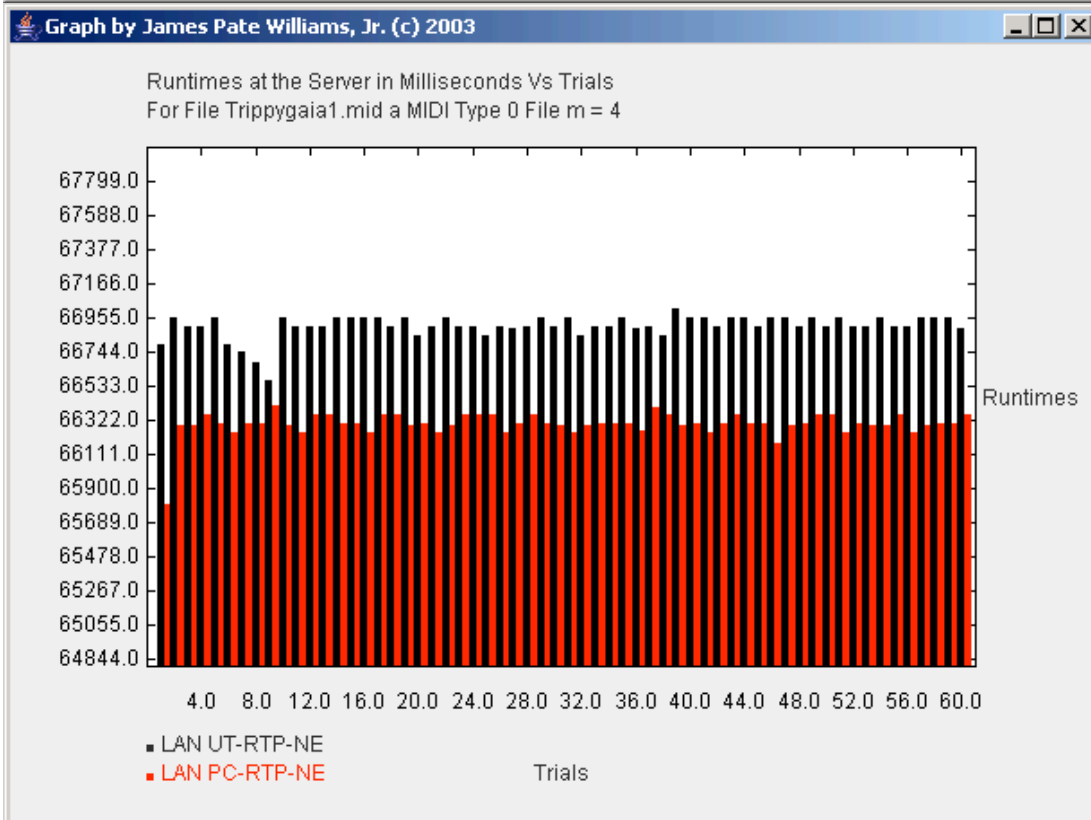
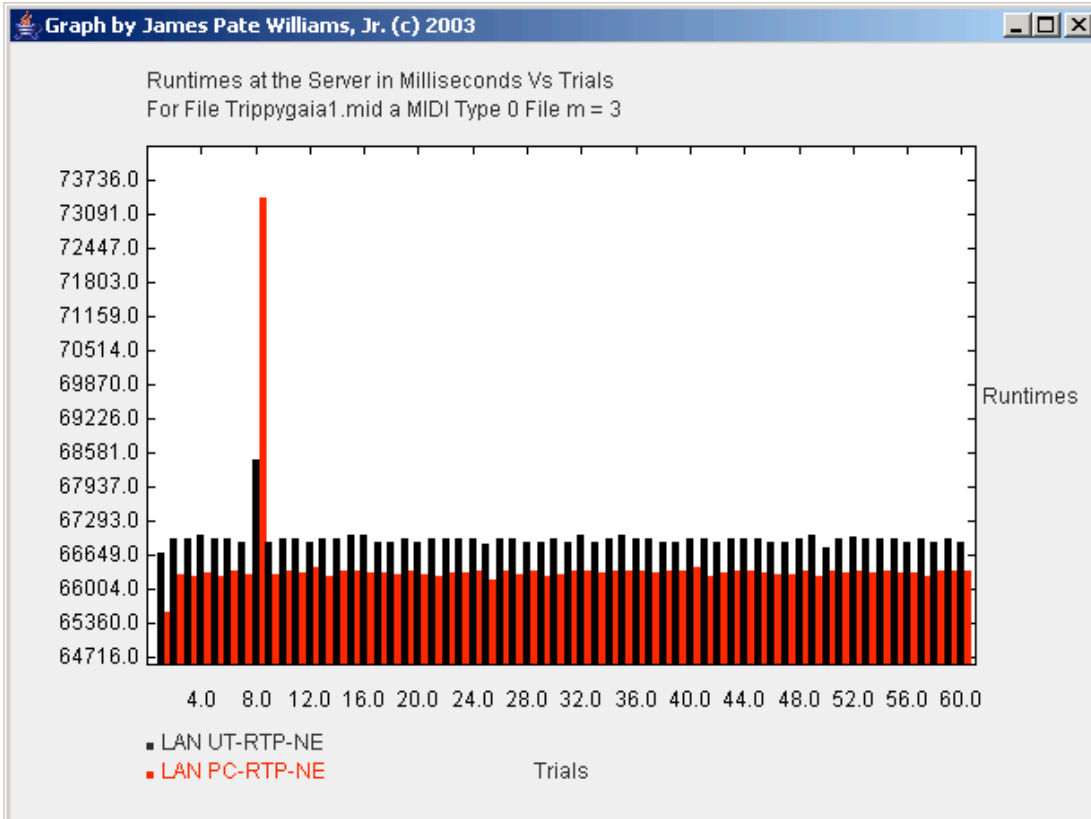




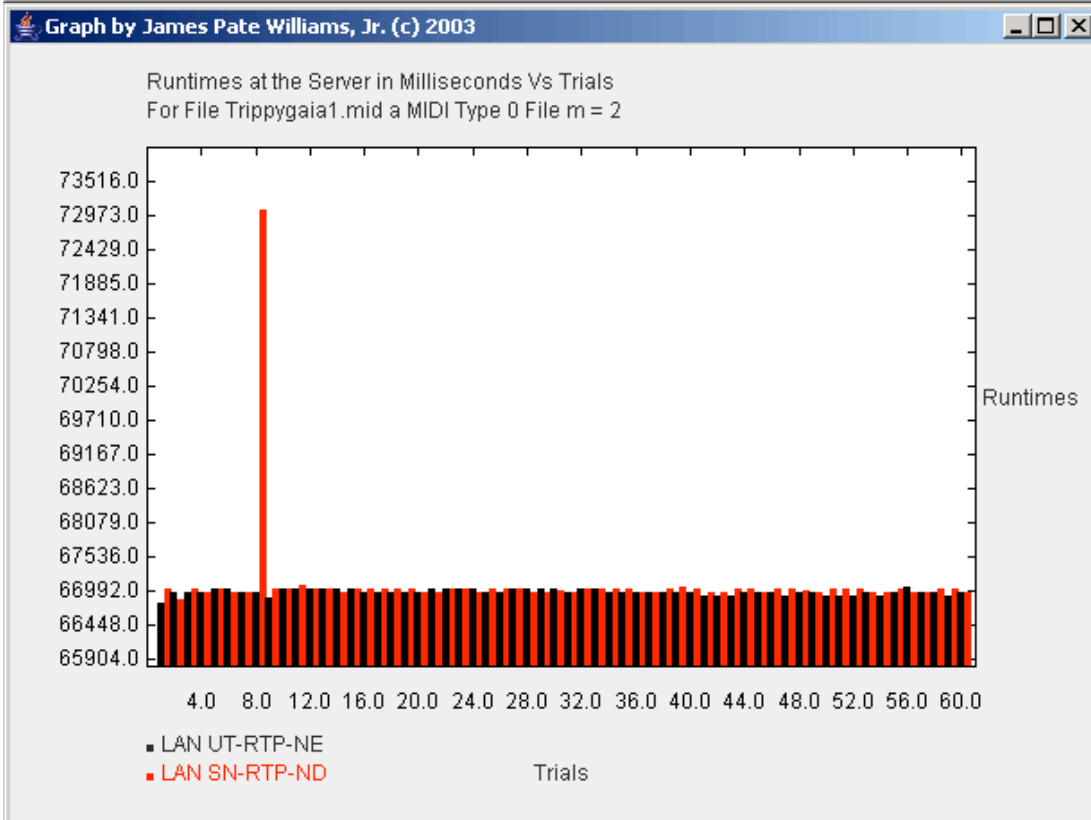
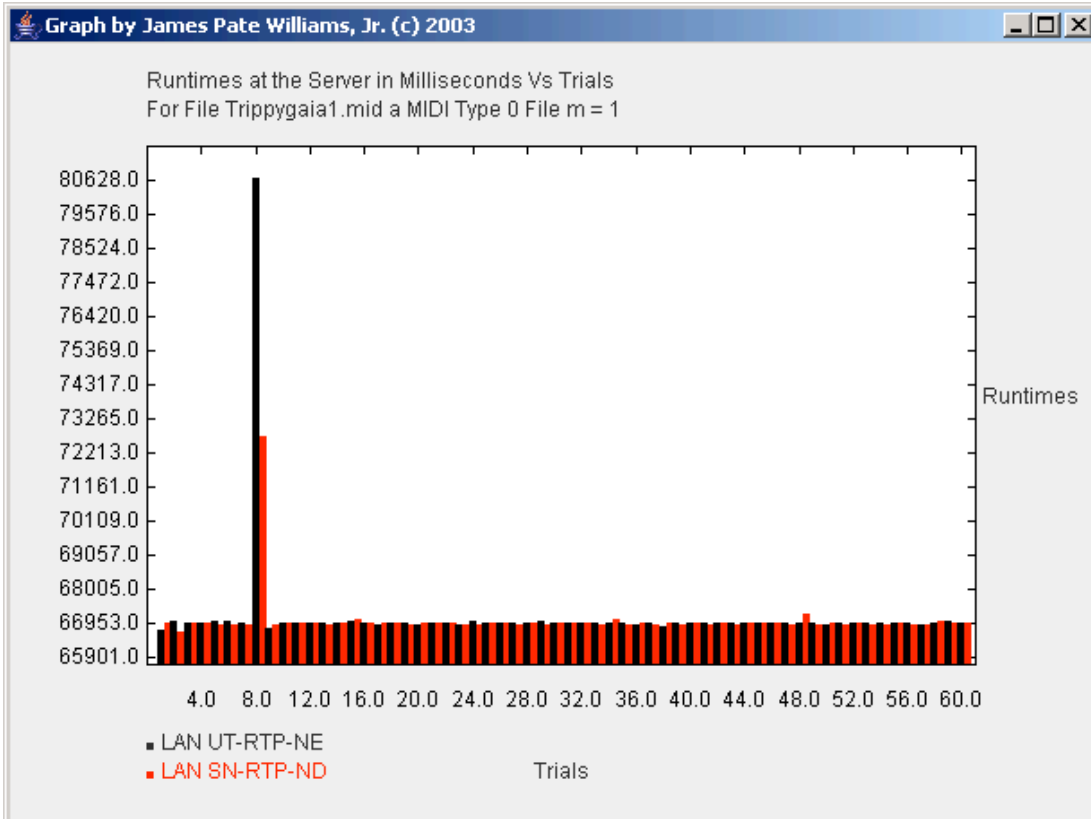


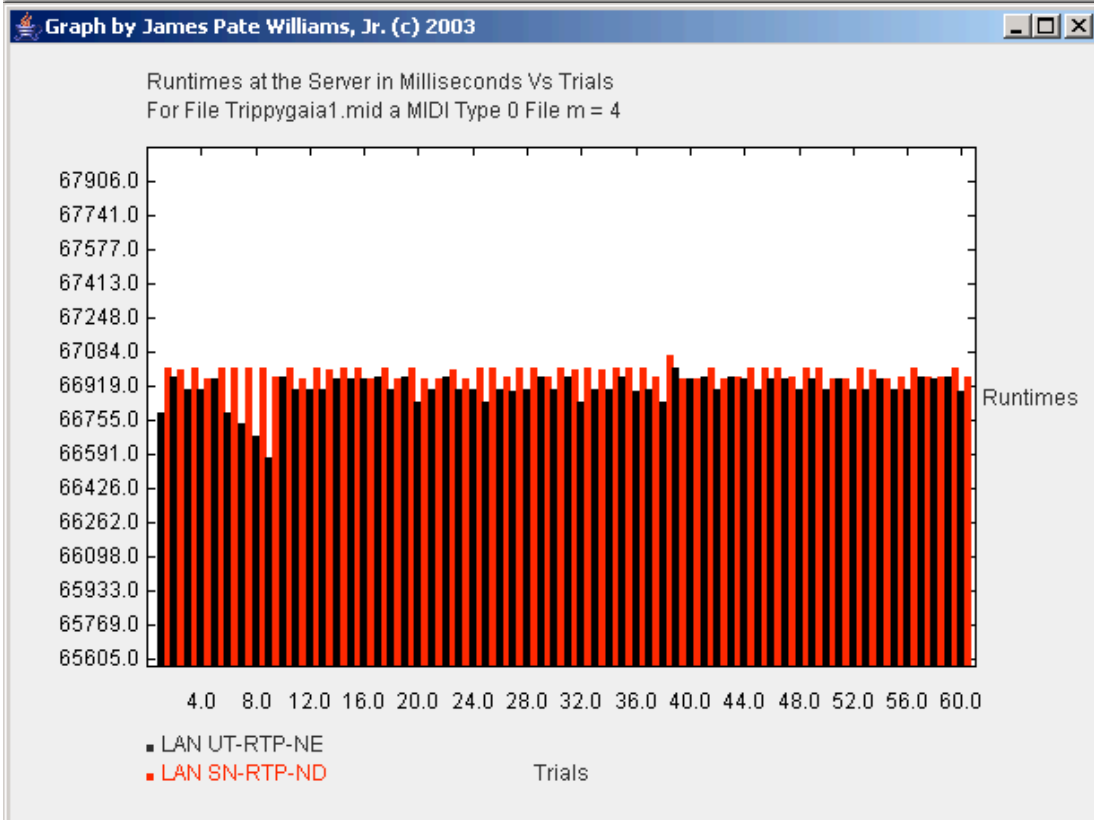
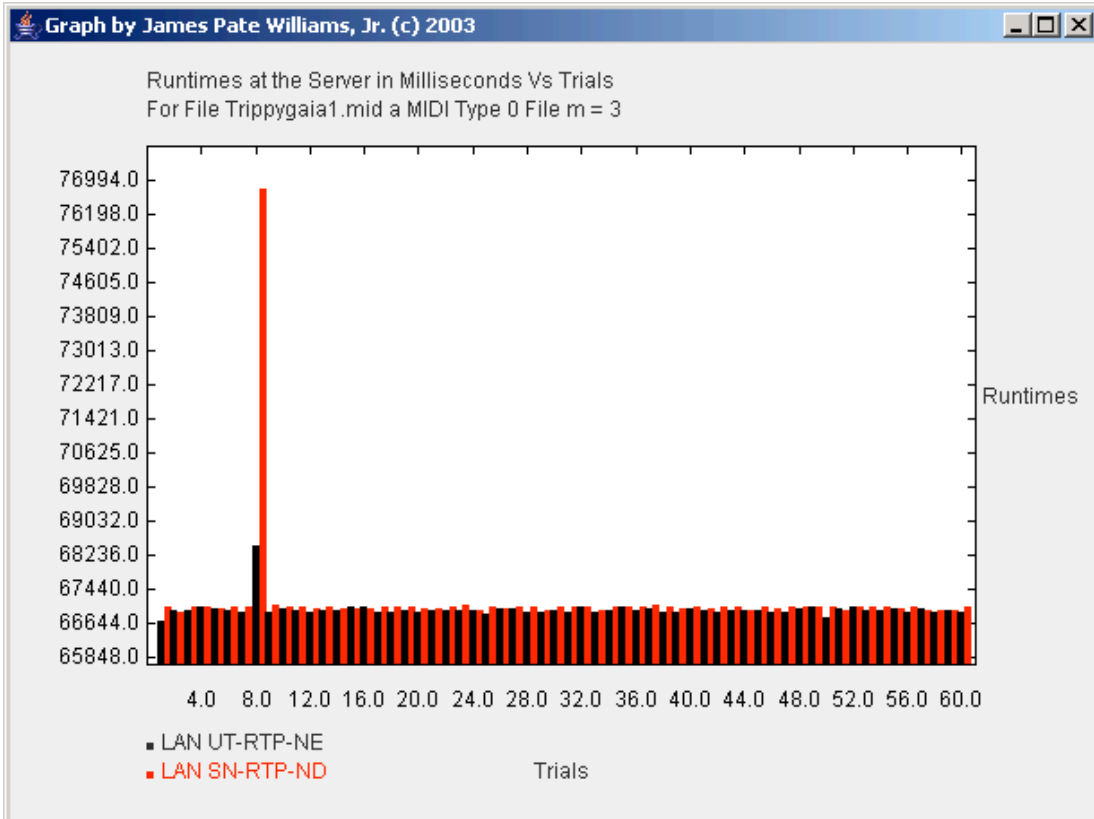


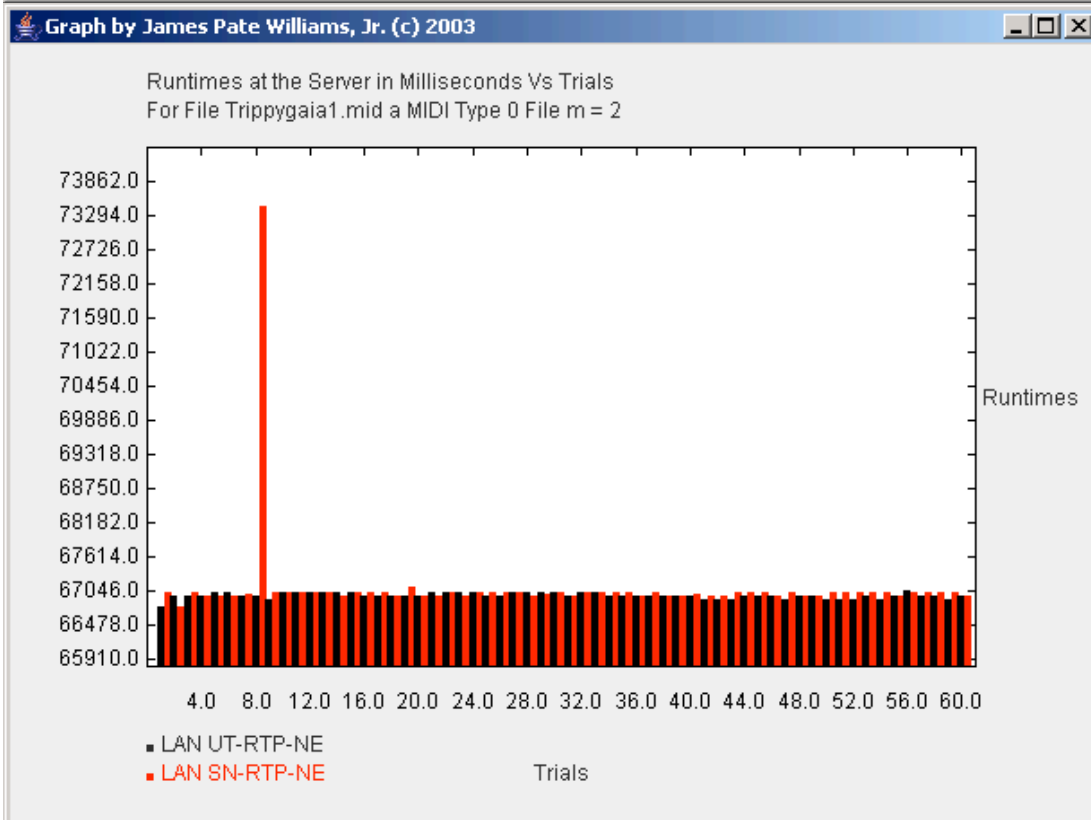
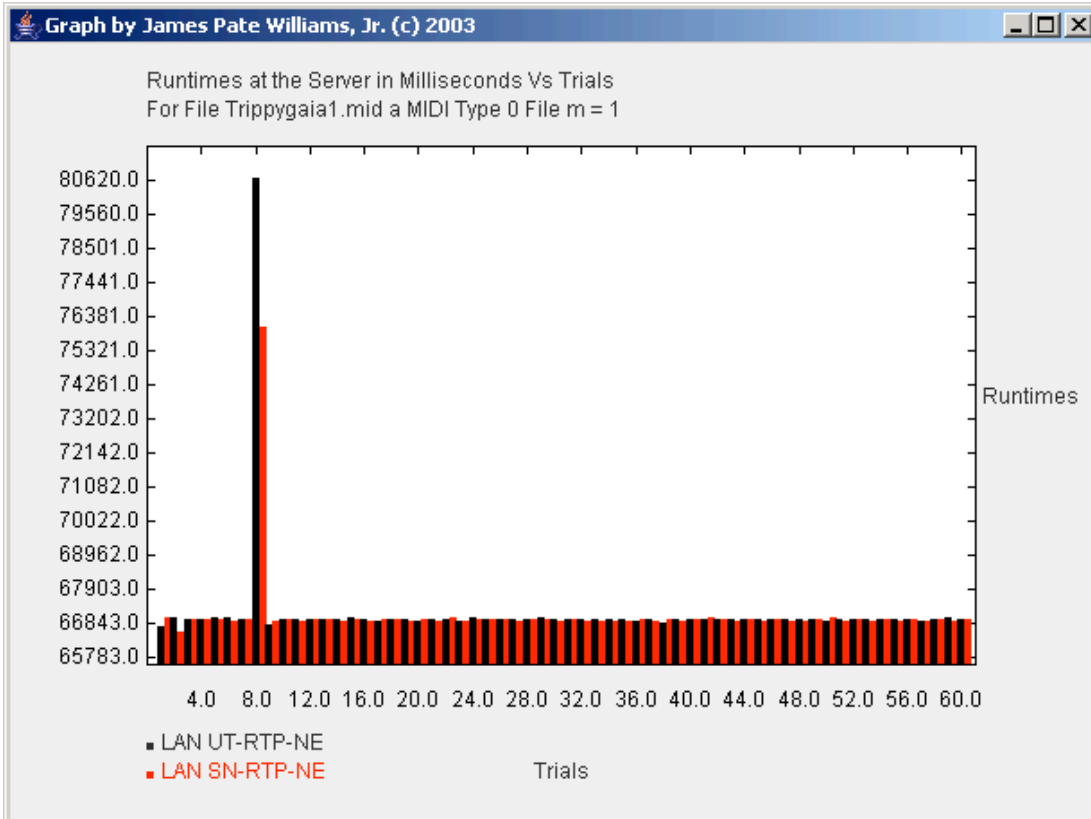


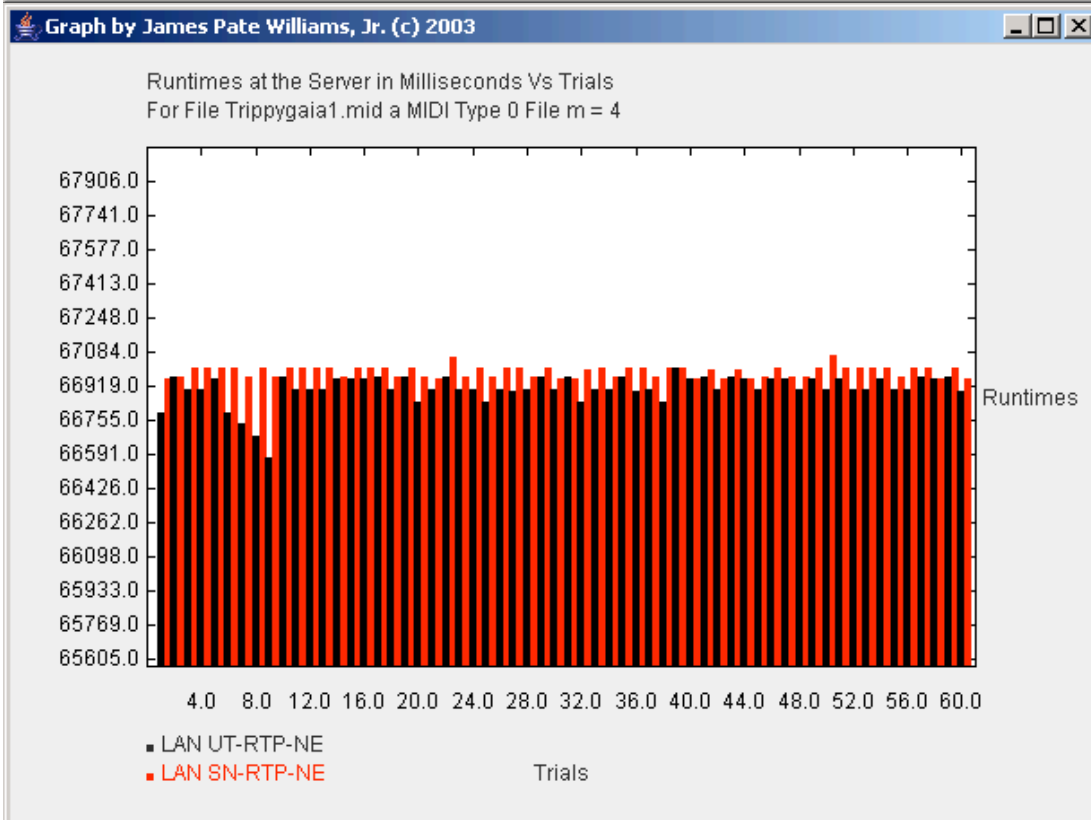
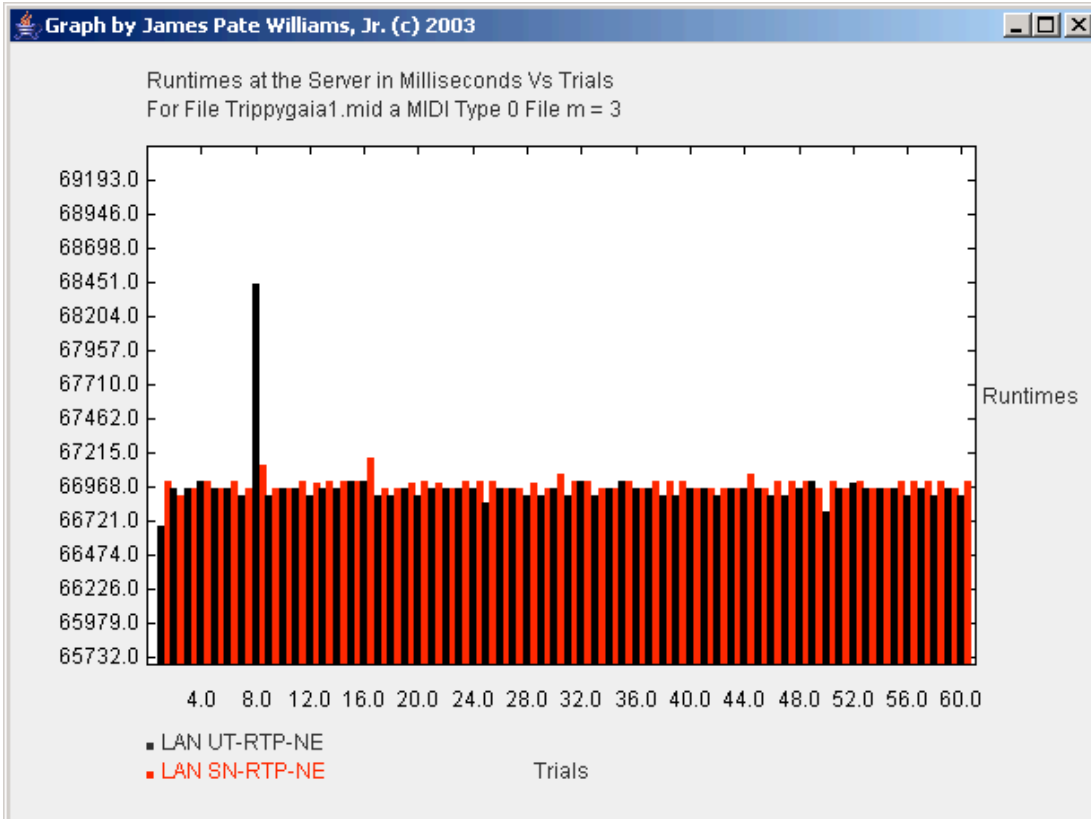


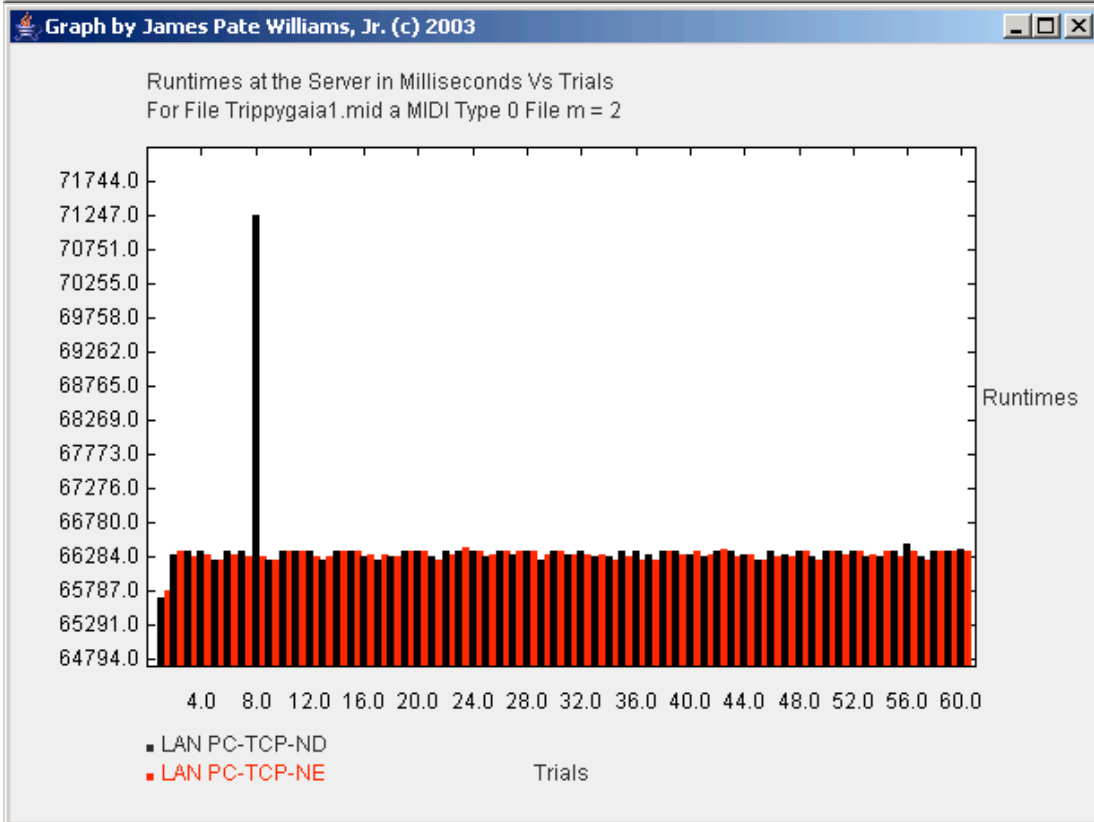
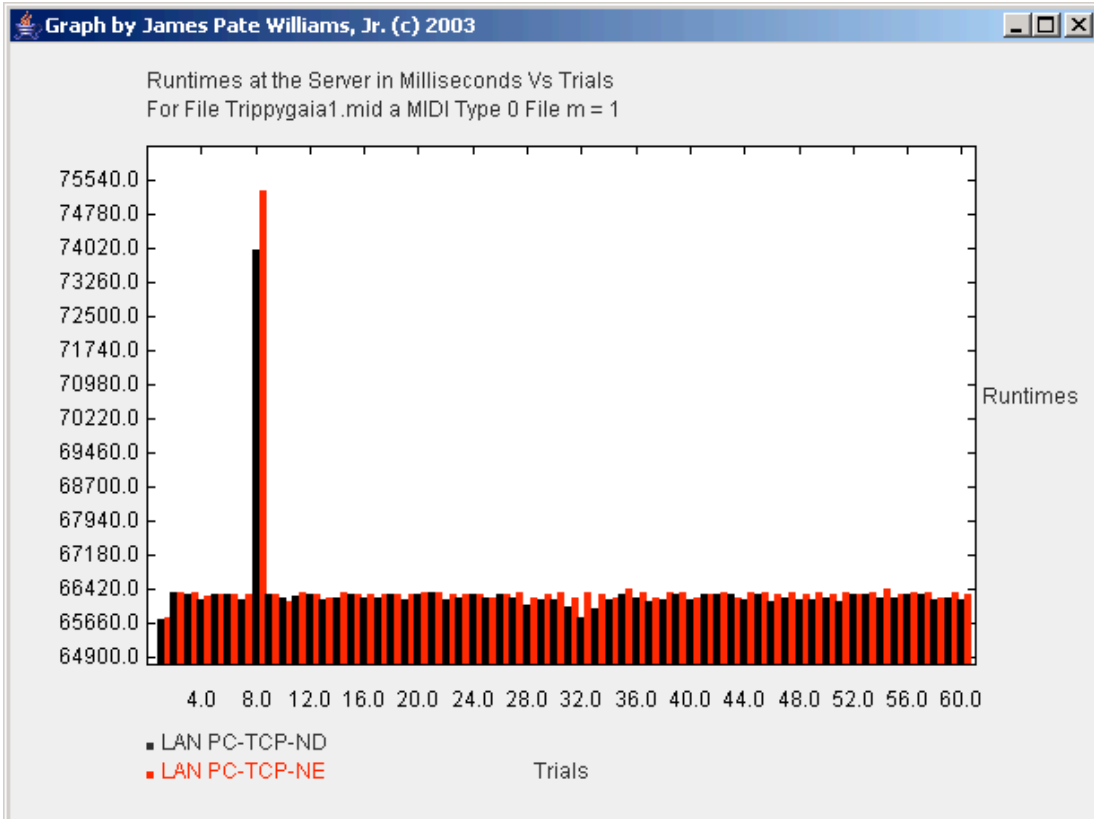


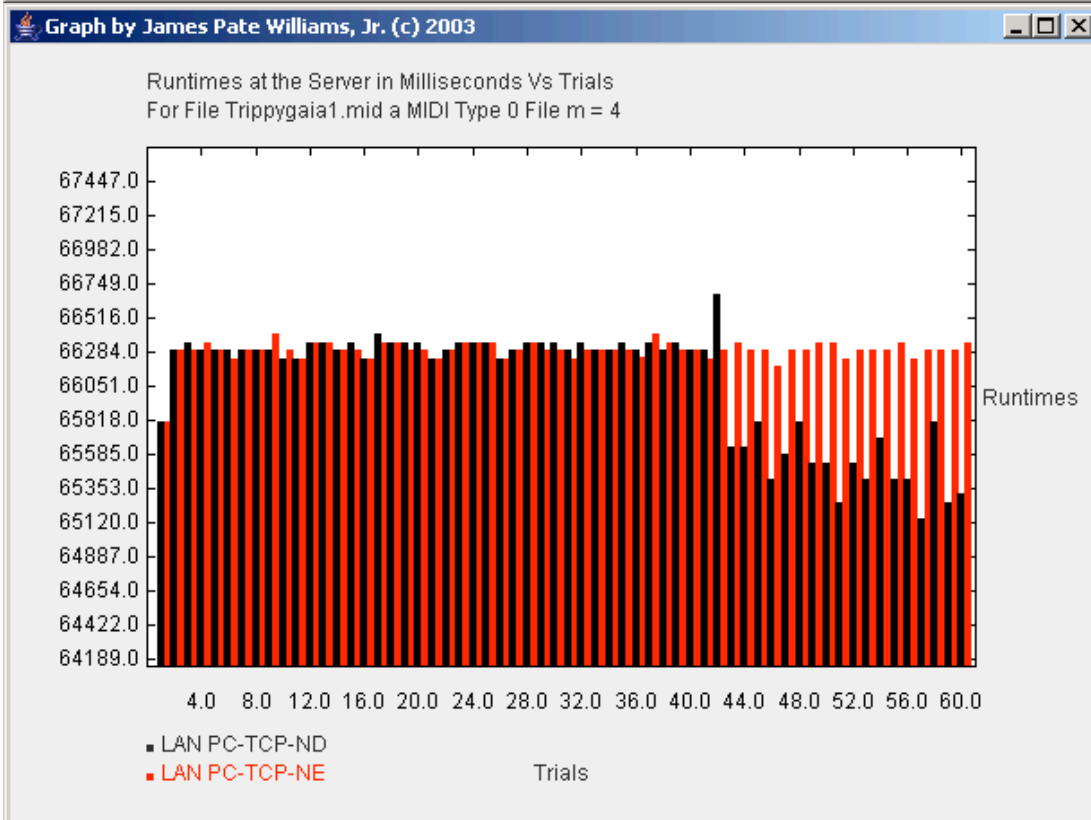
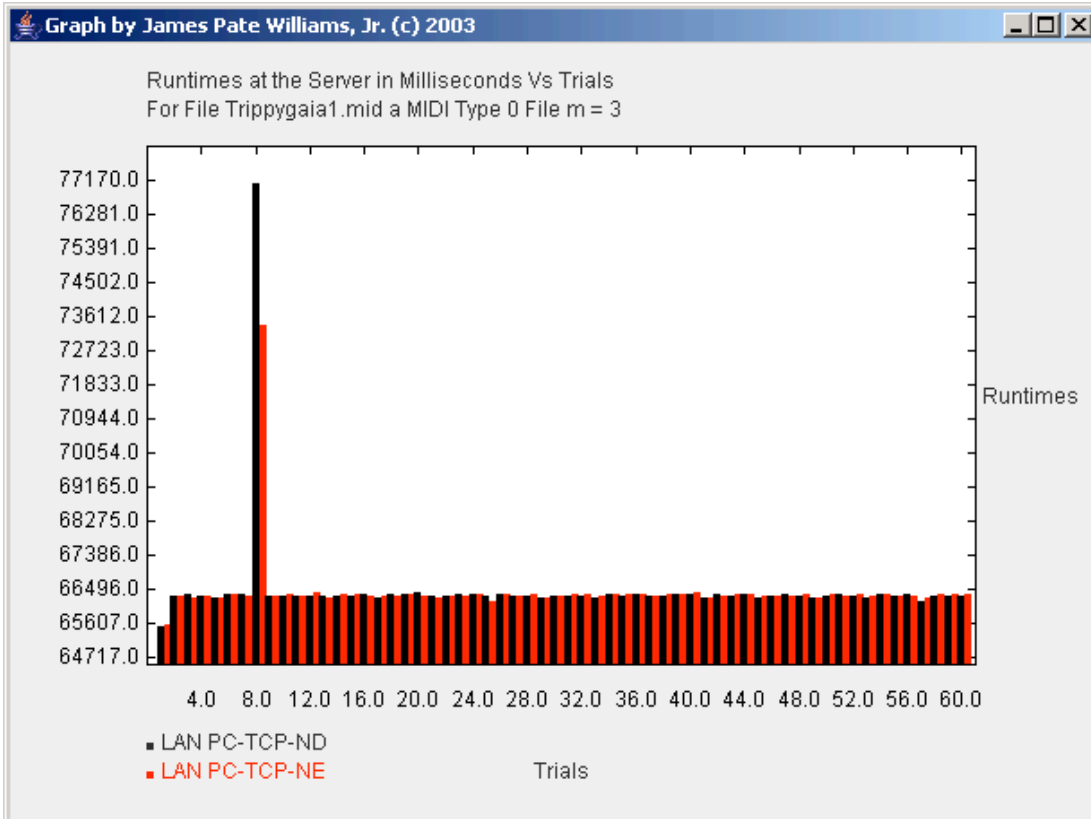


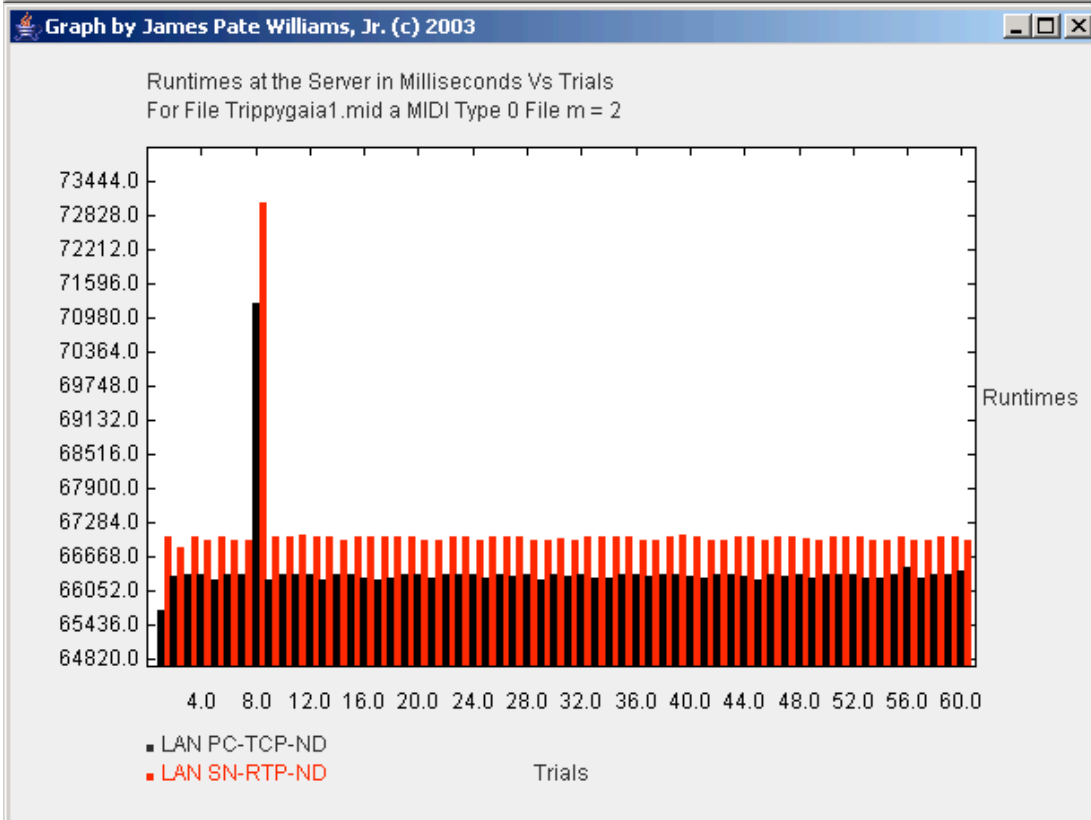
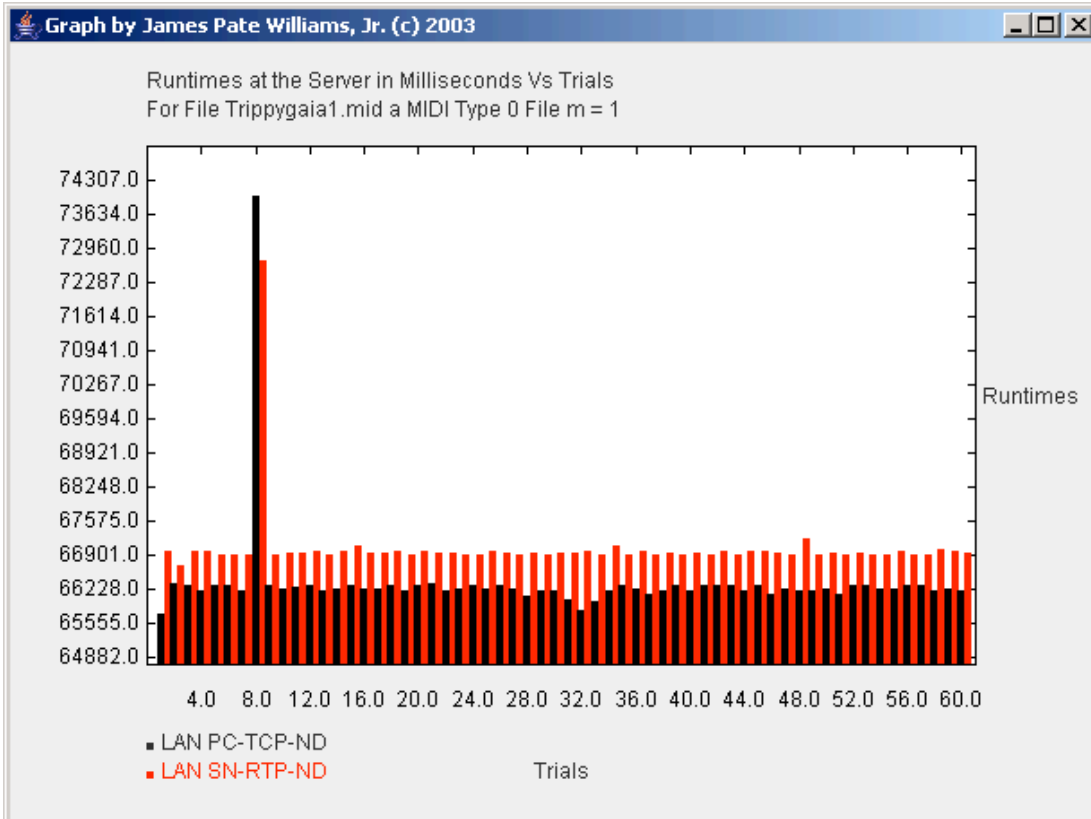


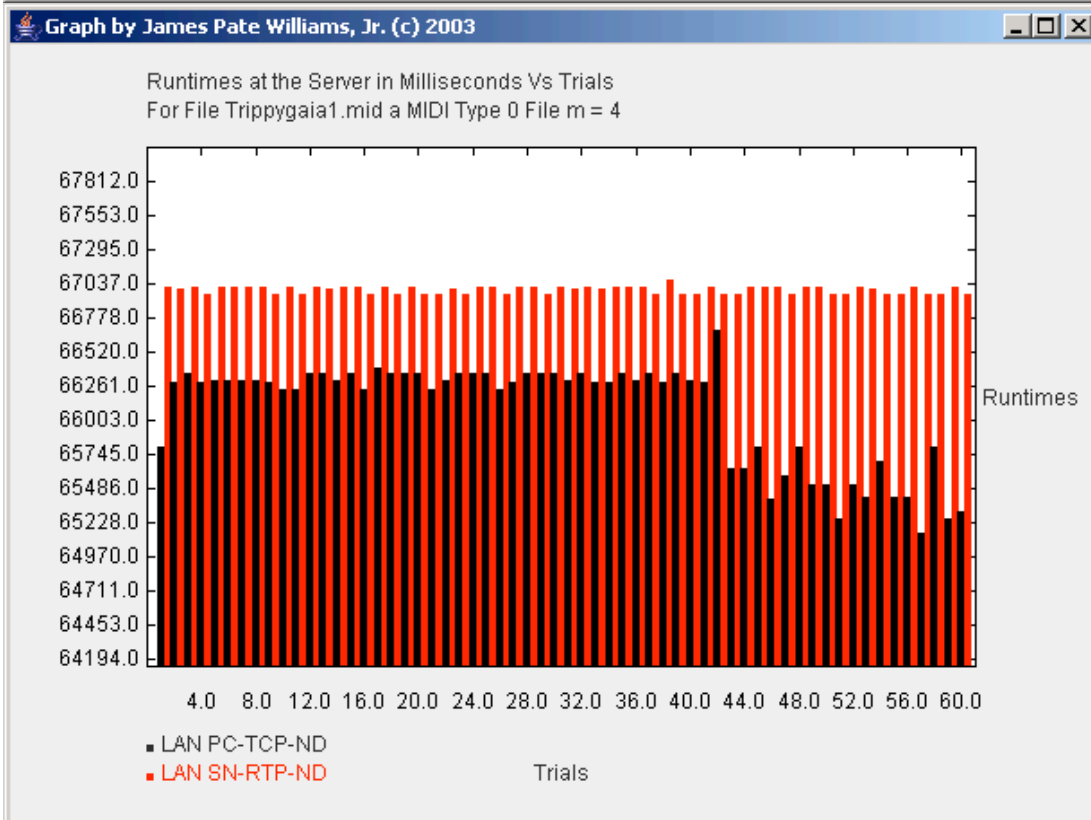
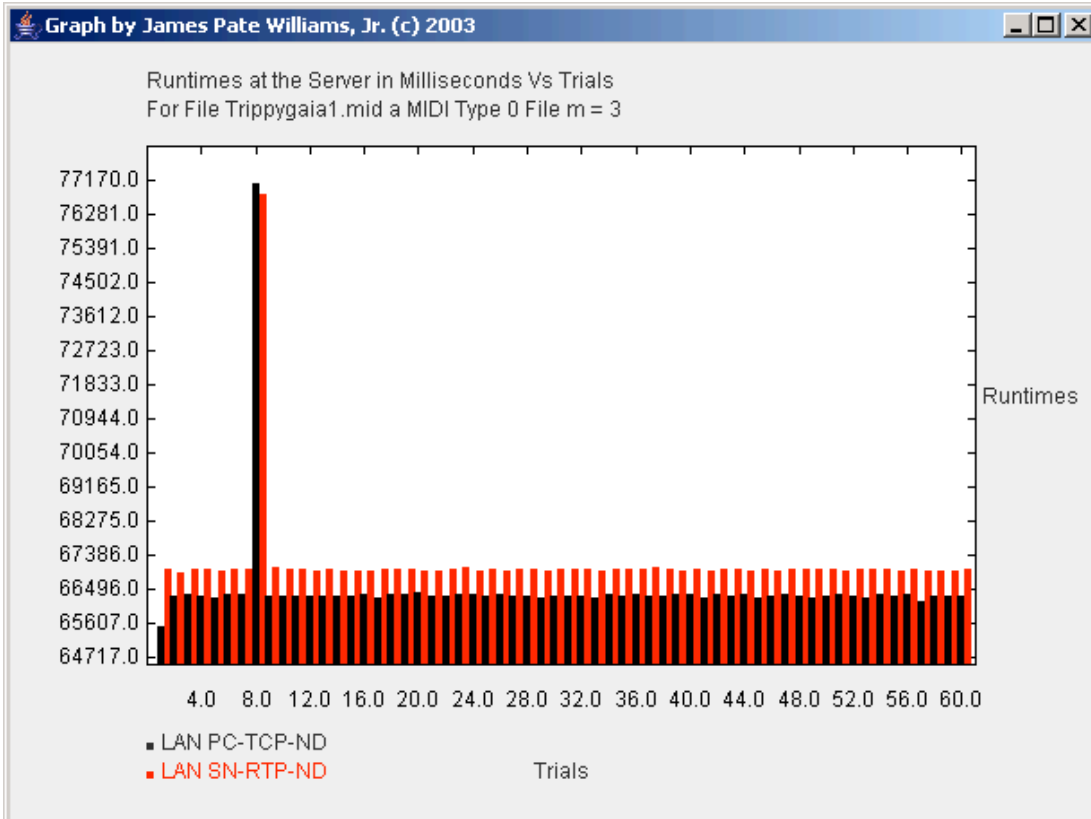




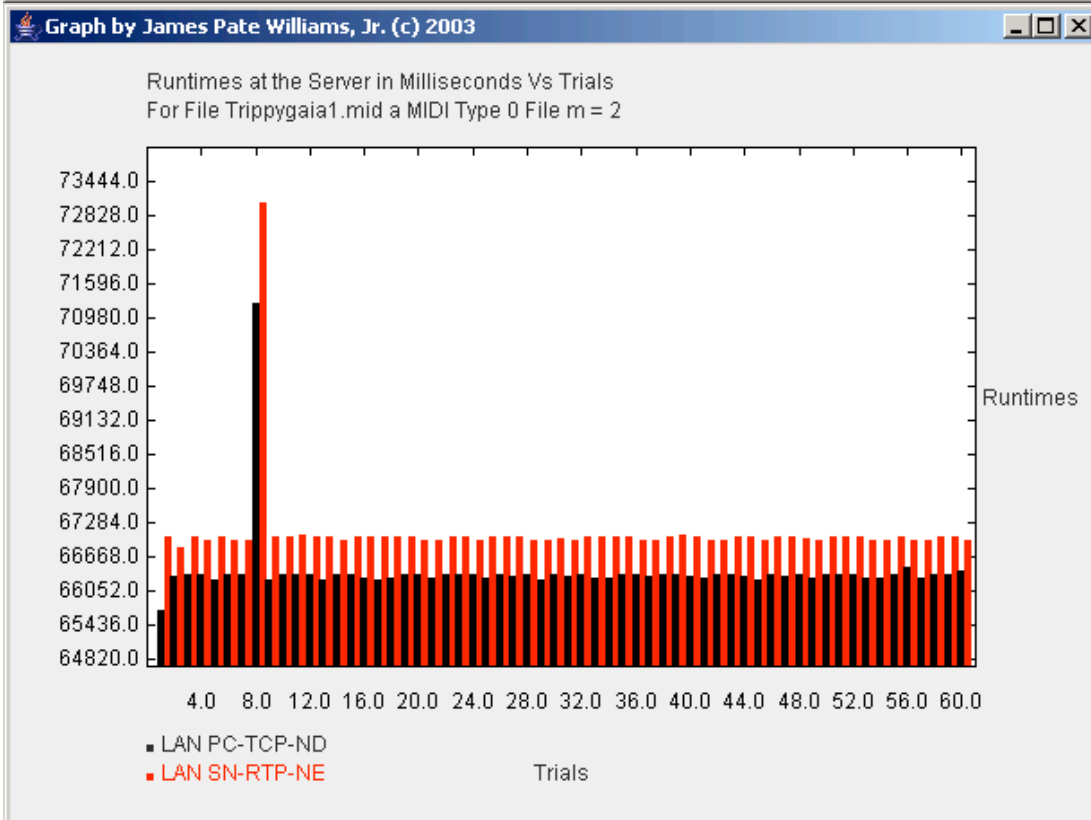
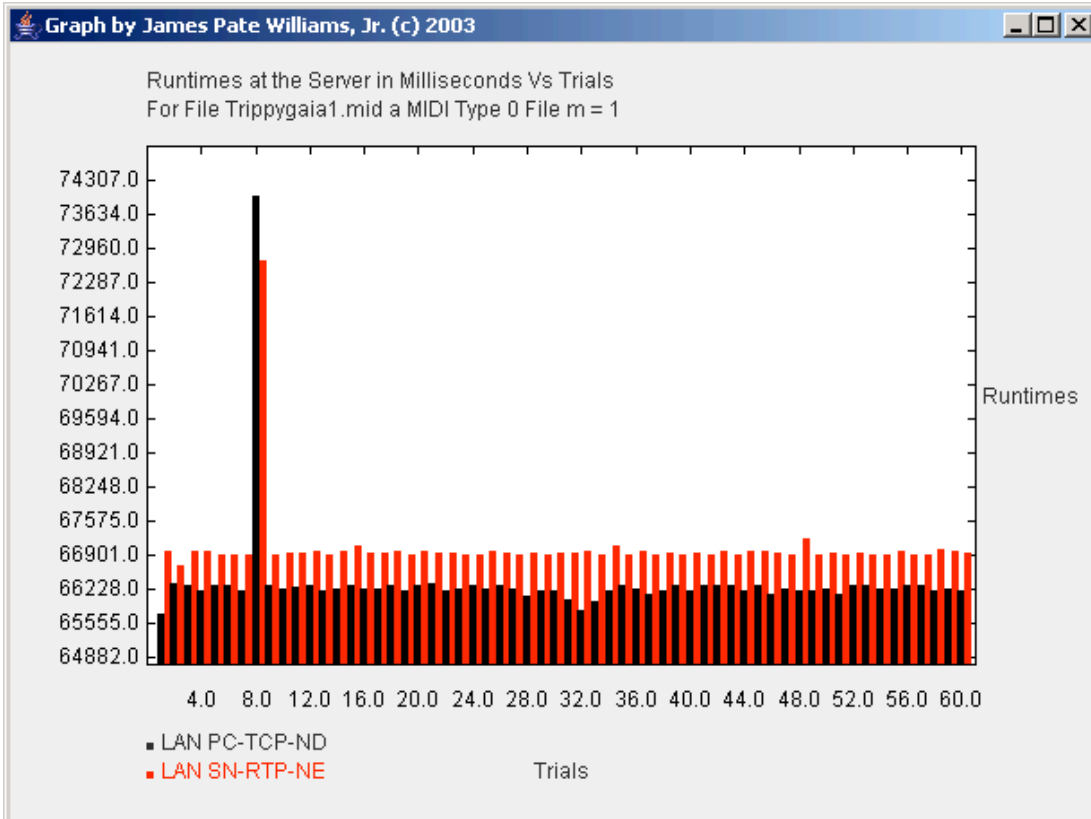


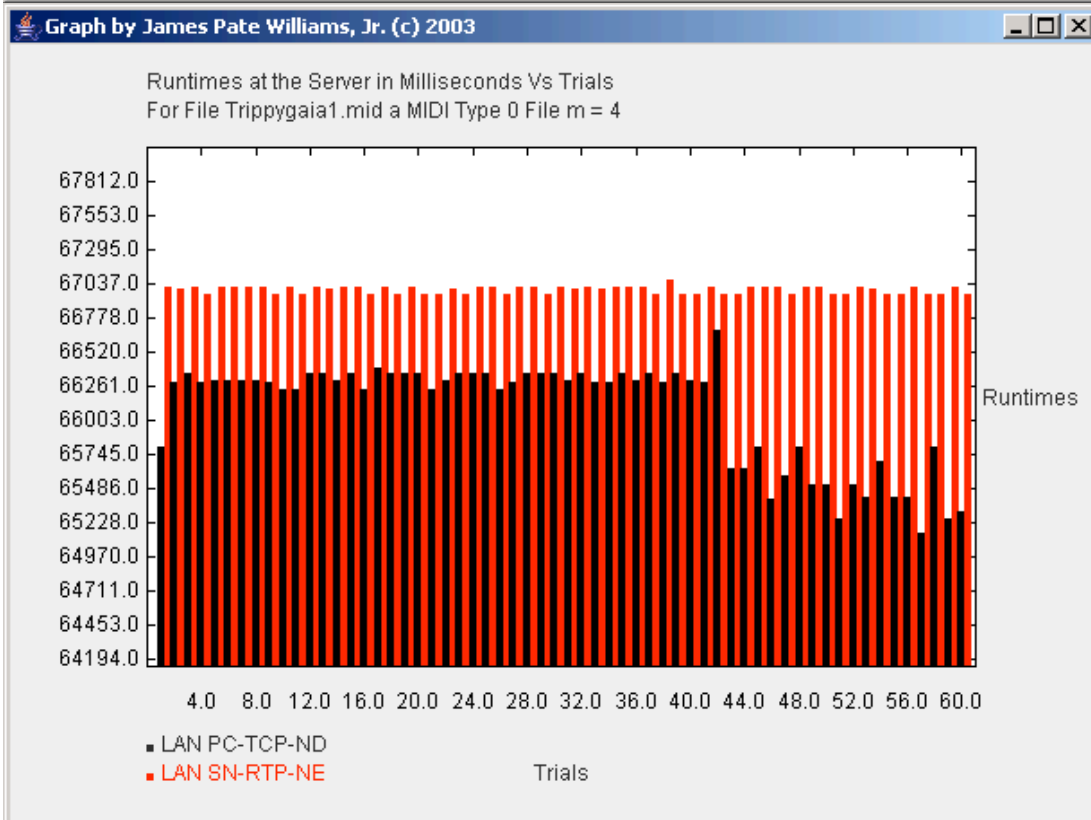
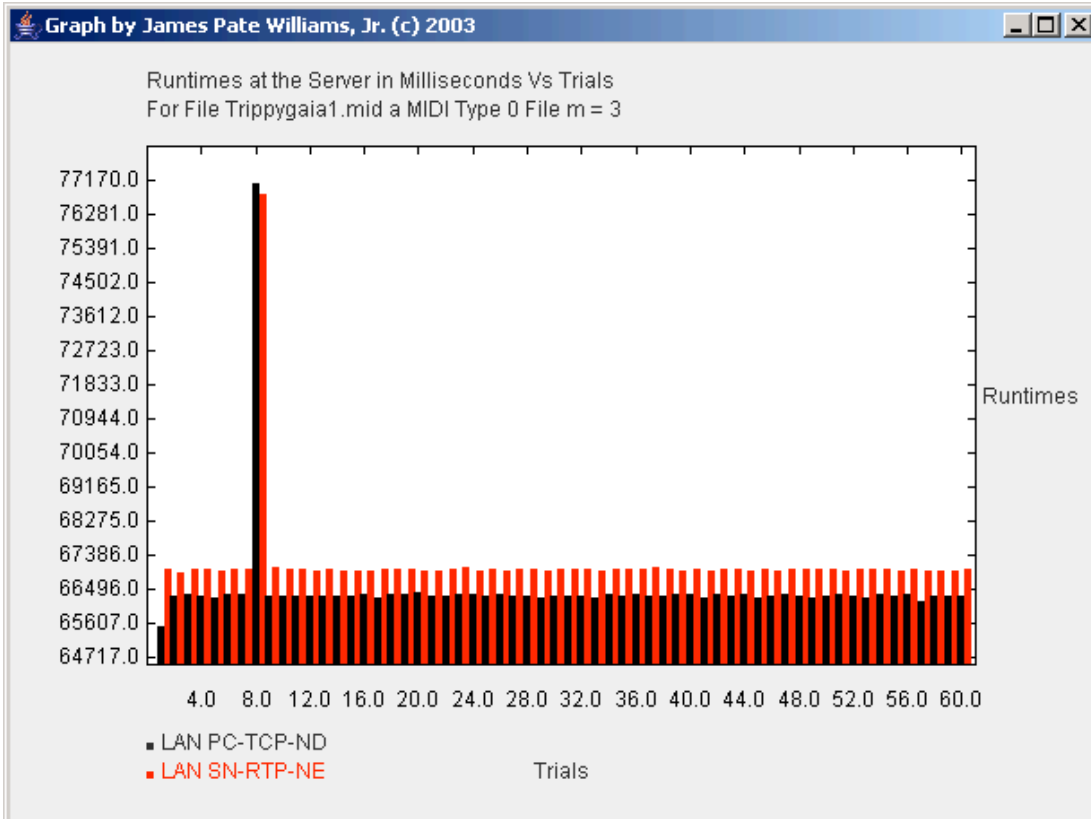


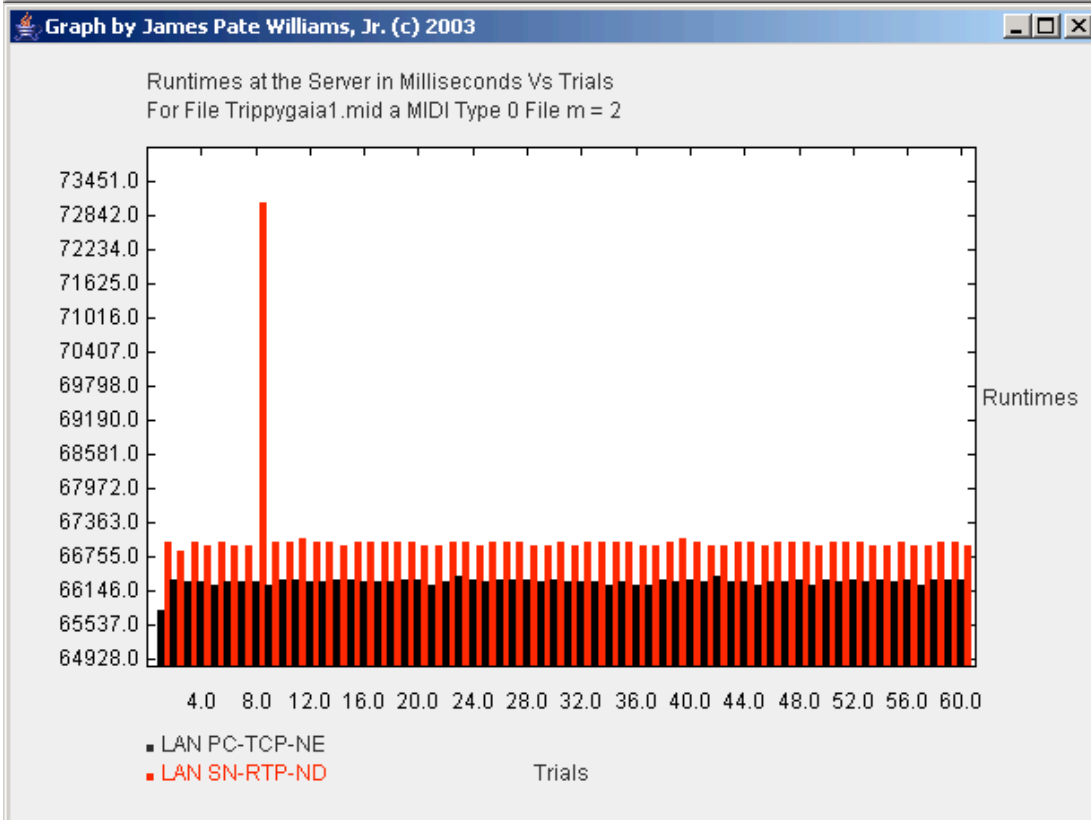
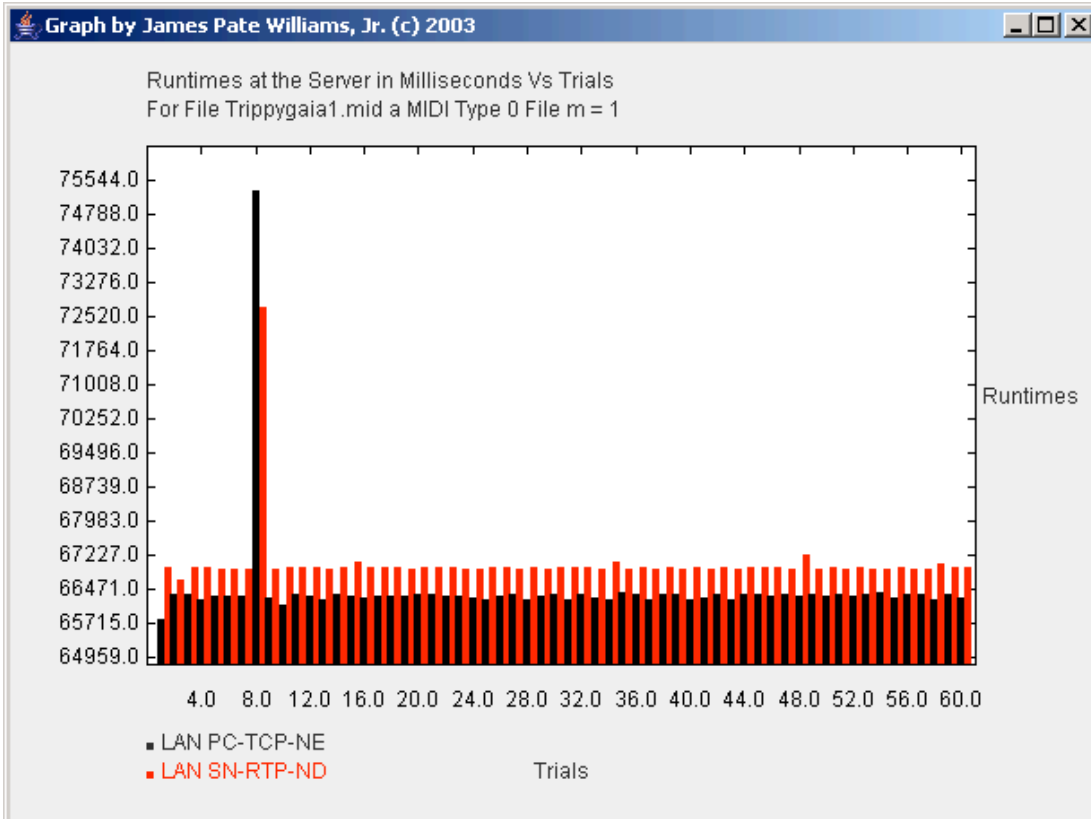


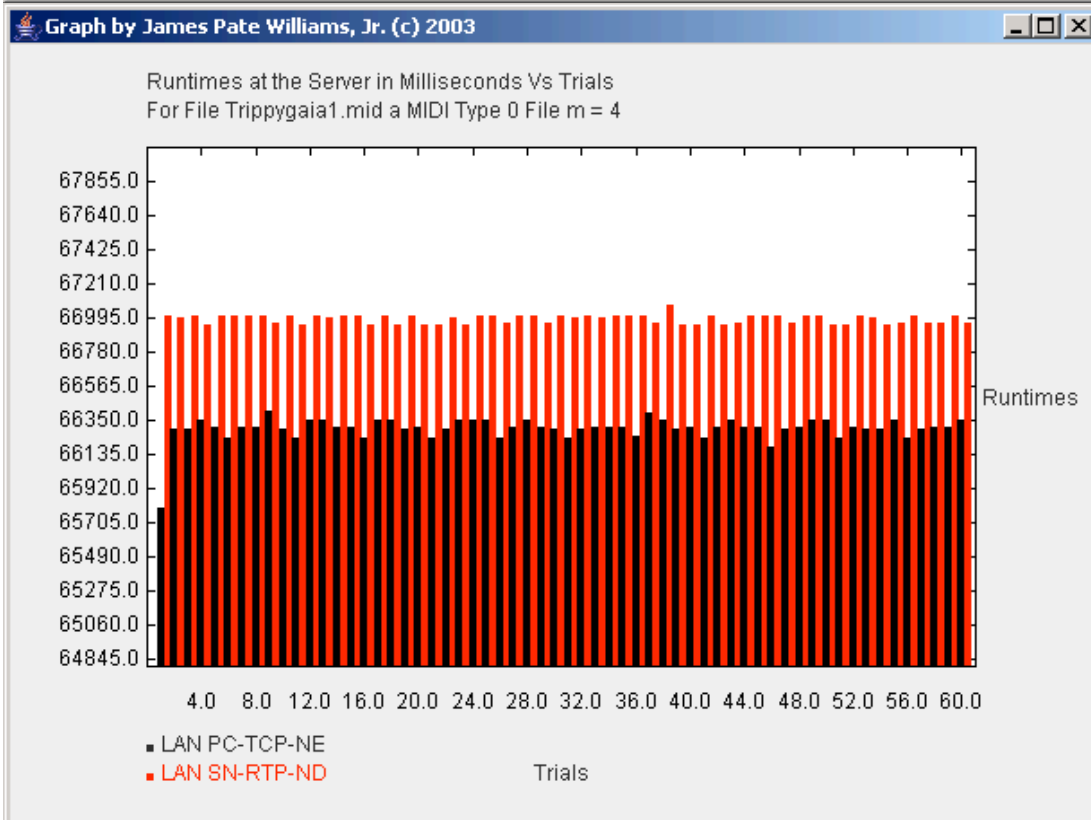
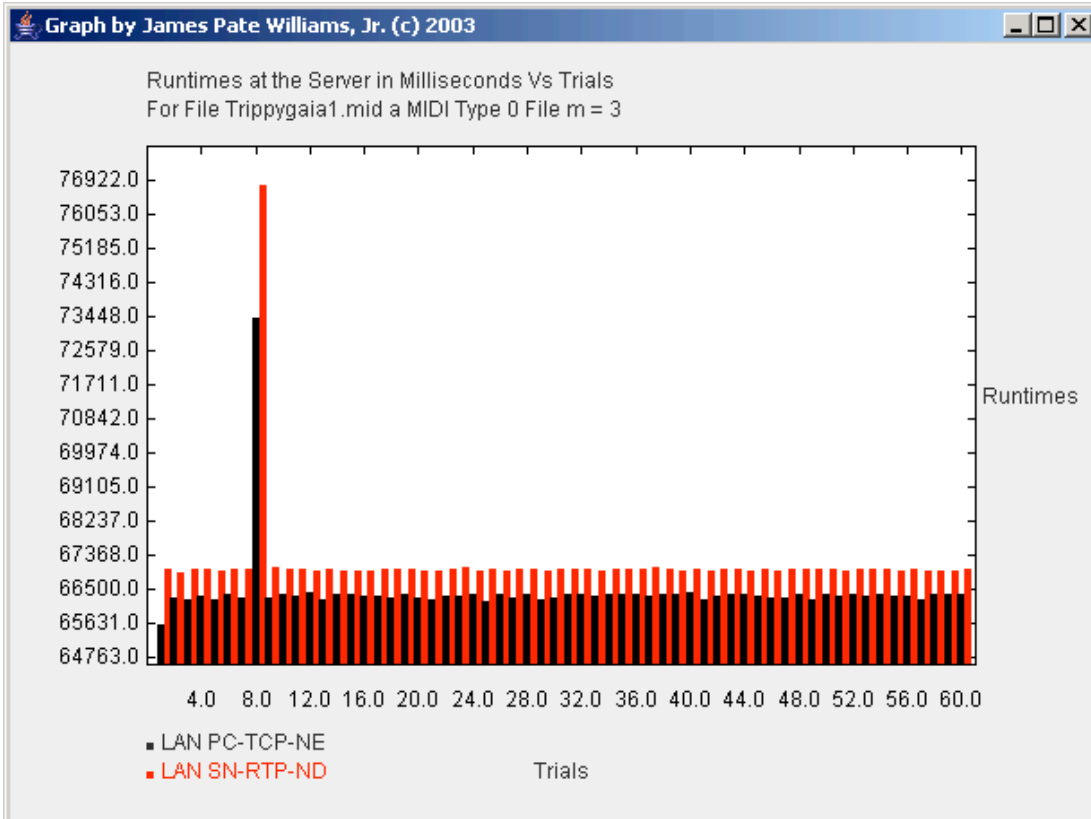


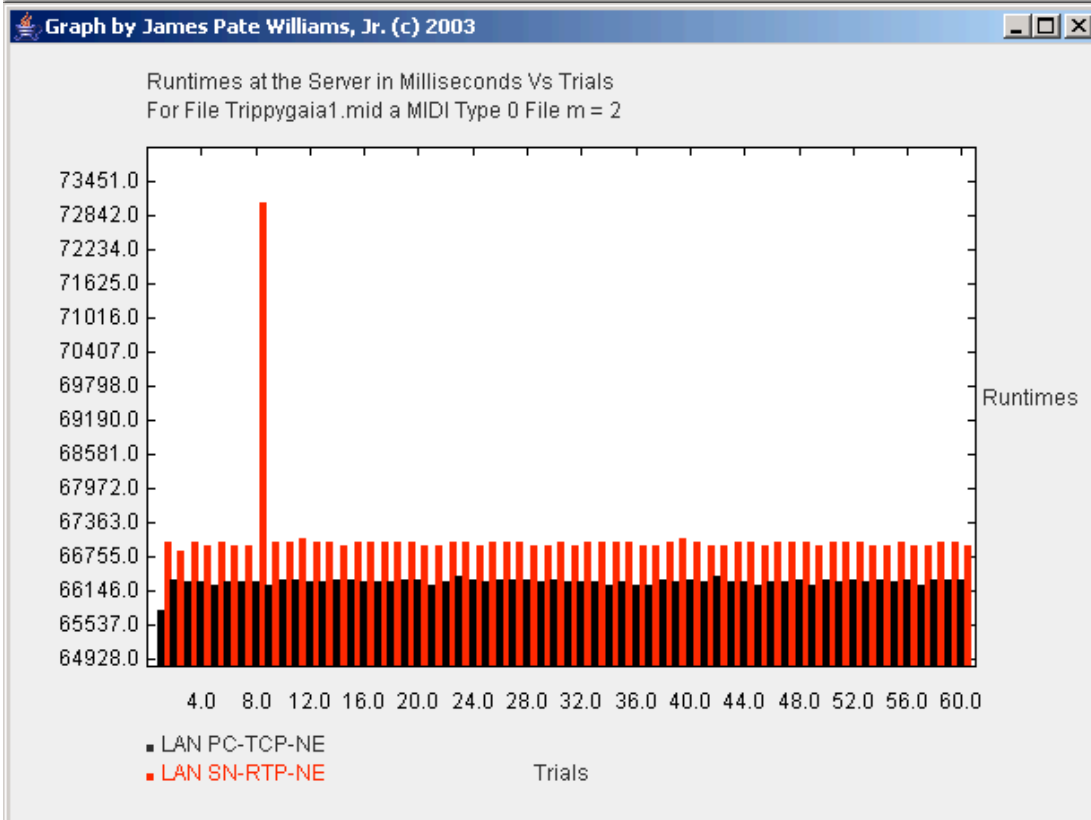
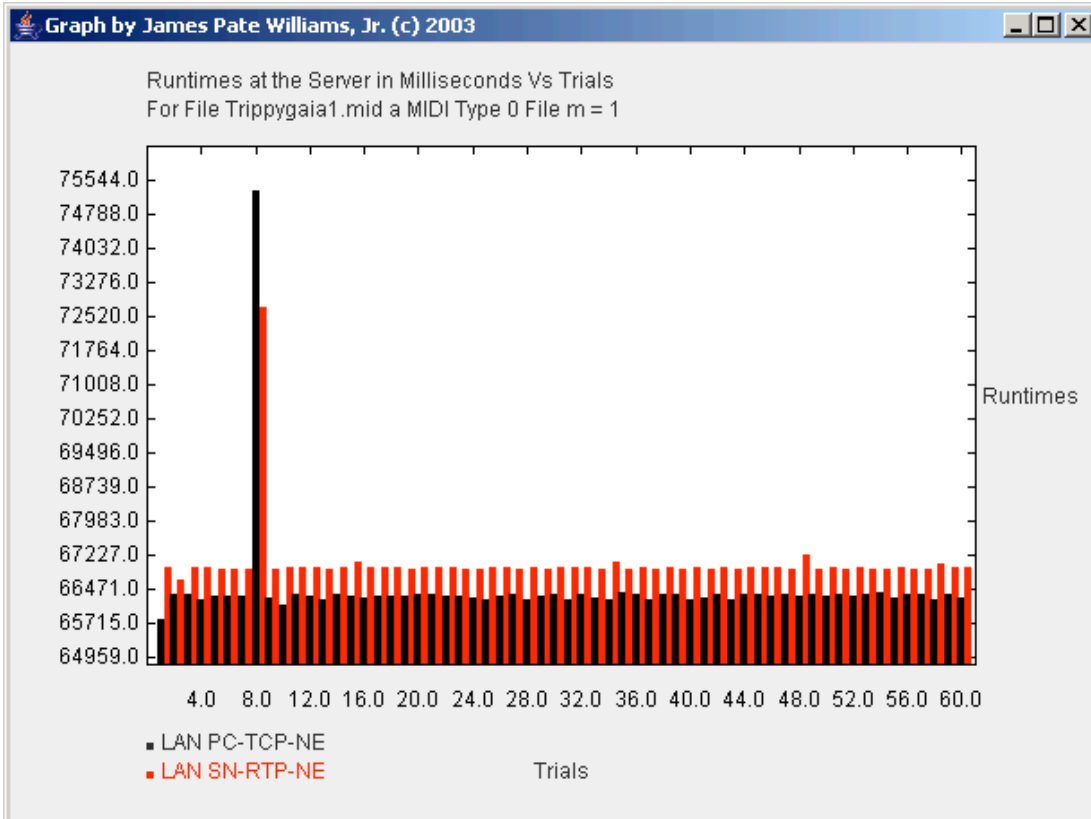


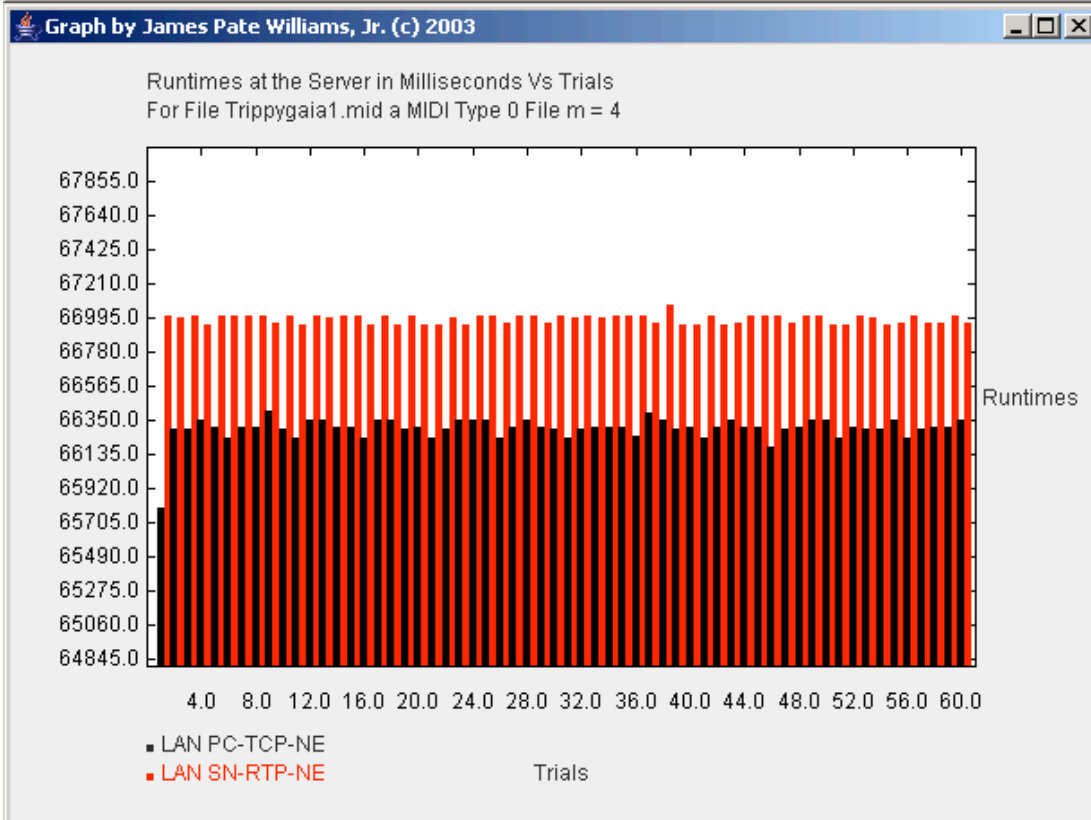
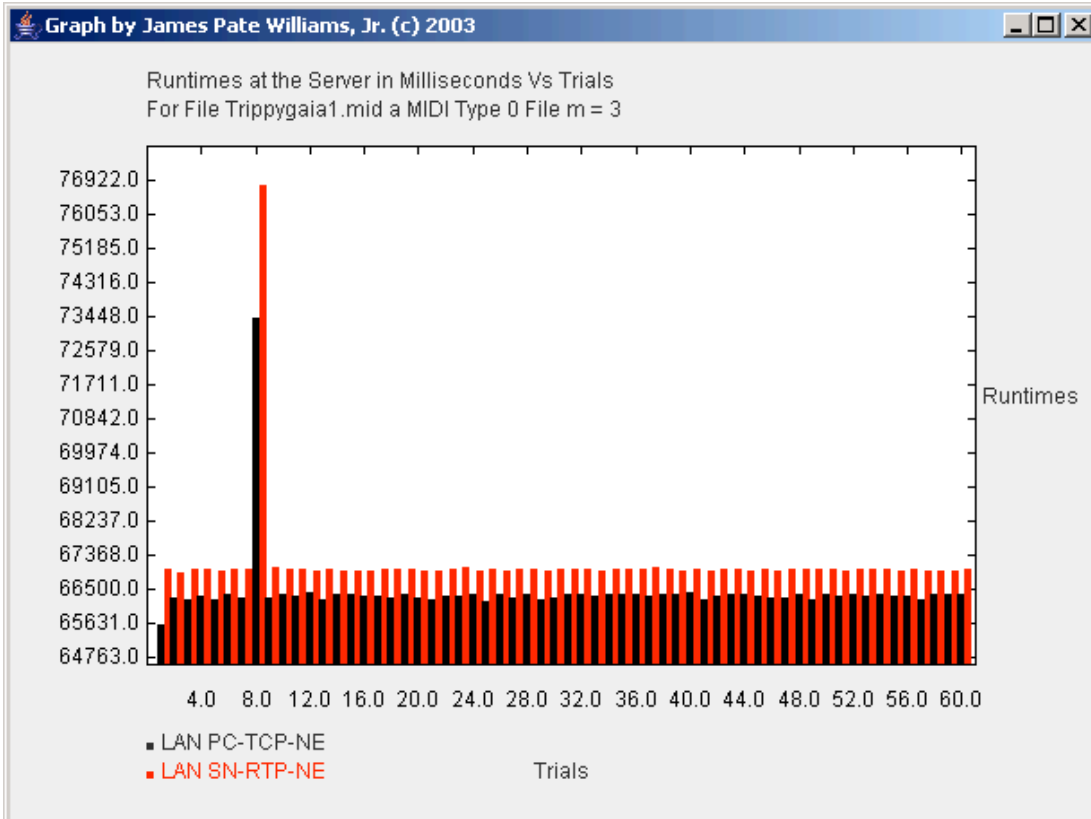


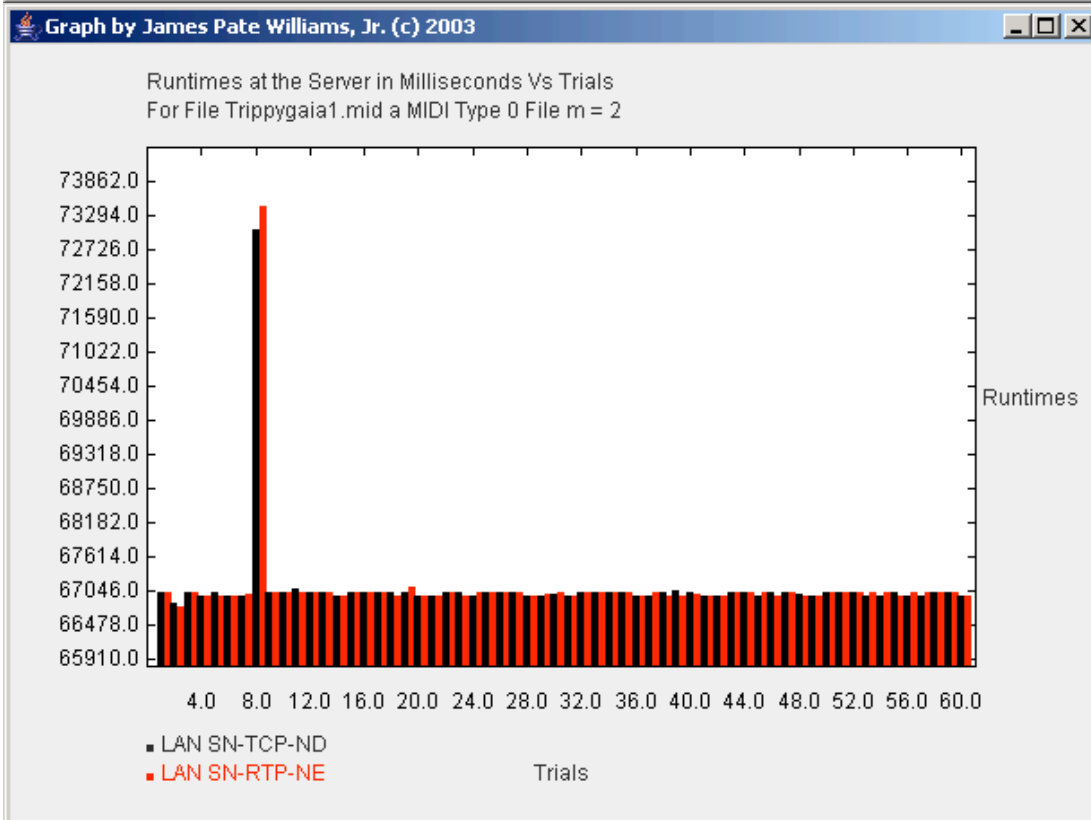
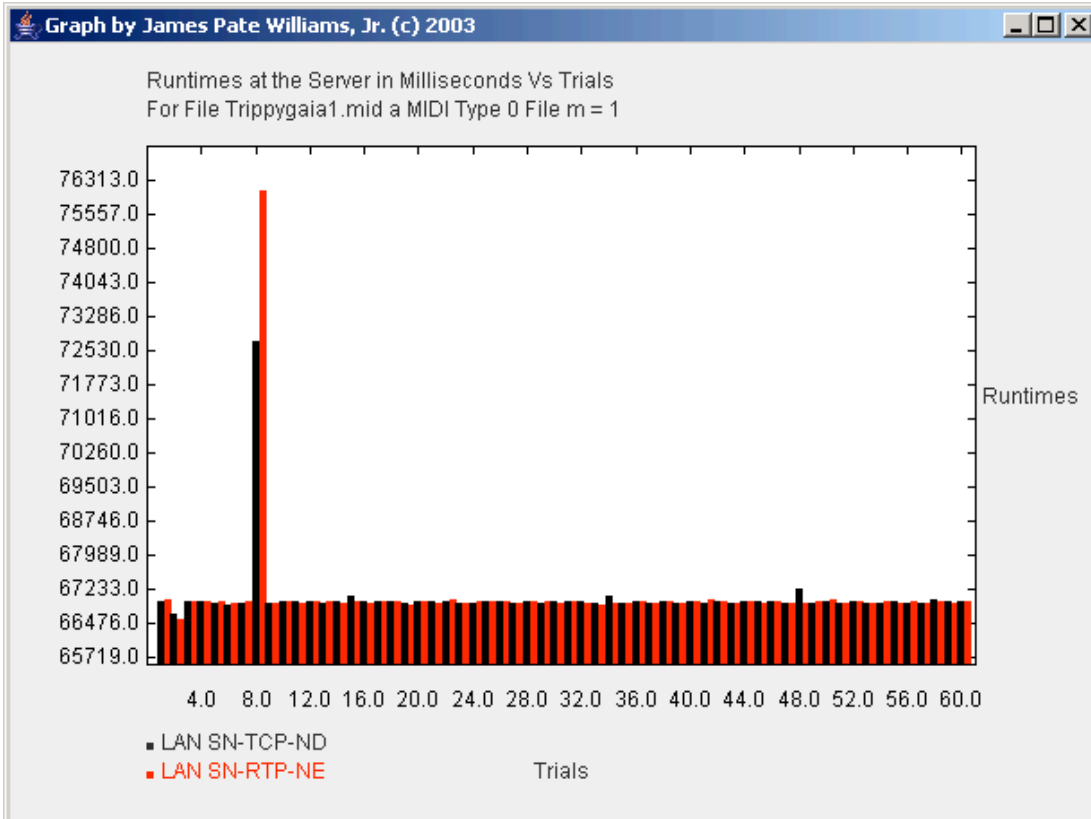


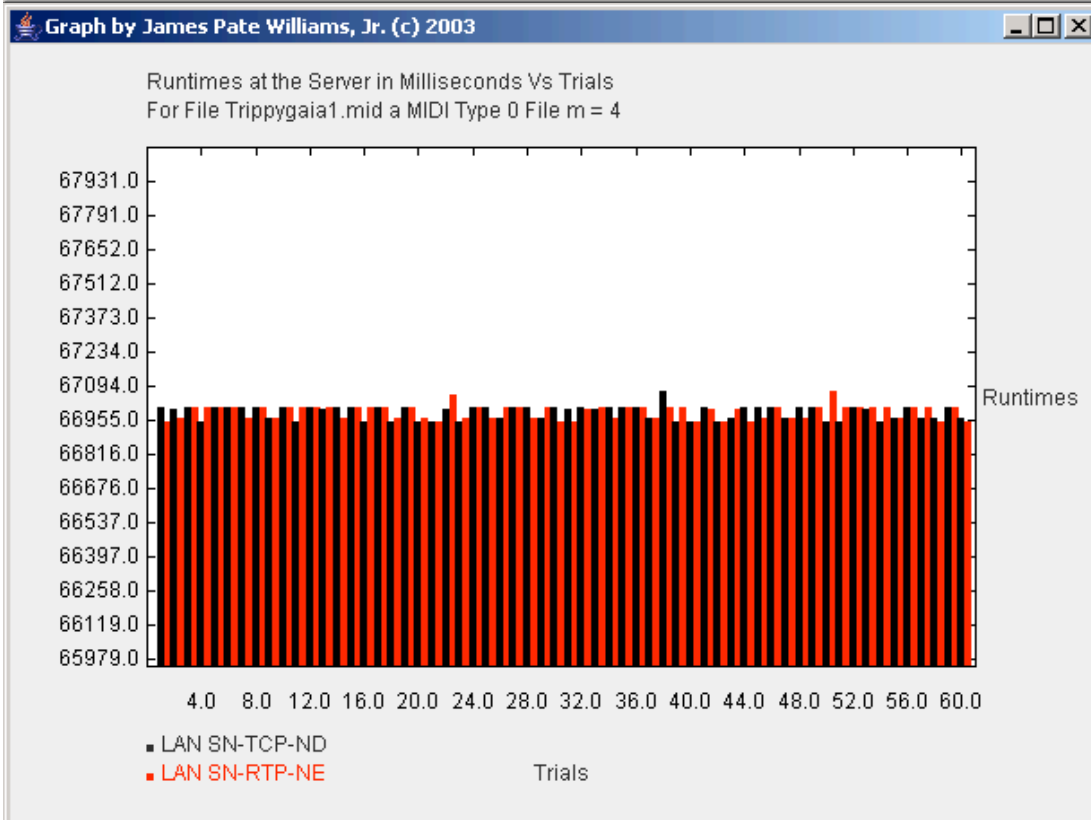
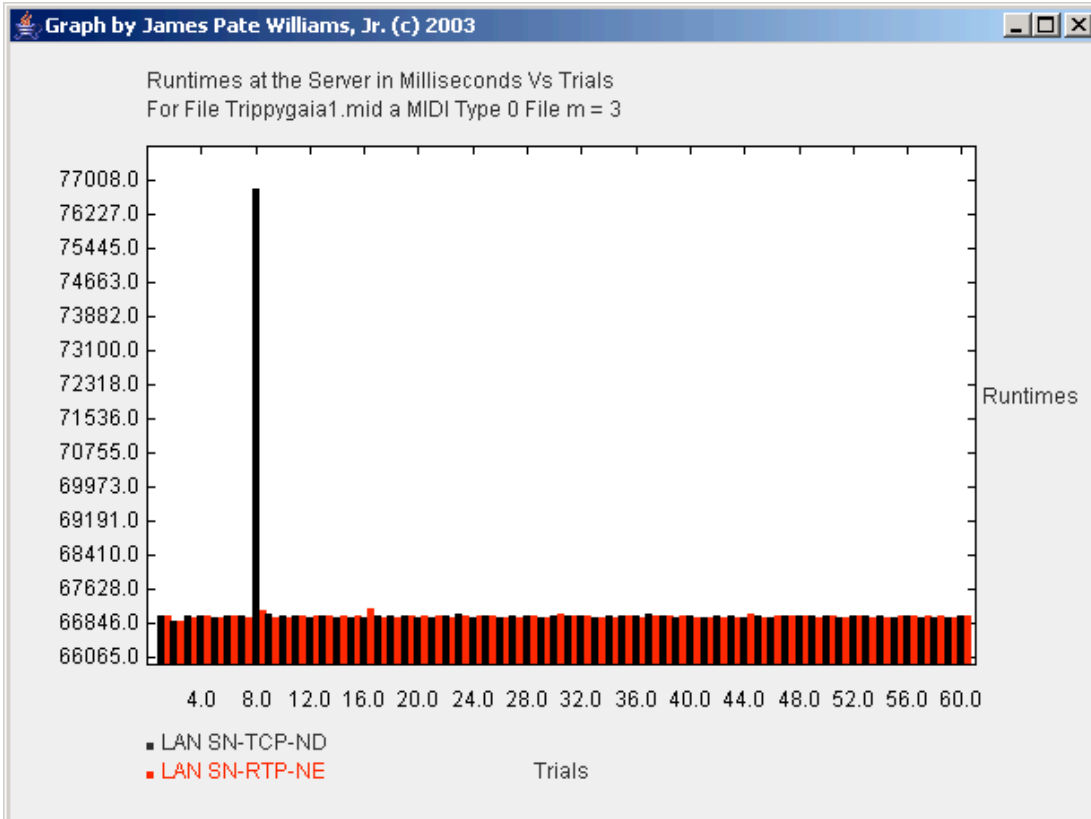








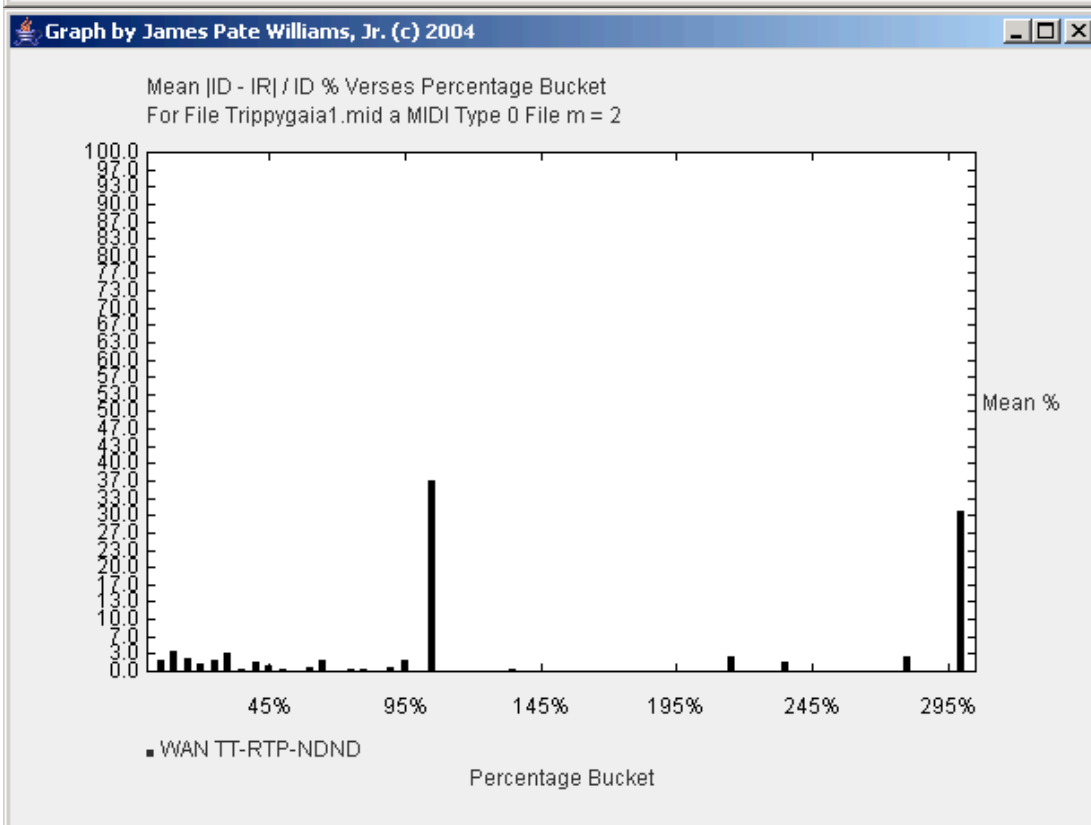
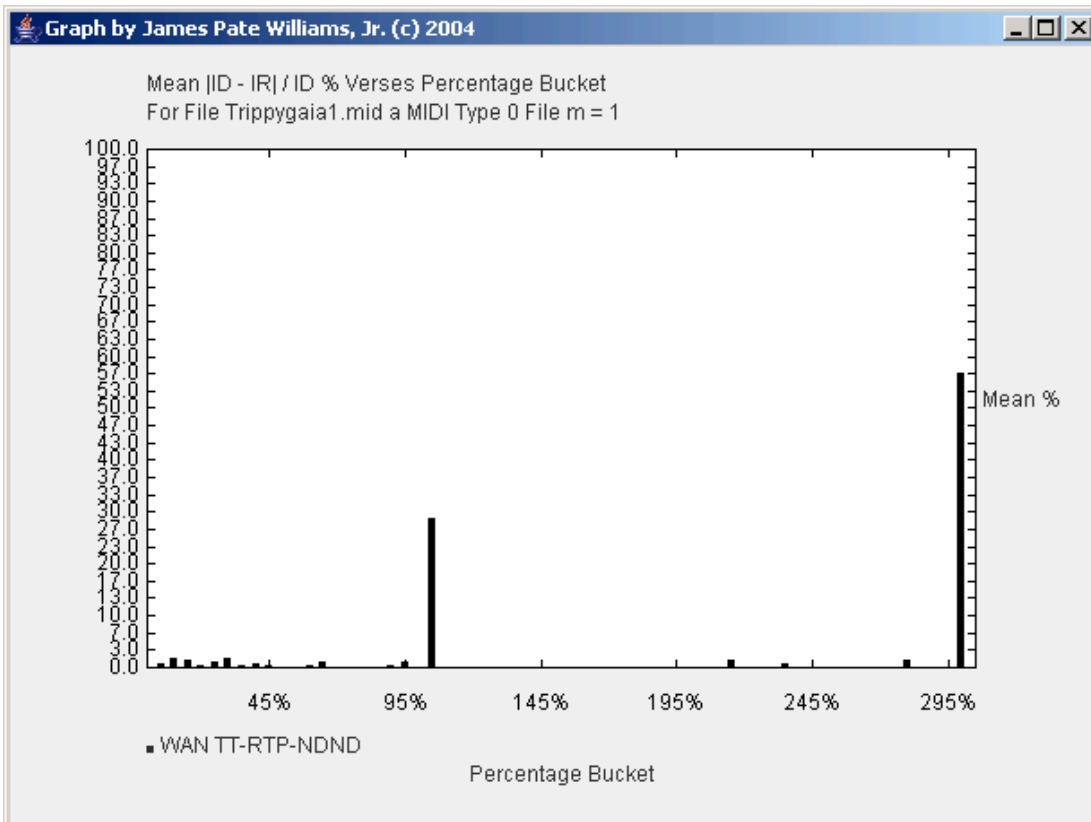


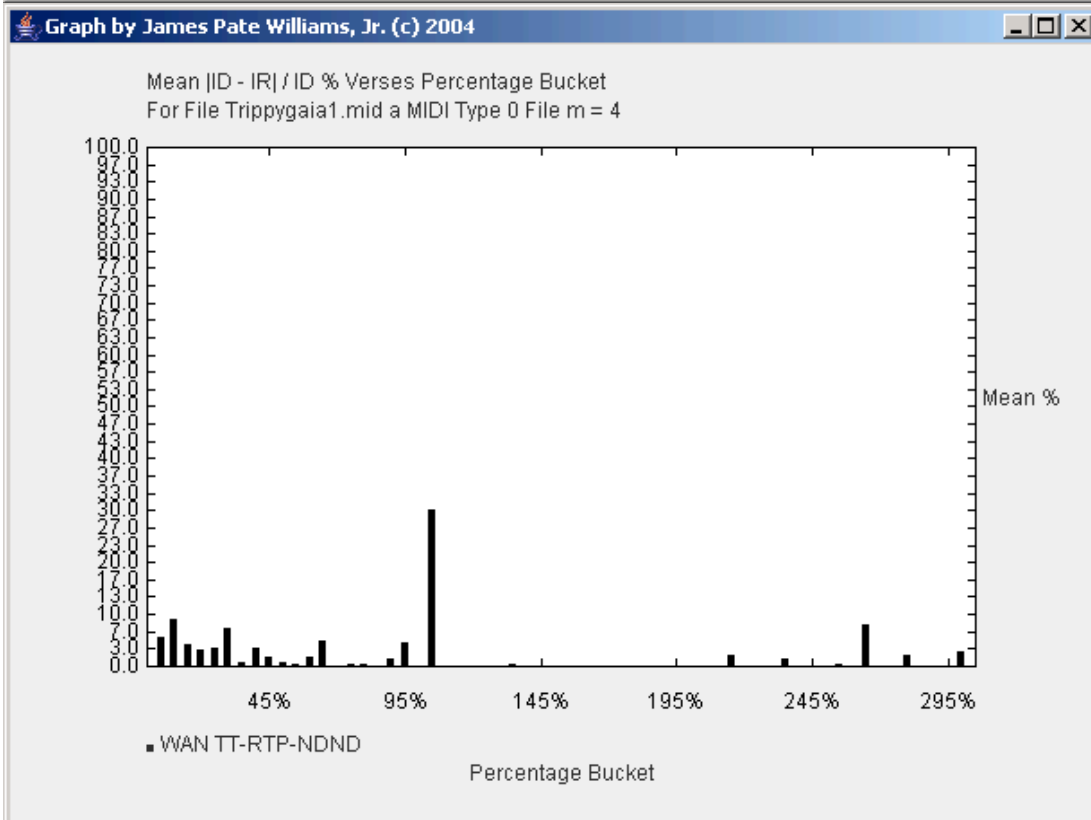
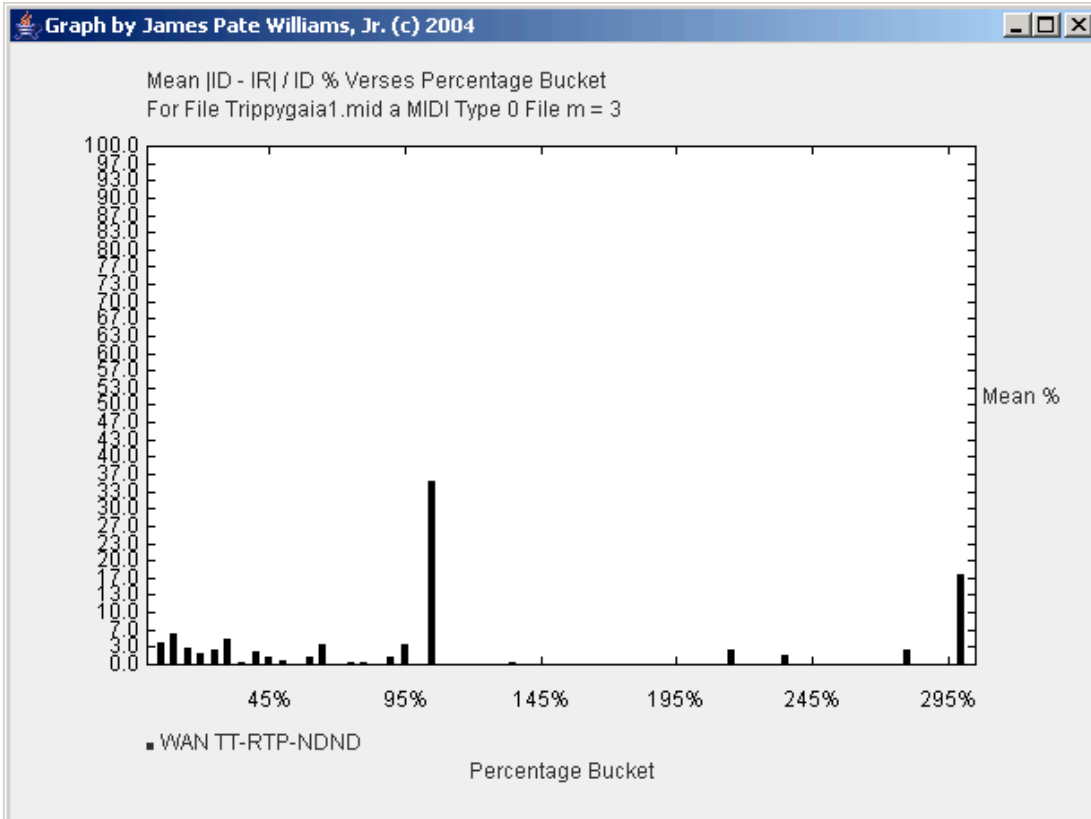


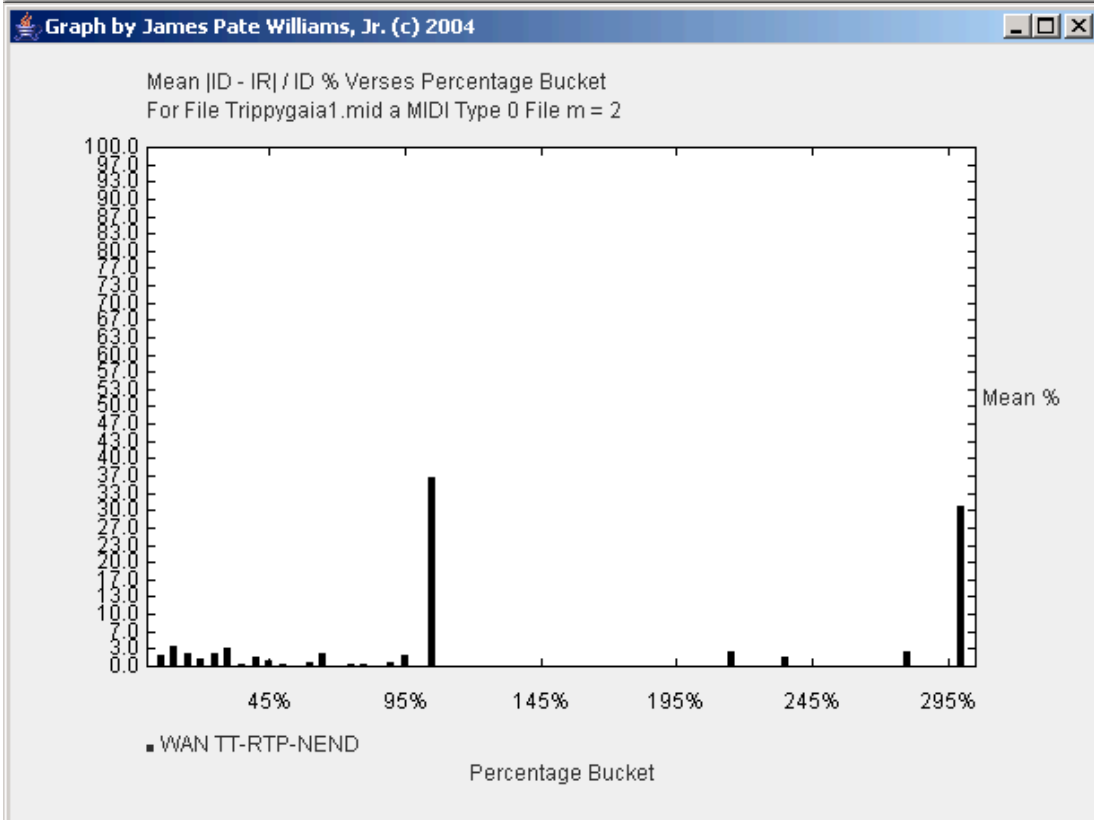
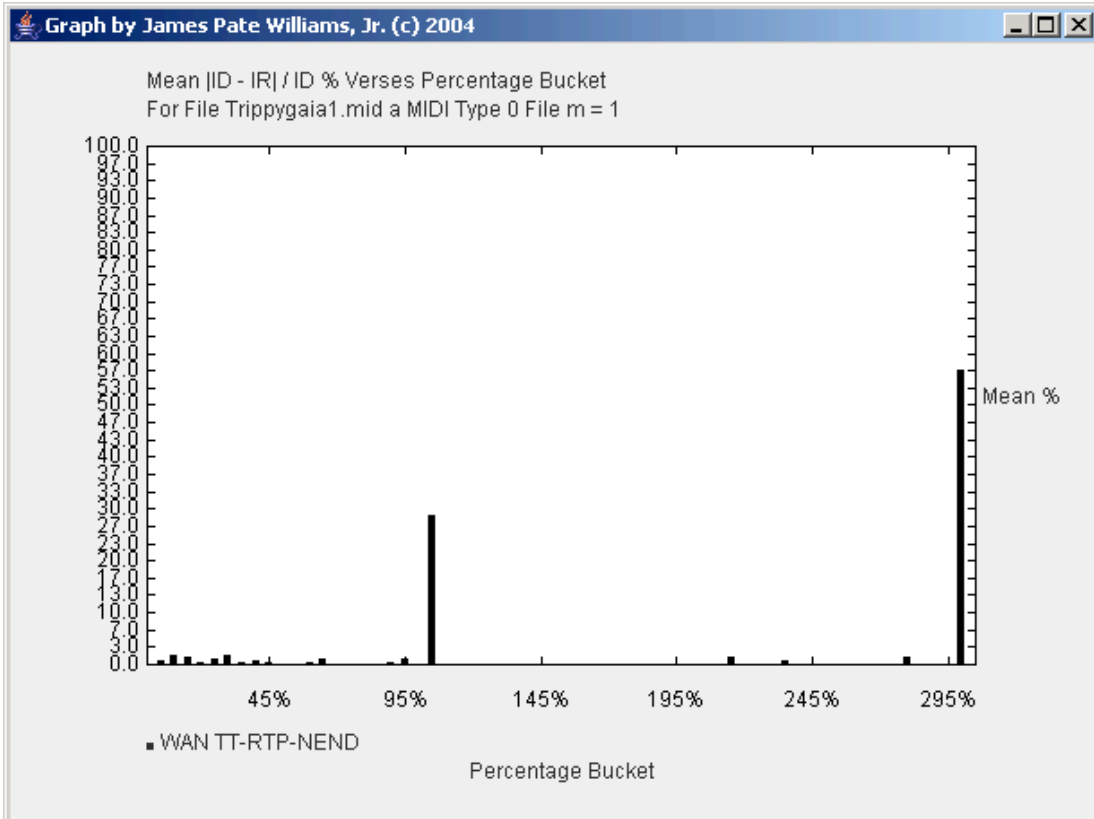


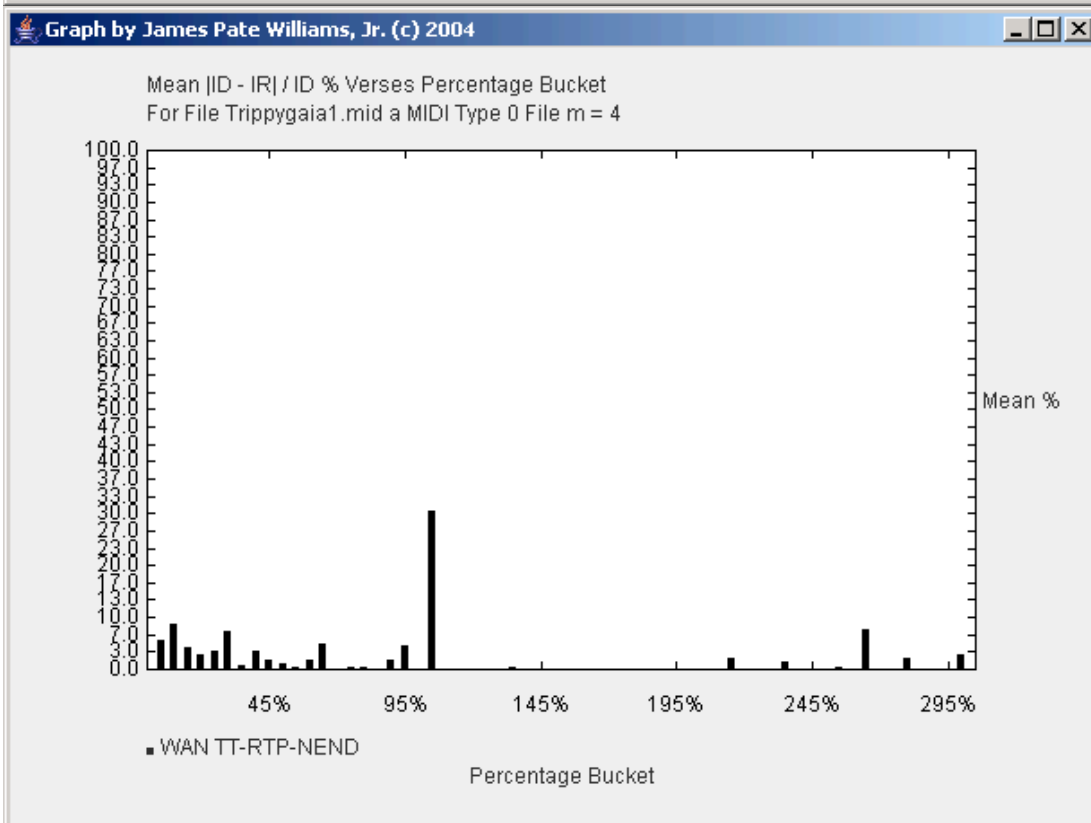
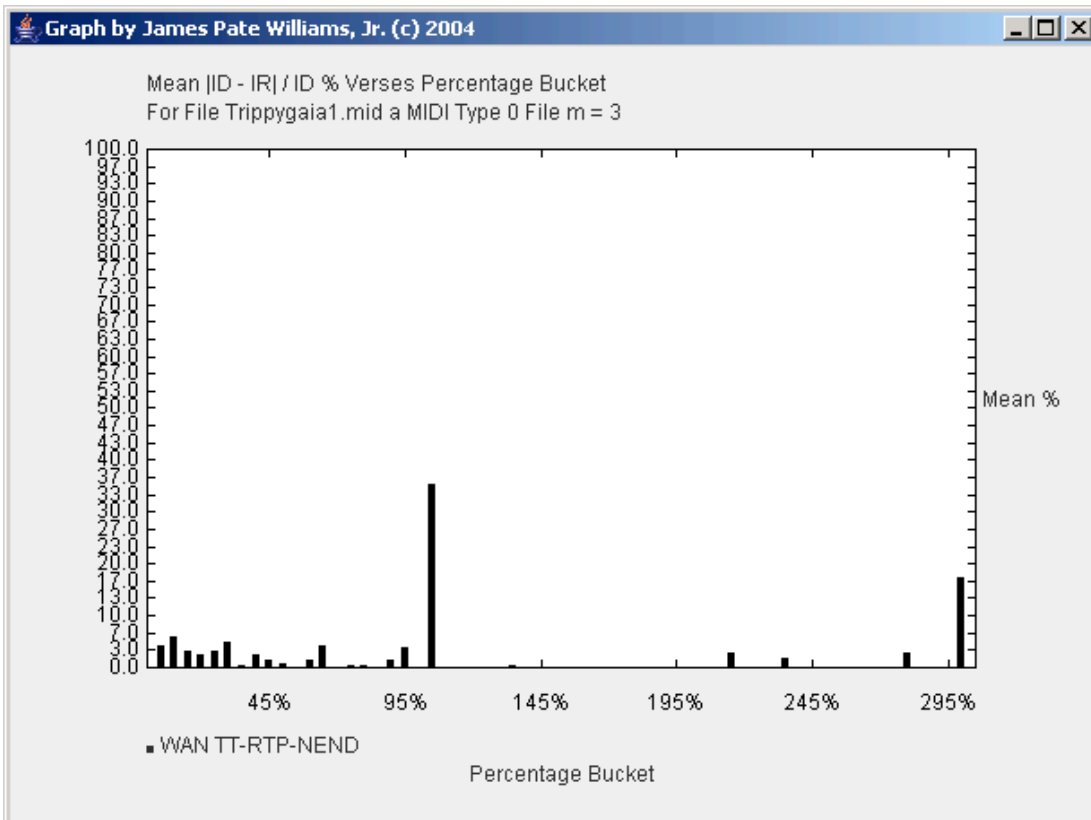
## APPENDIX C WAN EQUATION (1) CHAPTER 4 GRAPHS

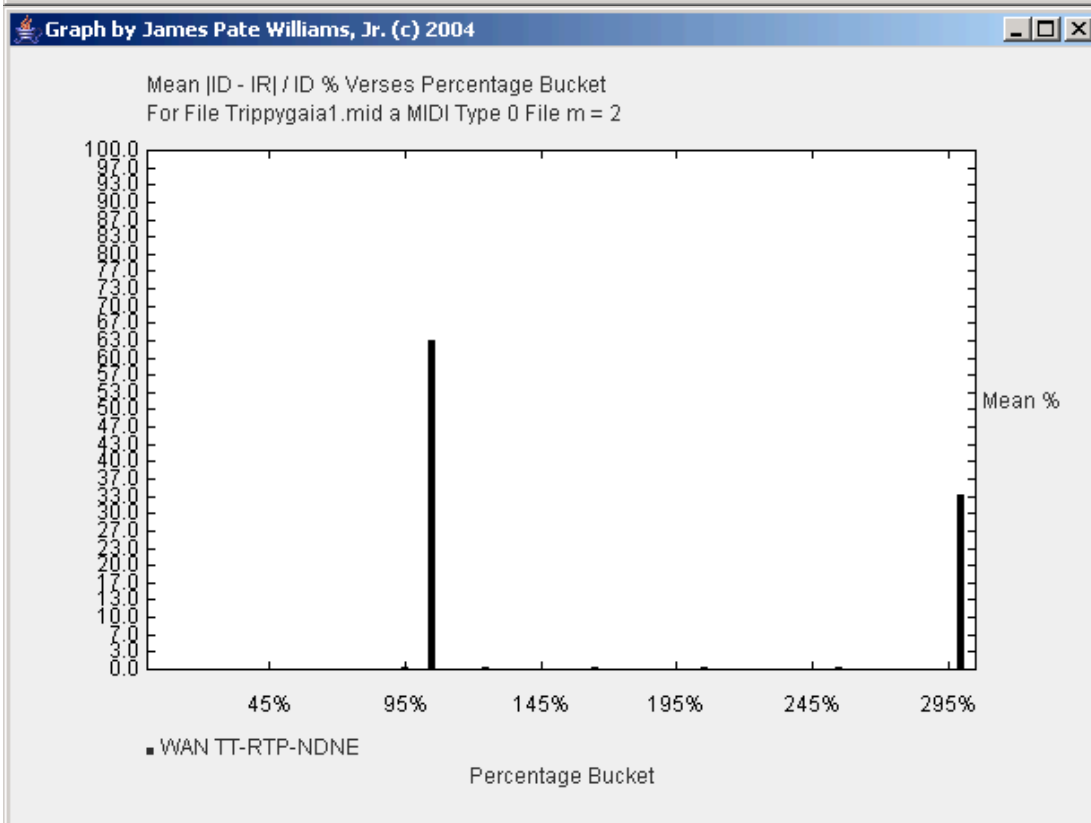
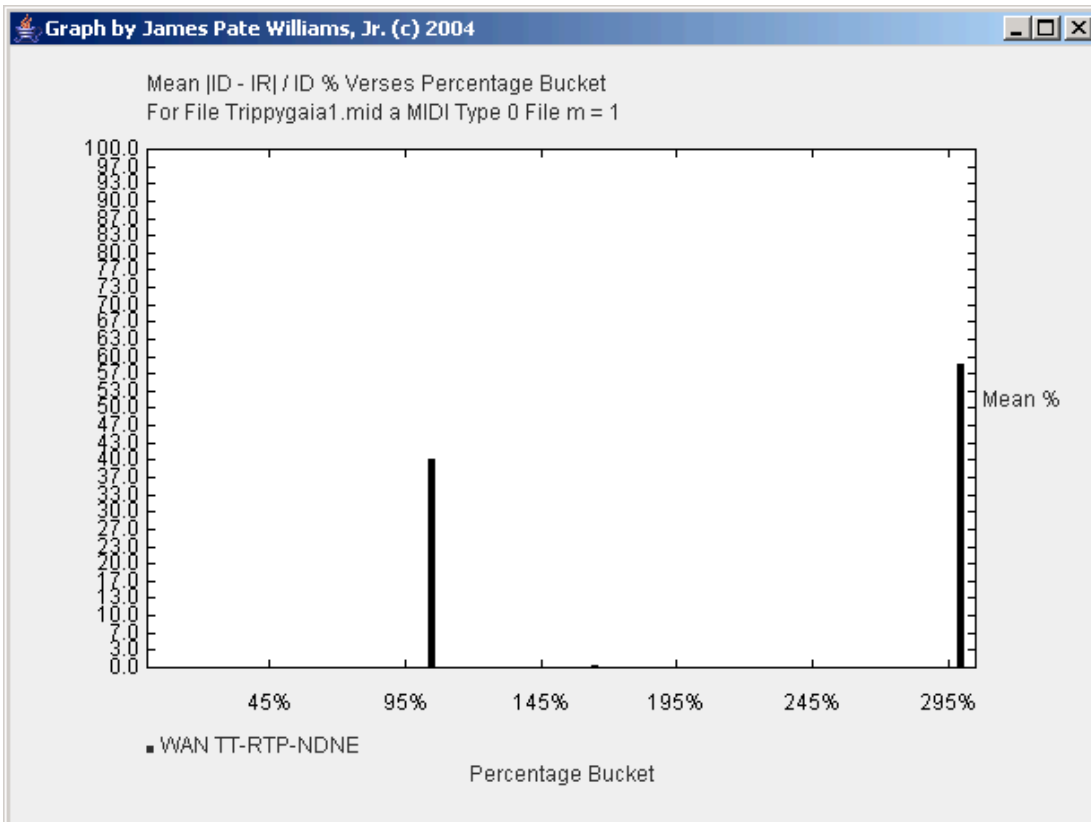
There are forty graphs in this appendix for a local area network (WAN) that were generated from Equation (1) in Chapter 4. There are four graphs each for all ten protocols in the suite of MIDI over IP protocols. To reiterate the protocols are as follows: TT-RTP-NDND, TT-RTP-NEND, TT-RTP-NDNE, TT-RTP-NENE, UT-RTP-ND, UT-RTP-NE, PC-RTP-ND, PC-TCP-NE, SN-TCP-ND, and SN-TCP-NE, where ND = Nagle algorithm disabled and NE = Nagle algorithm enabled (typically the default Internet setting).

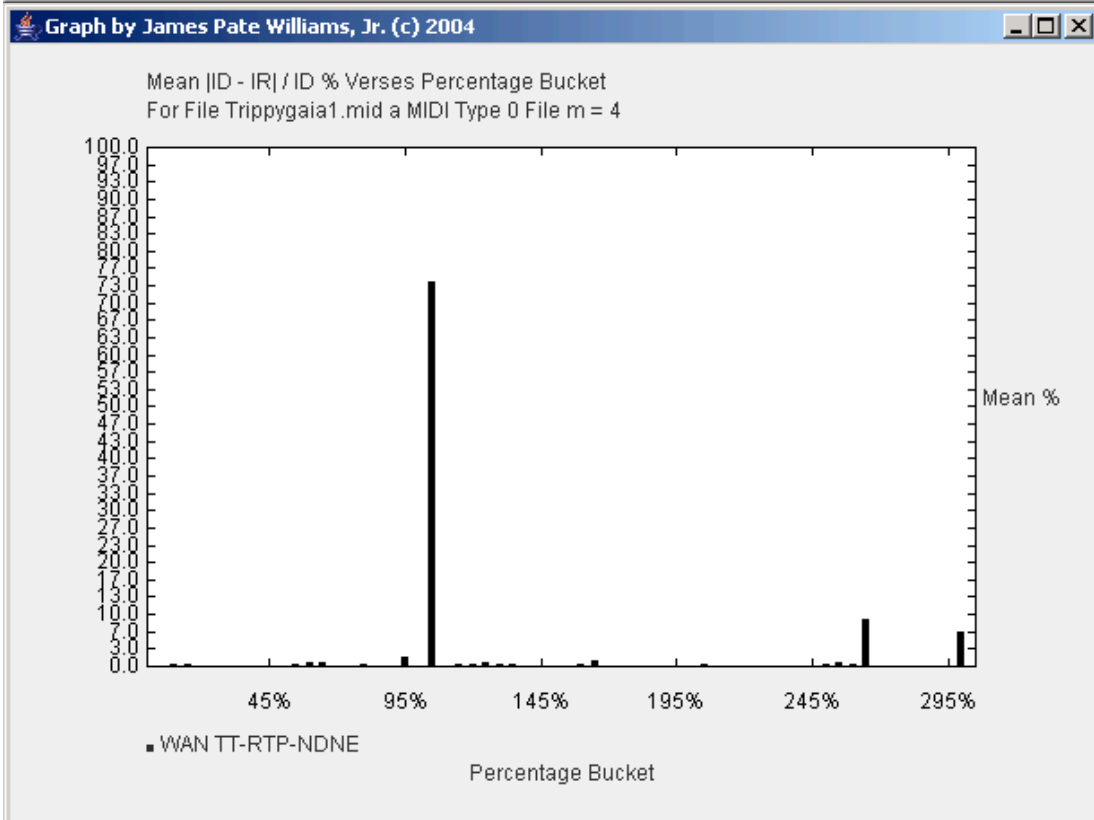
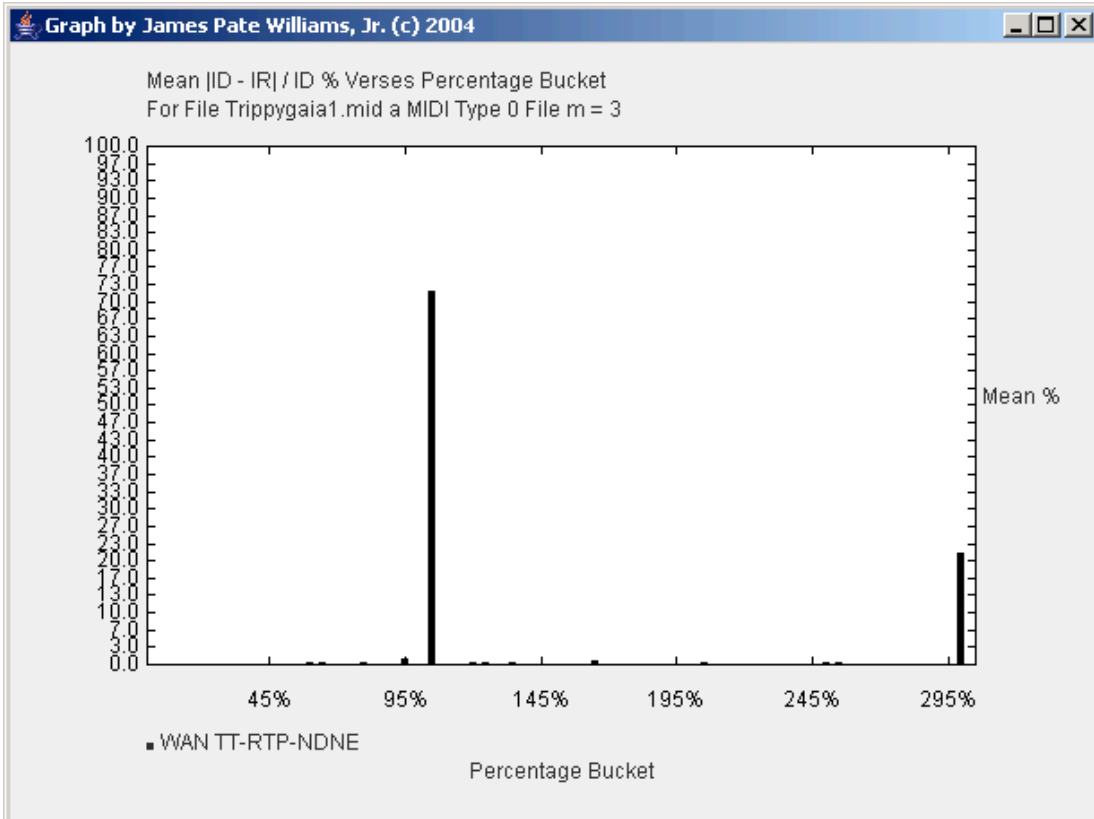


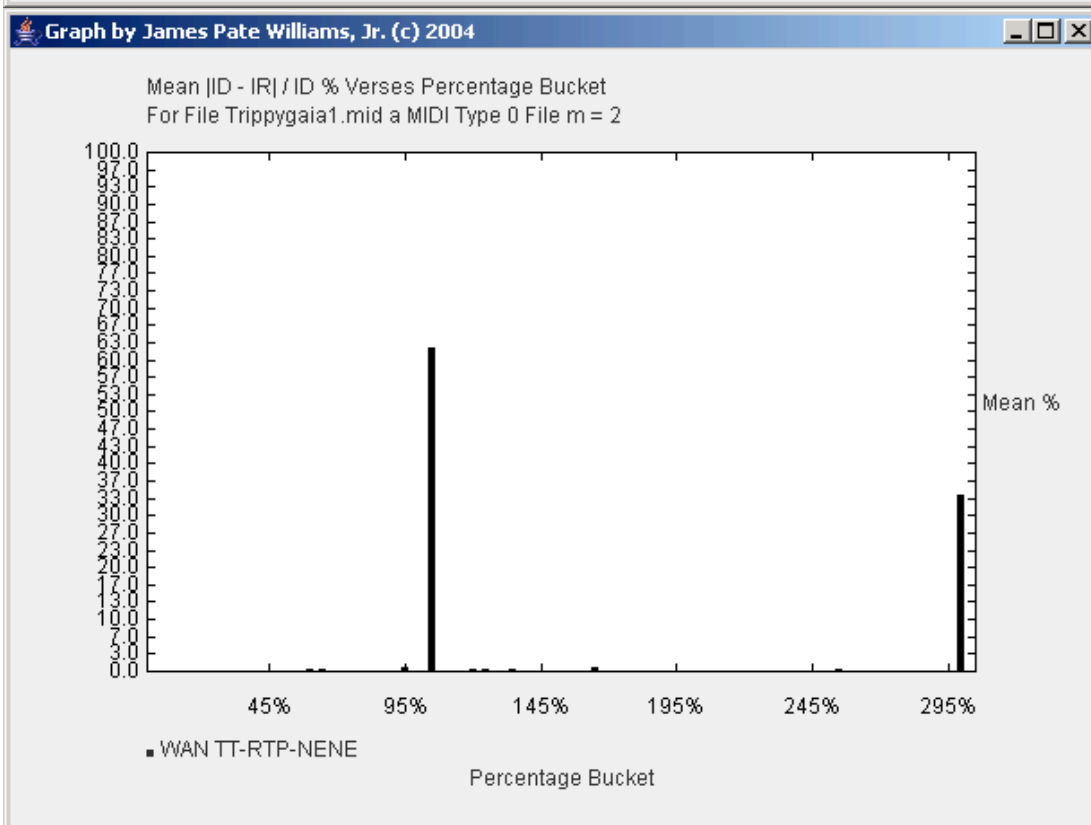
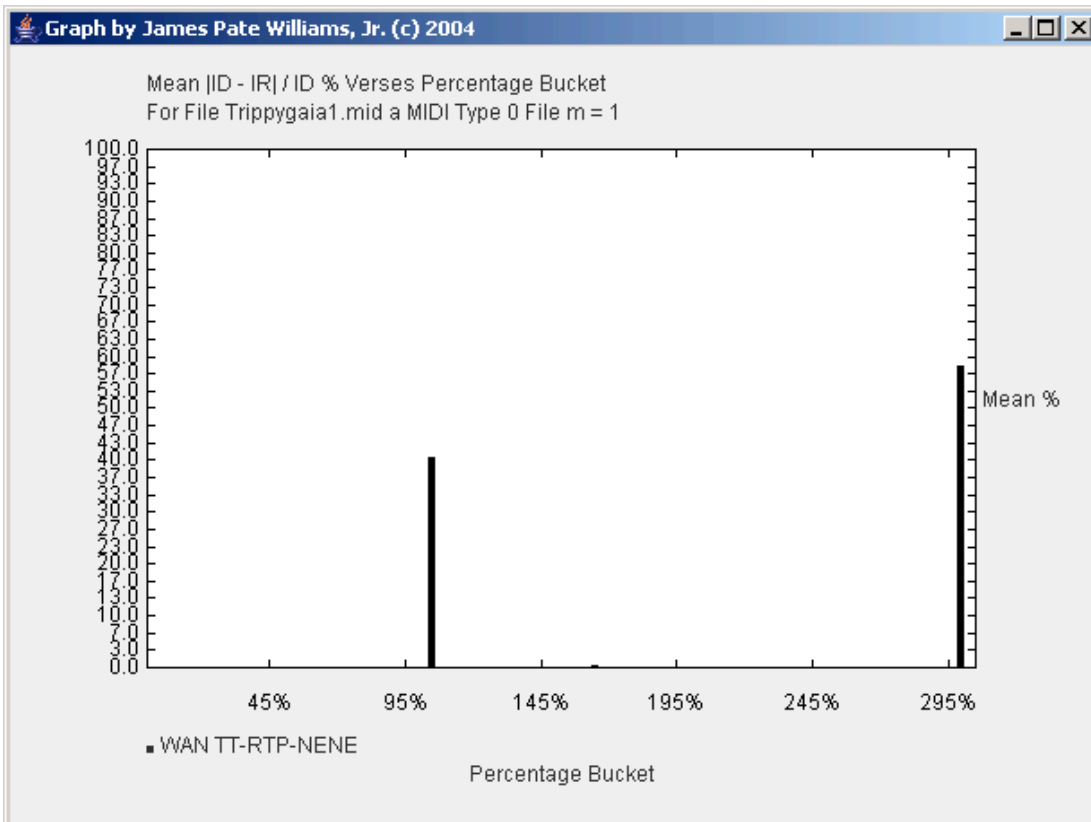




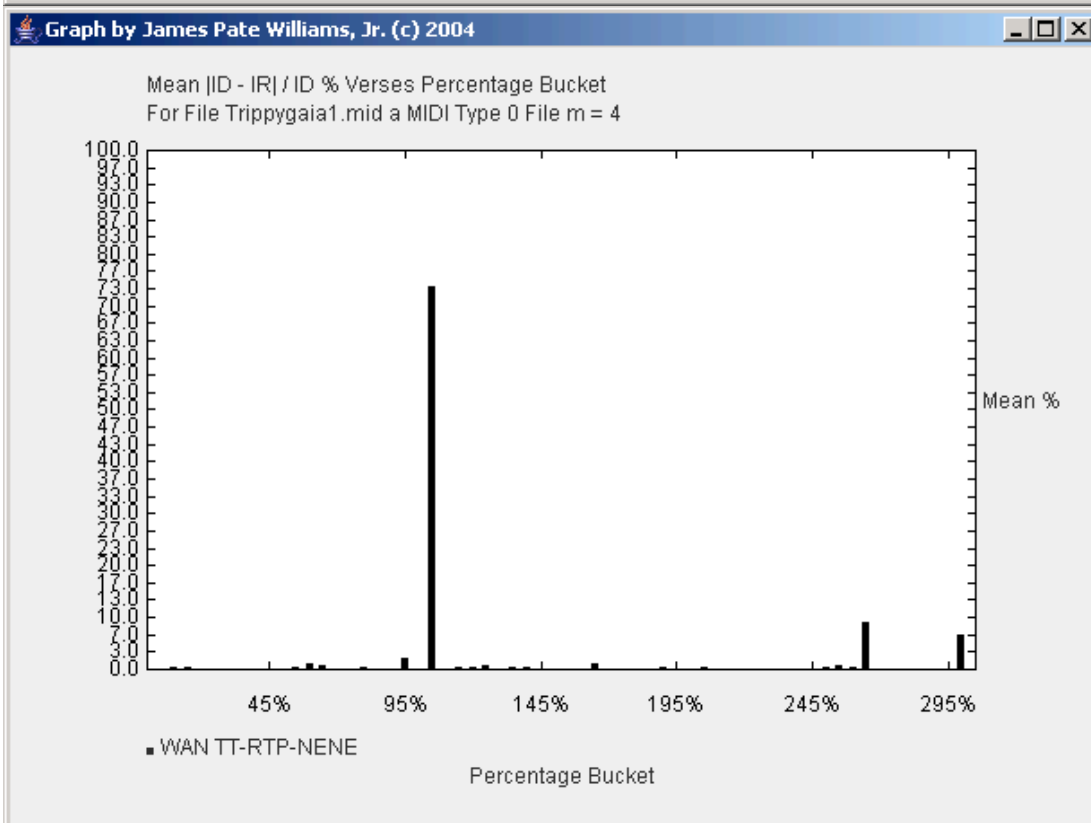
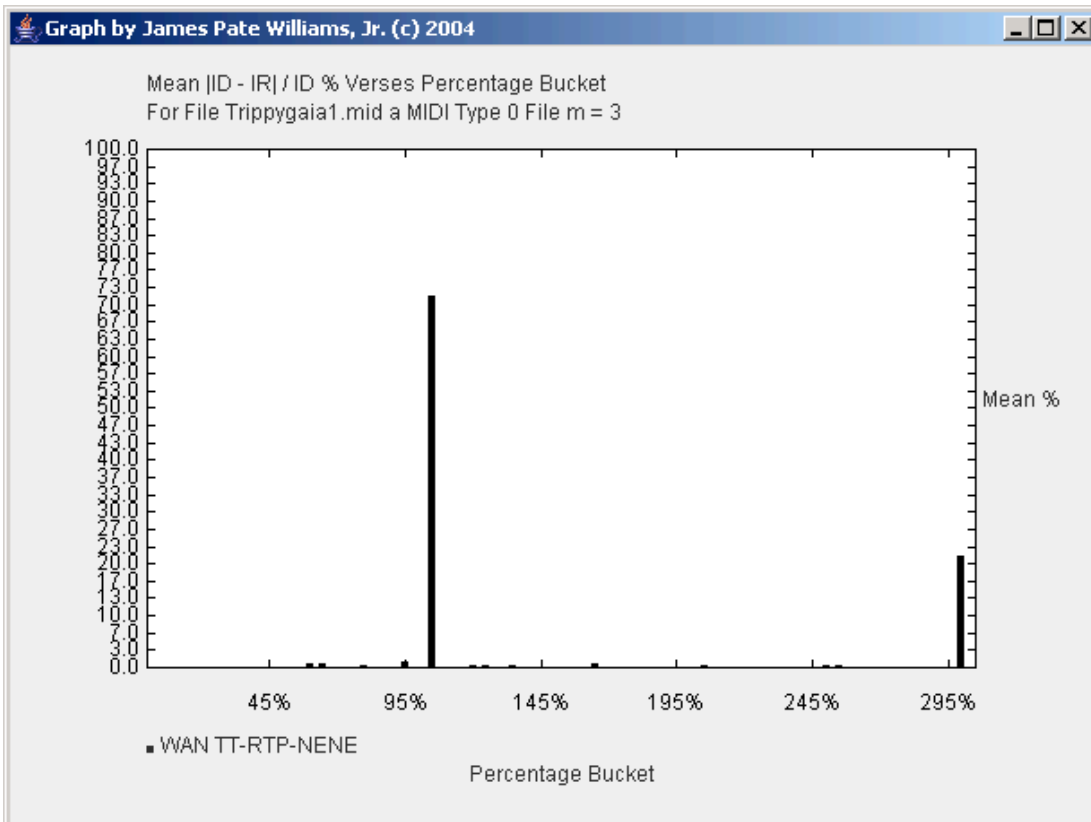


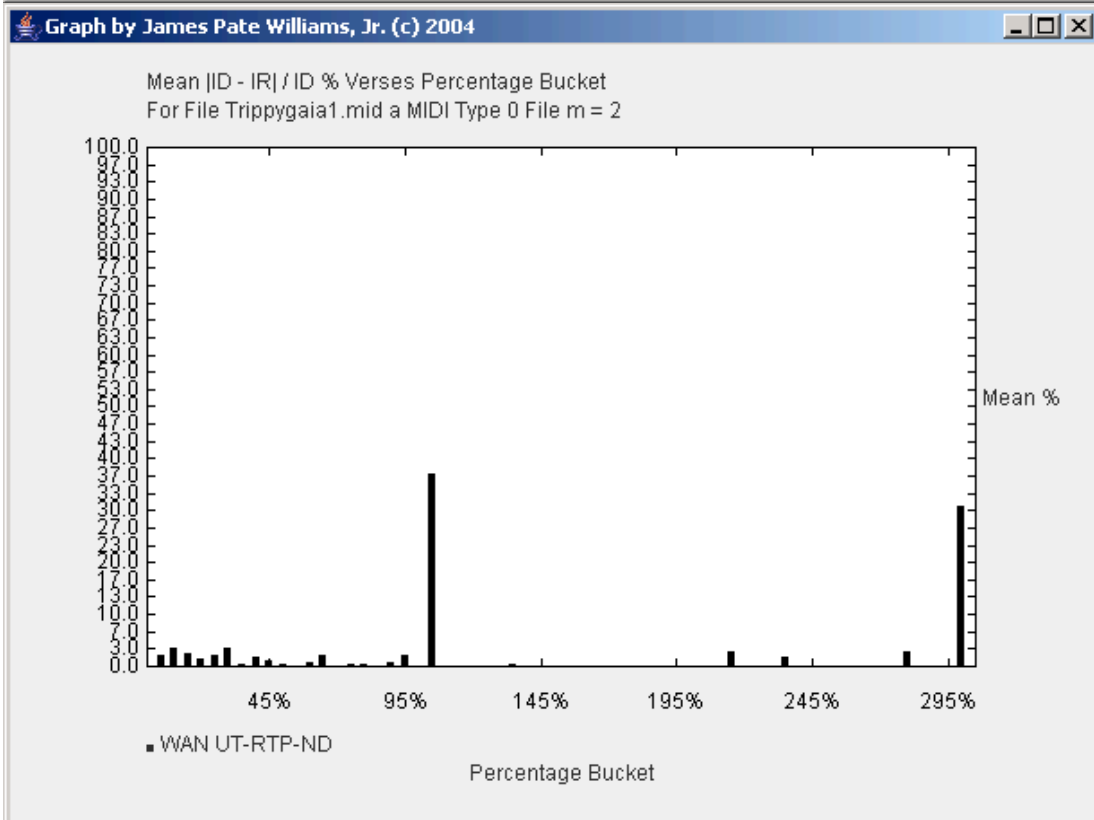
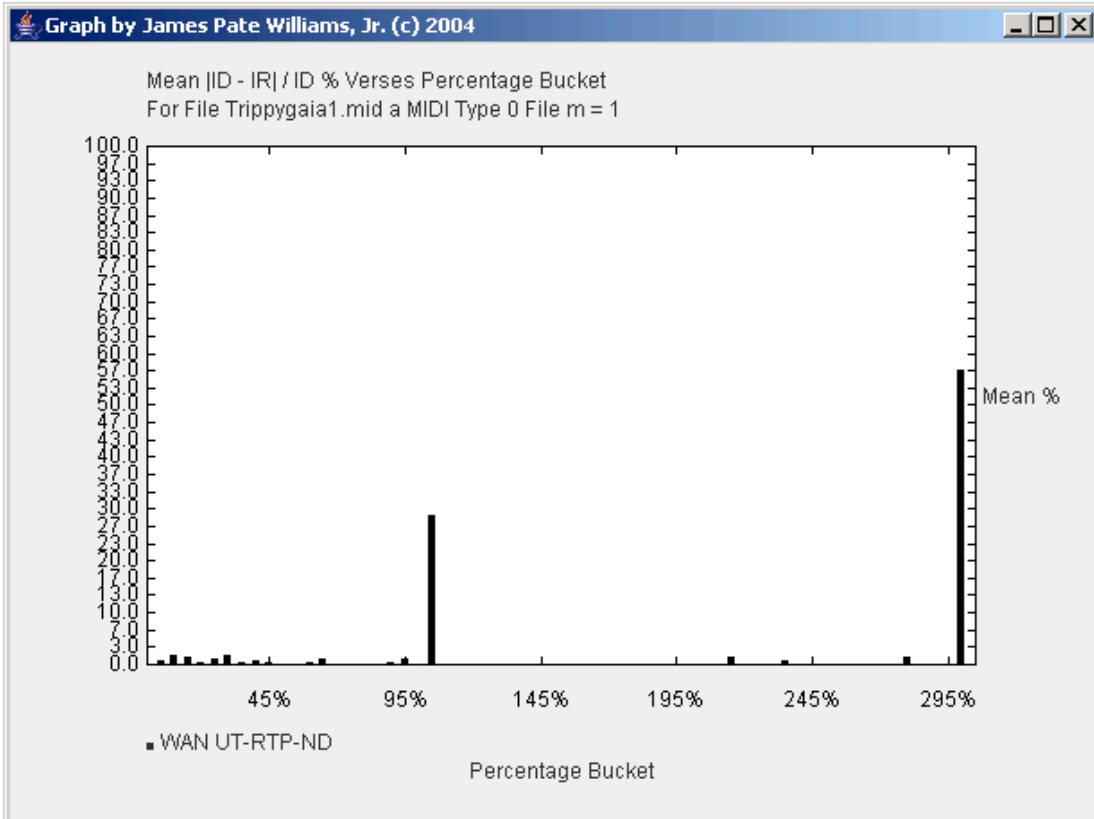


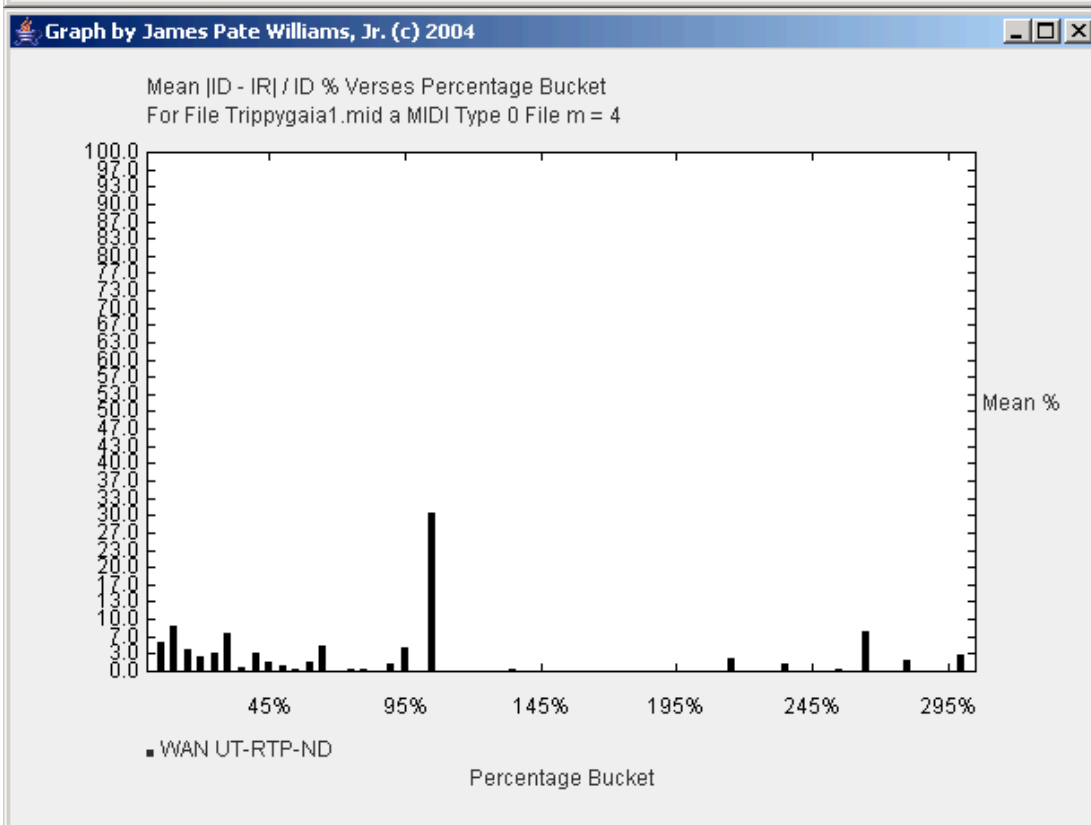
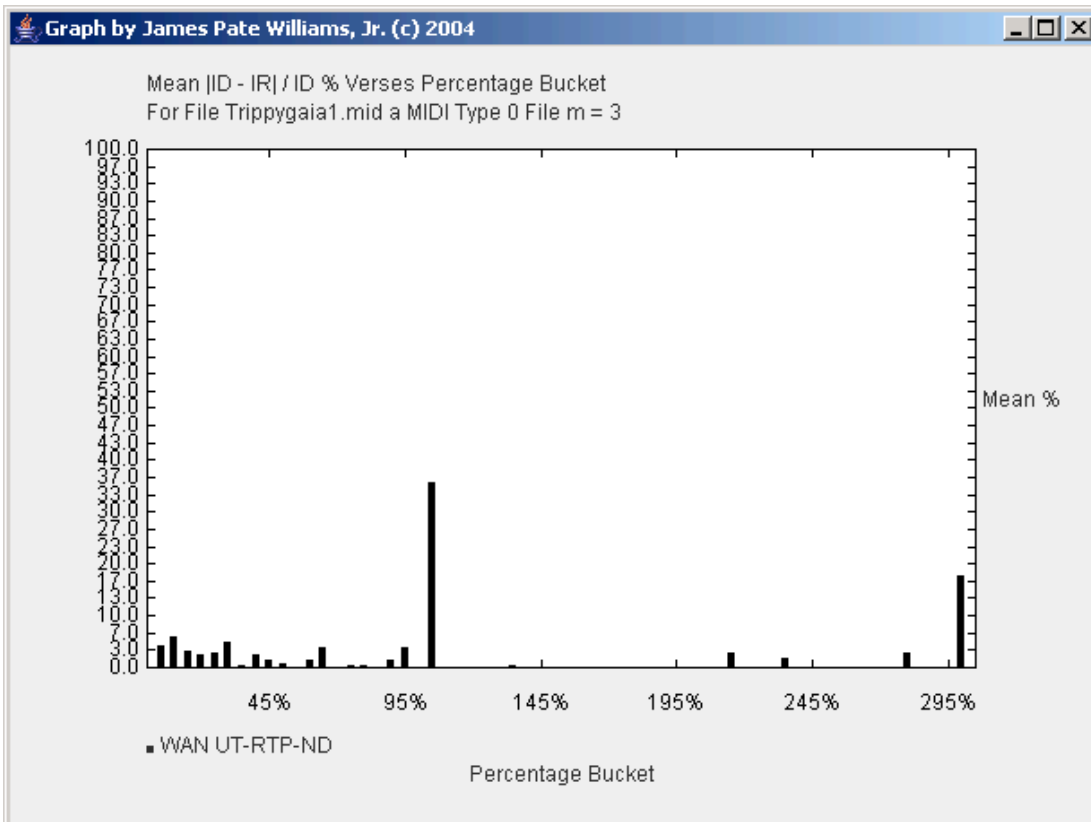


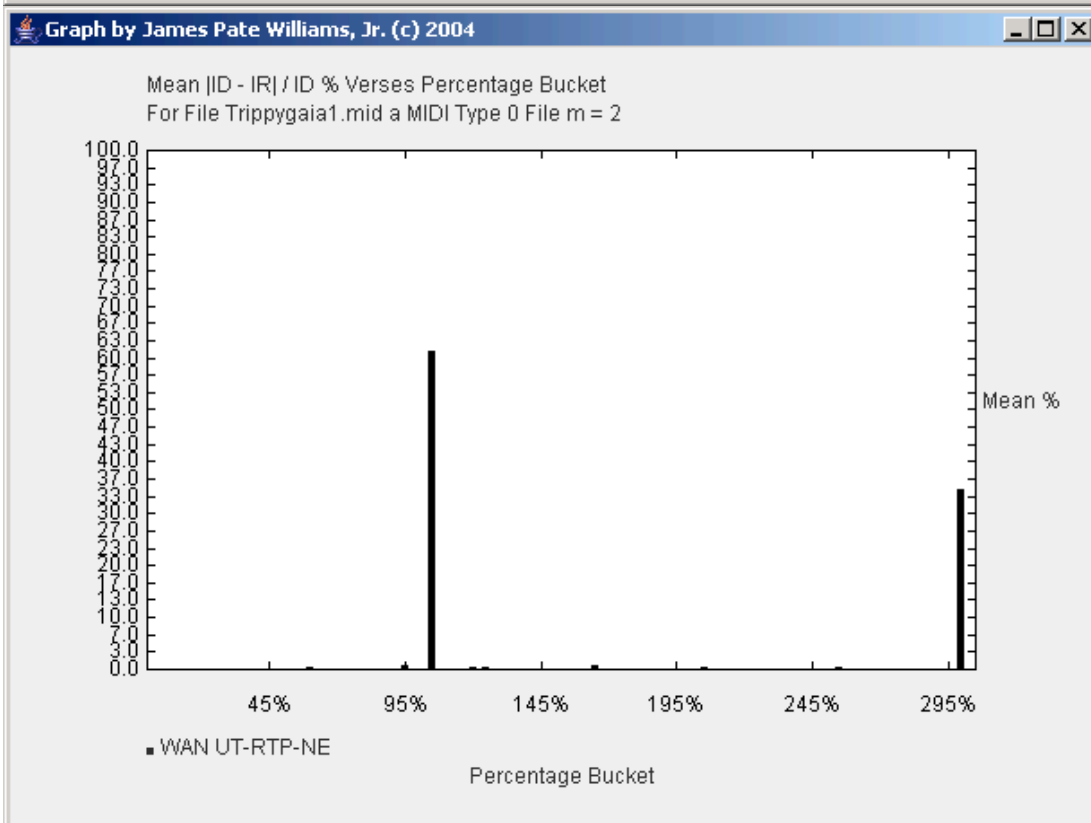
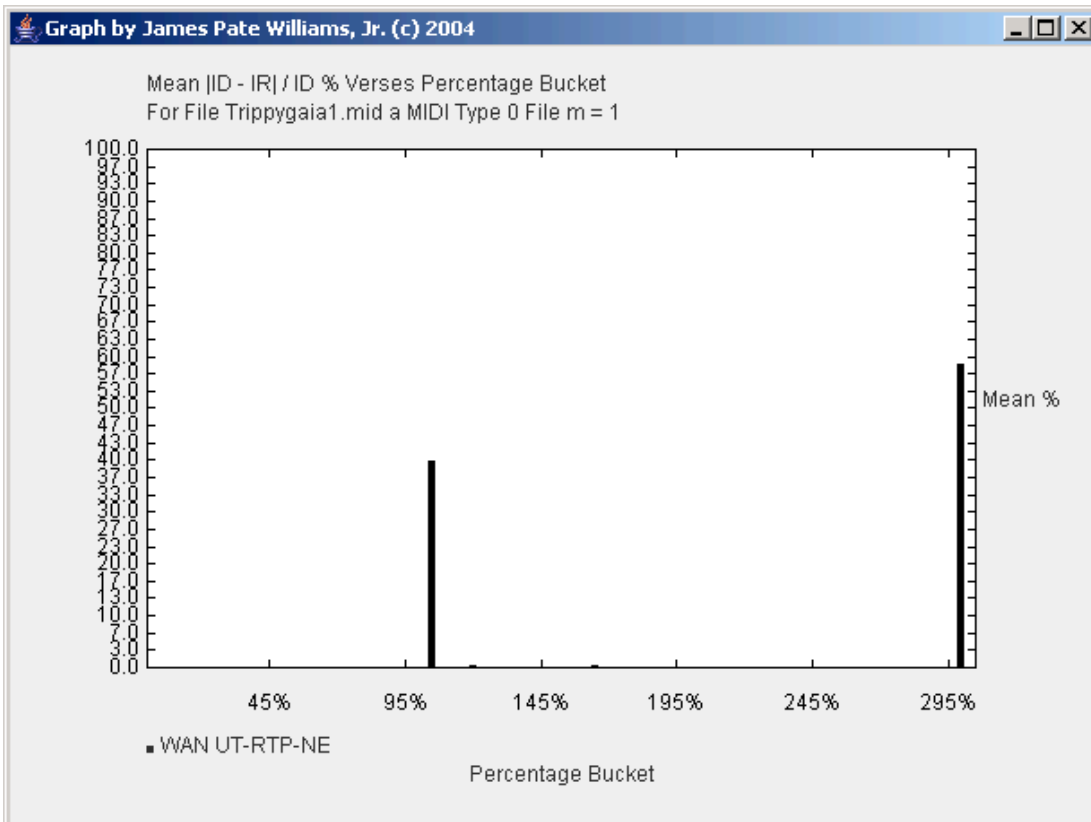


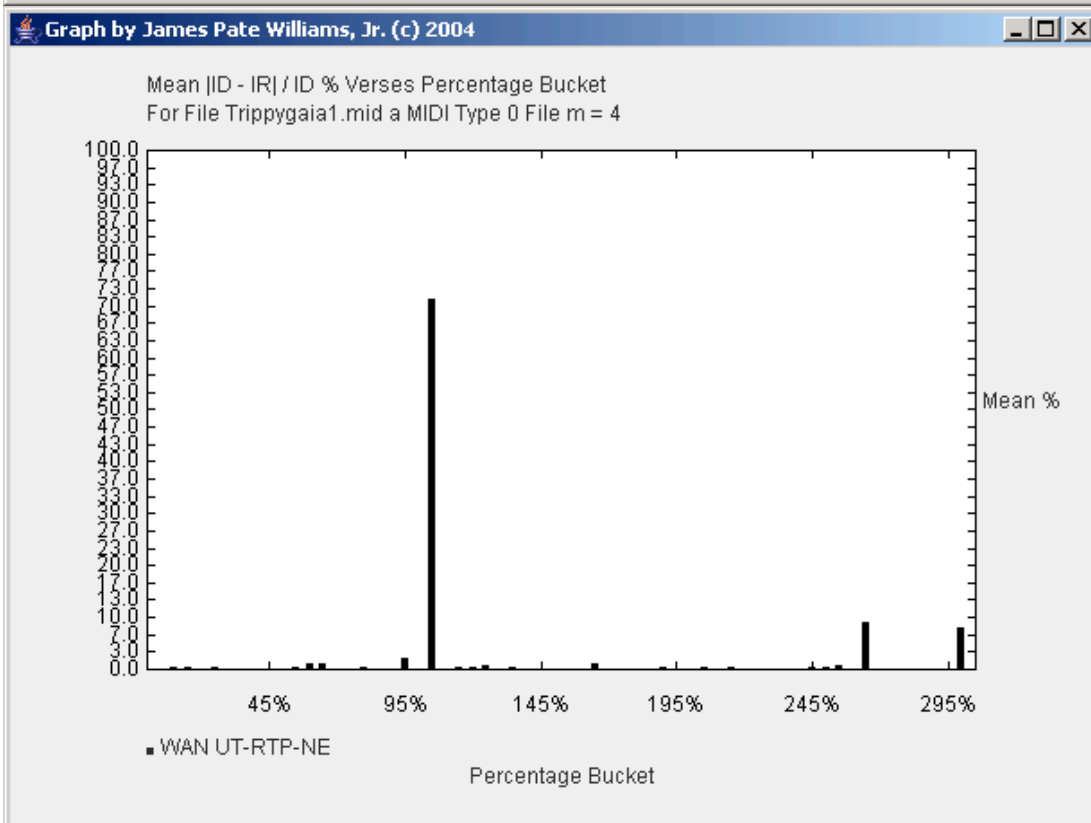
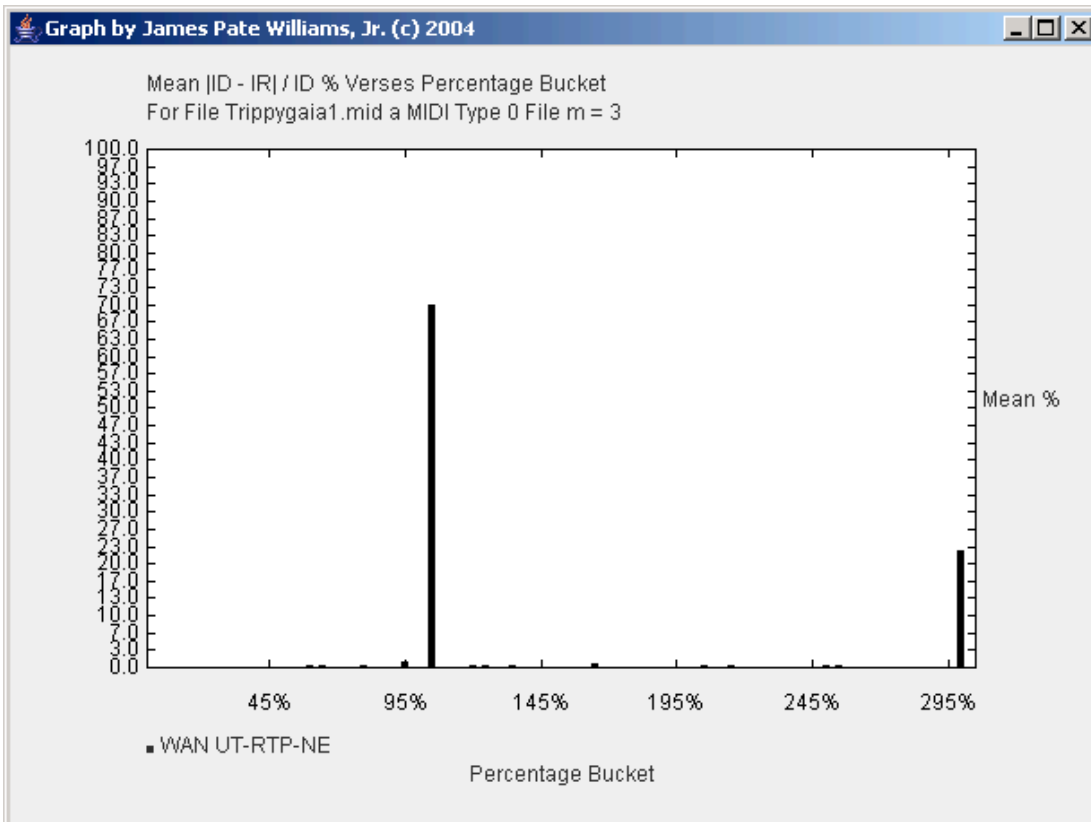


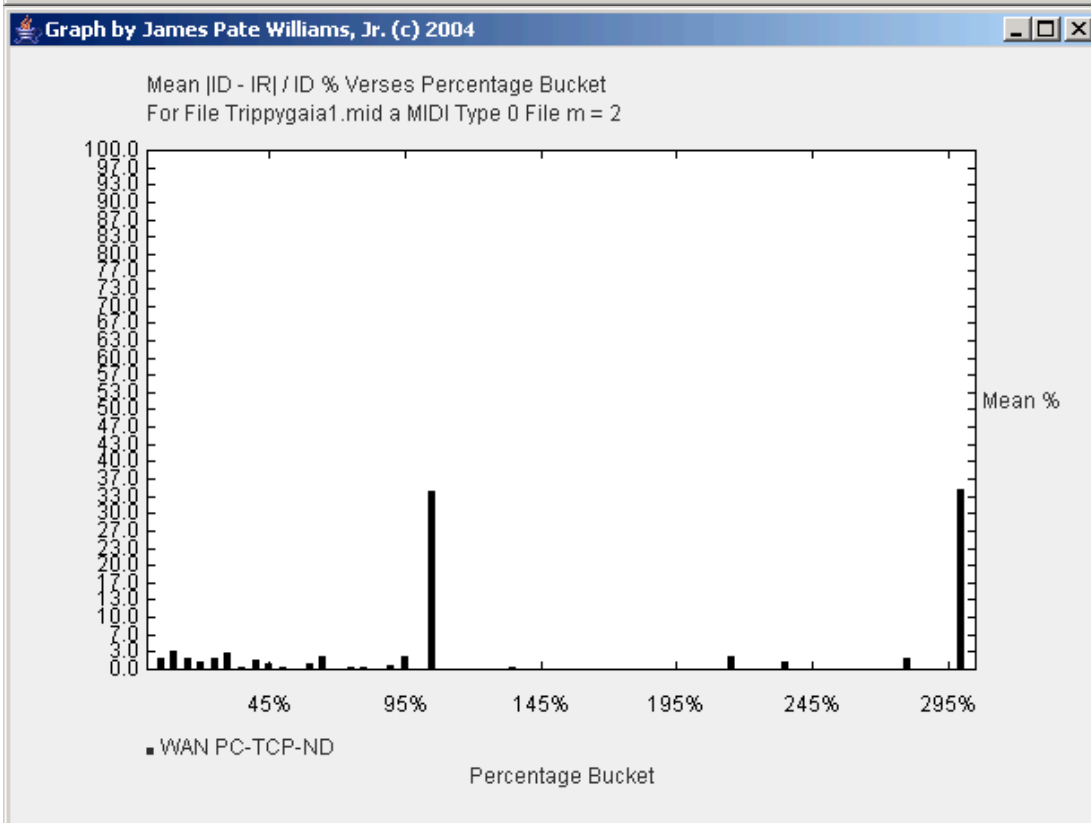
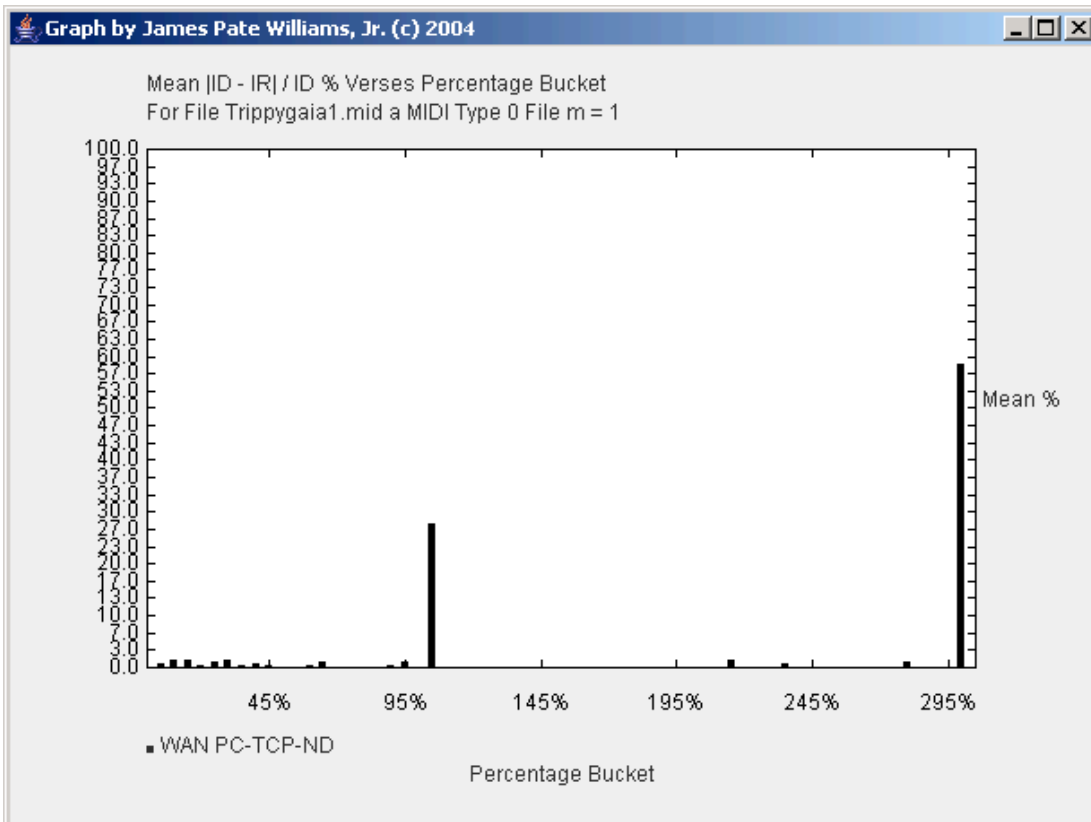


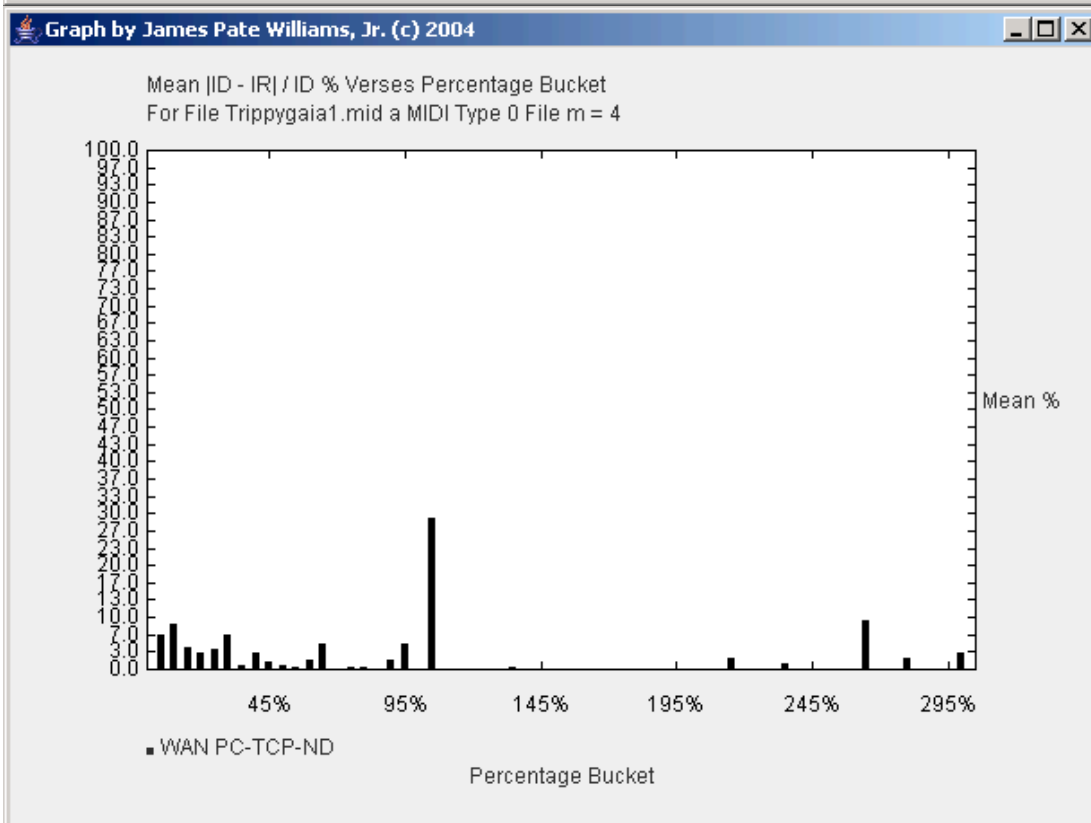
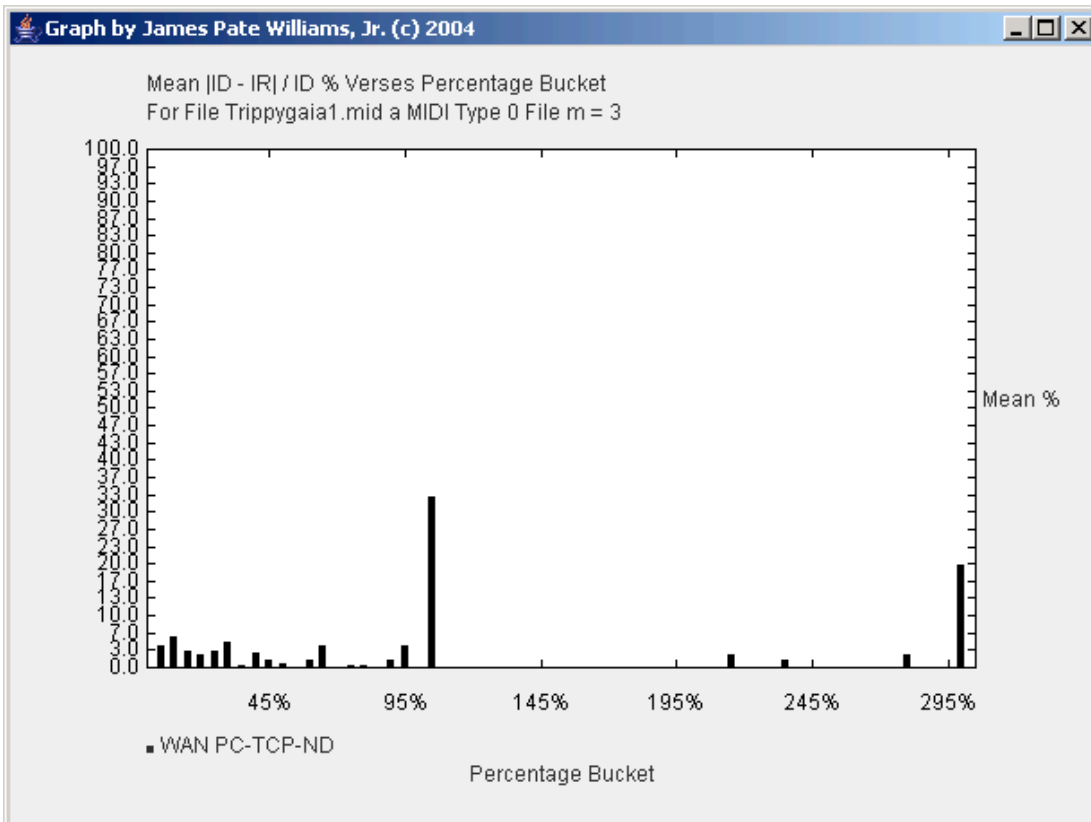


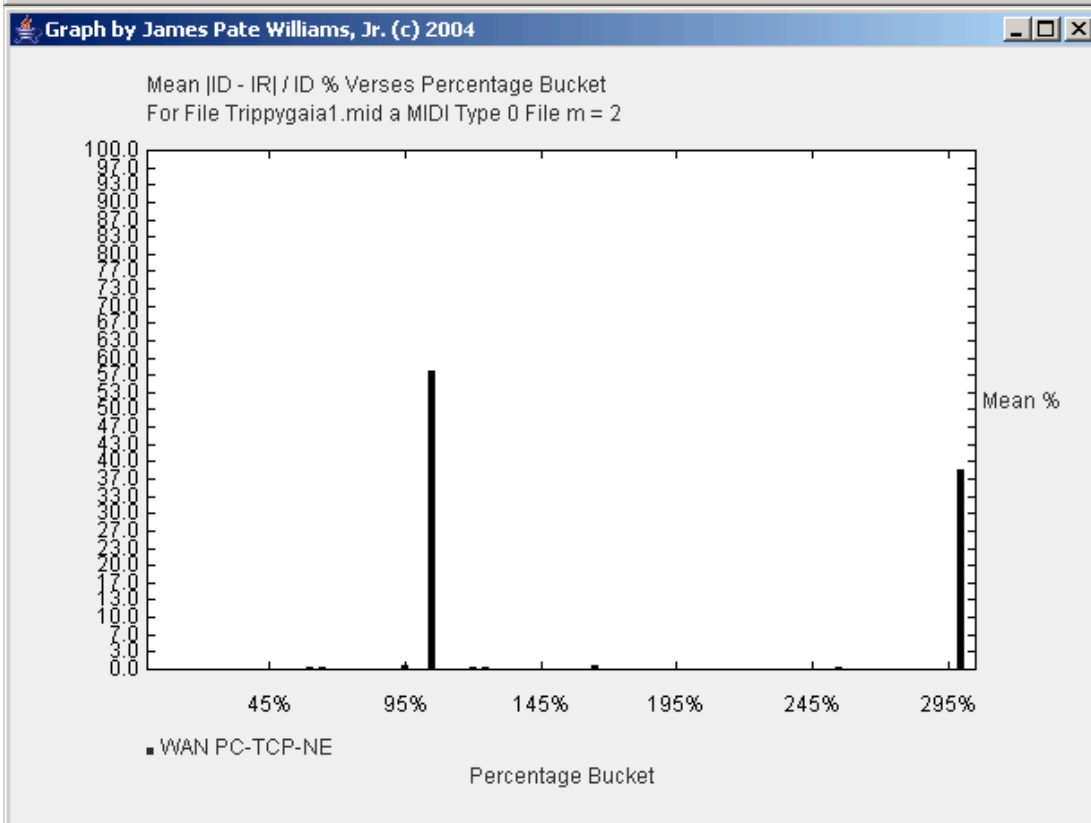
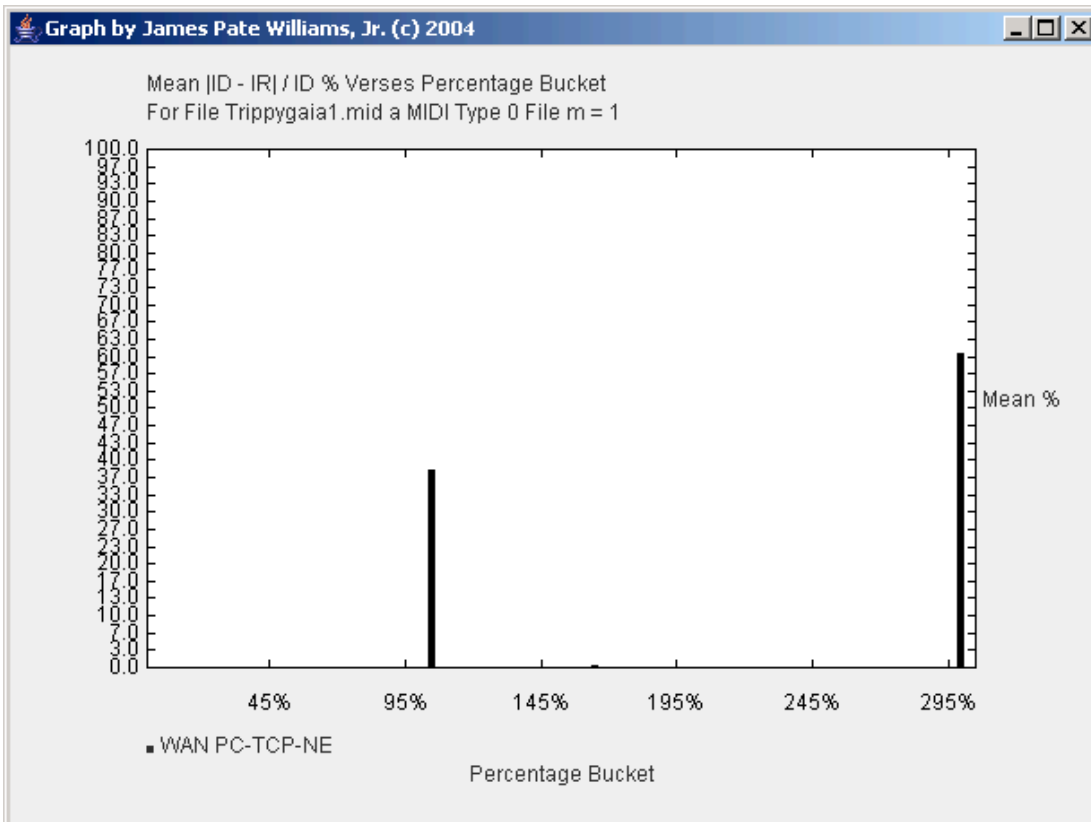




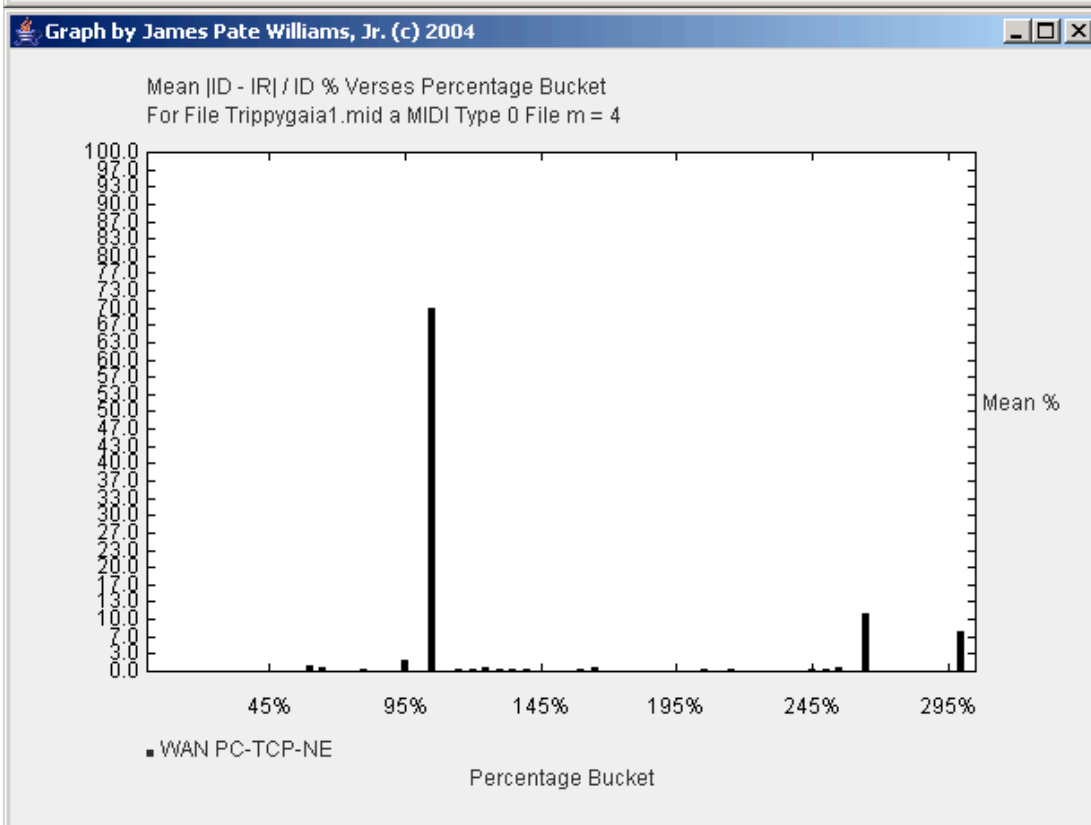
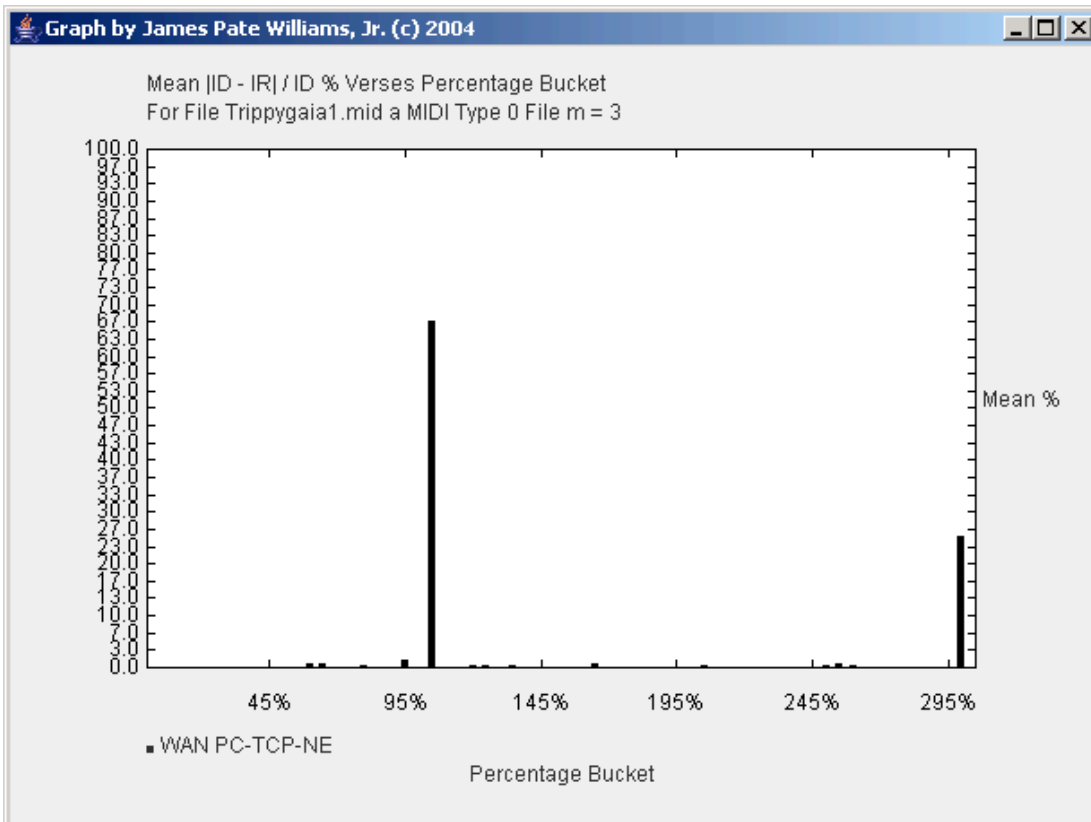


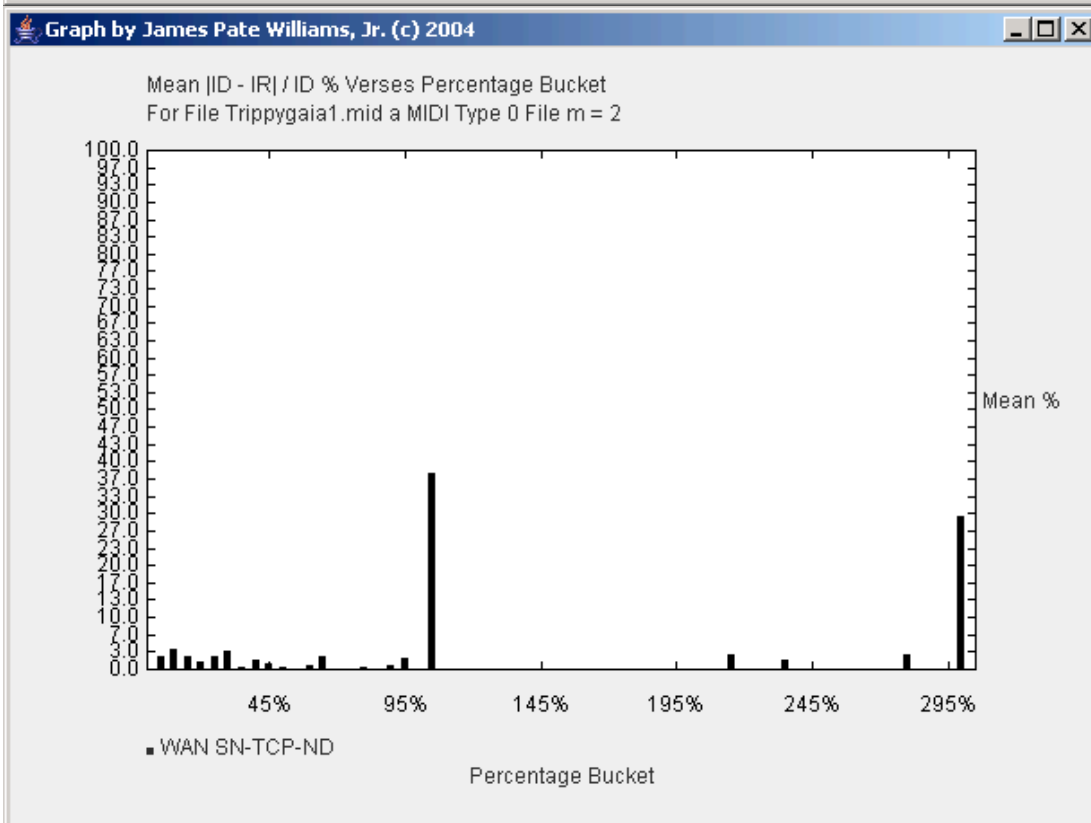
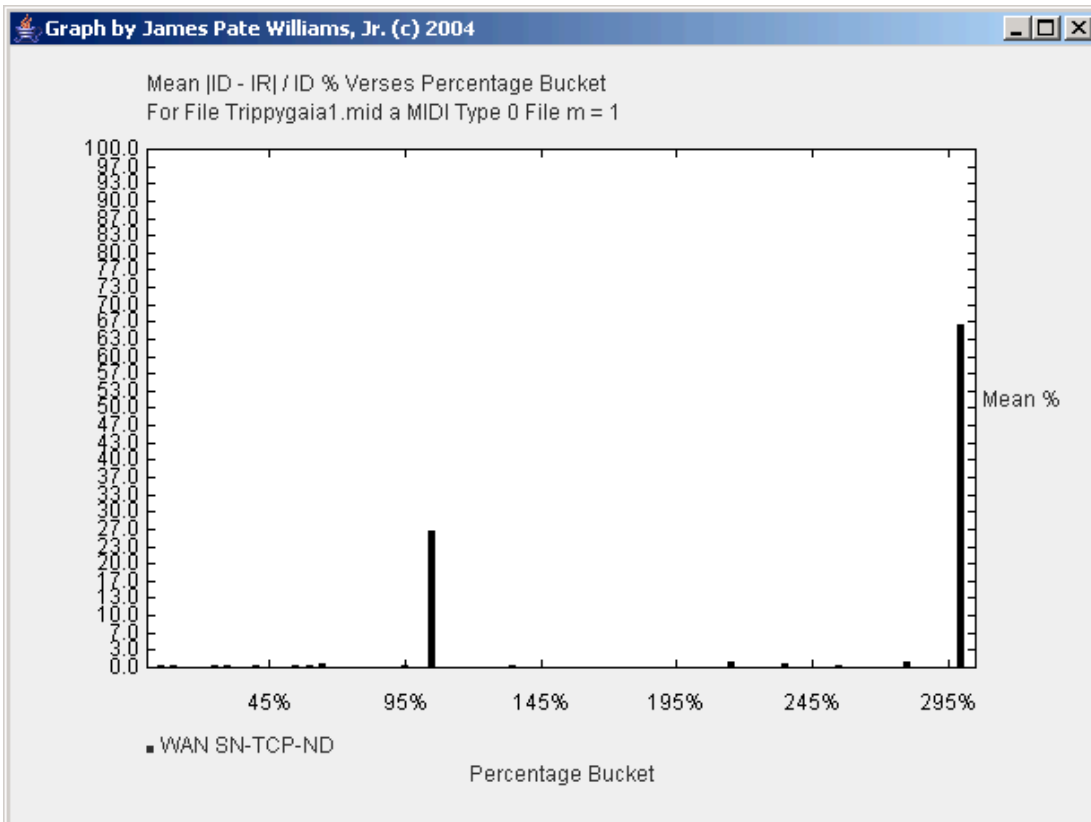


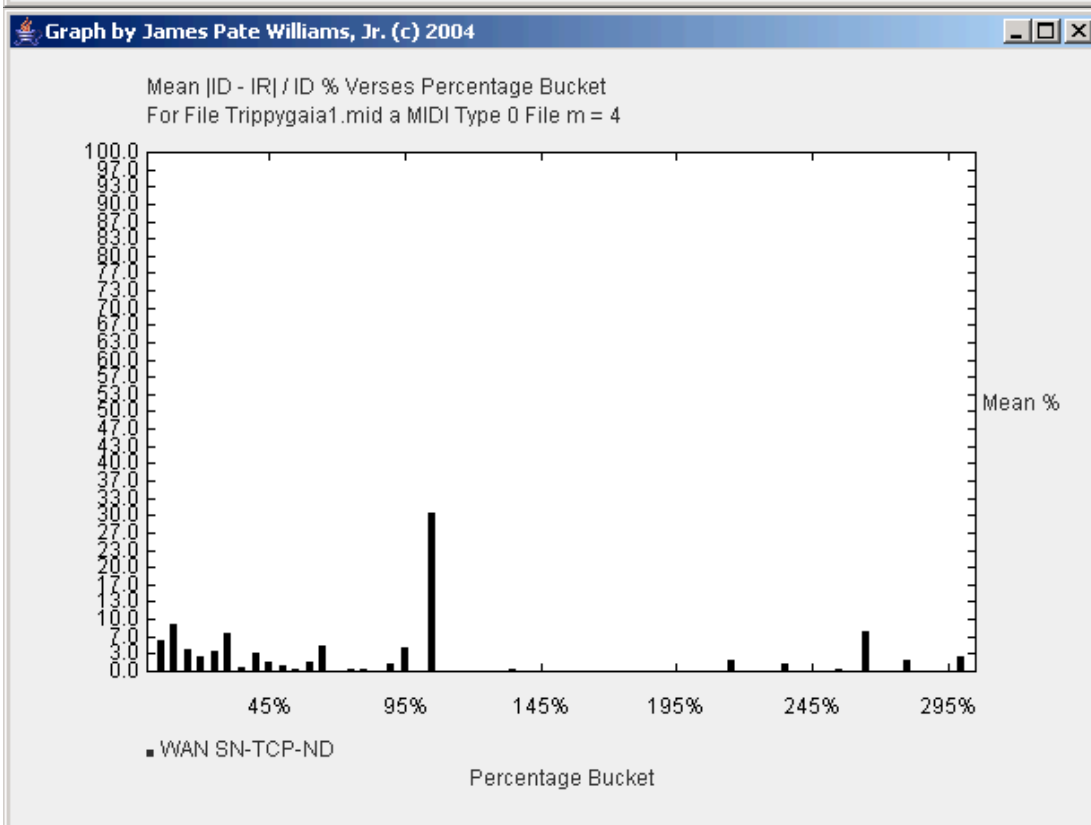
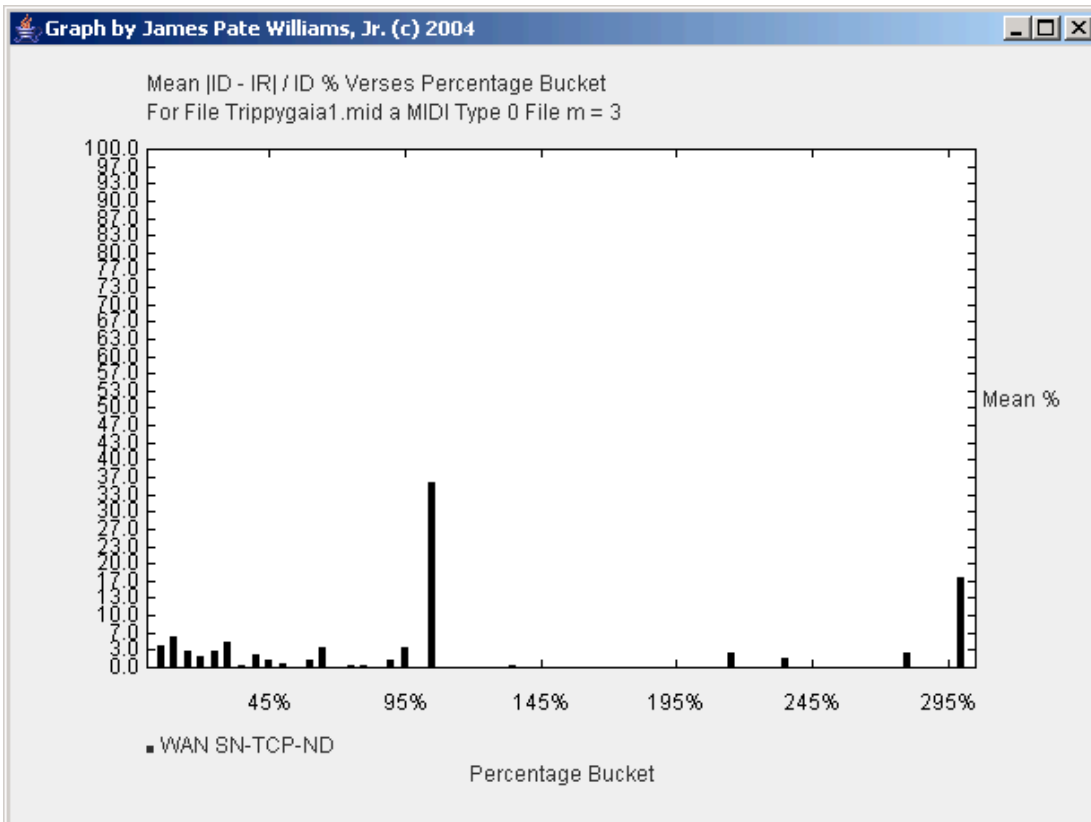


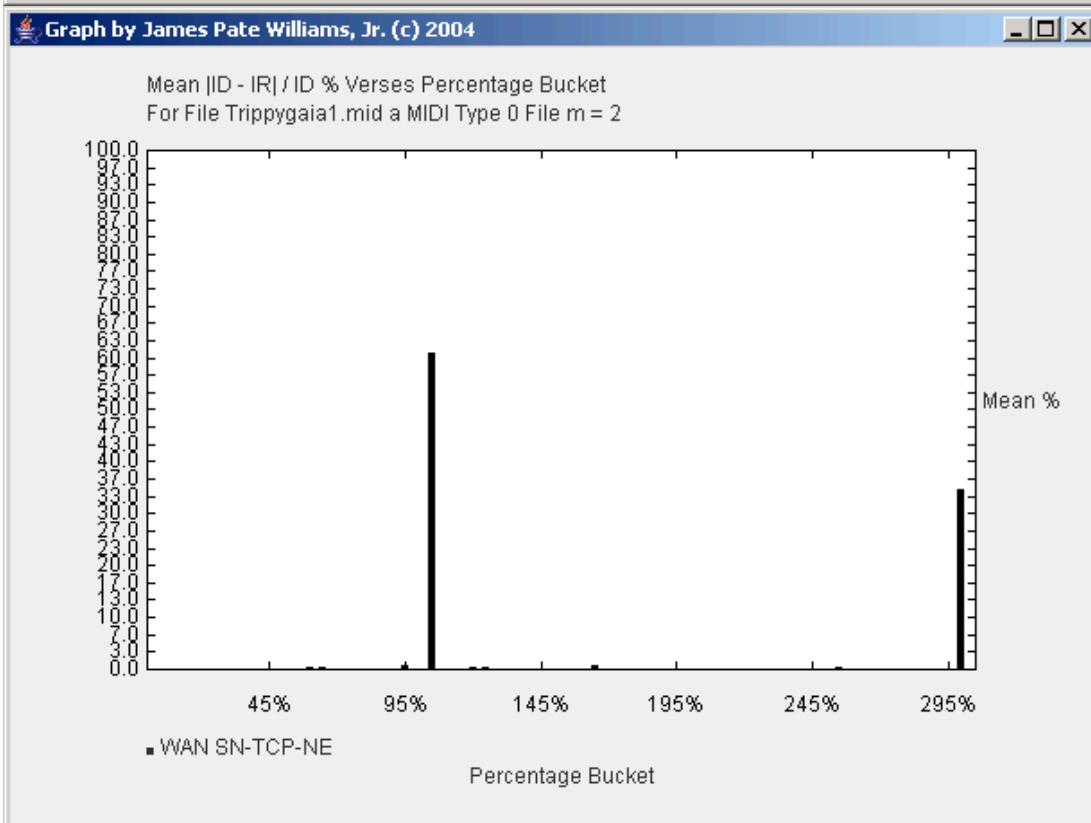
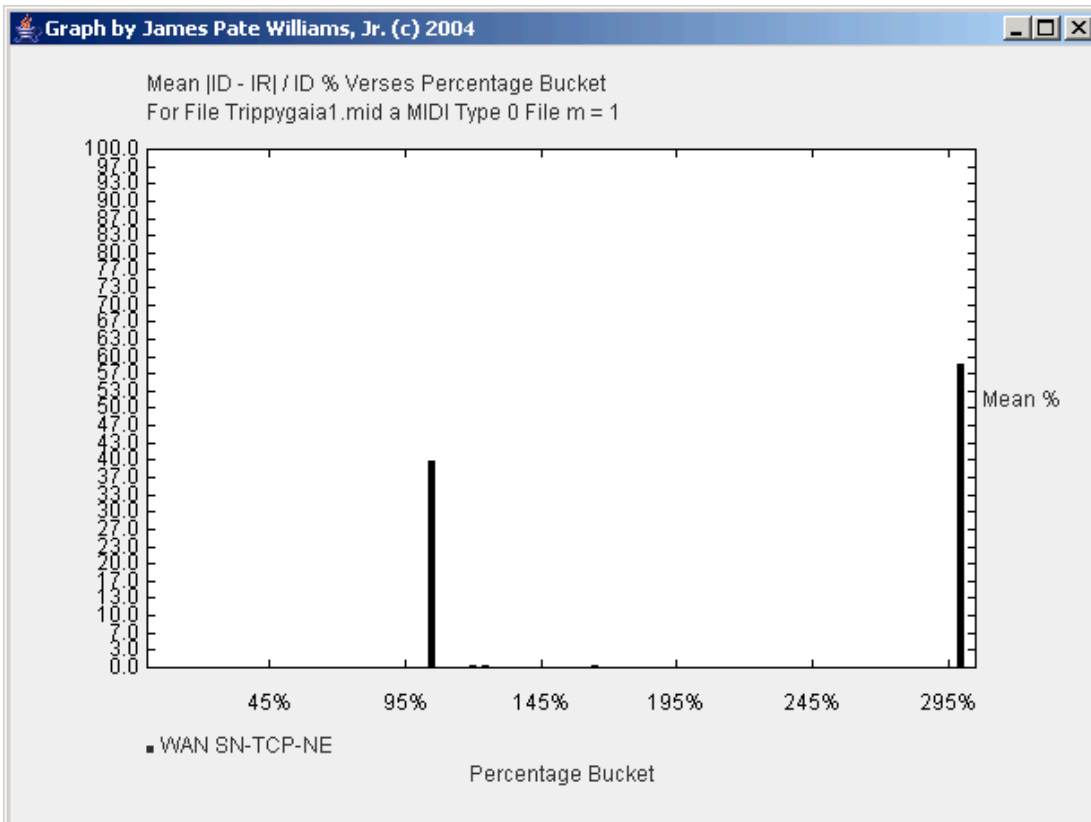


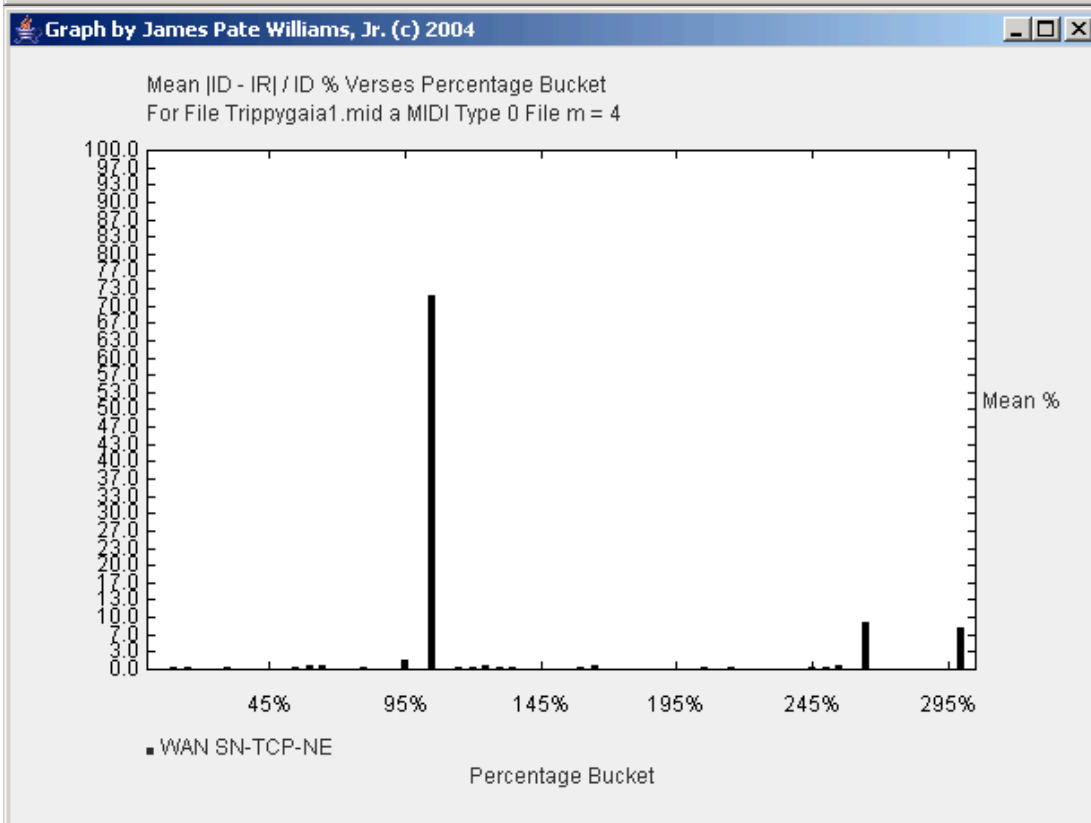
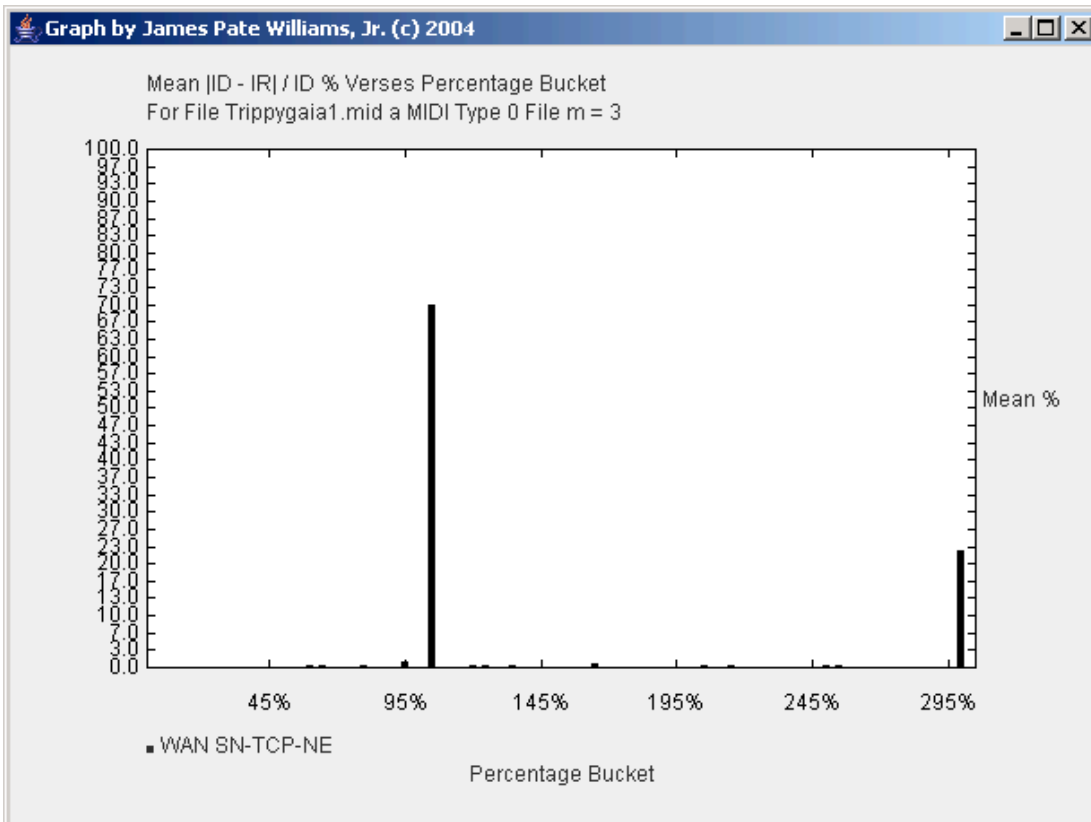






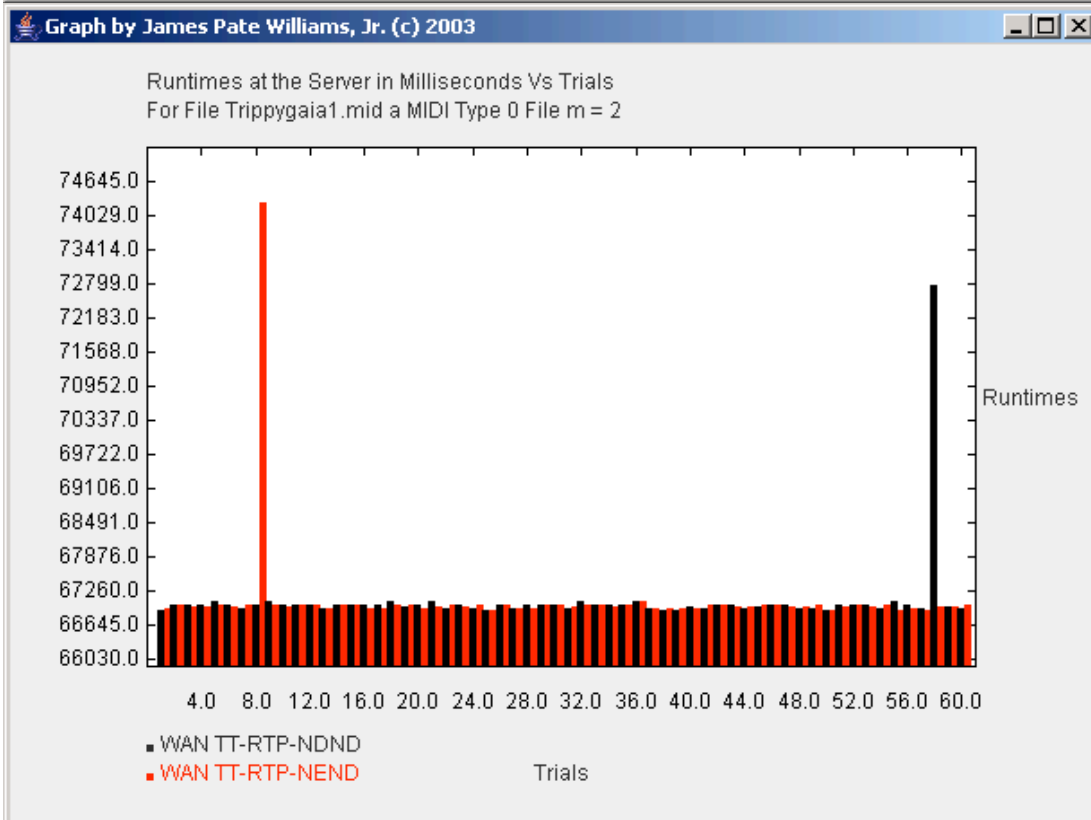
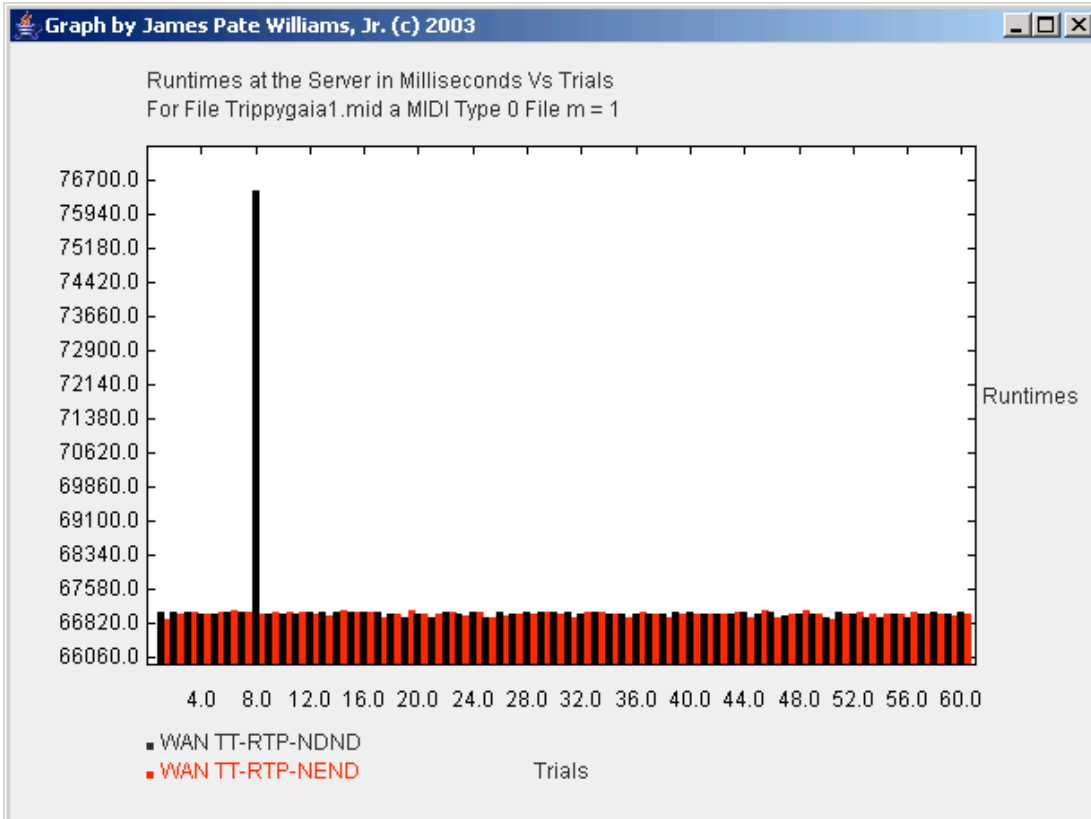


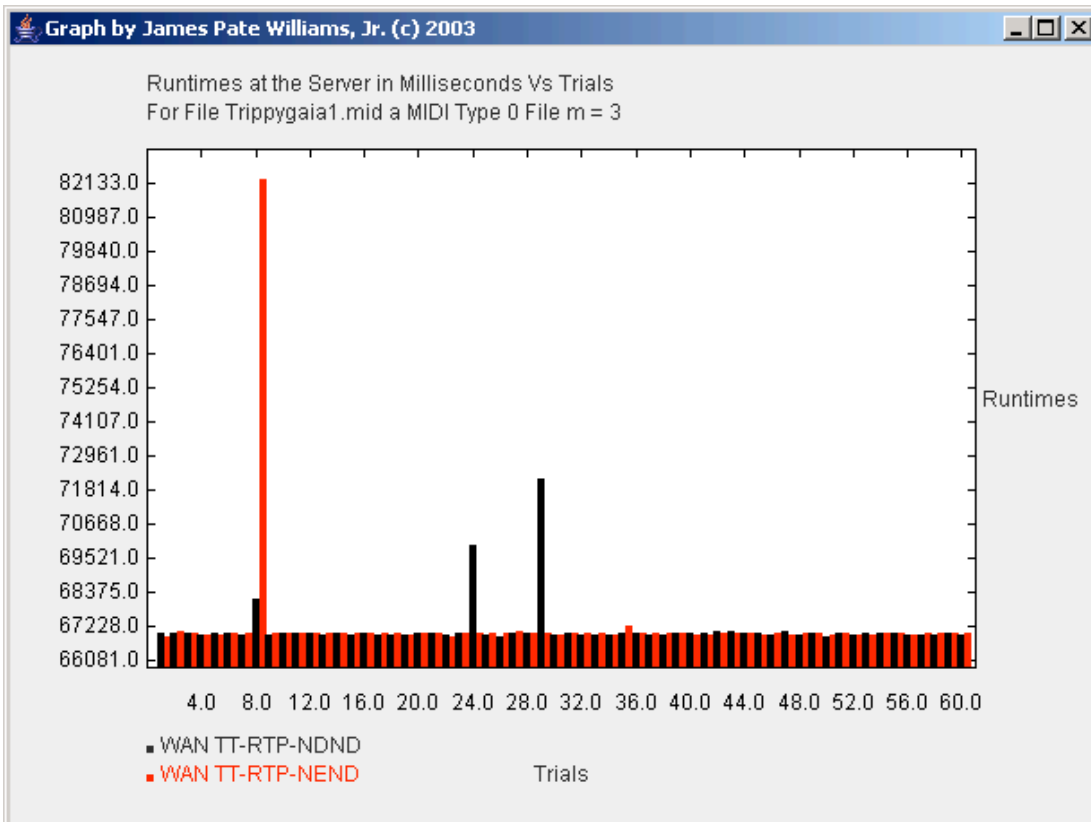




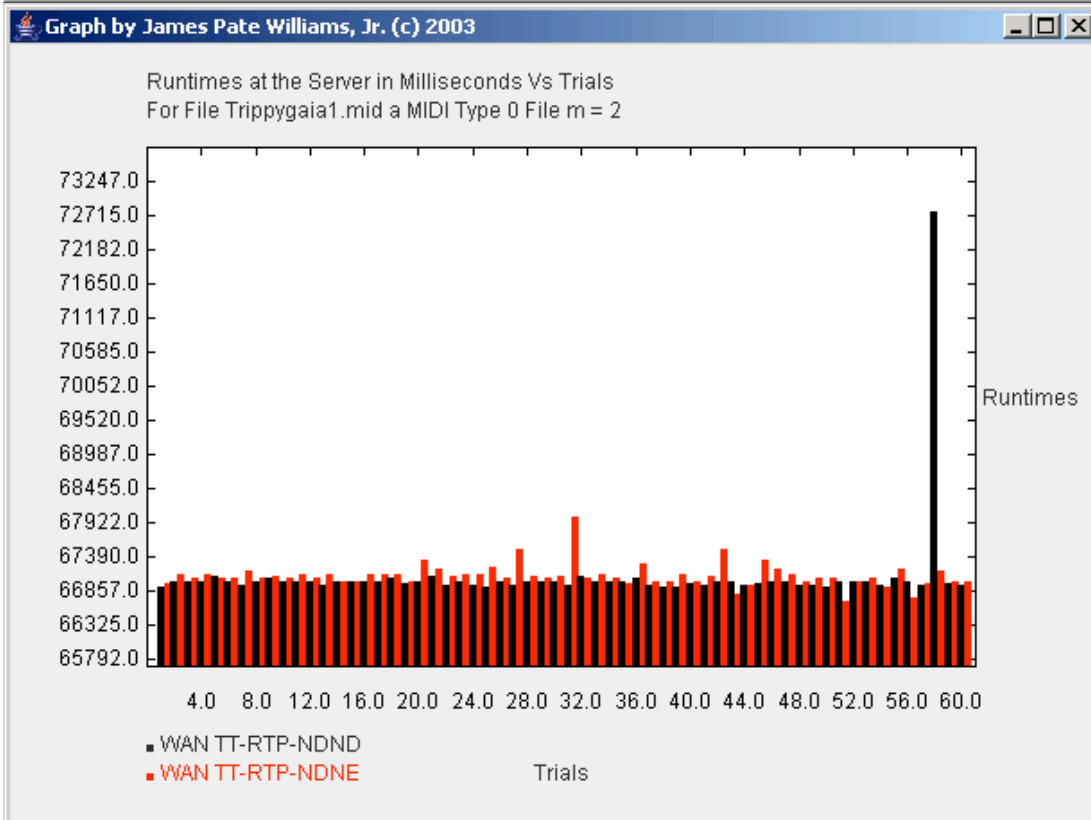
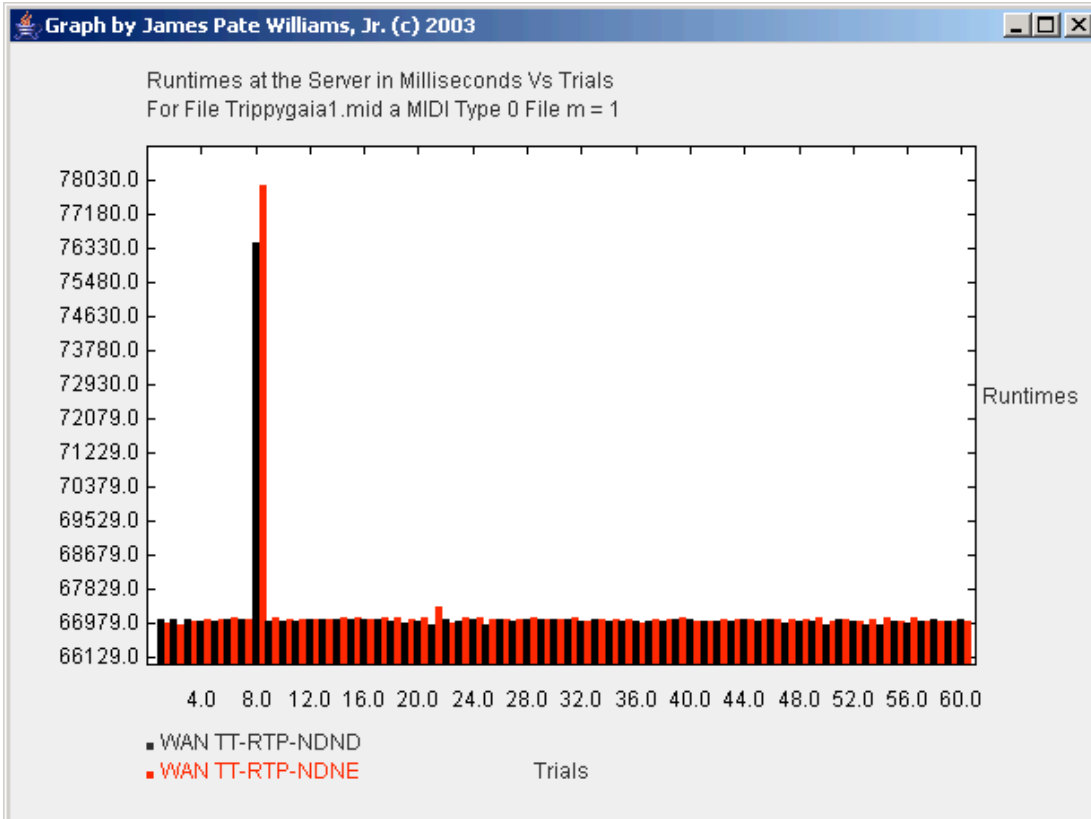
## APPENDIX D WAN PAIRED MEANS COMPARISON GRAPHS

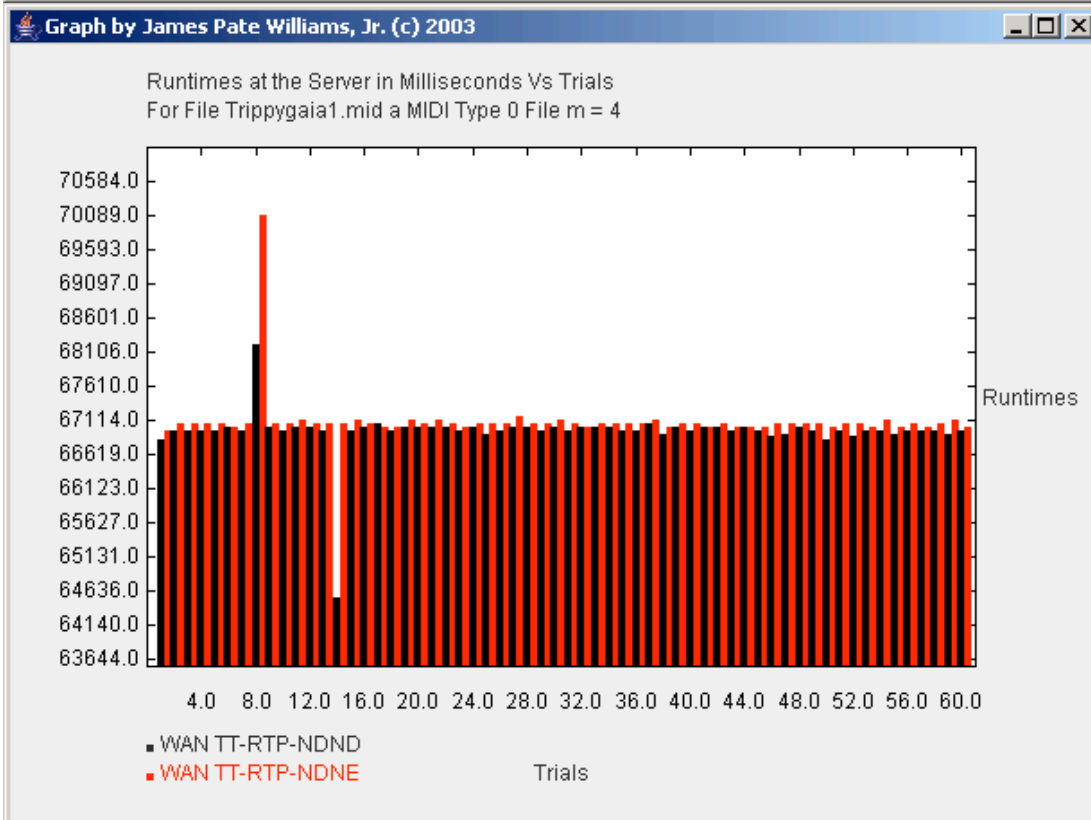
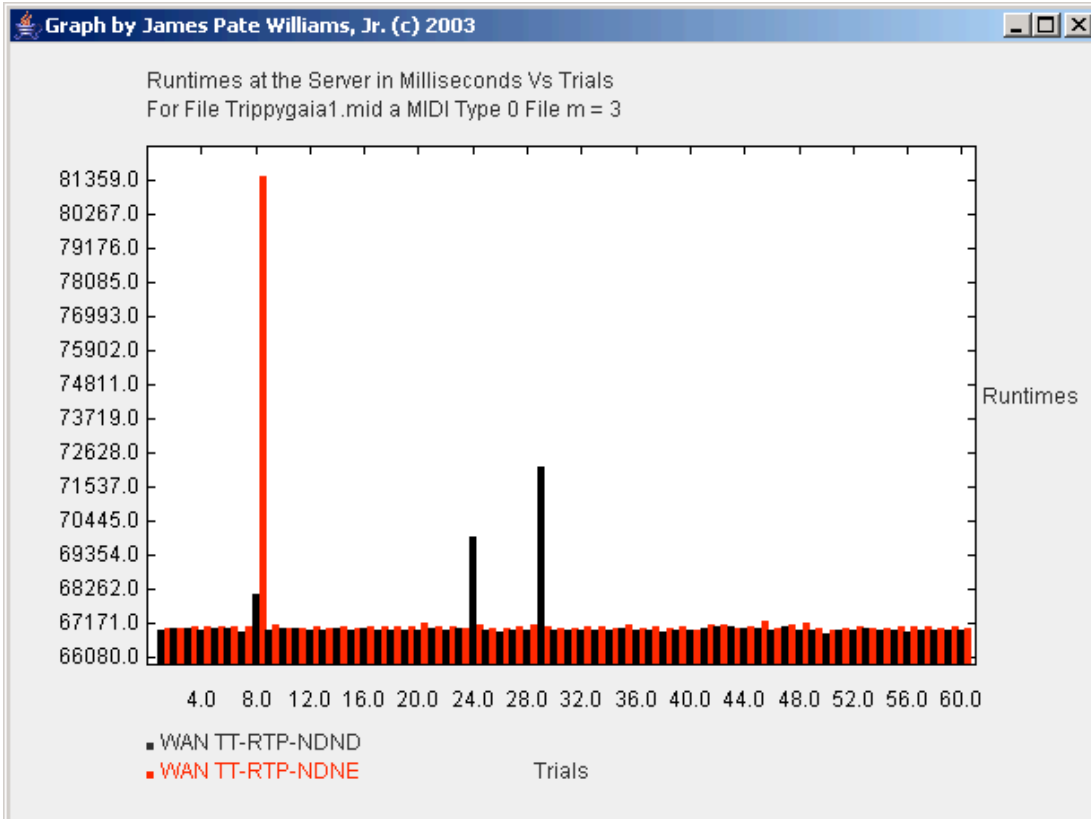
This appendix consists of 180 graphs of the mean runtime of a MIDI sequence on the destination host on a WAN. The number of graphs can be determined by calculating the total number of possible pairings of the protocols as  $(10 * 9) / 2 = 45$  and multiplying 45 by 4 to get 180, where 4 is the number of values of  $m$ , the number of MIDI short messages per TCP packet.

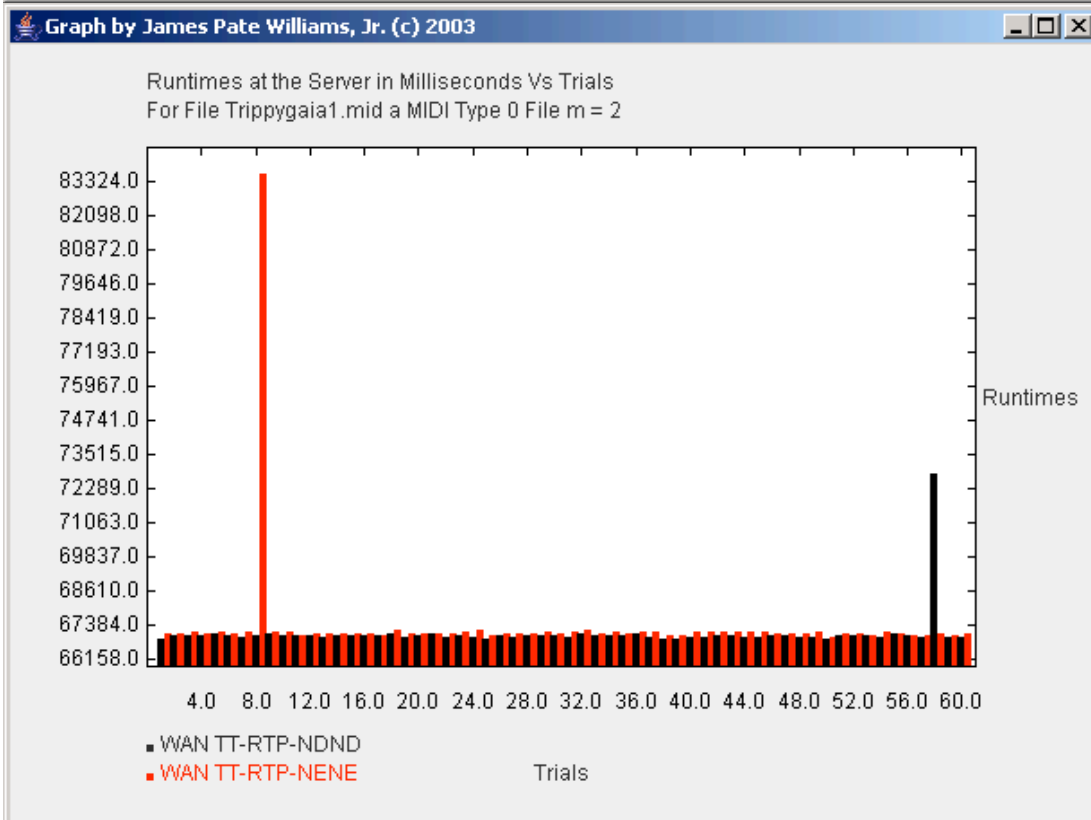
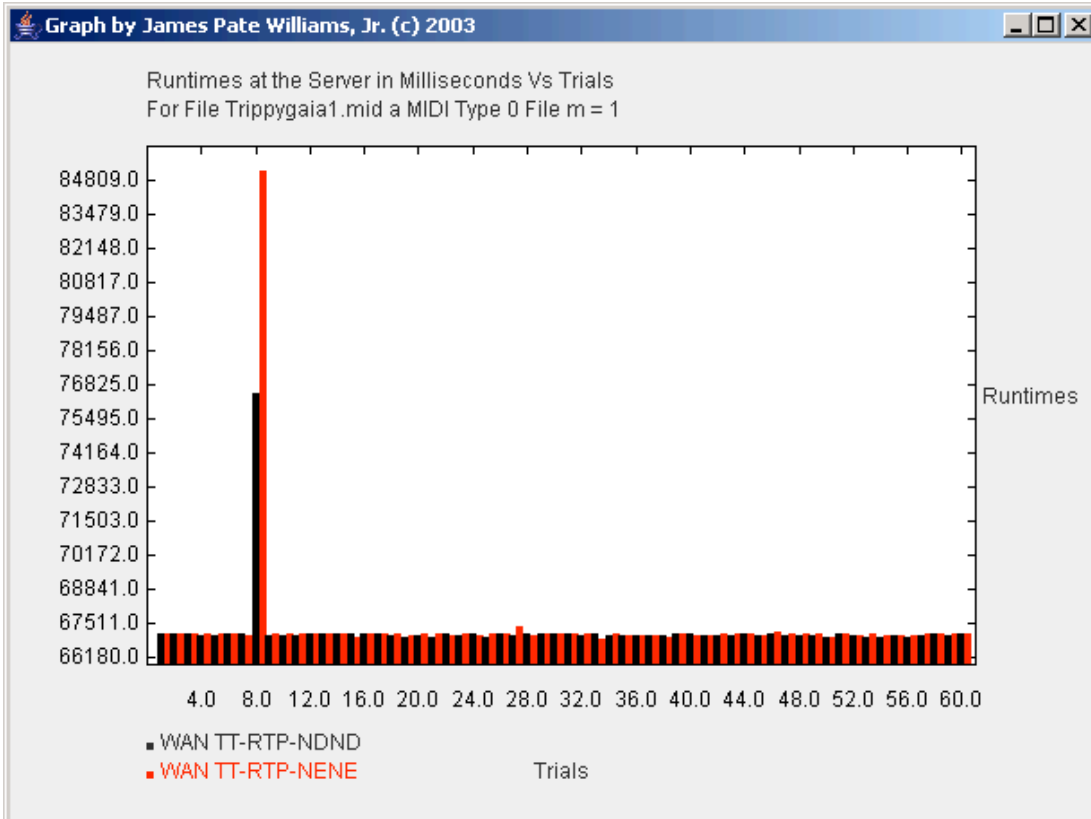


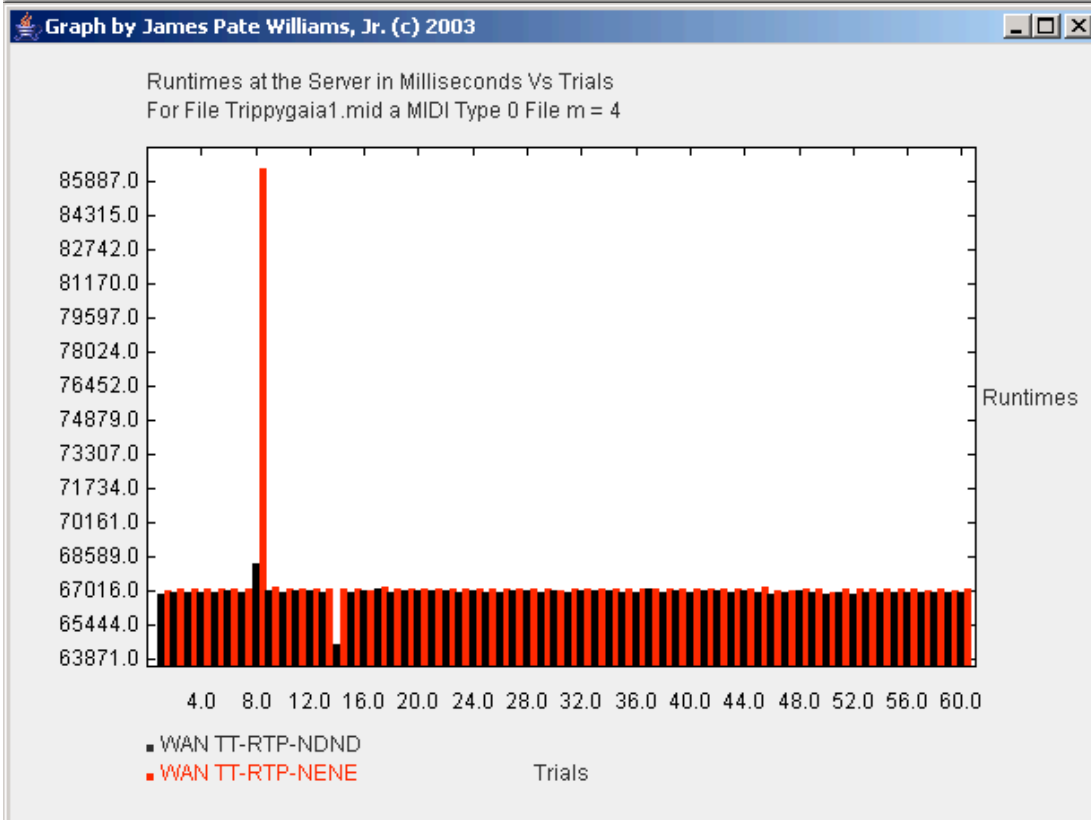
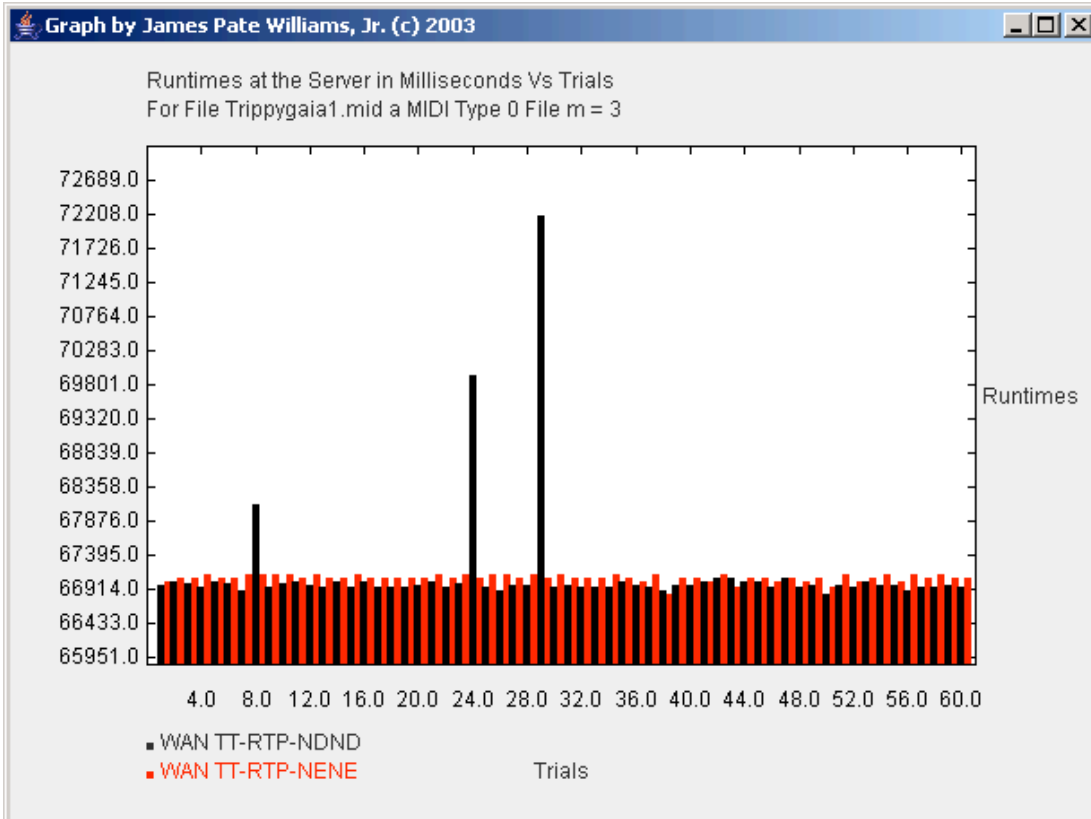


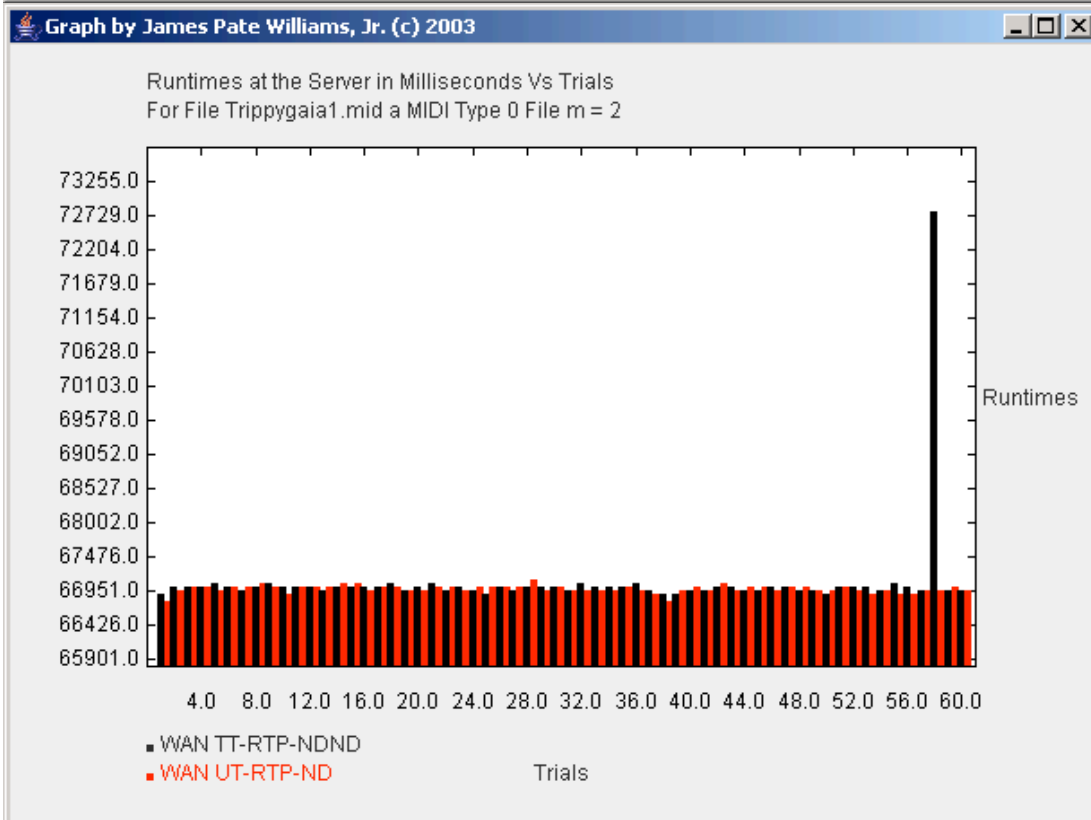
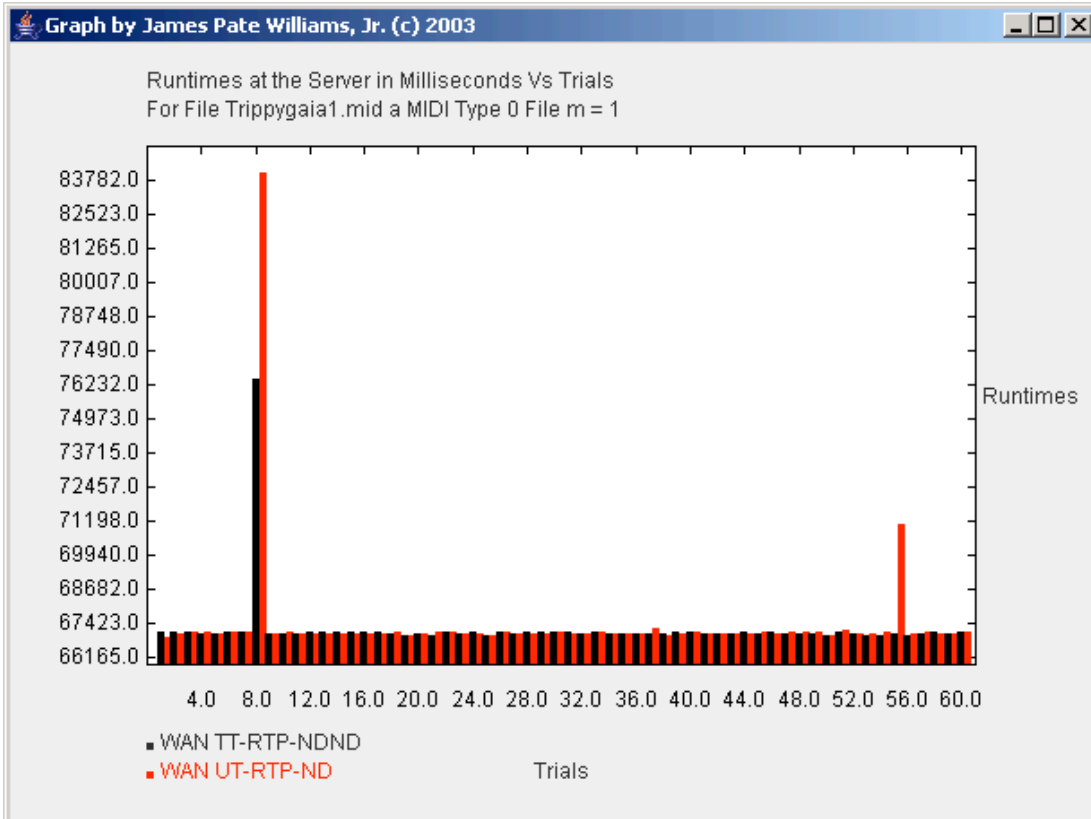


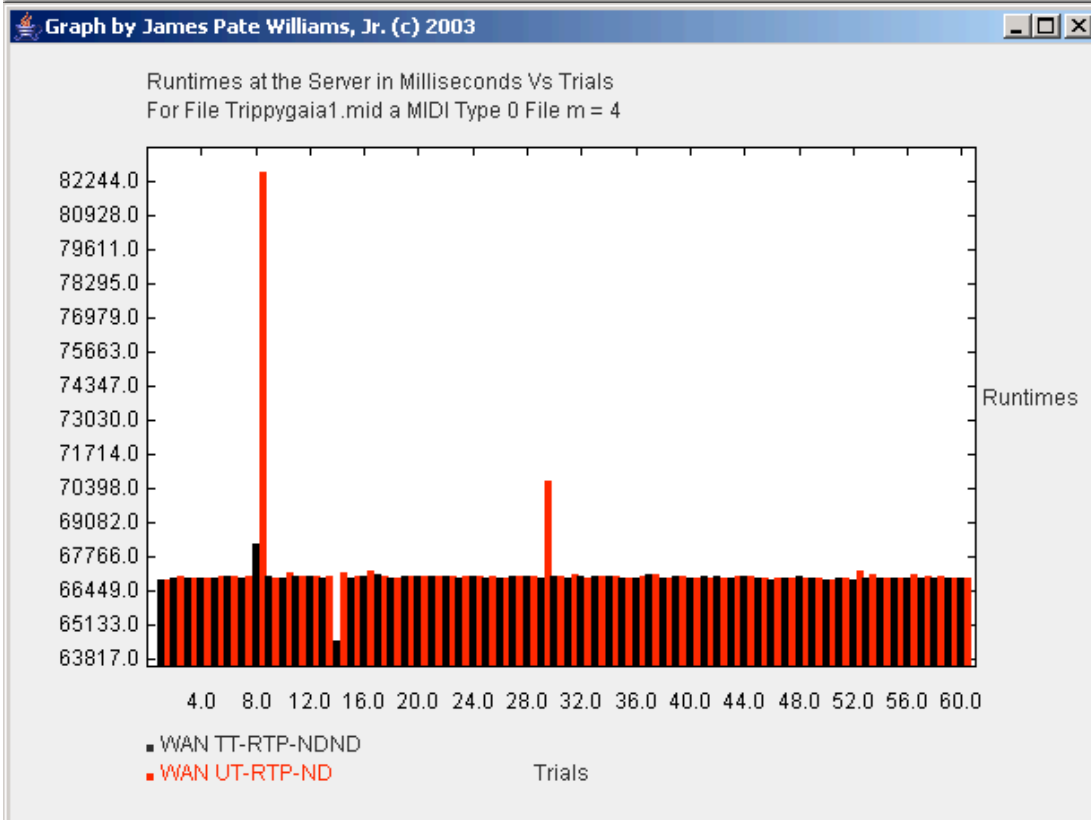
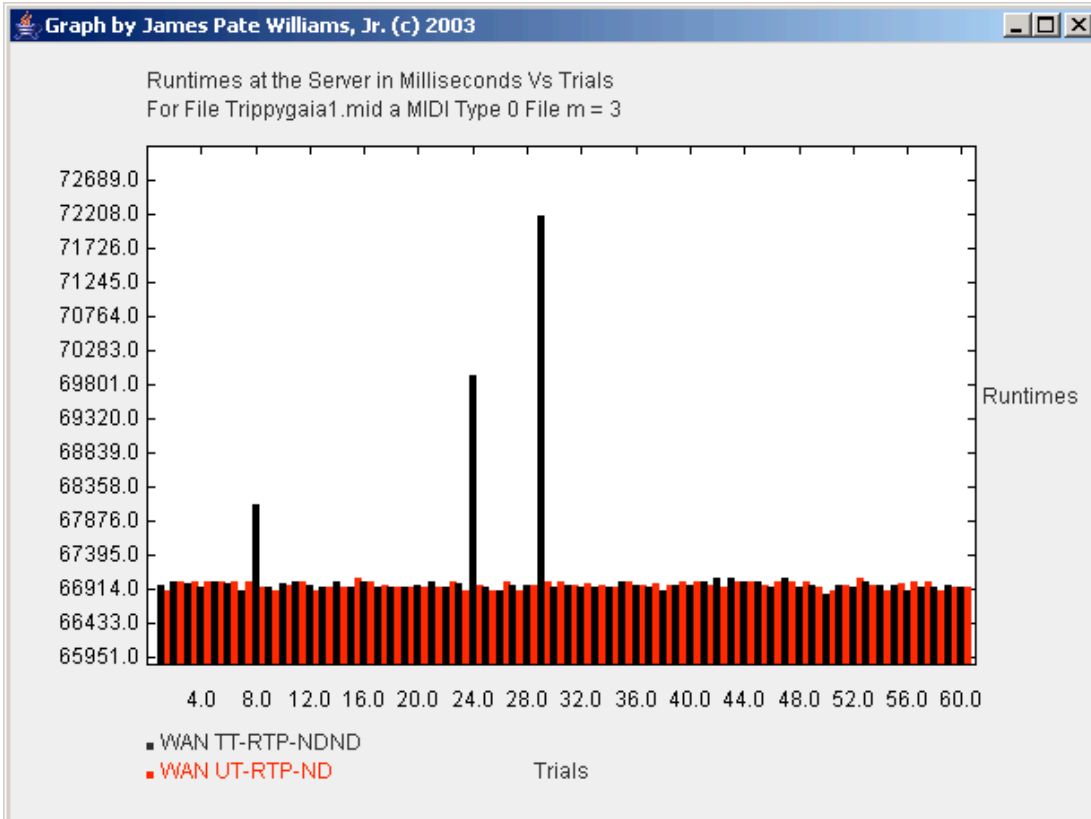


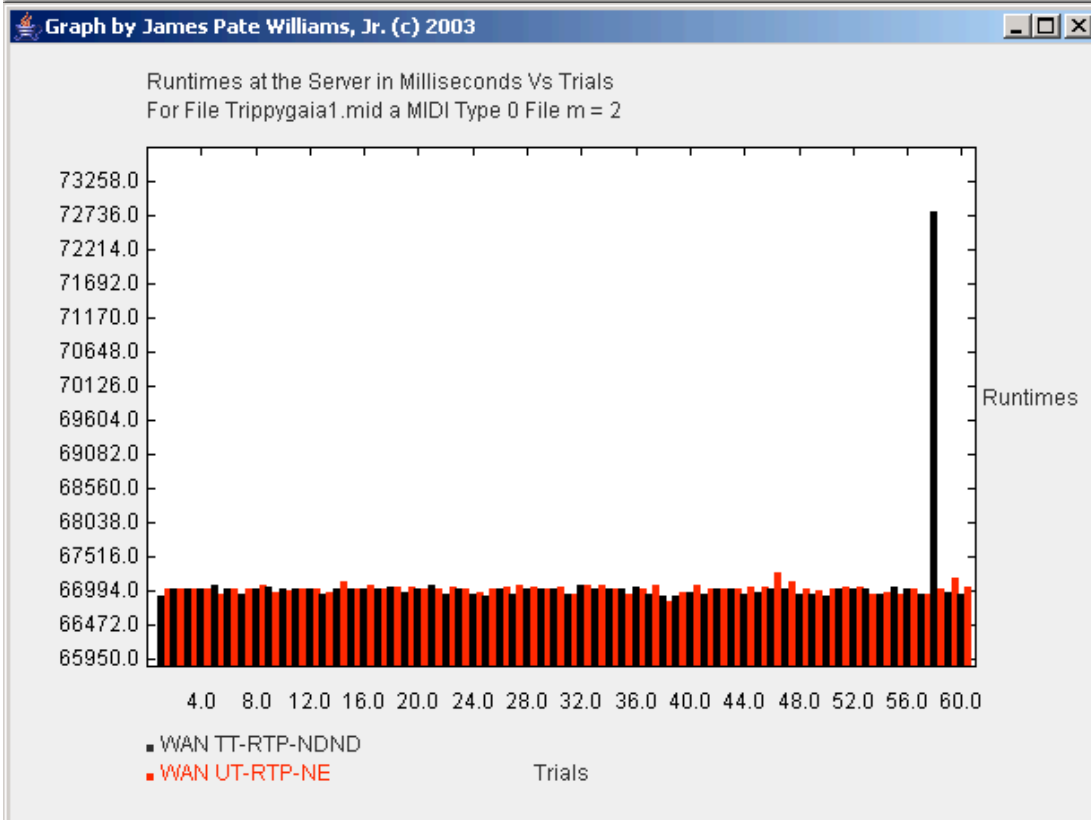
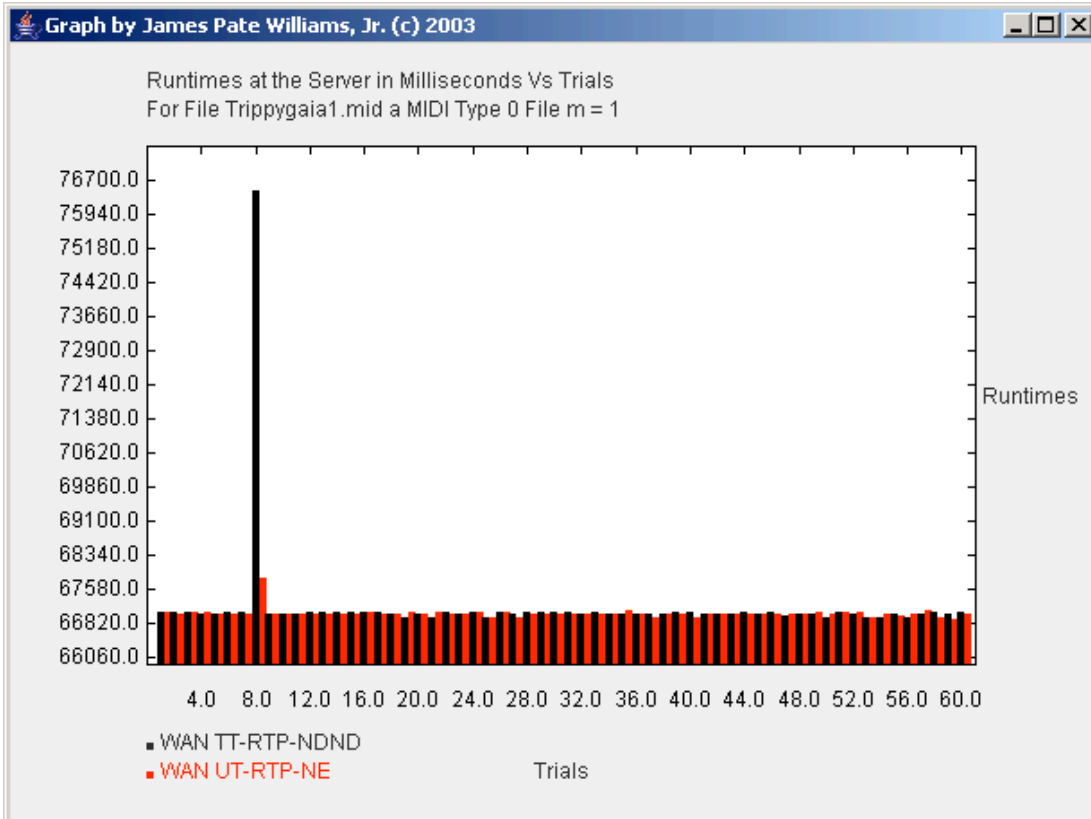


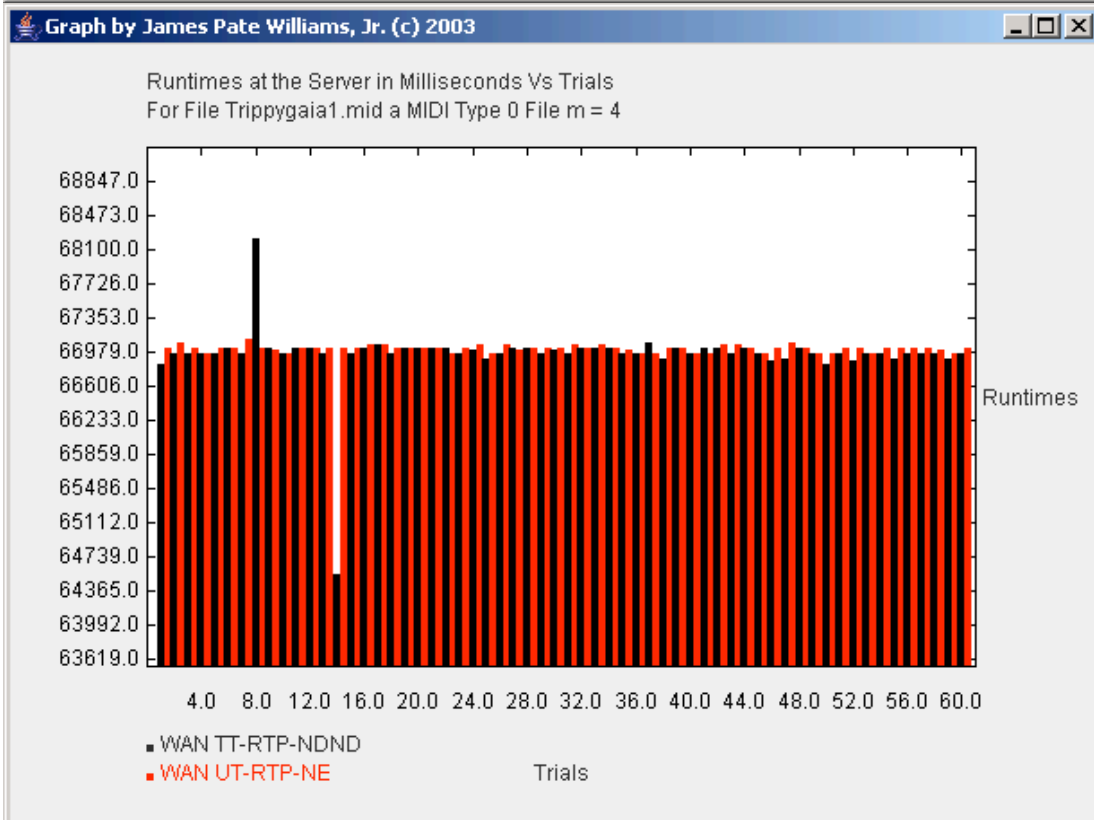
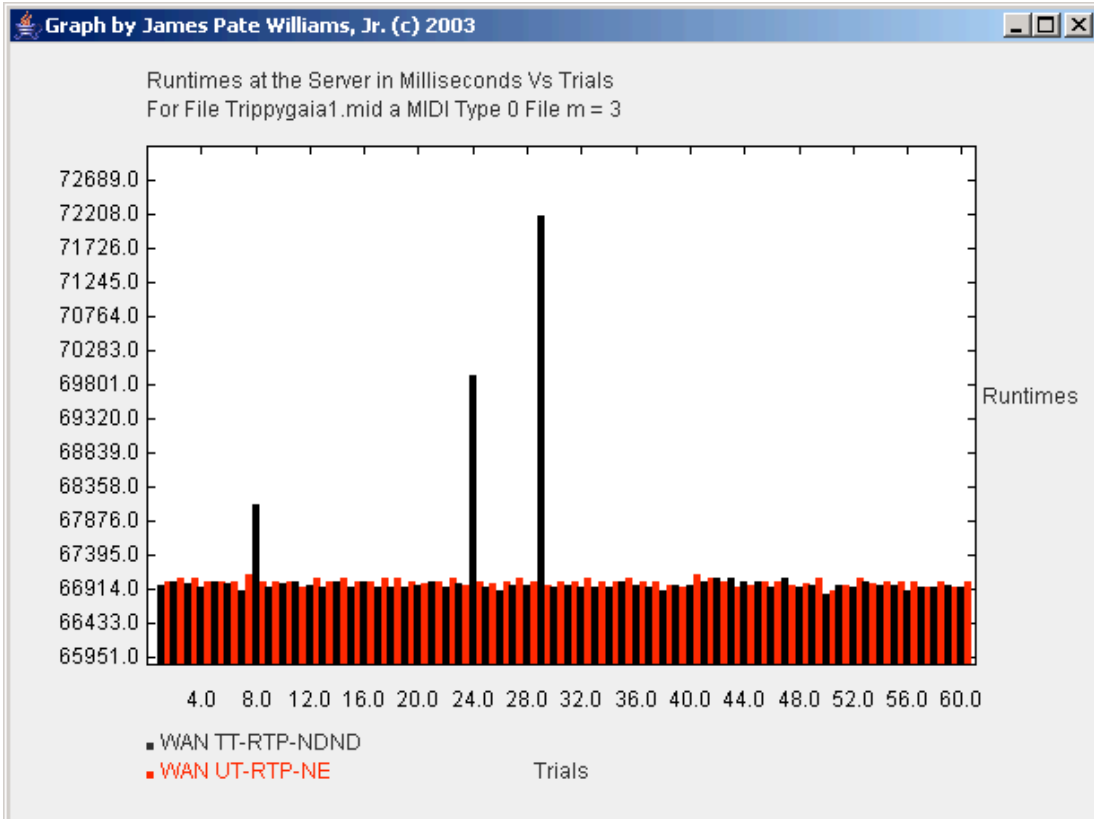




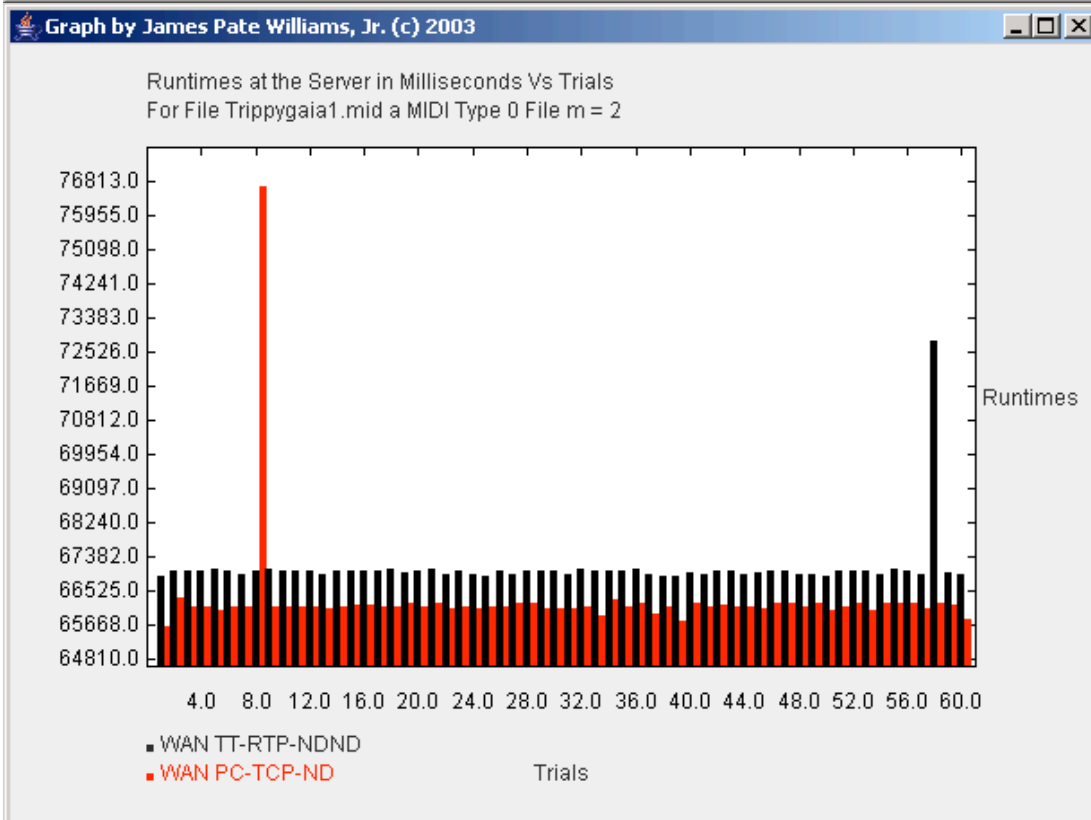
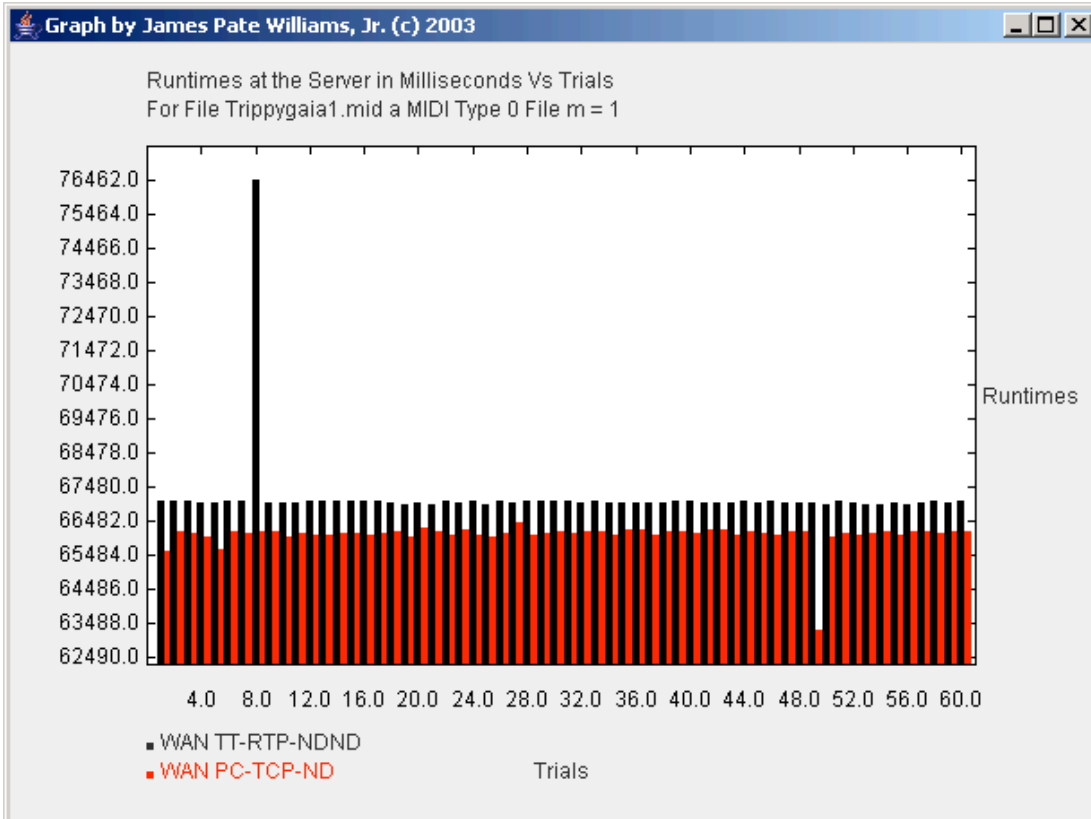


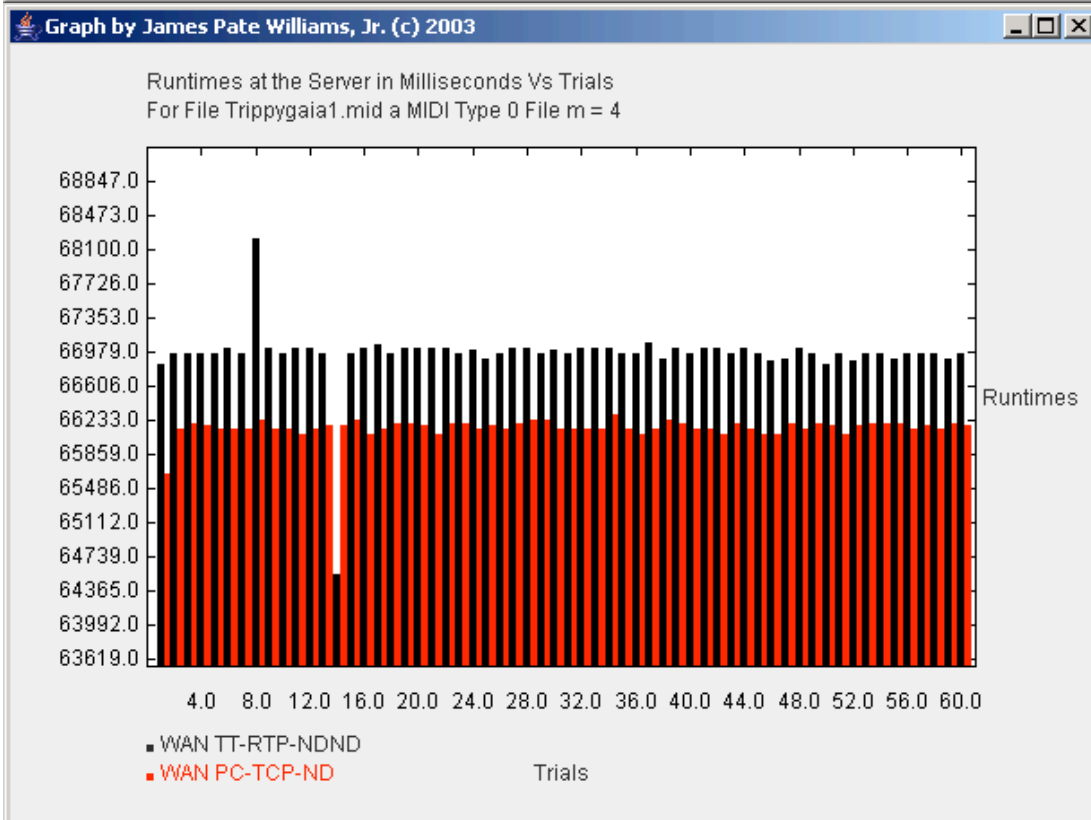
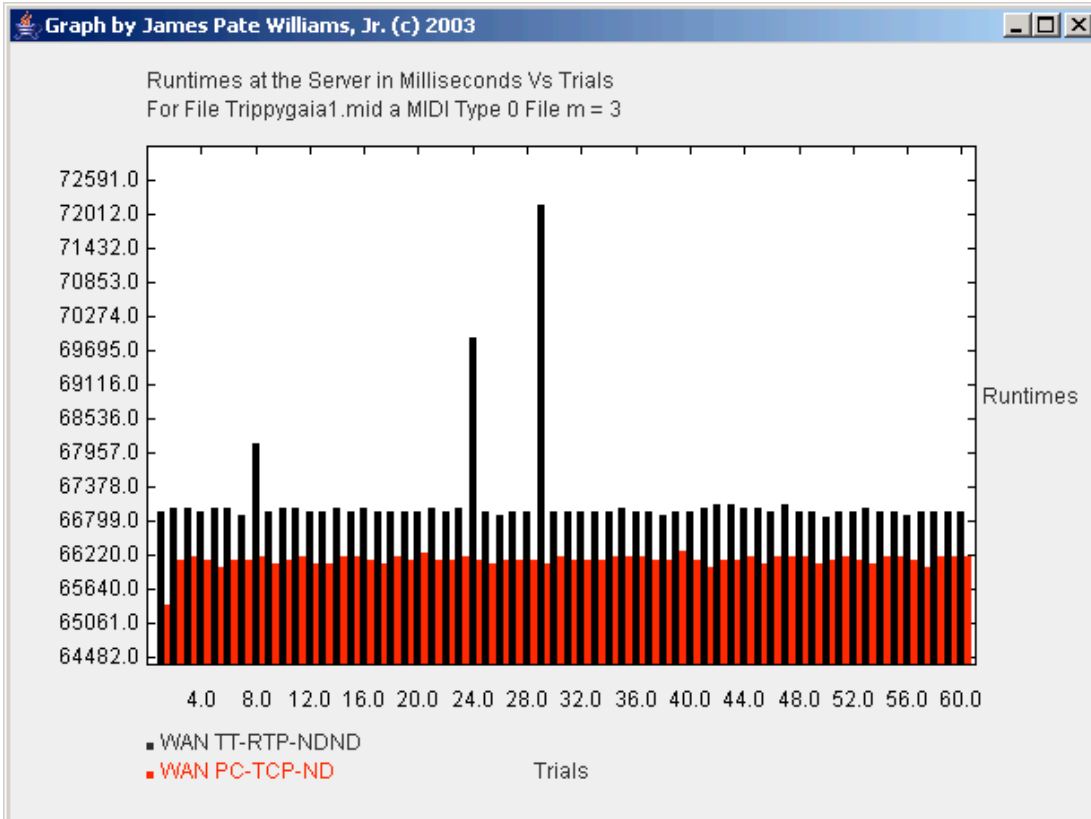


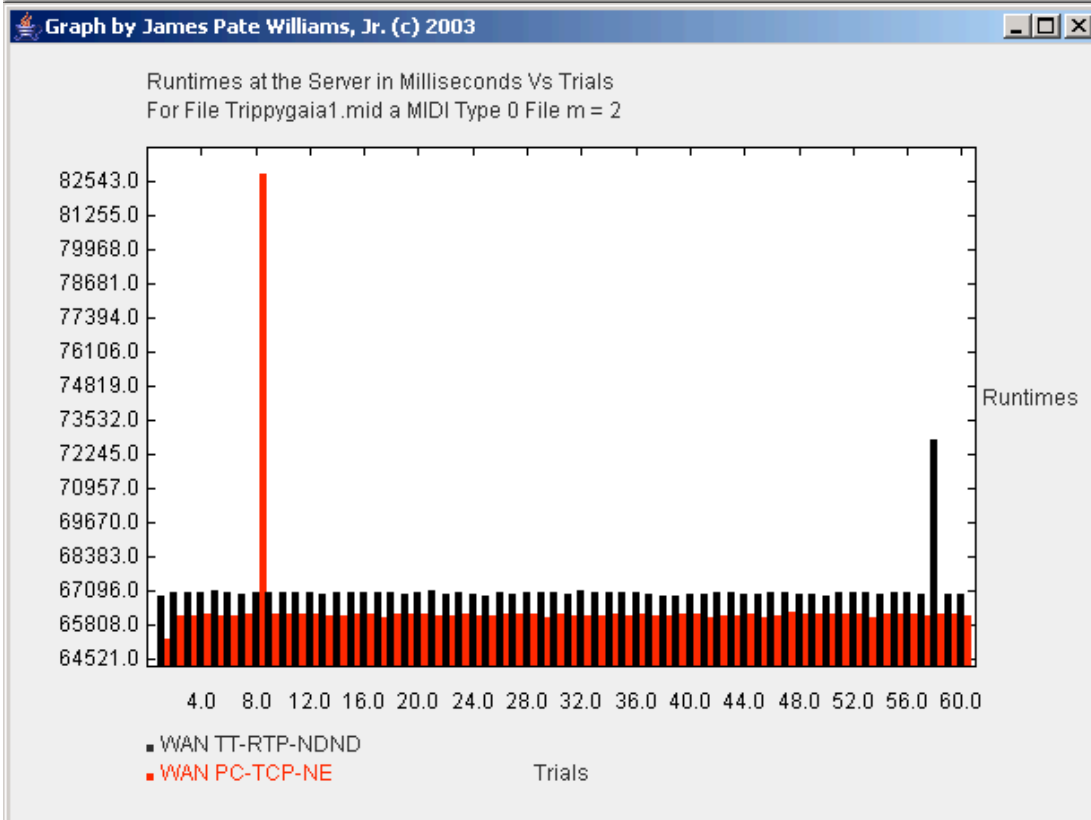
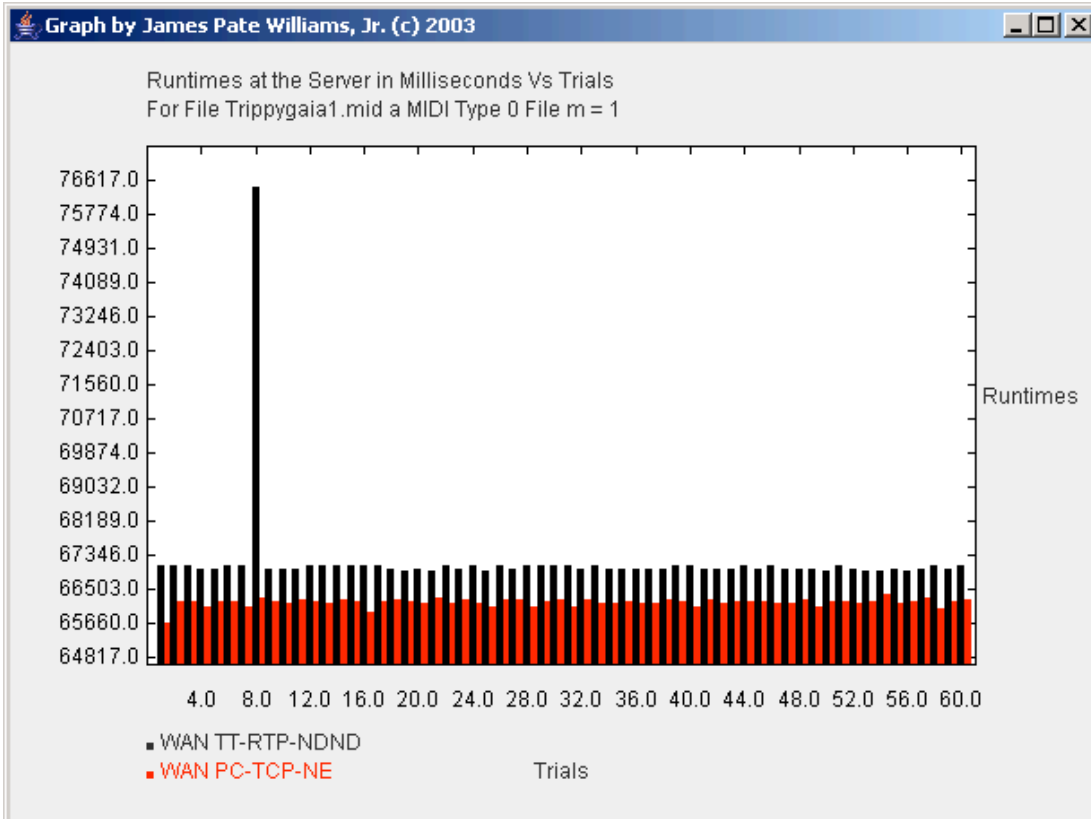


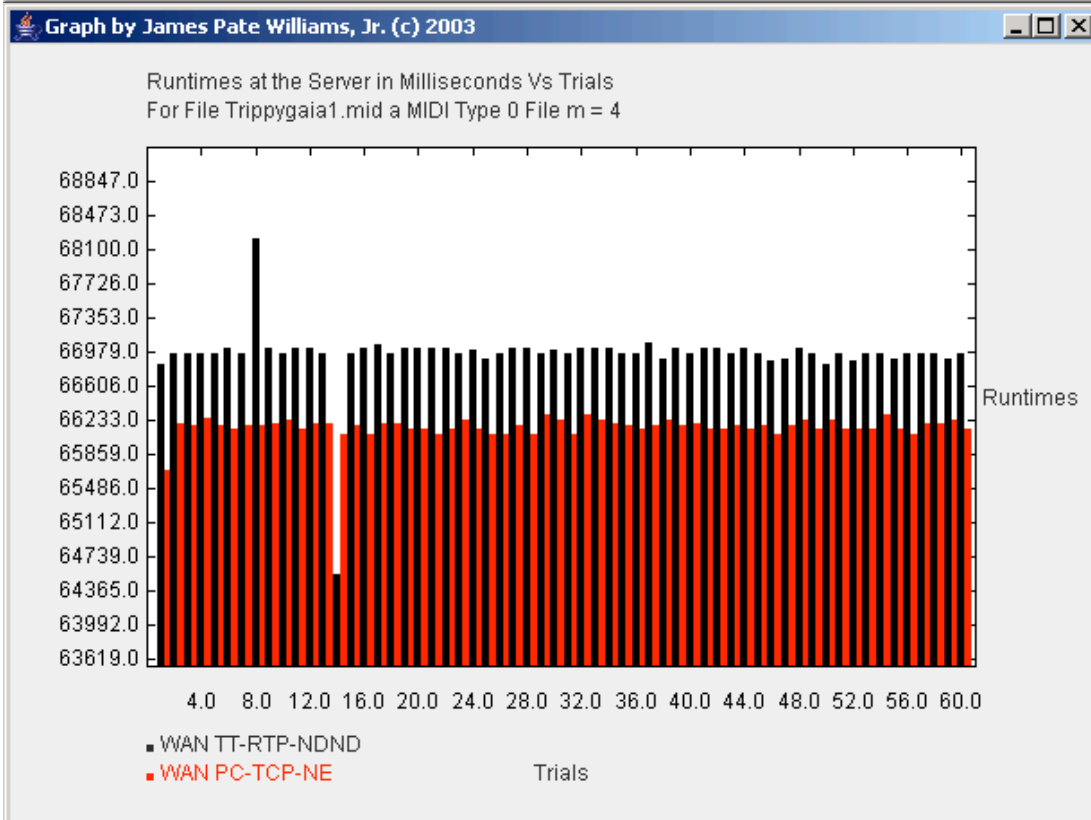
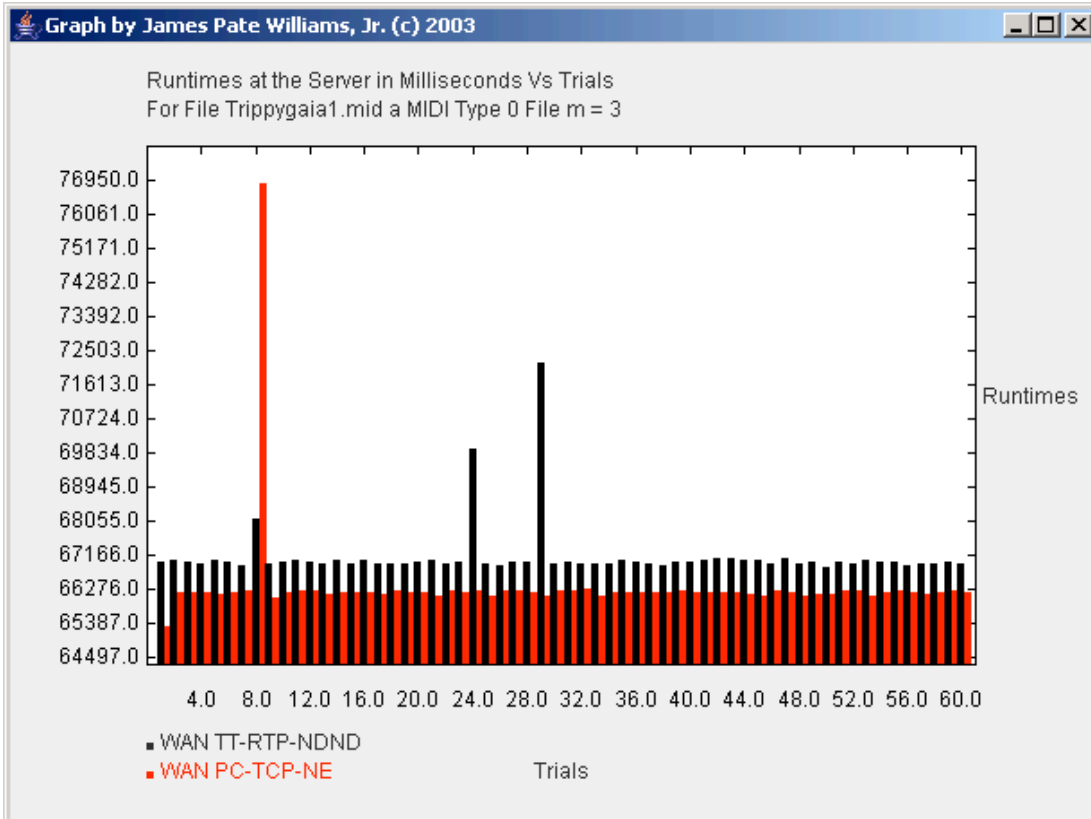


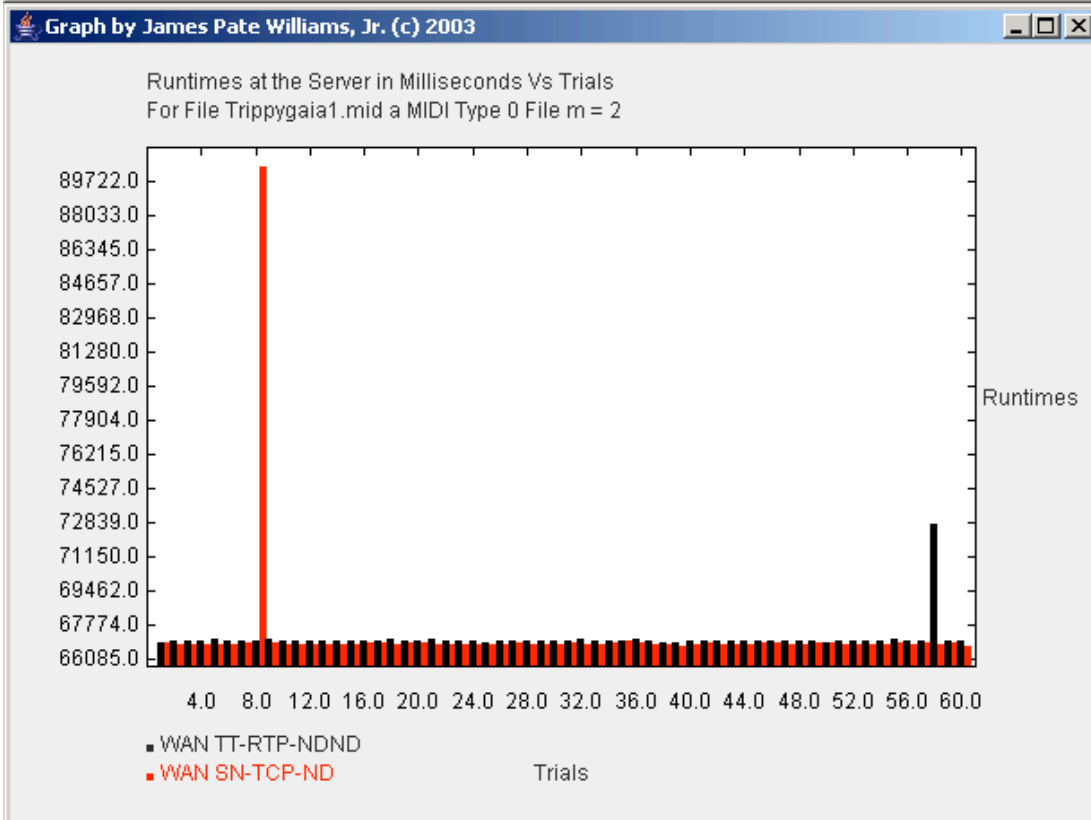
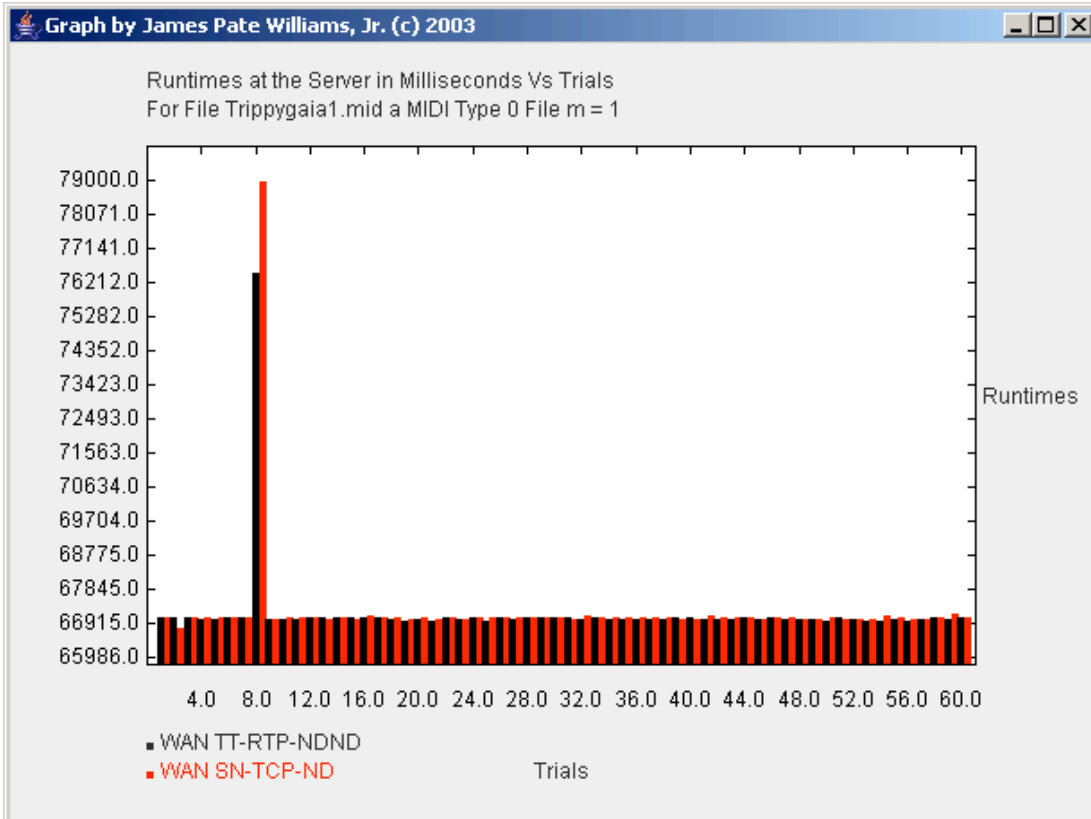


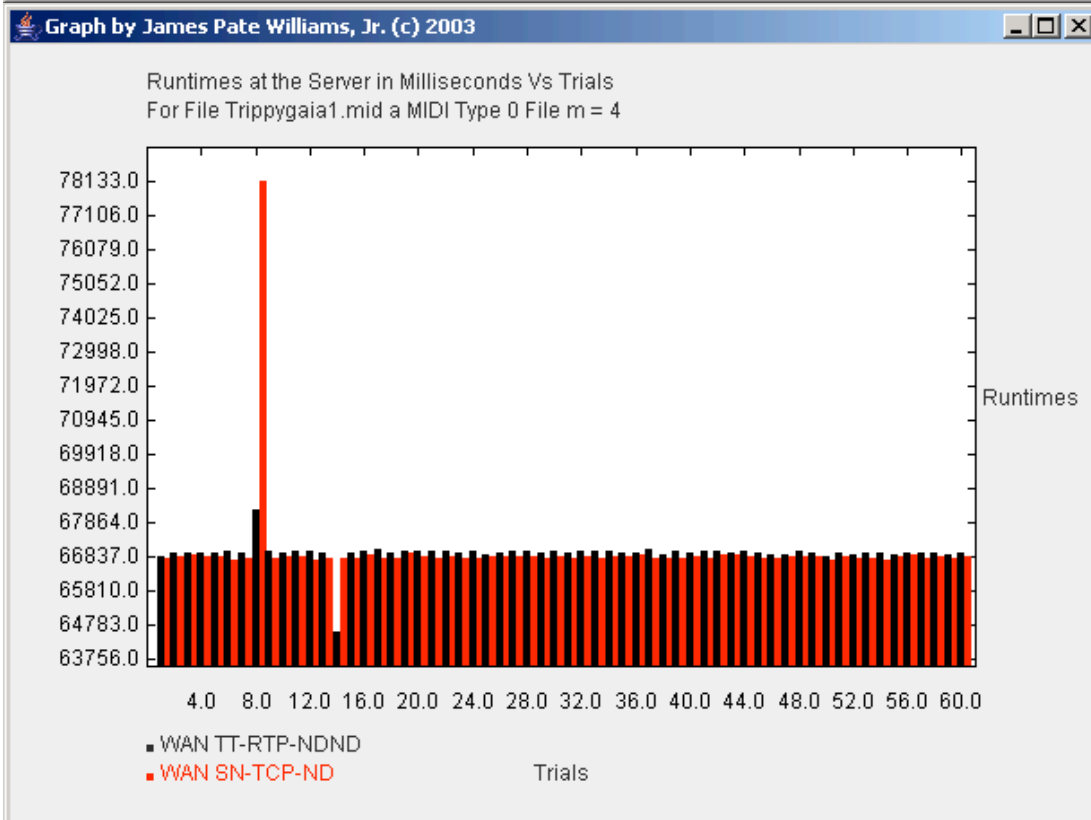
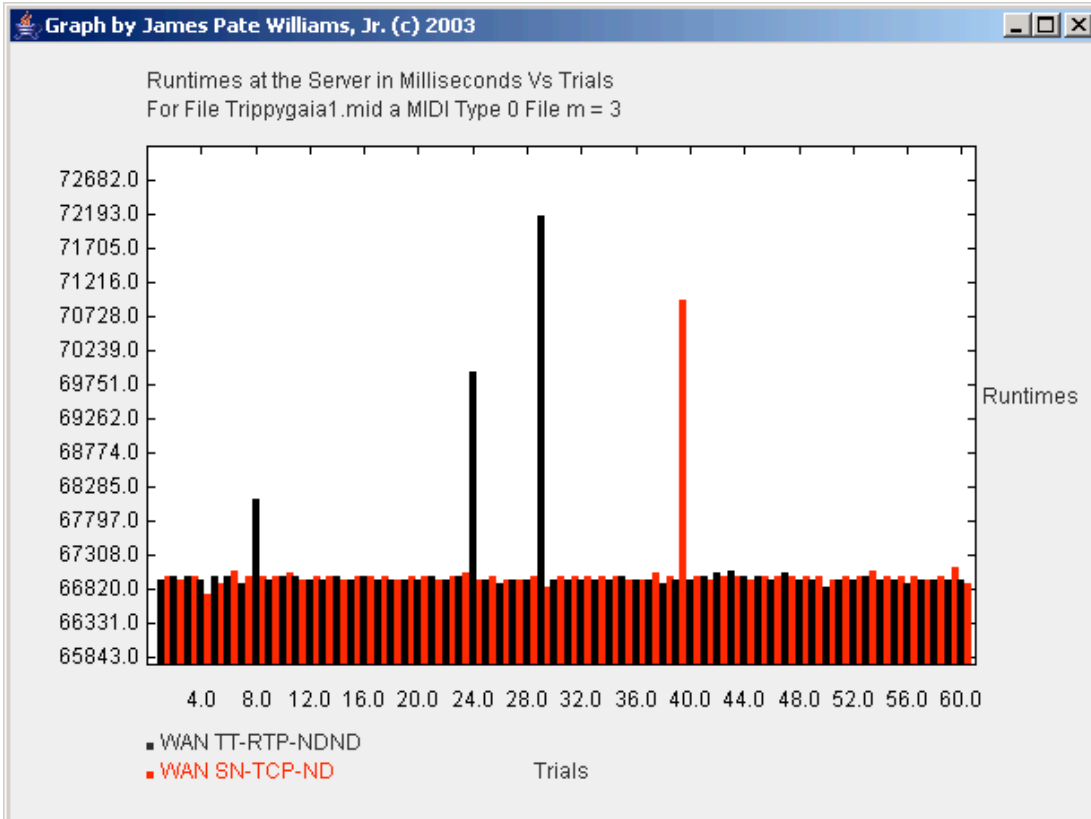


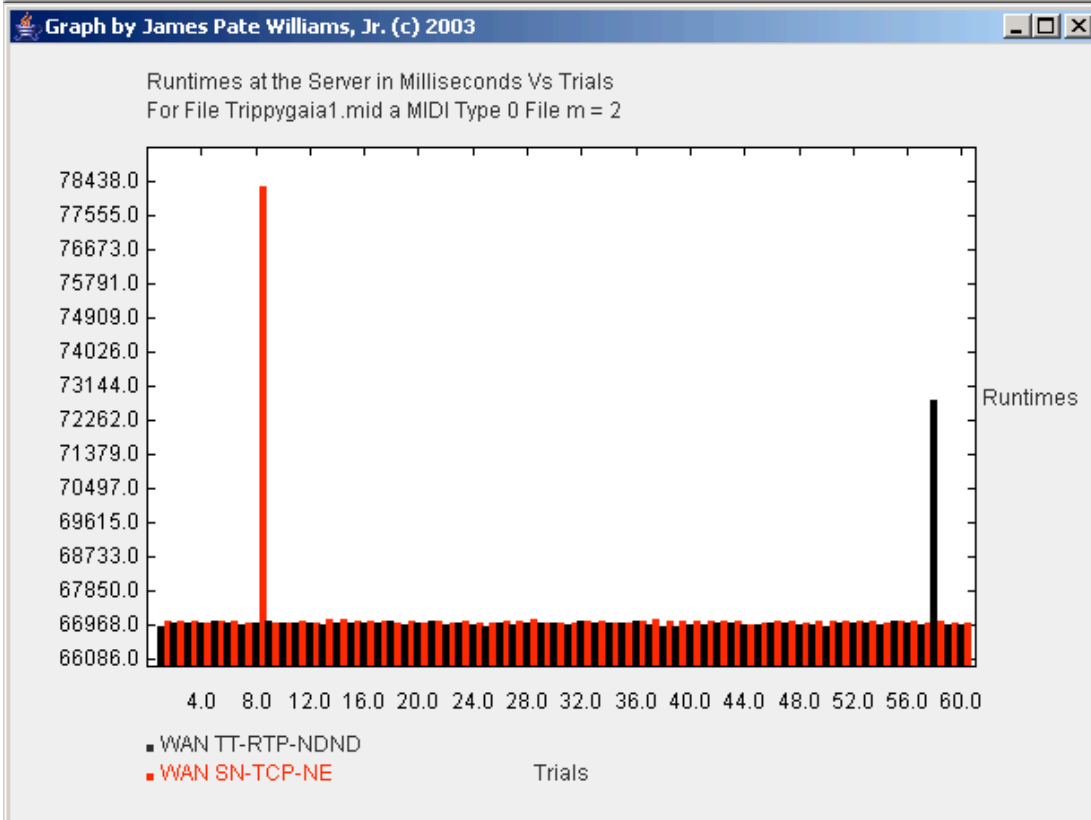
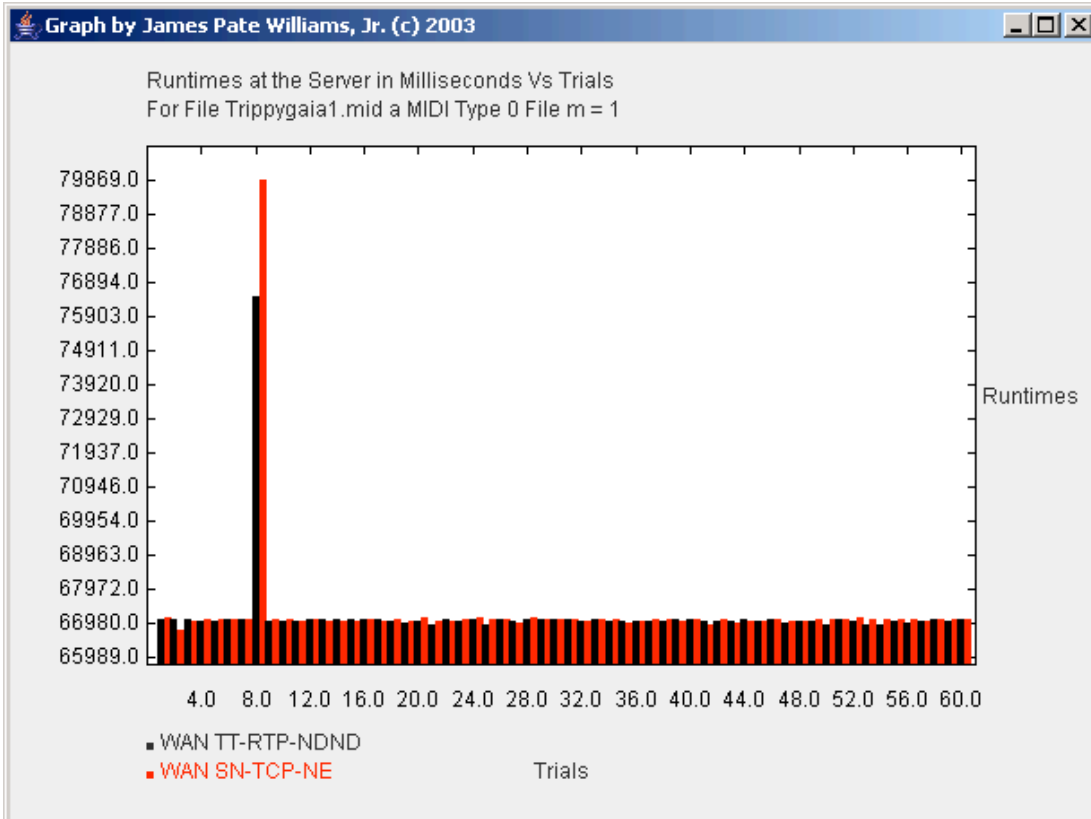


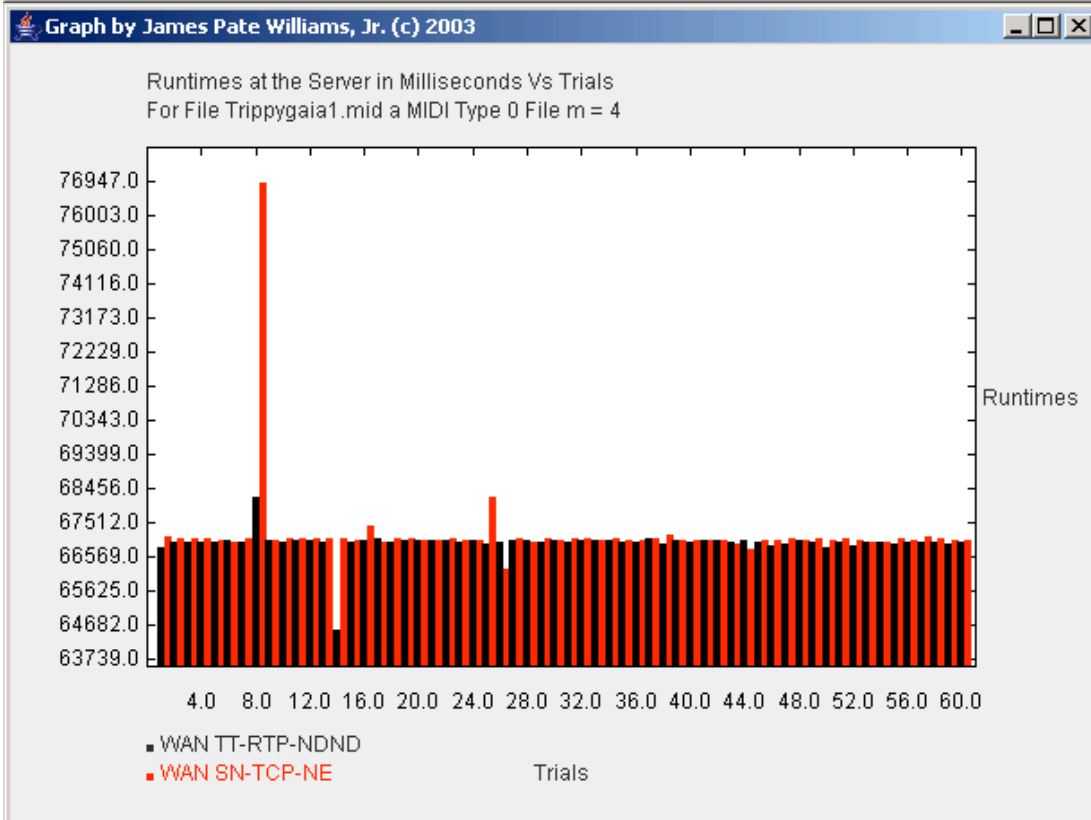
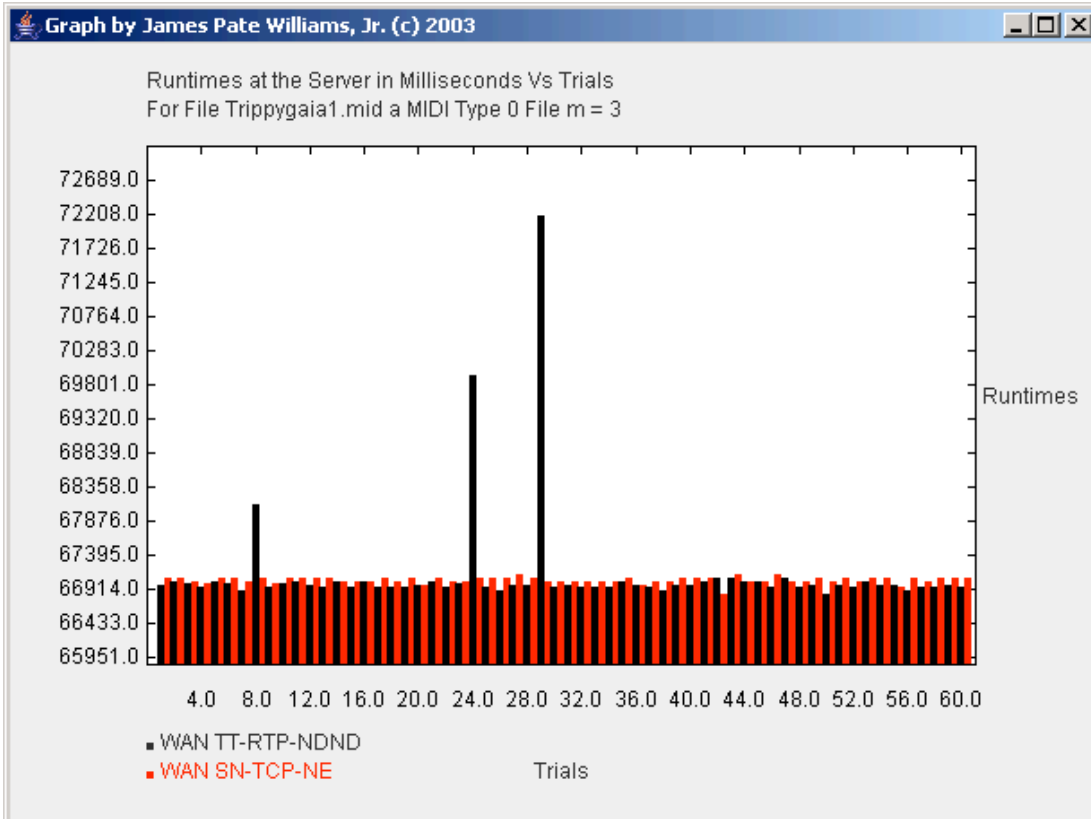




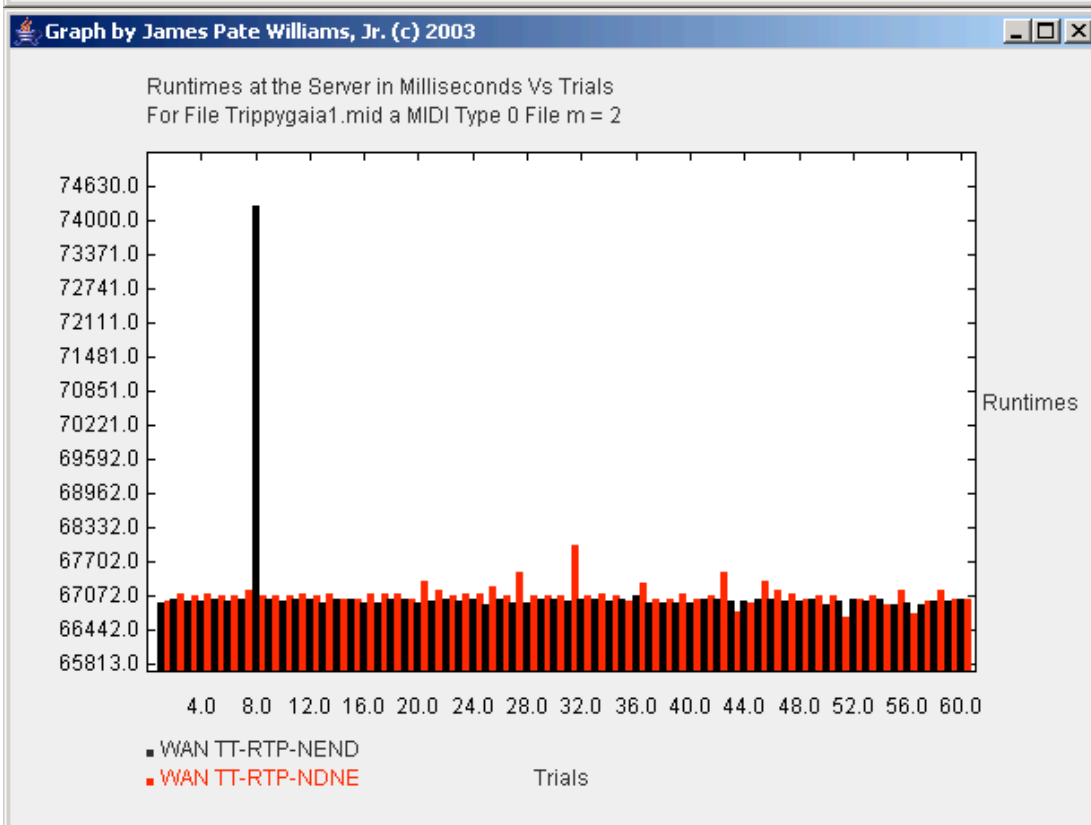
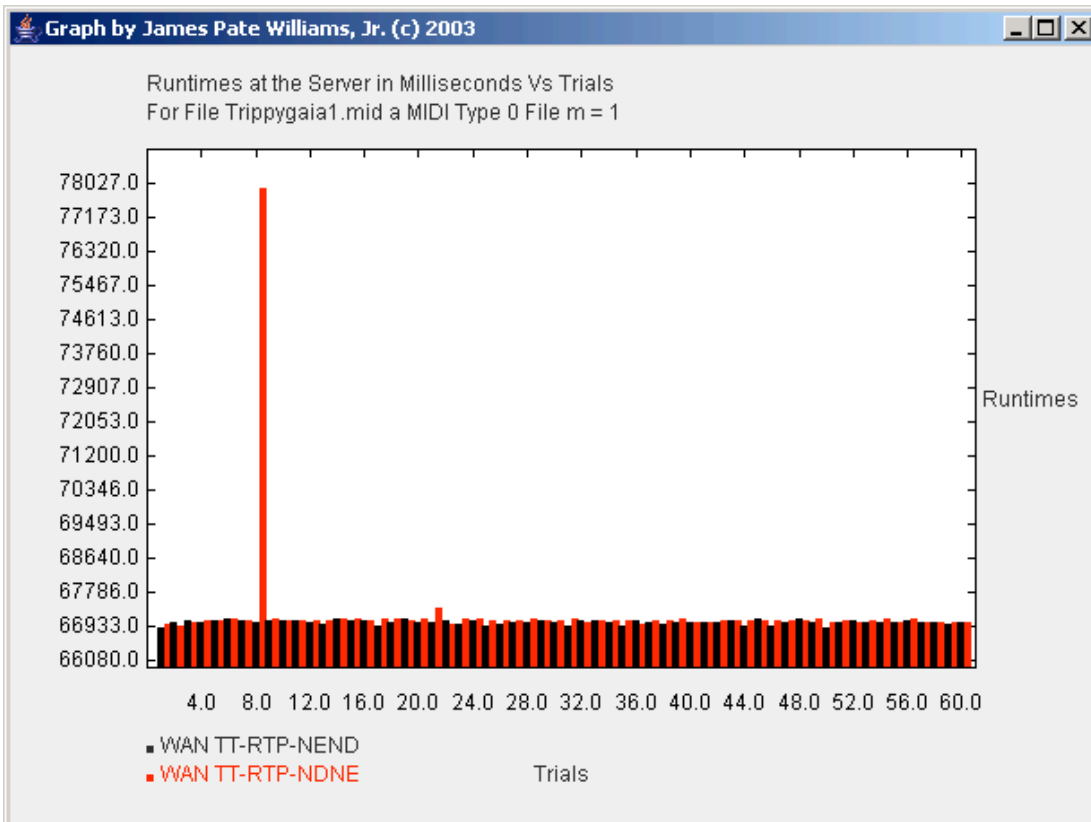


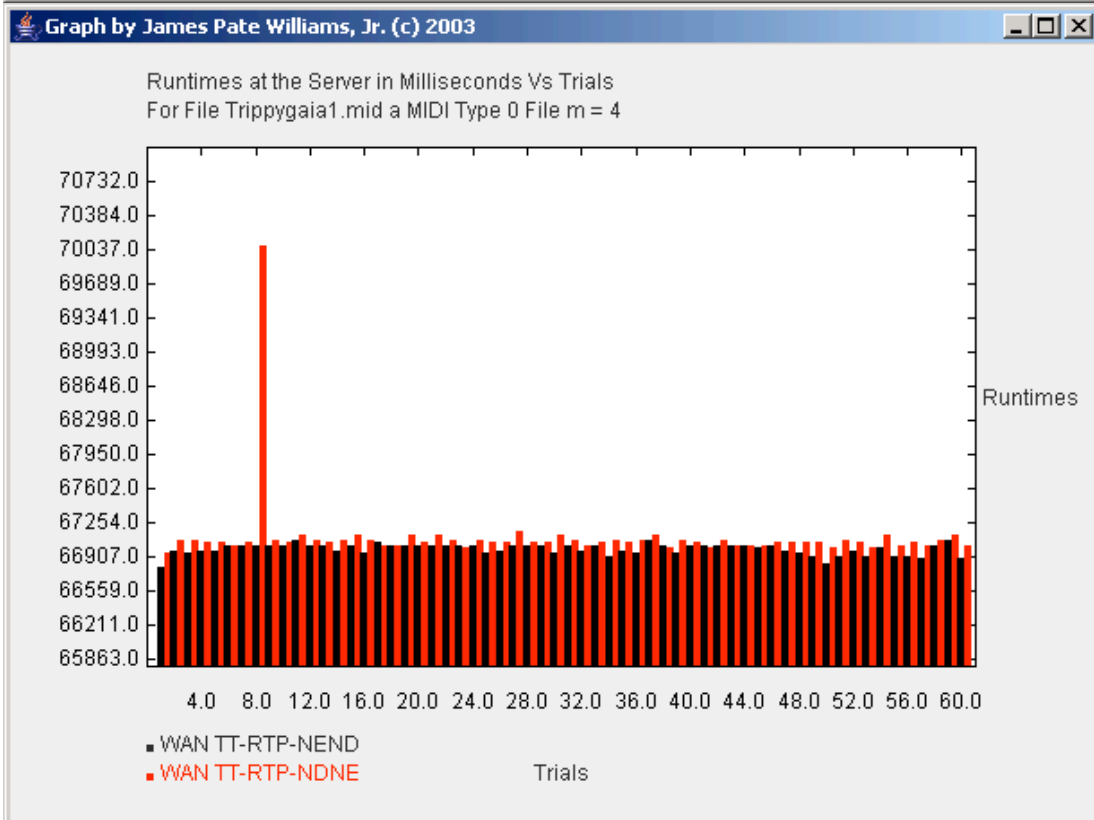
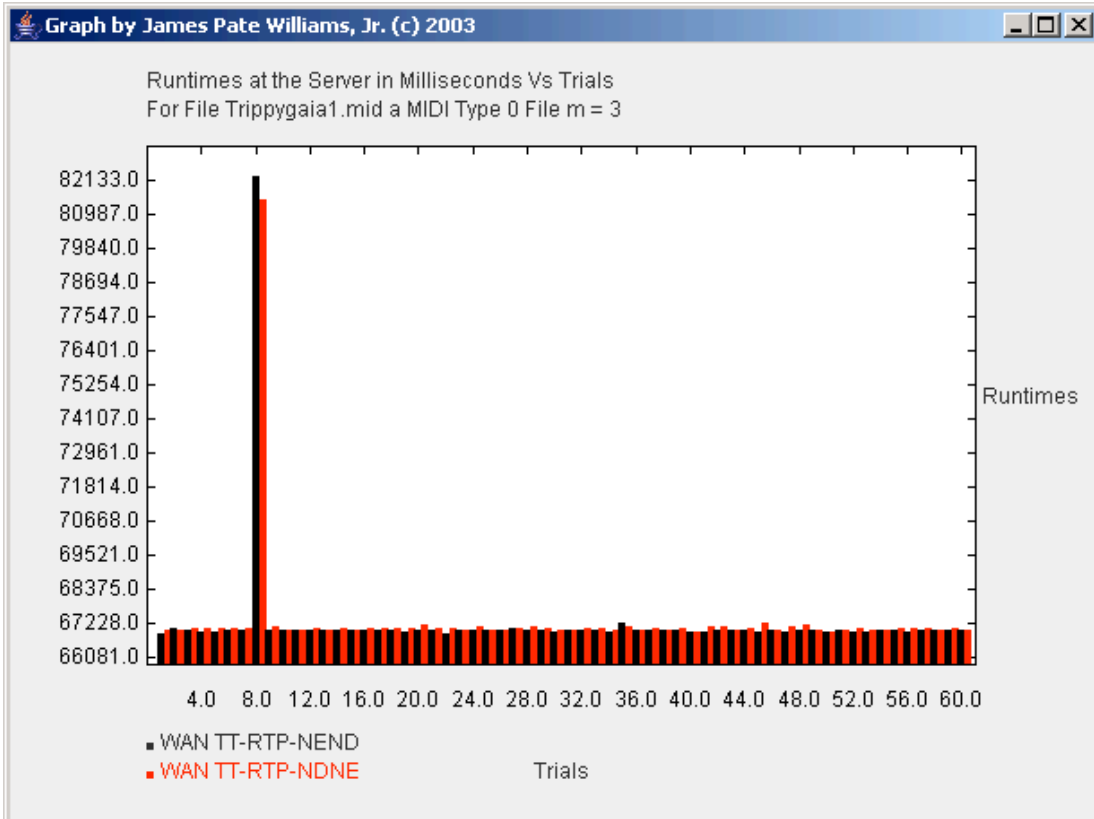


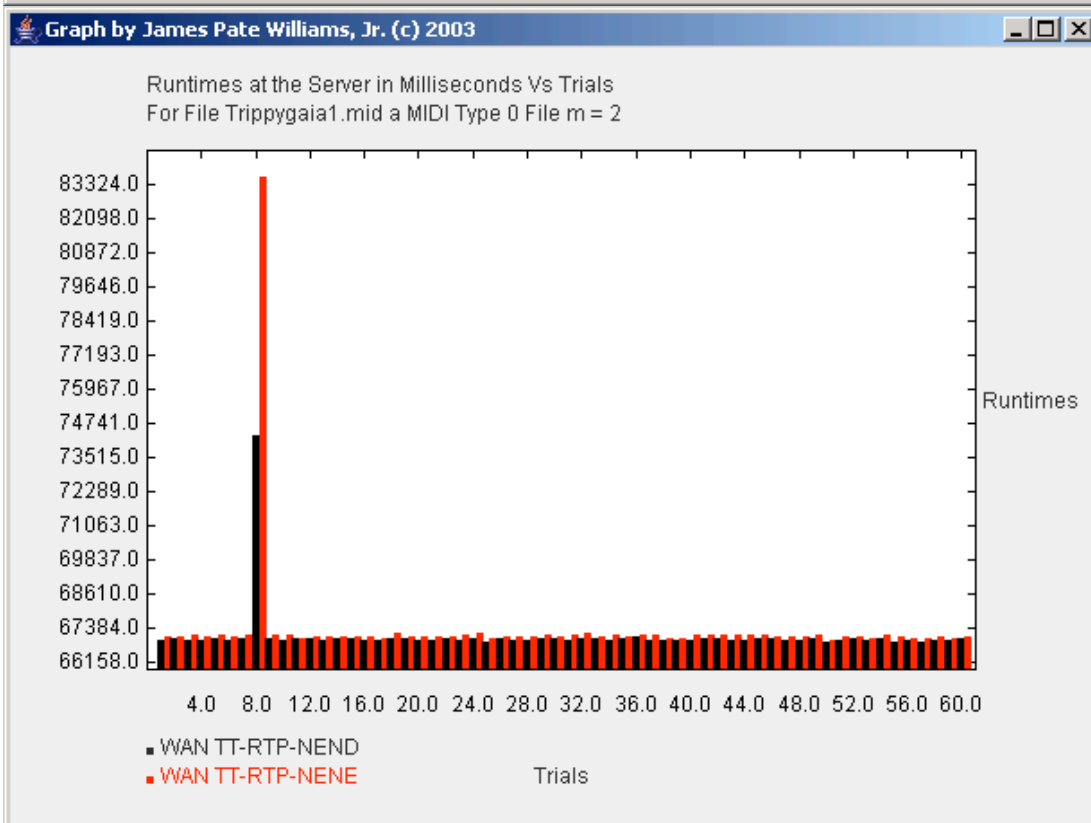
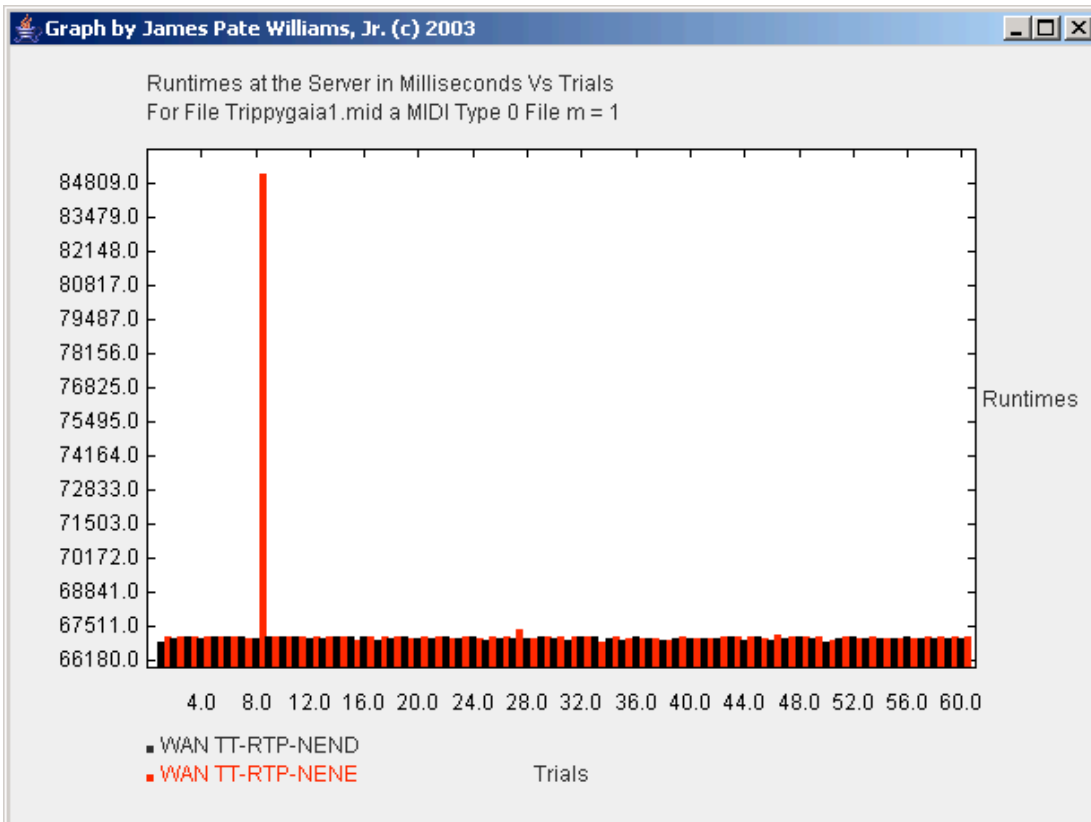


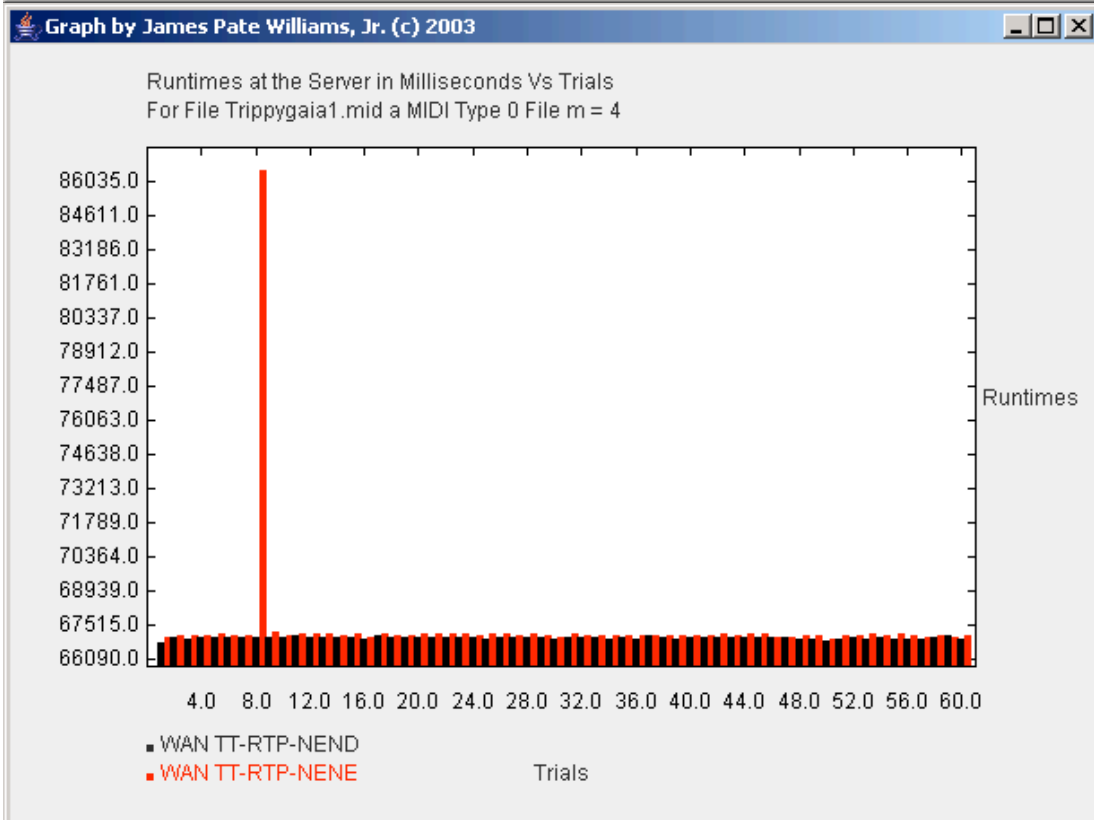
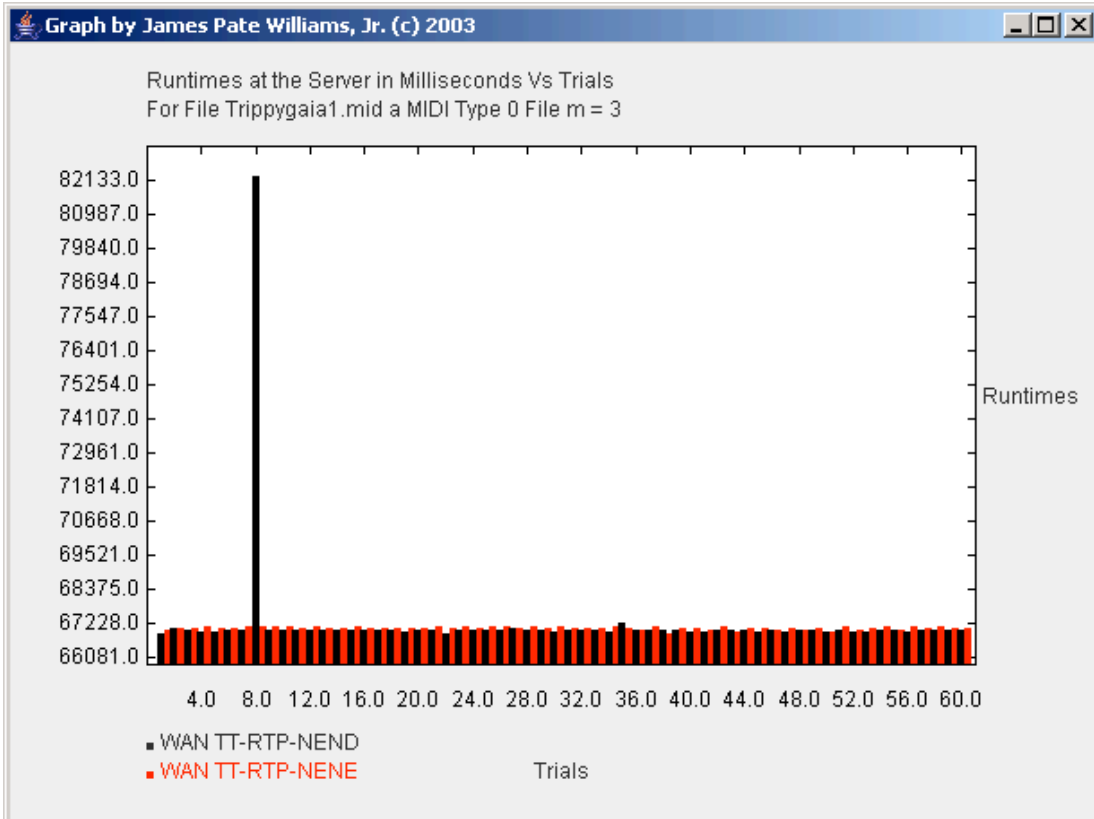


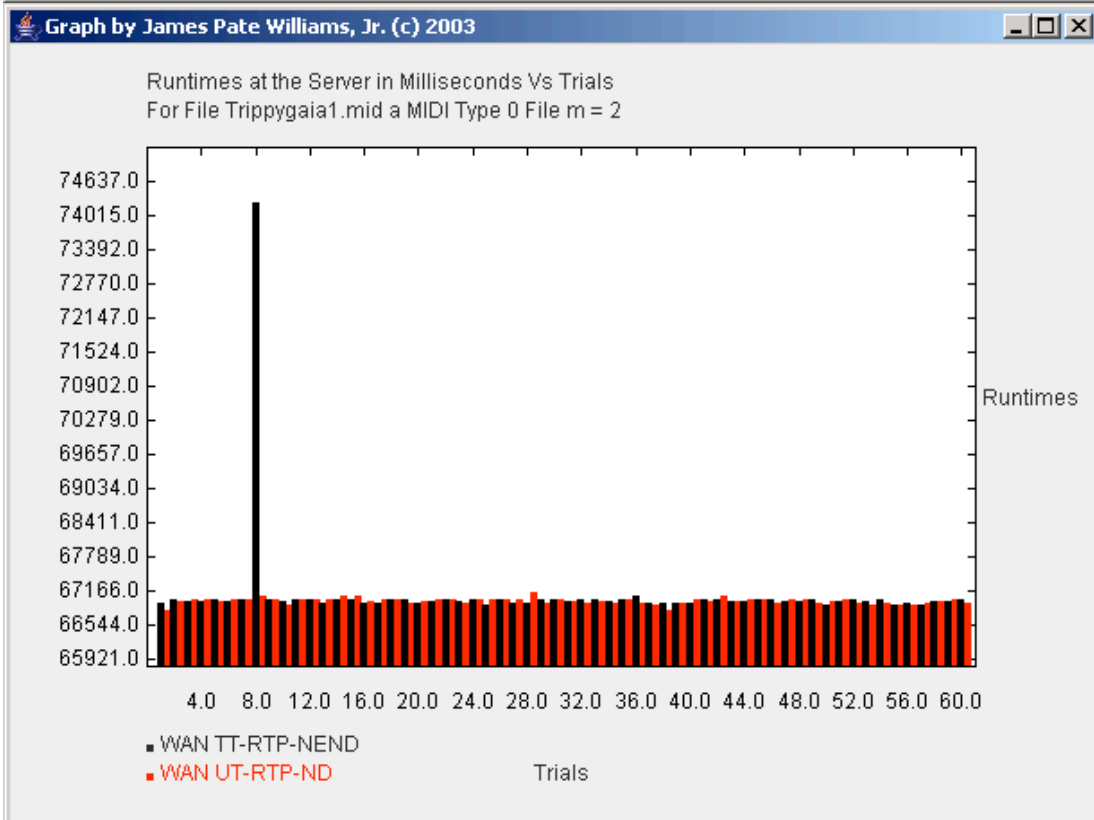
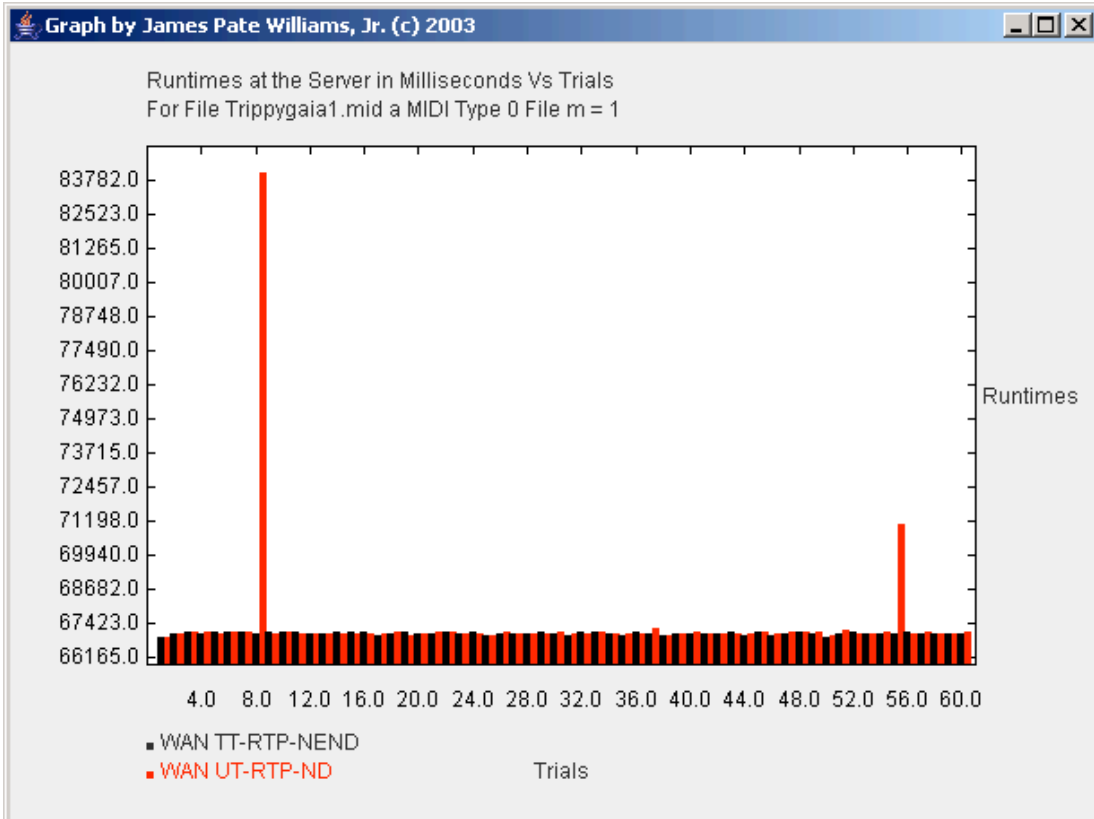


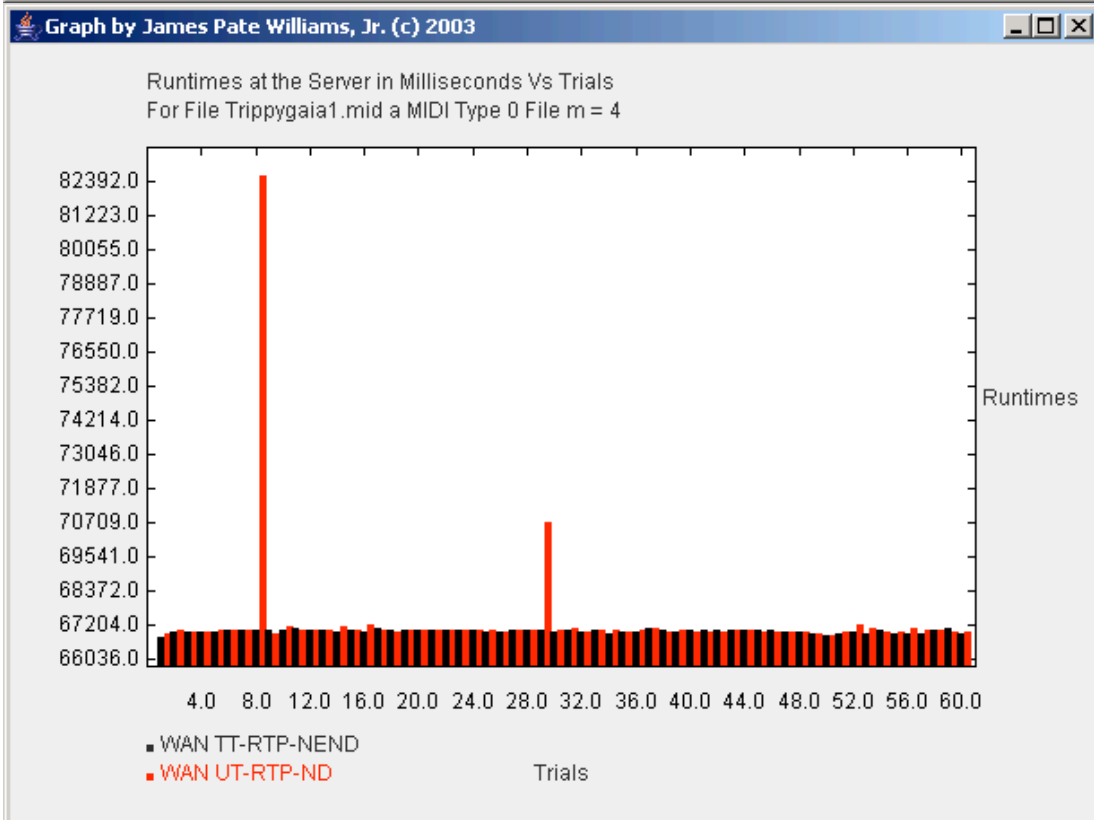
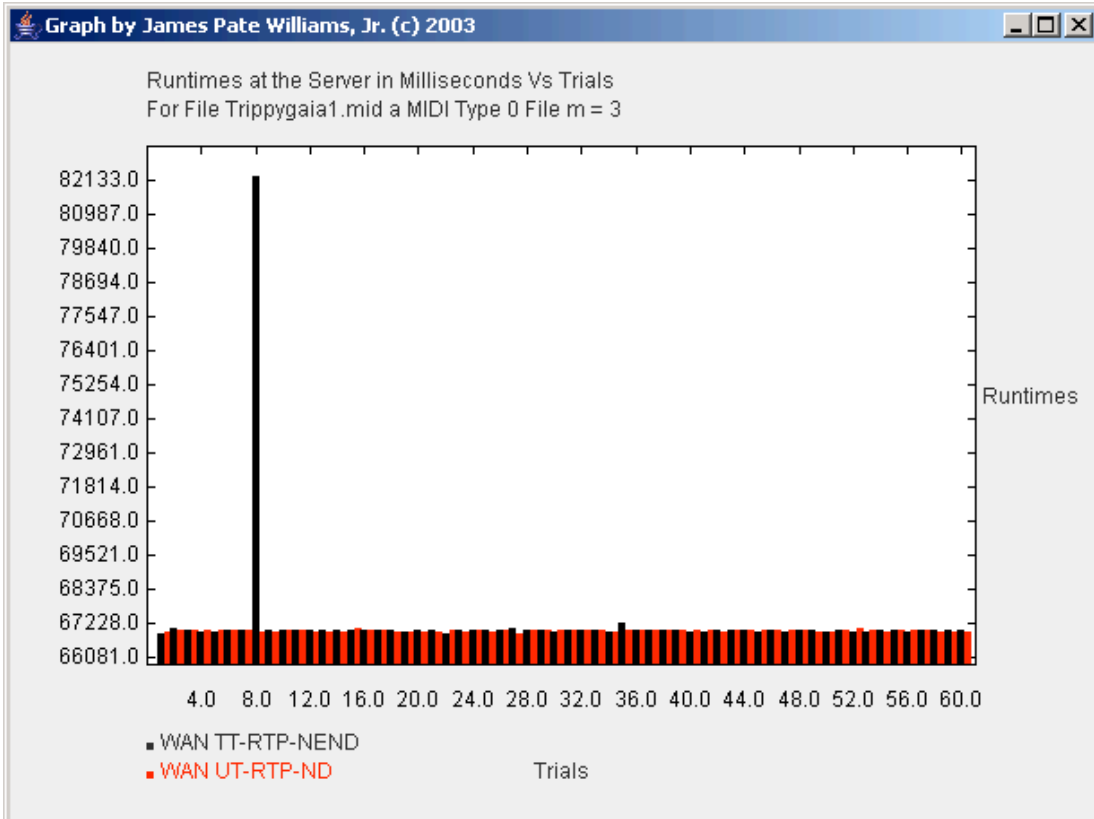


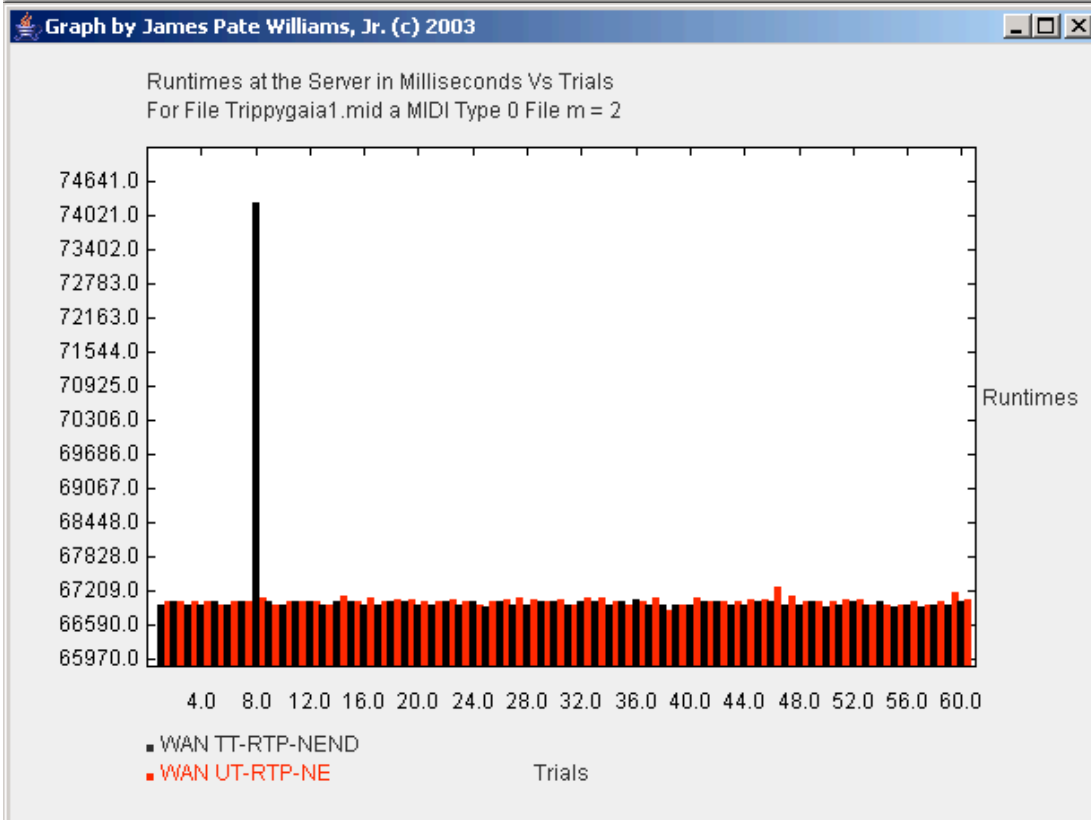
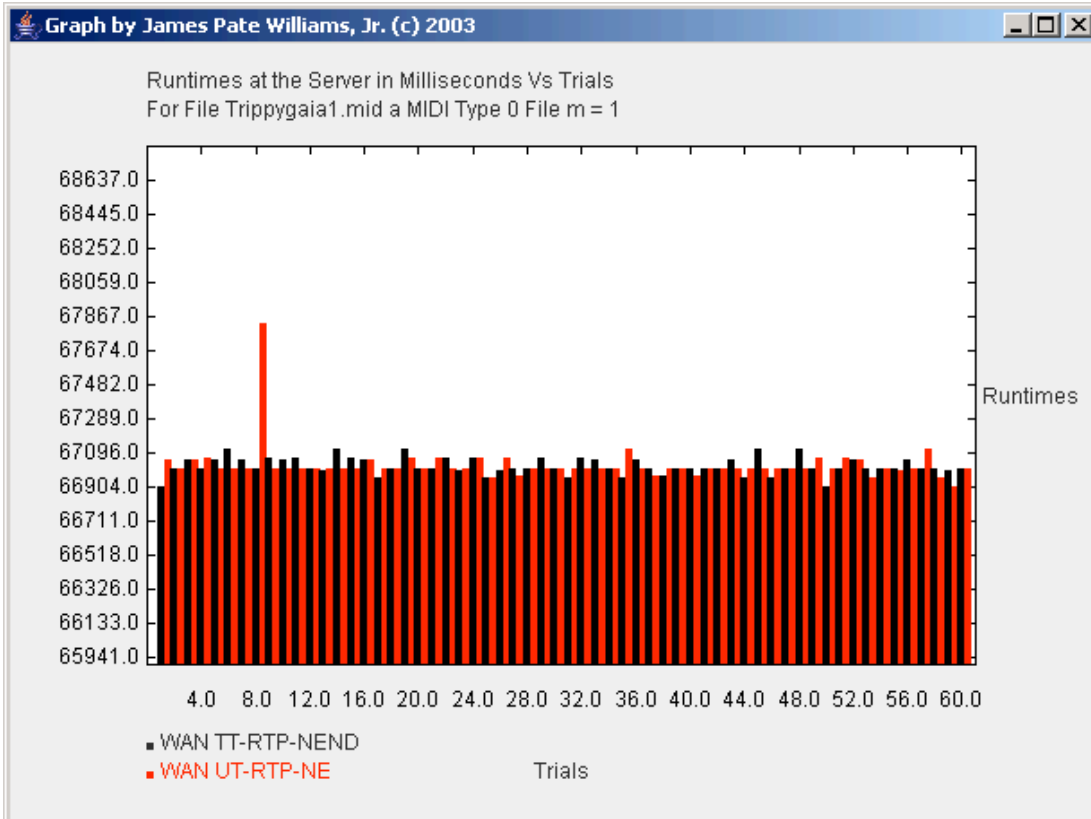


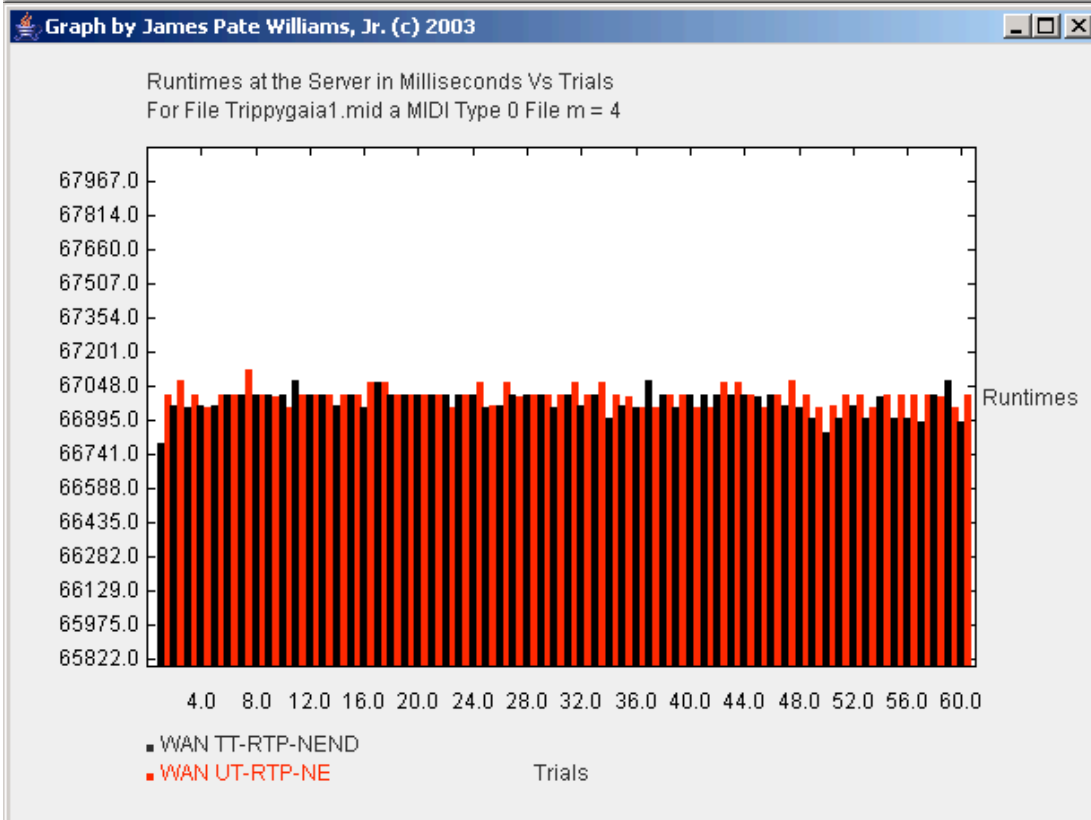
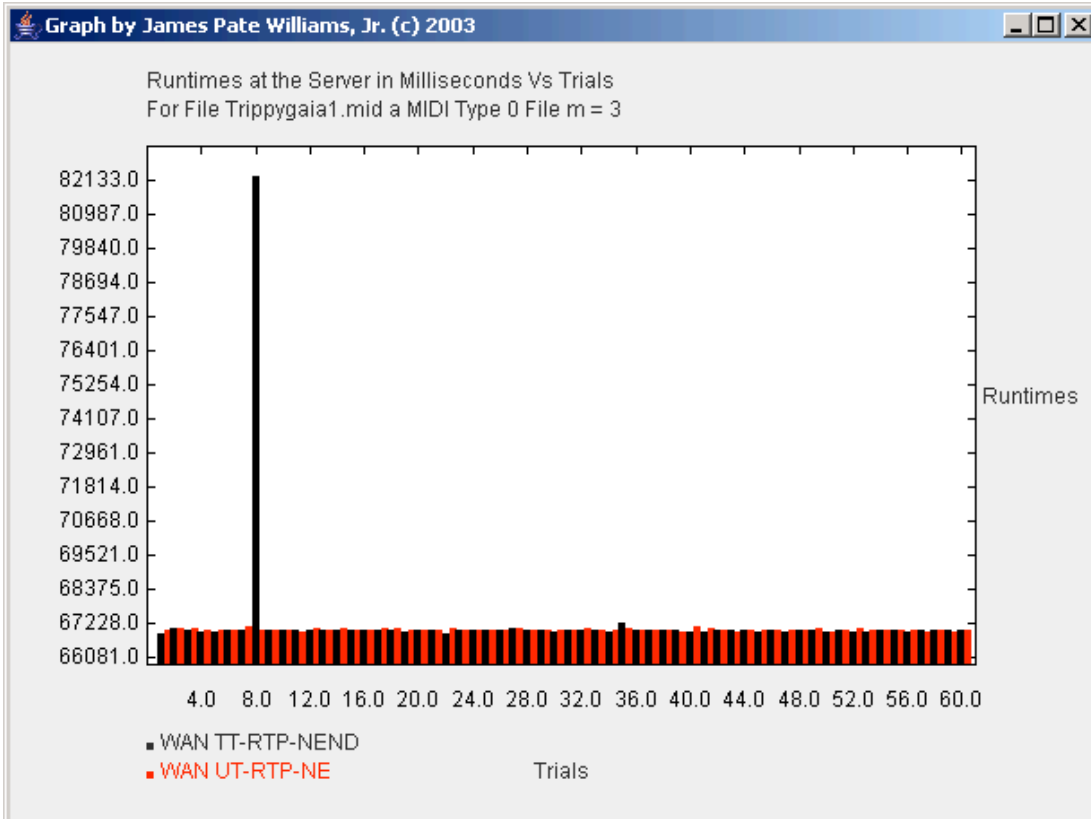




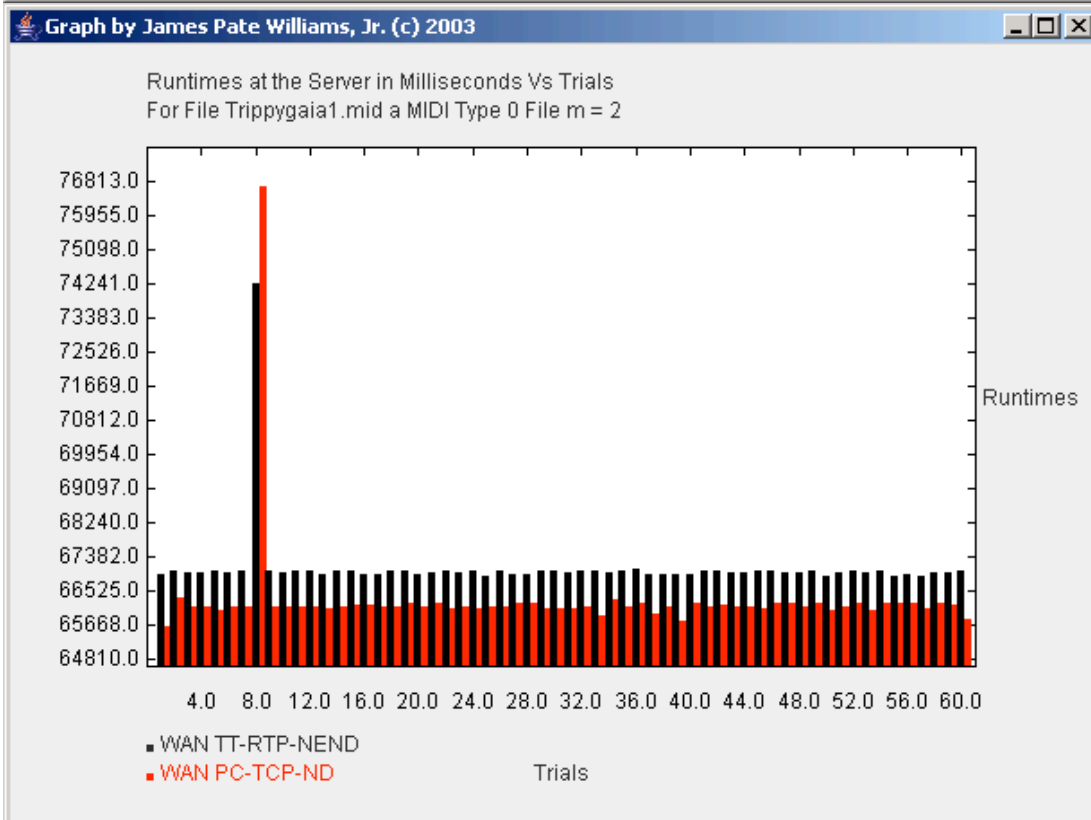
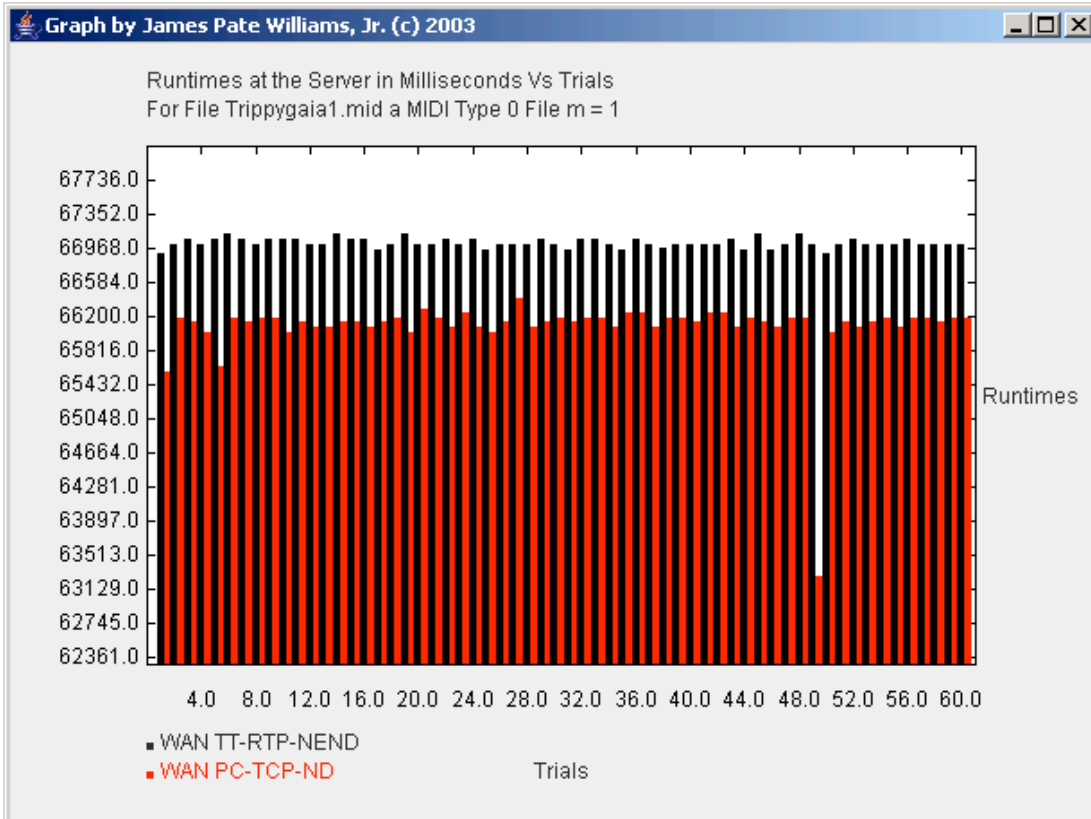


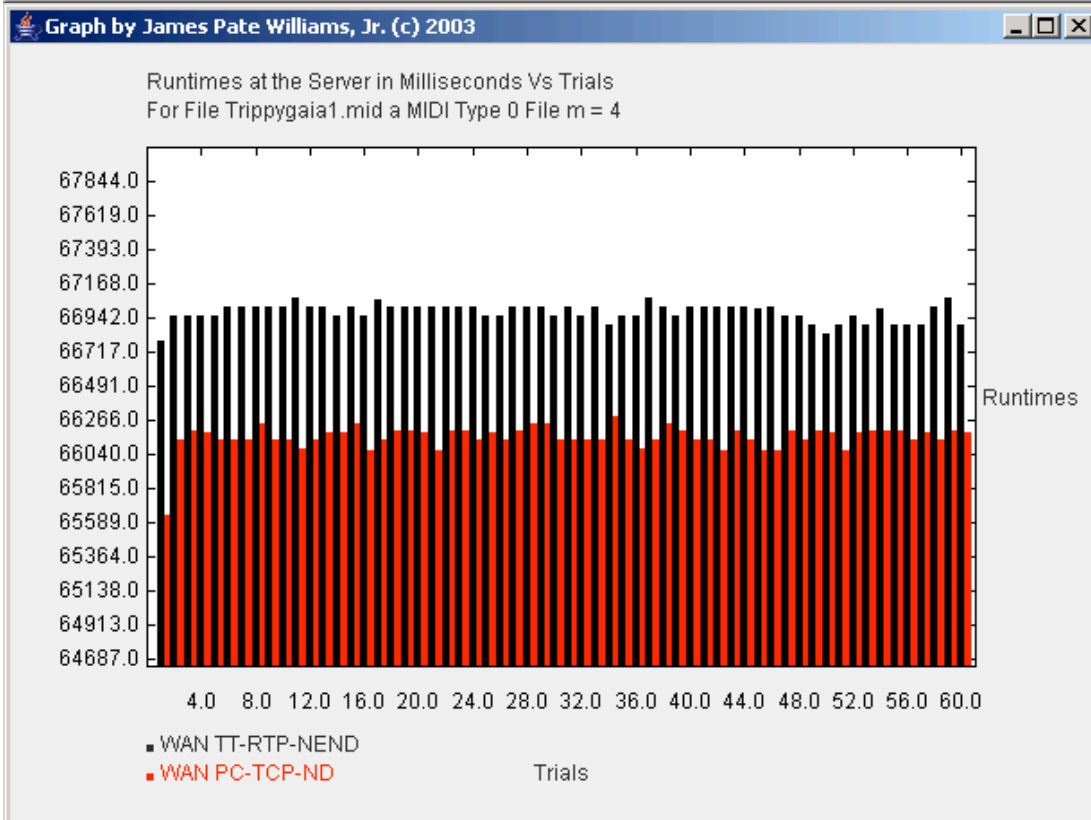
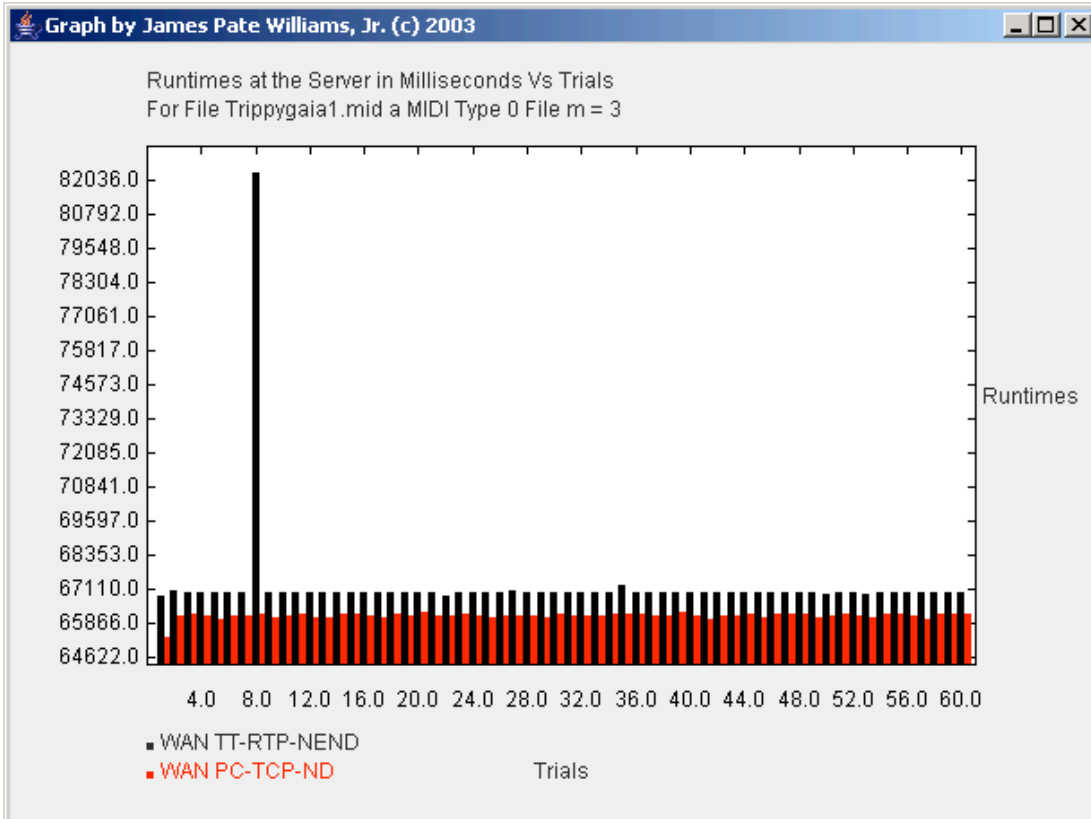


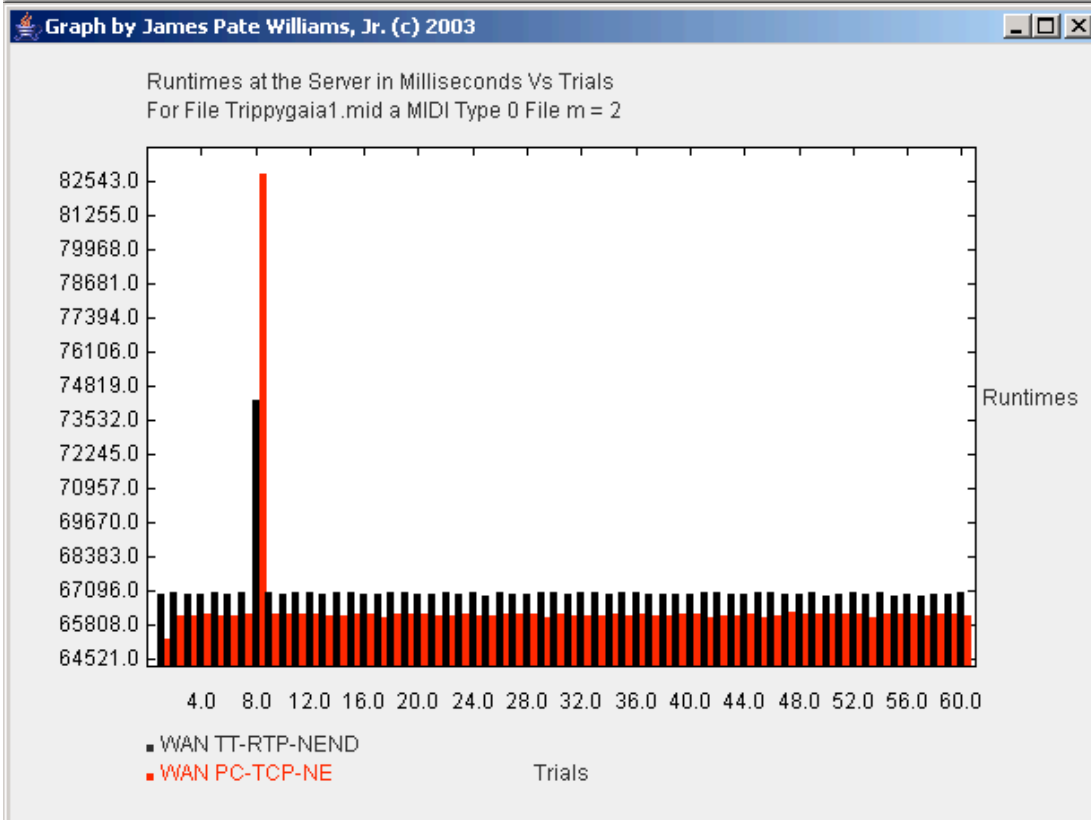
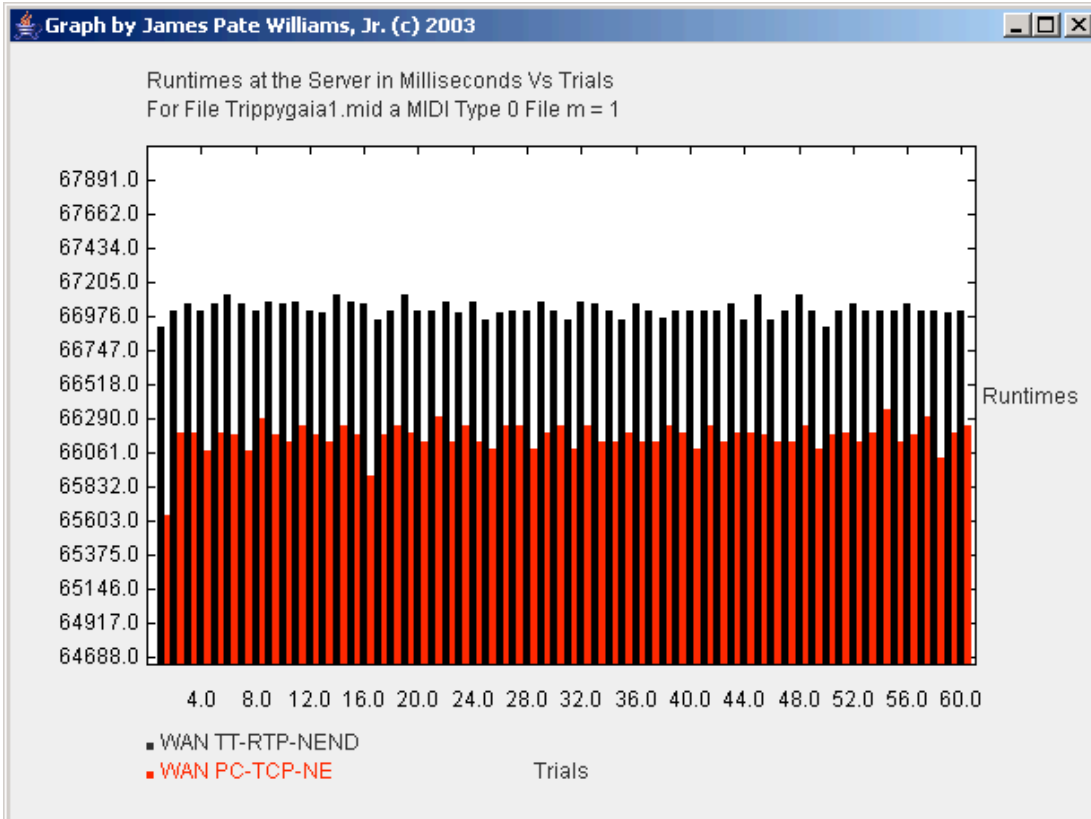


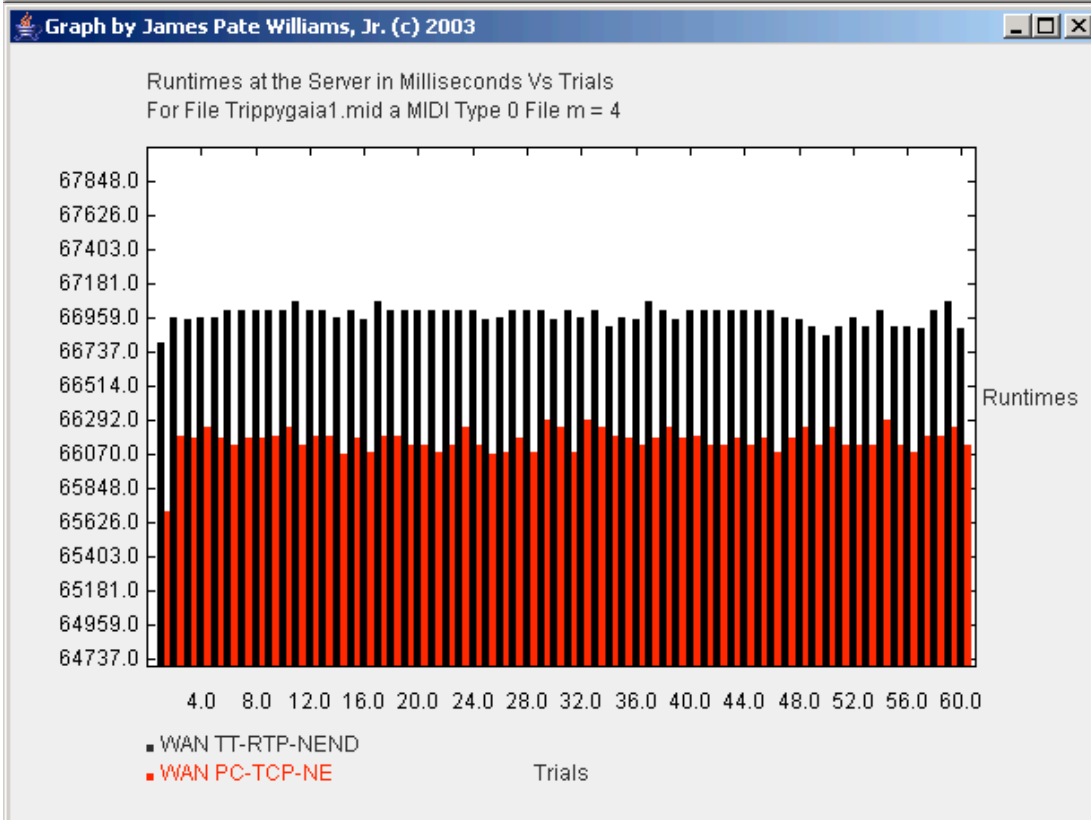
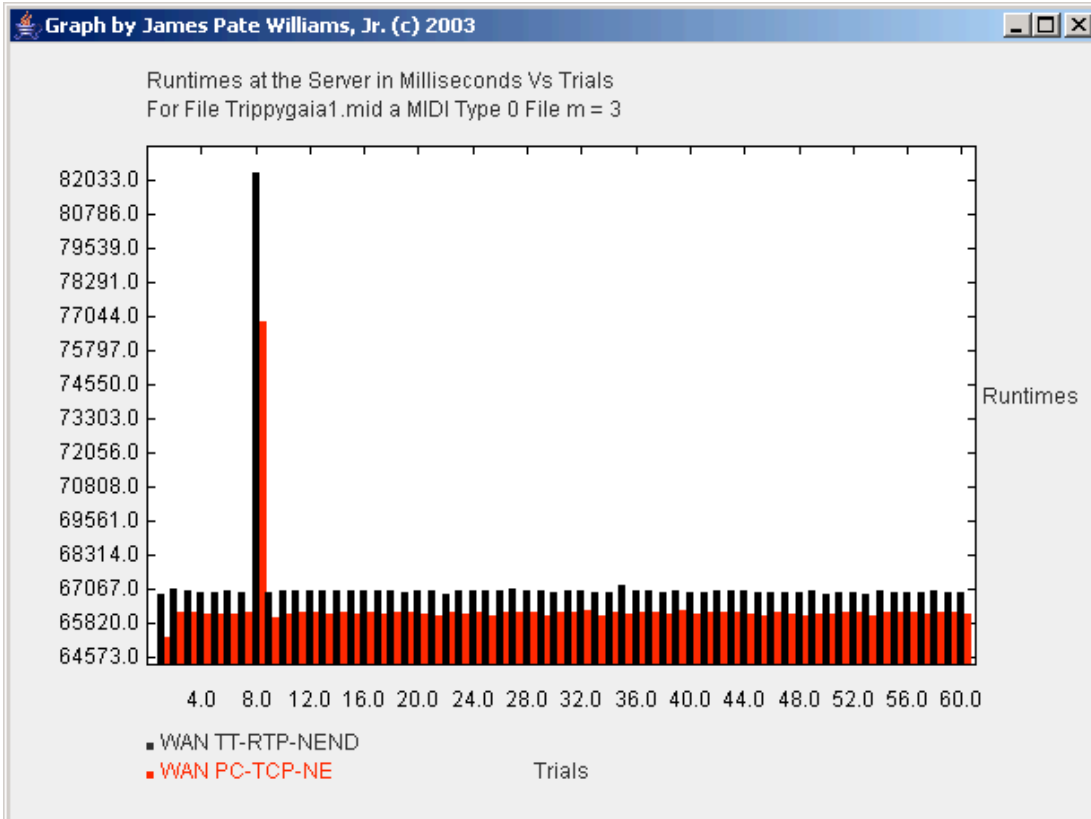


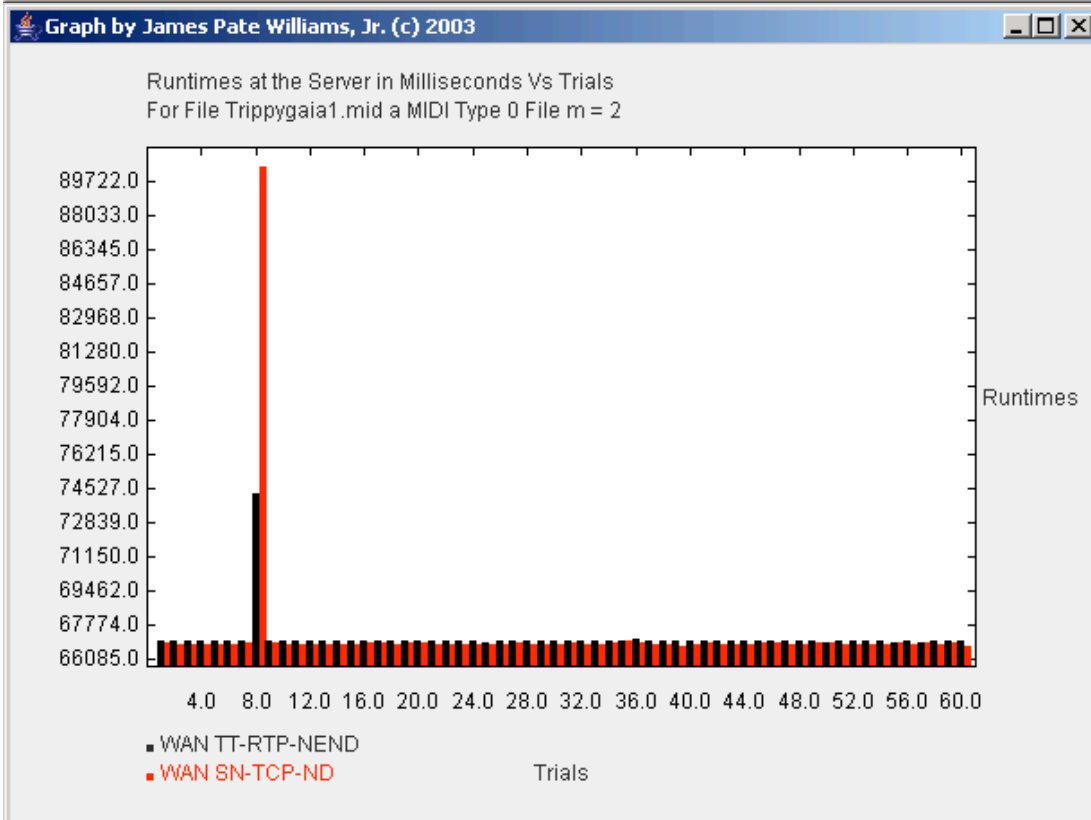
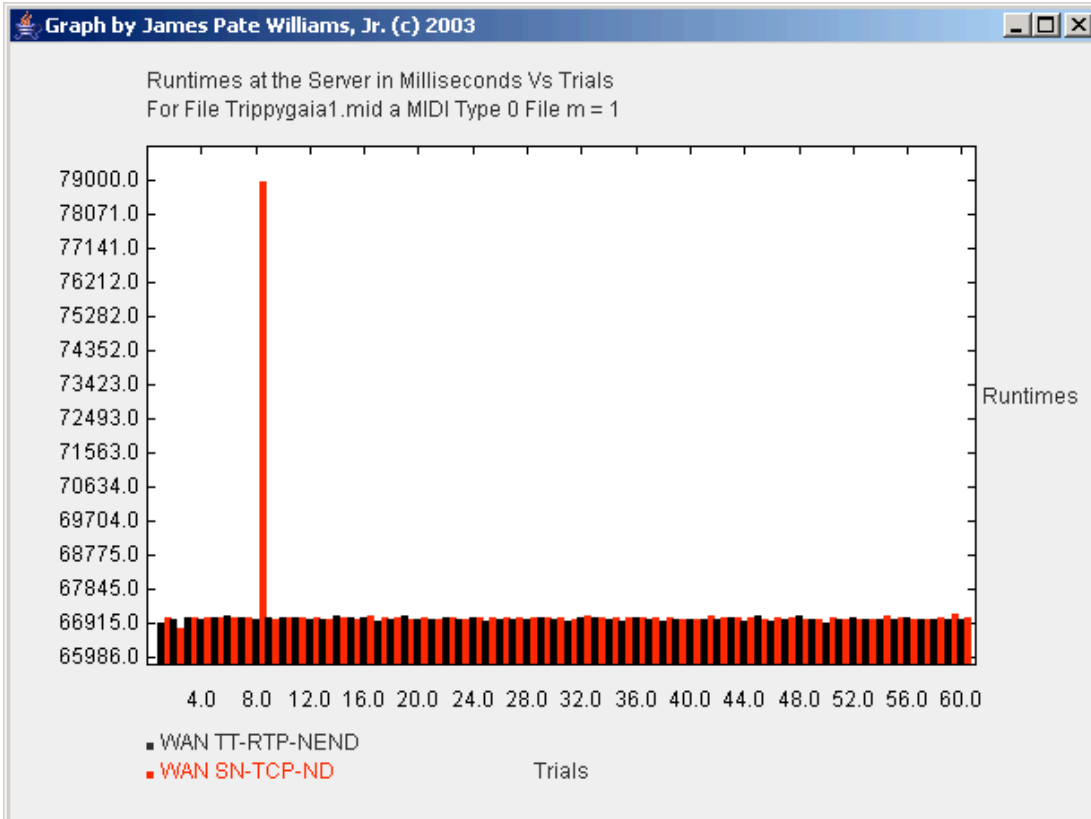


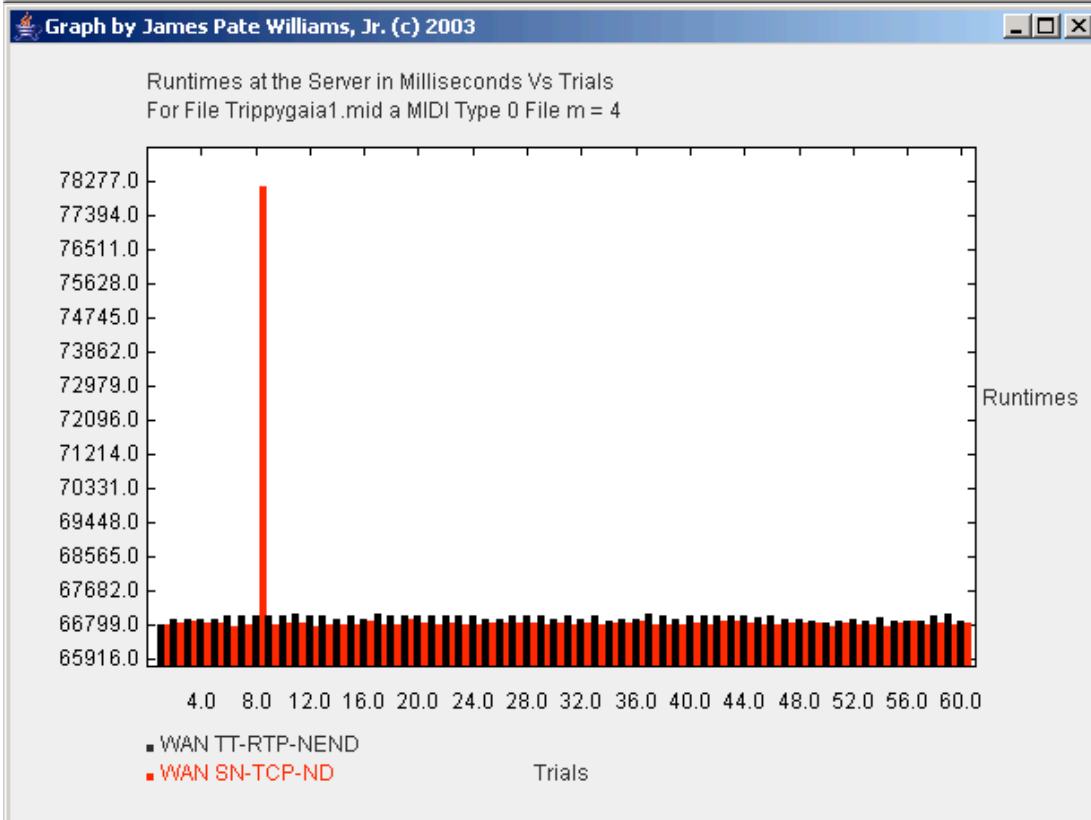
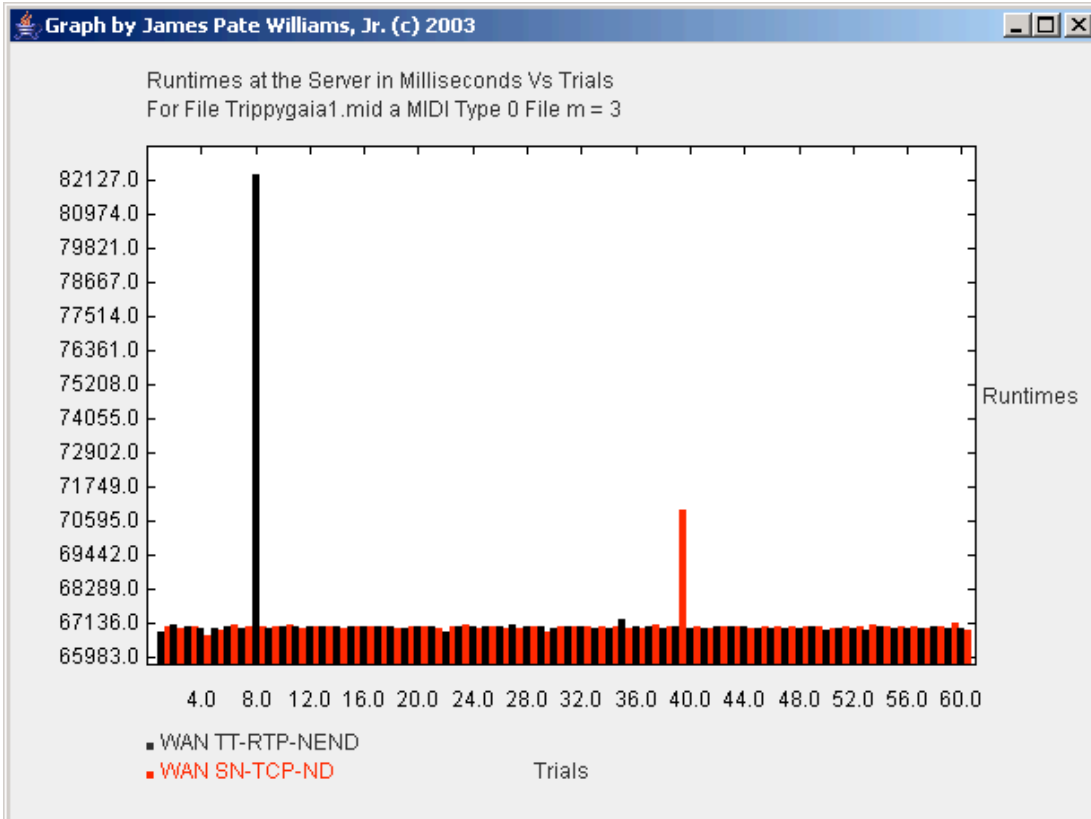


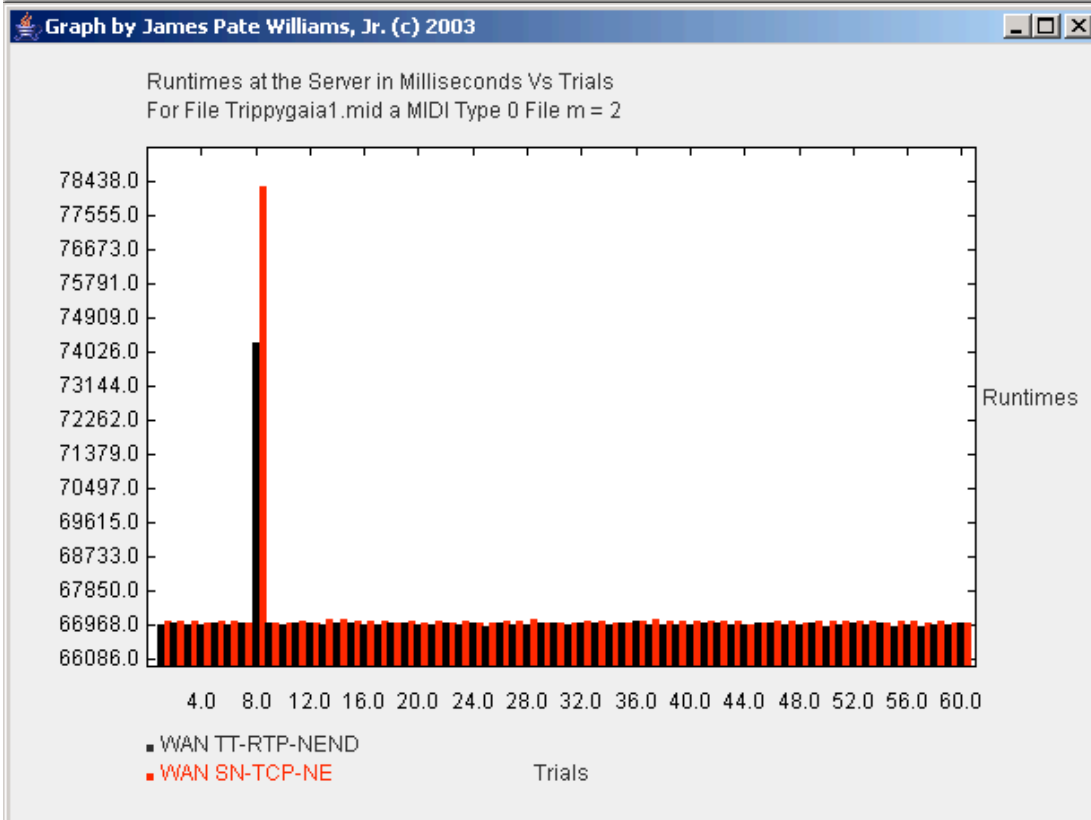
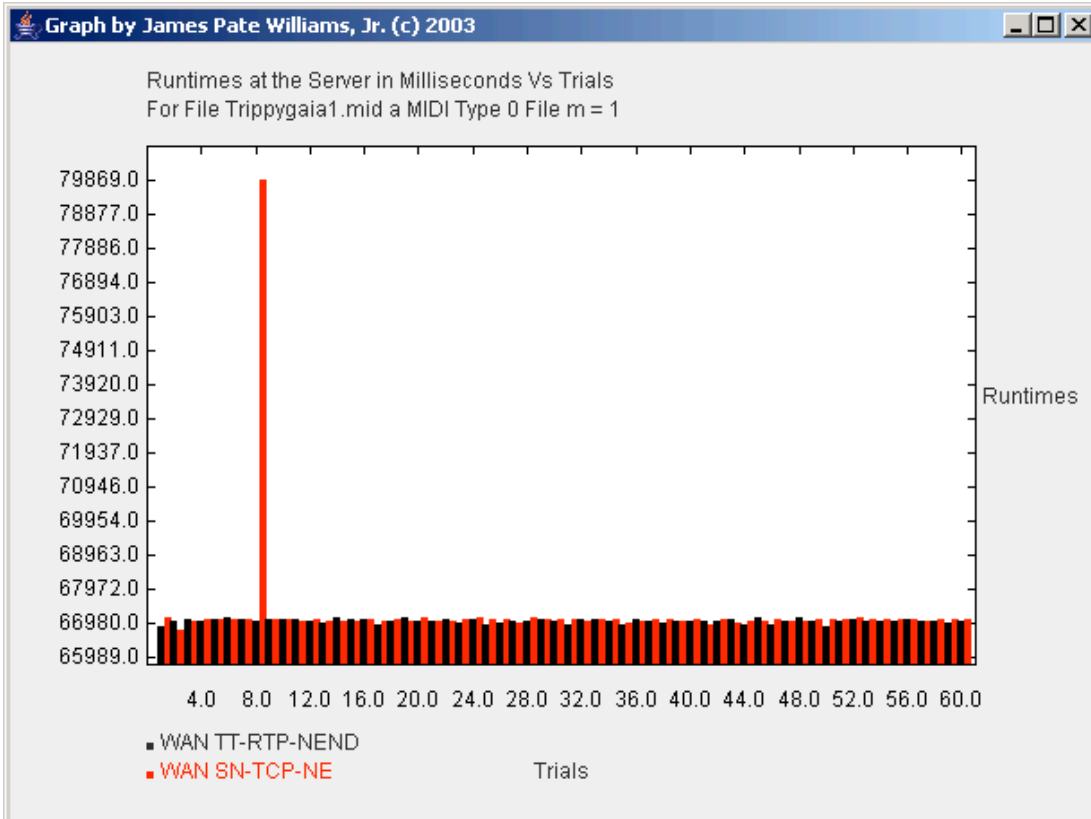


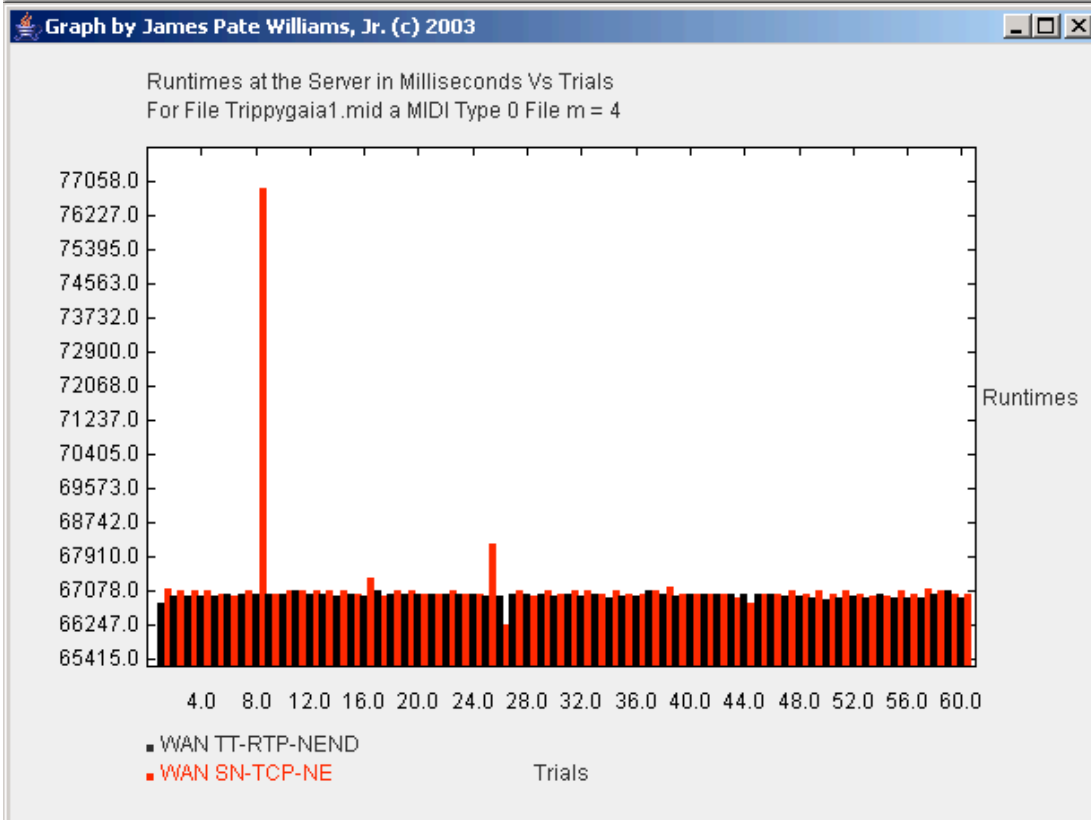
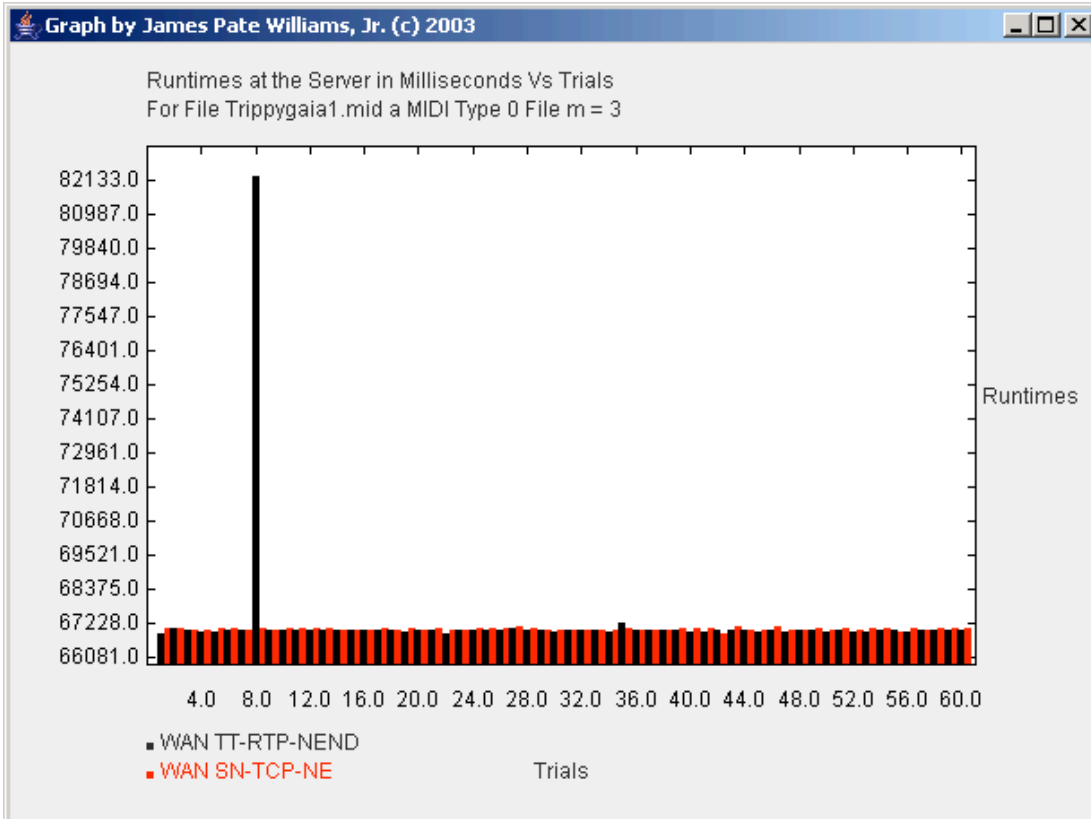




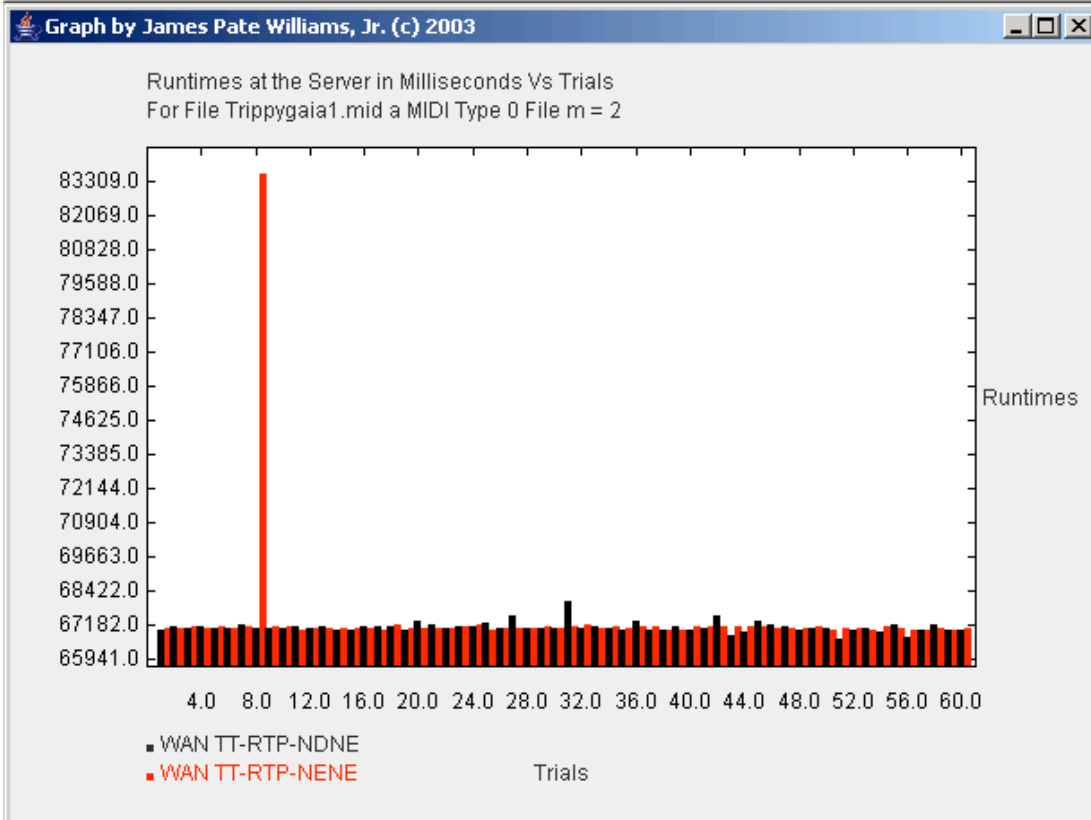
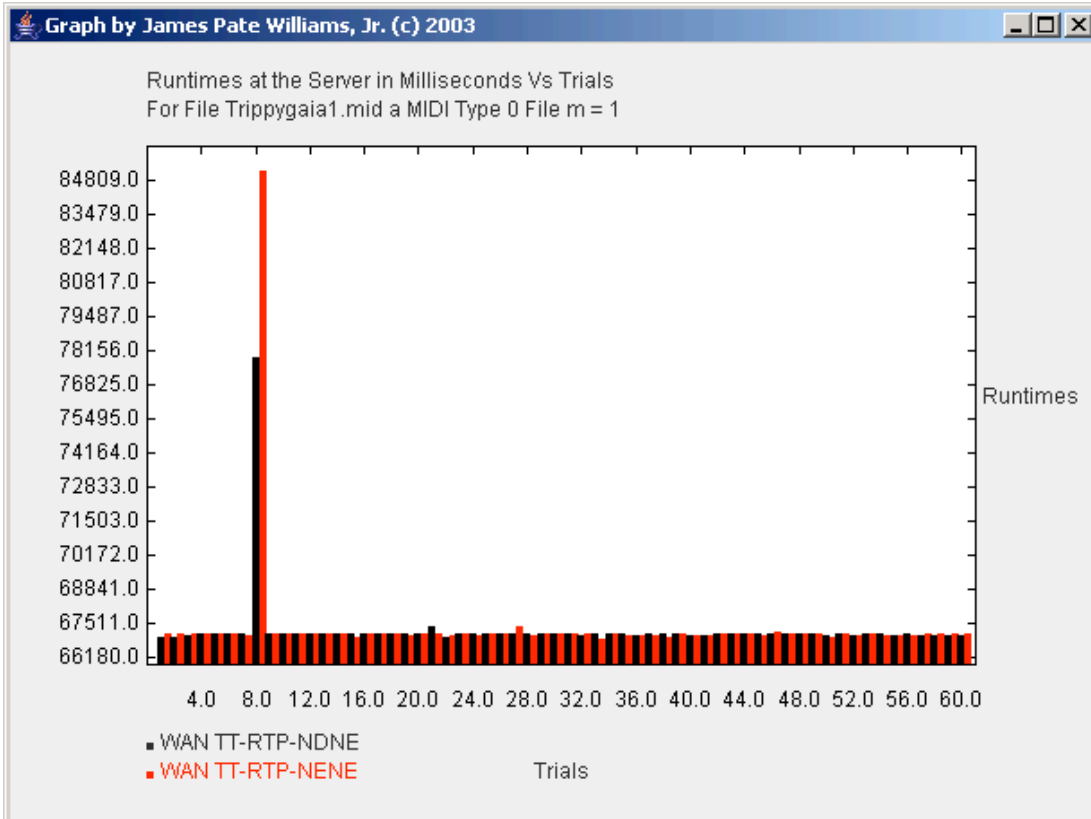


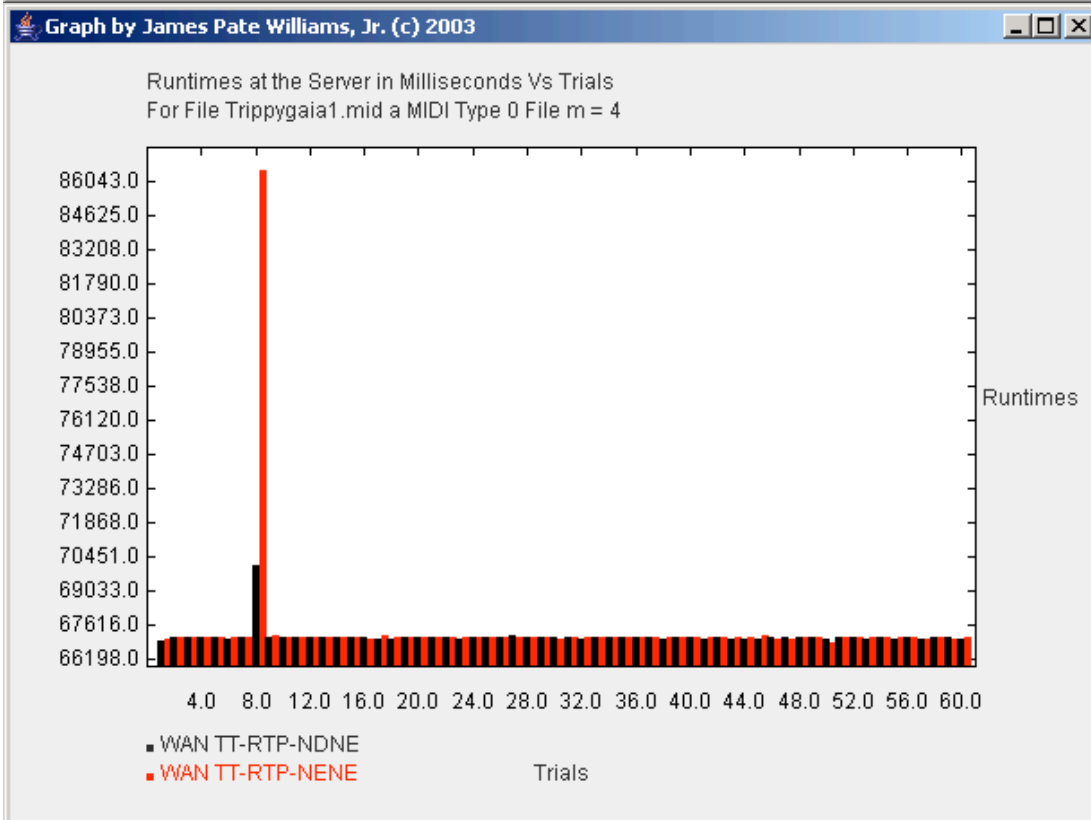
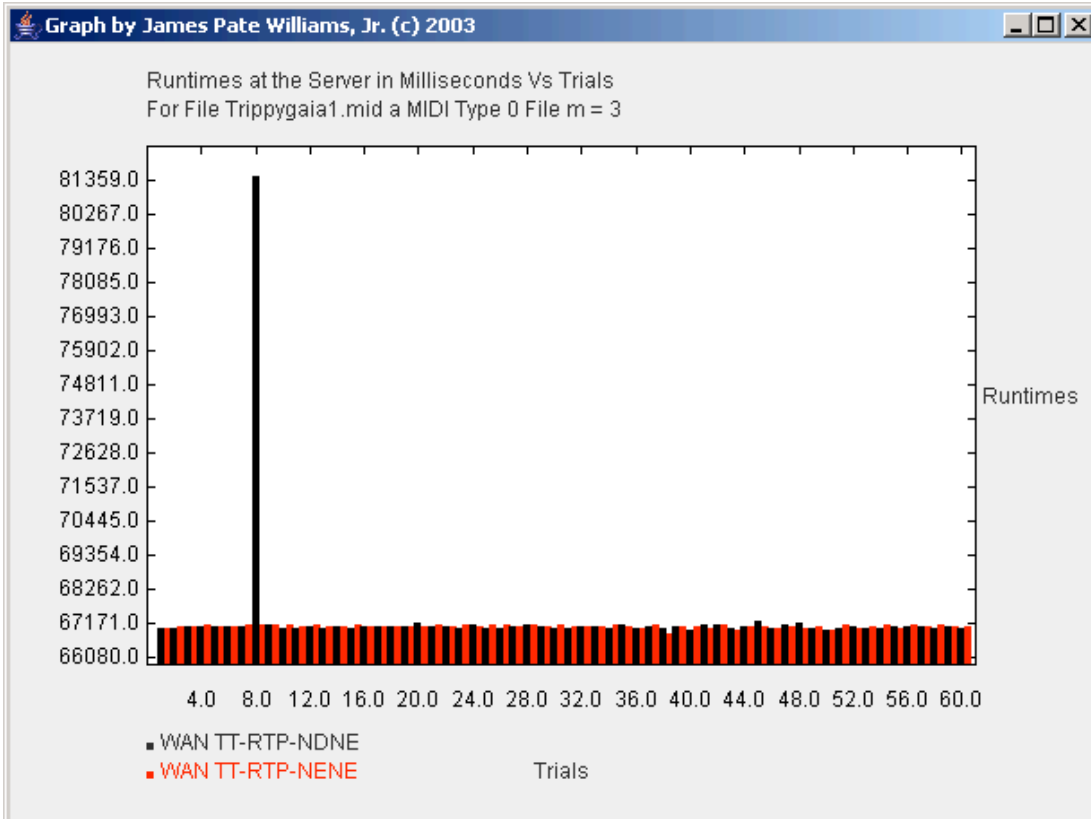


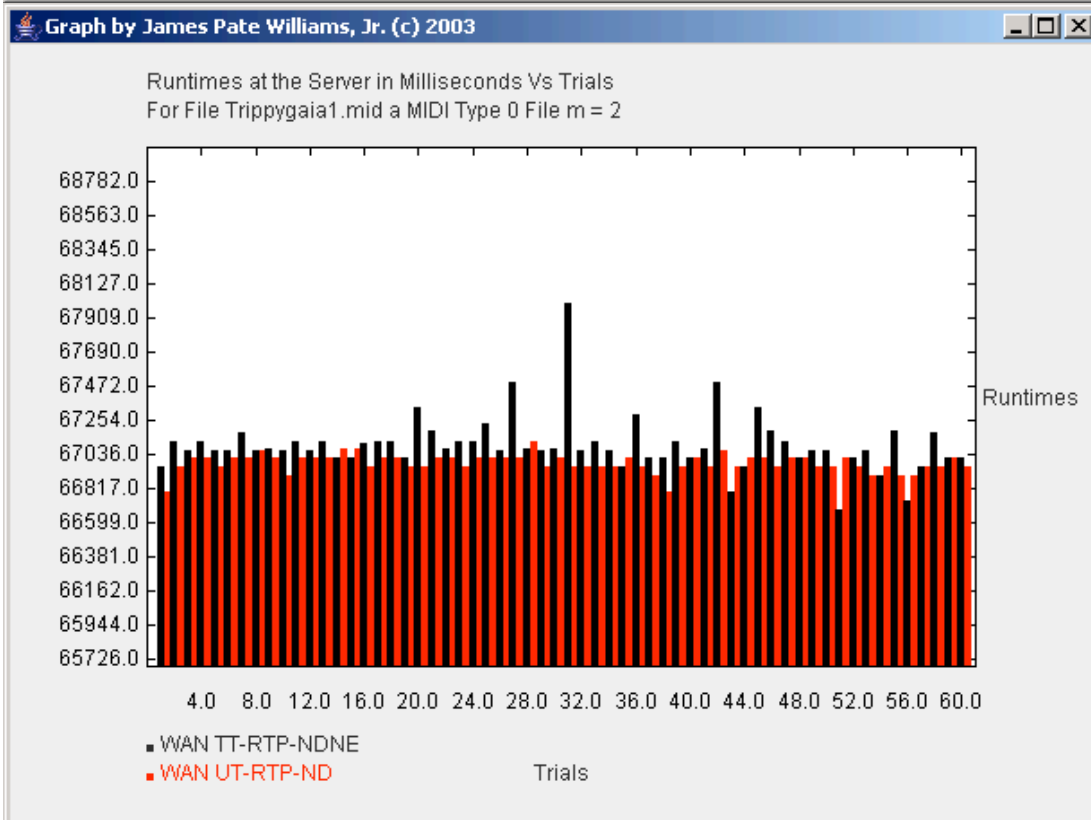
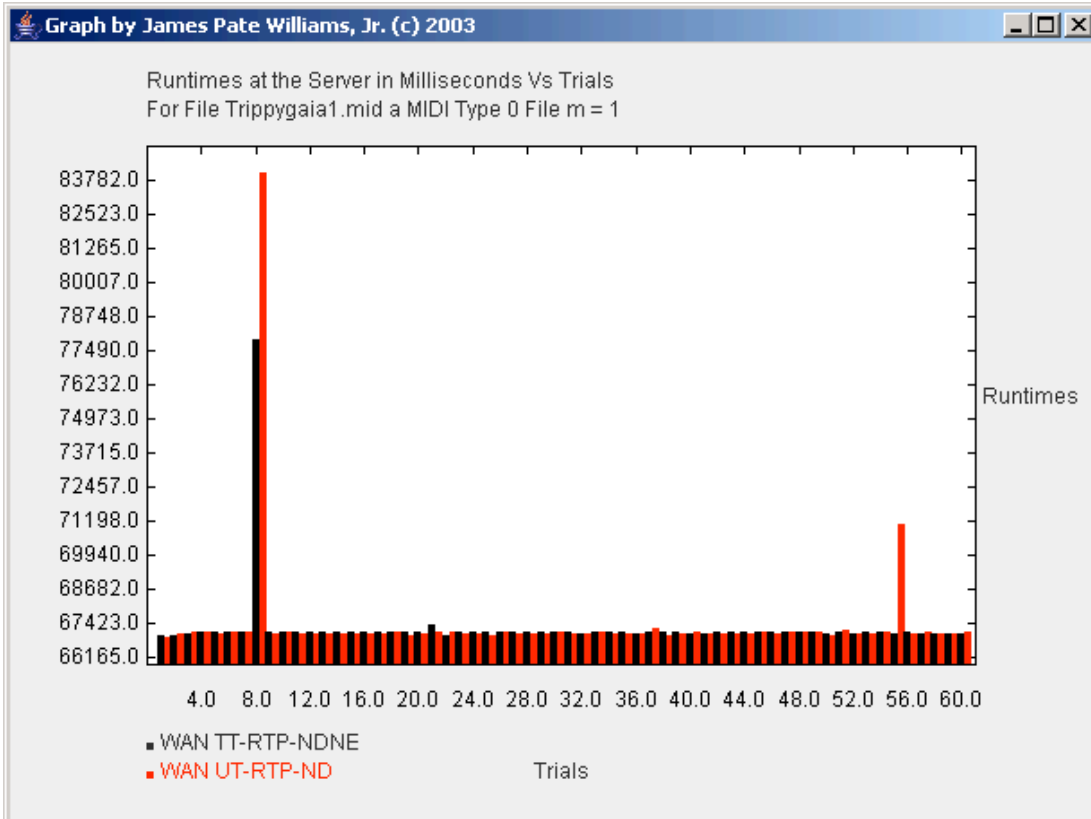


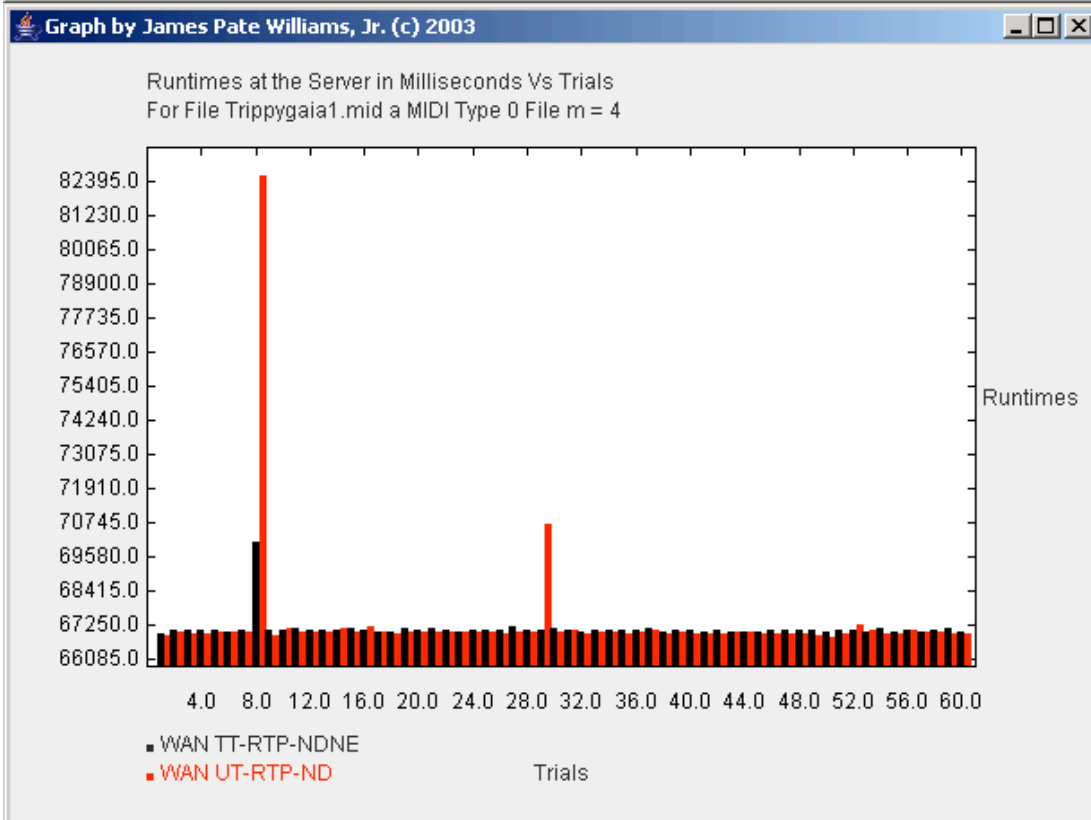
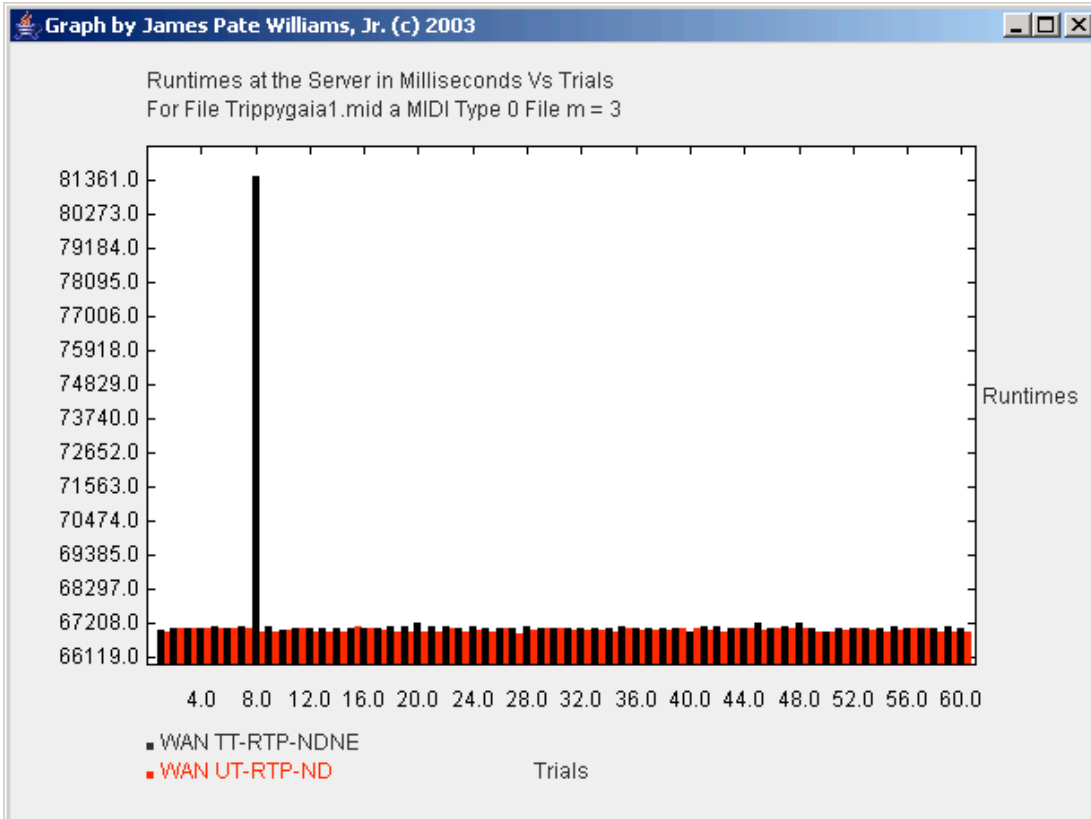


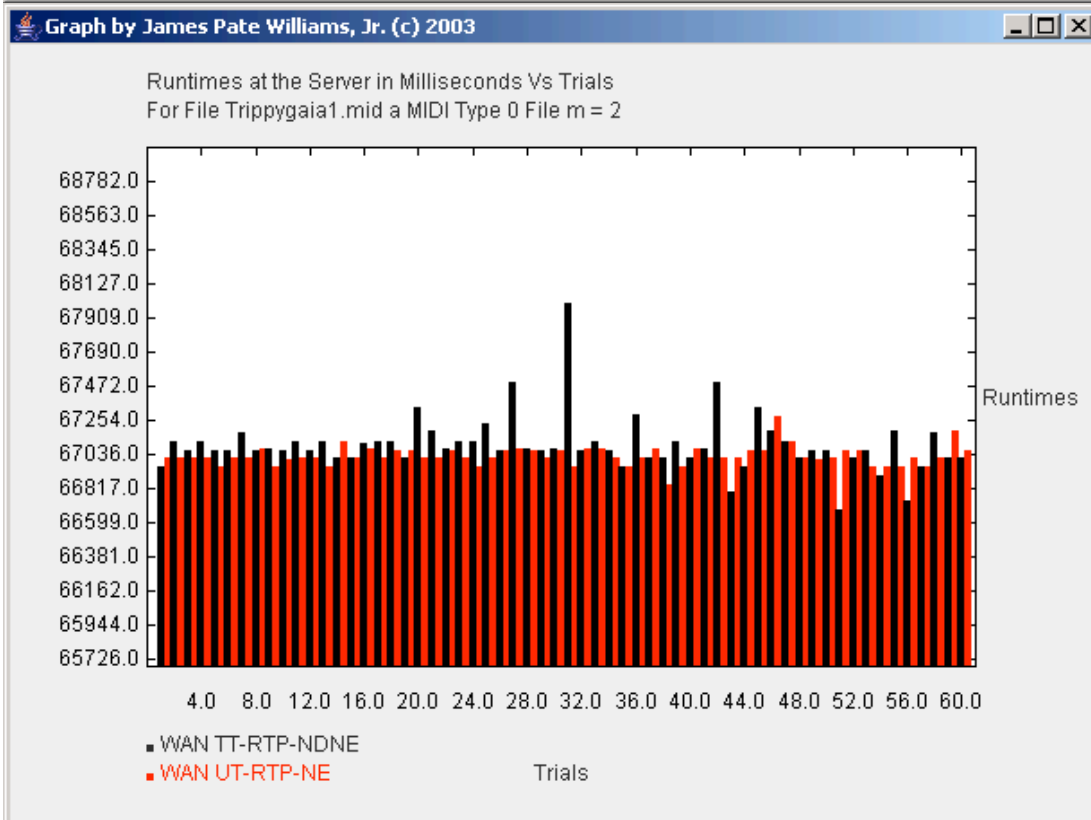
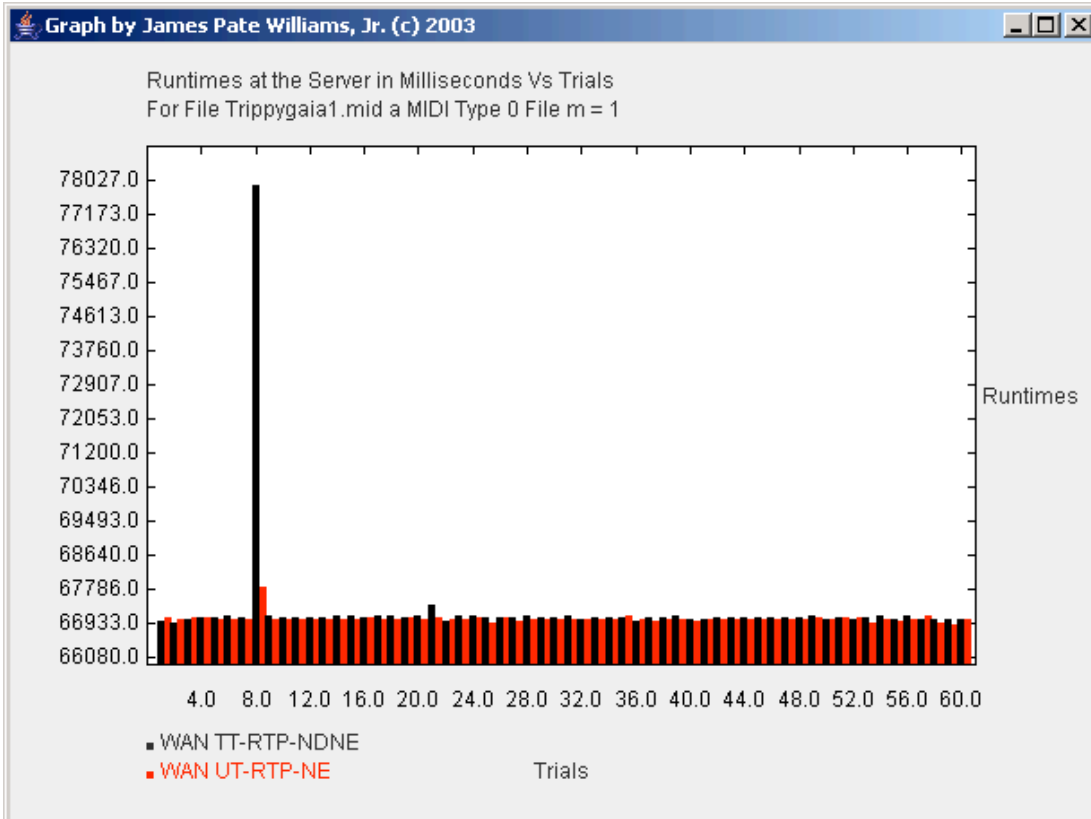


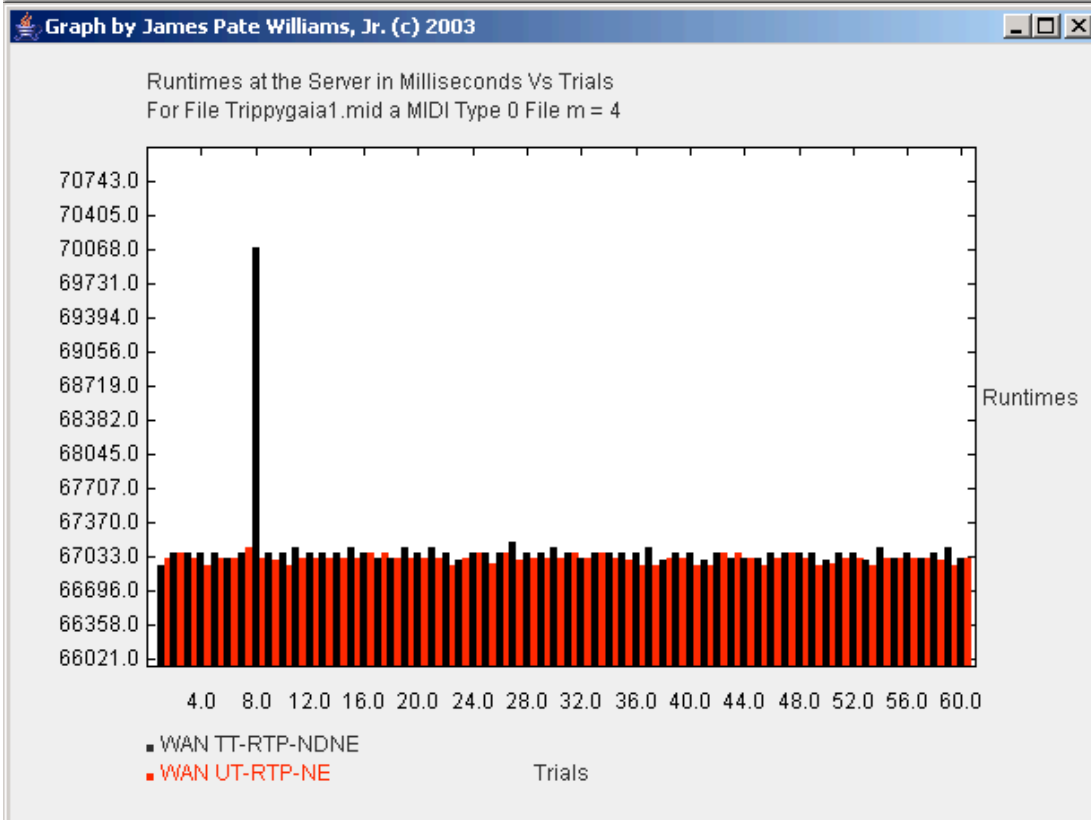
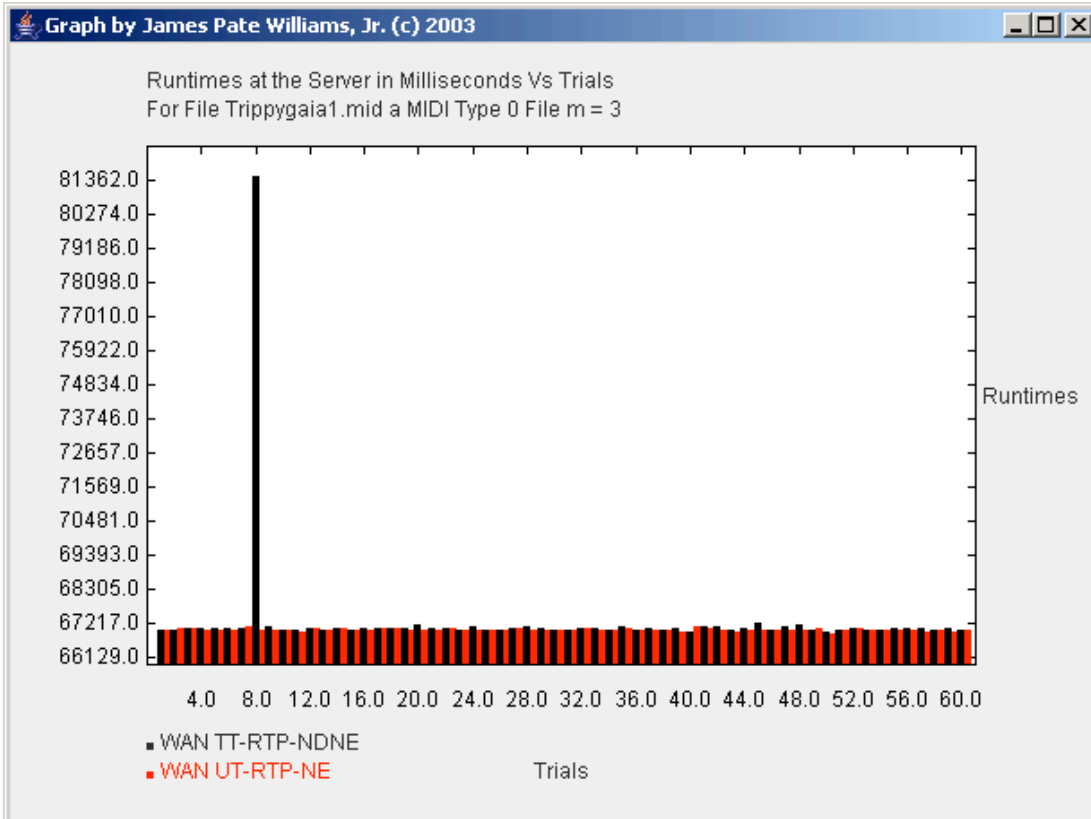


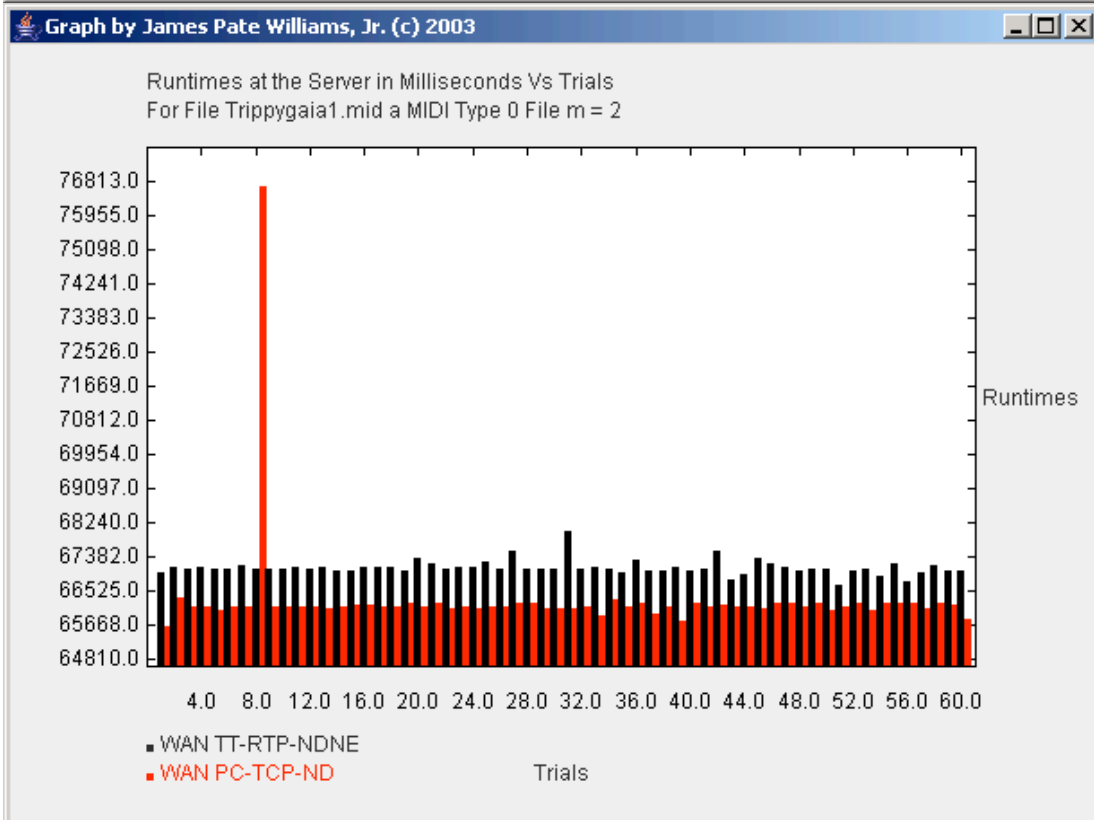
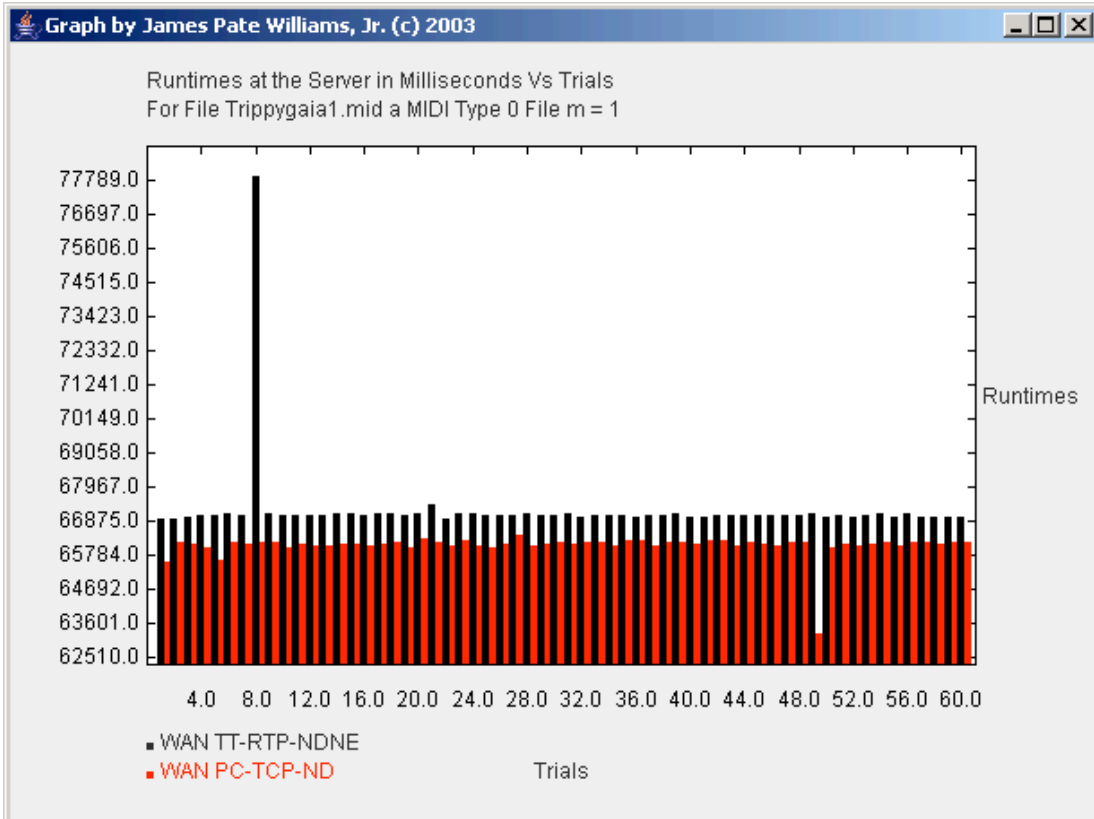


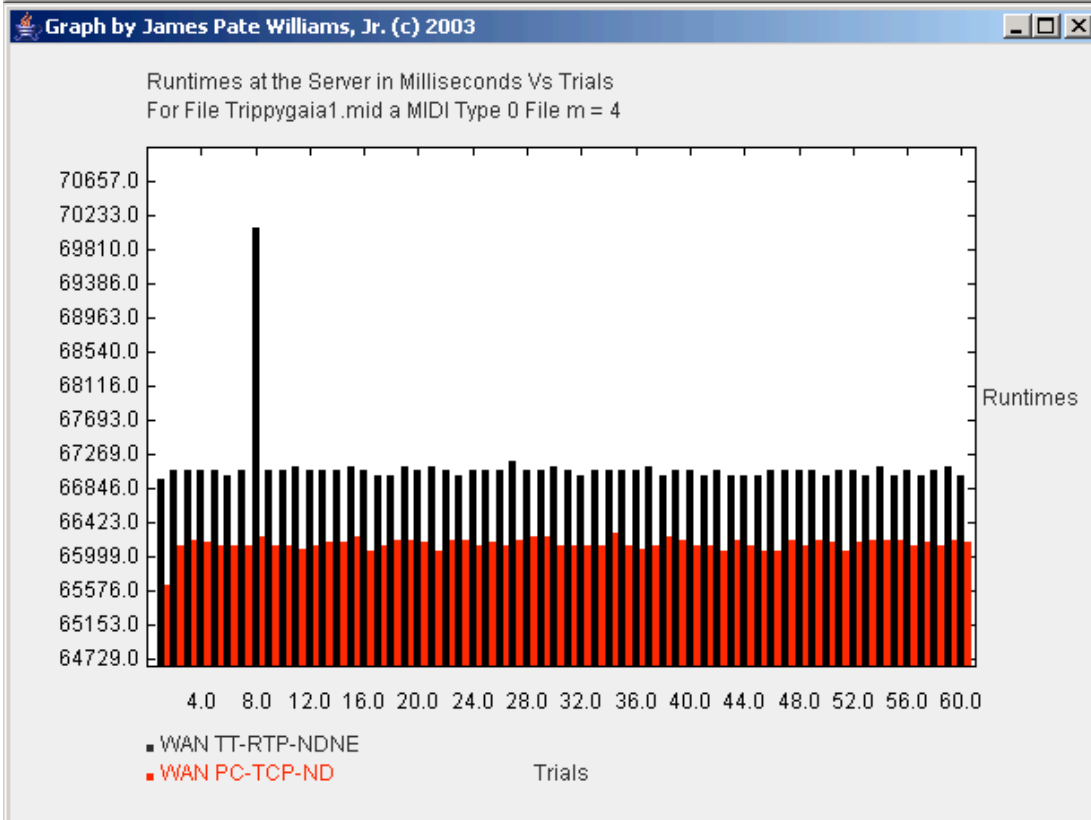
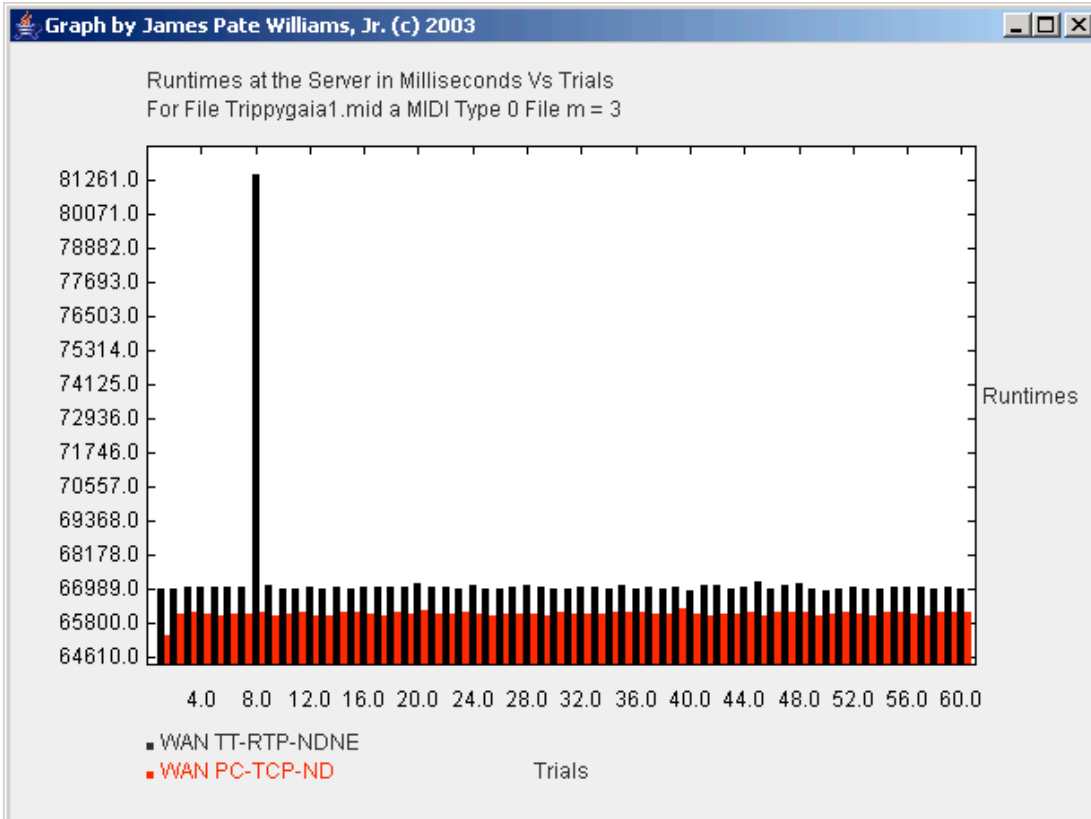




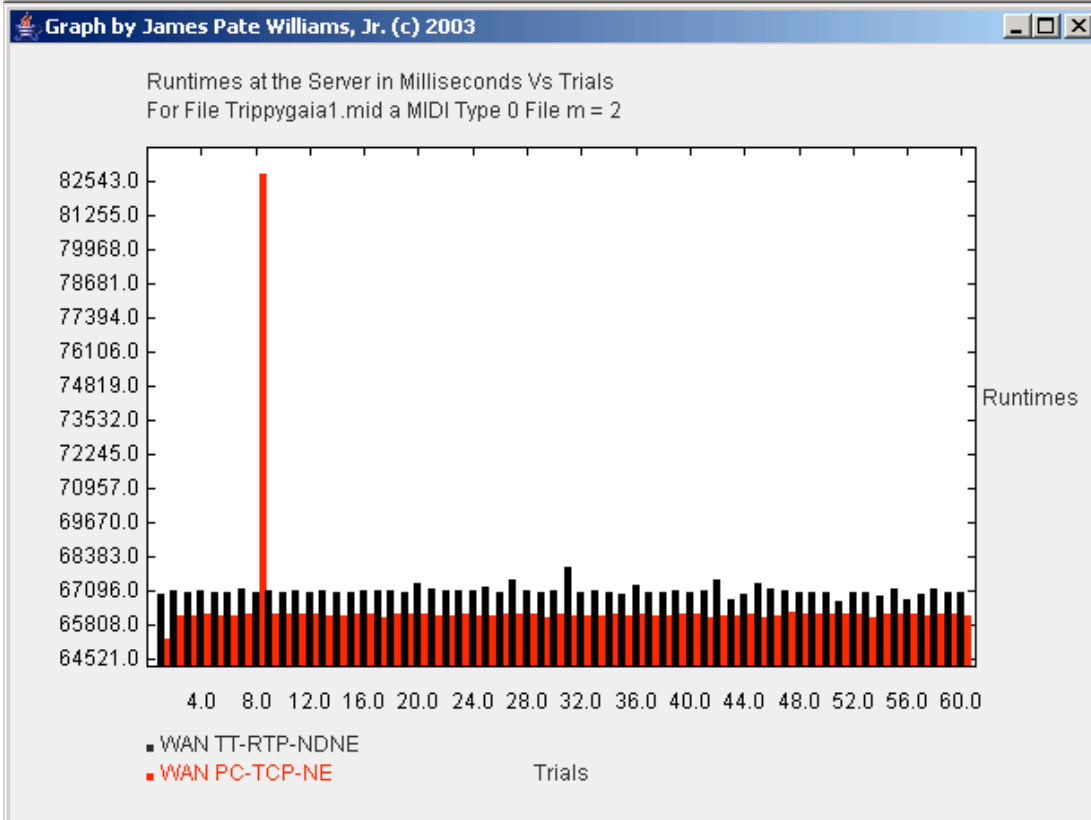
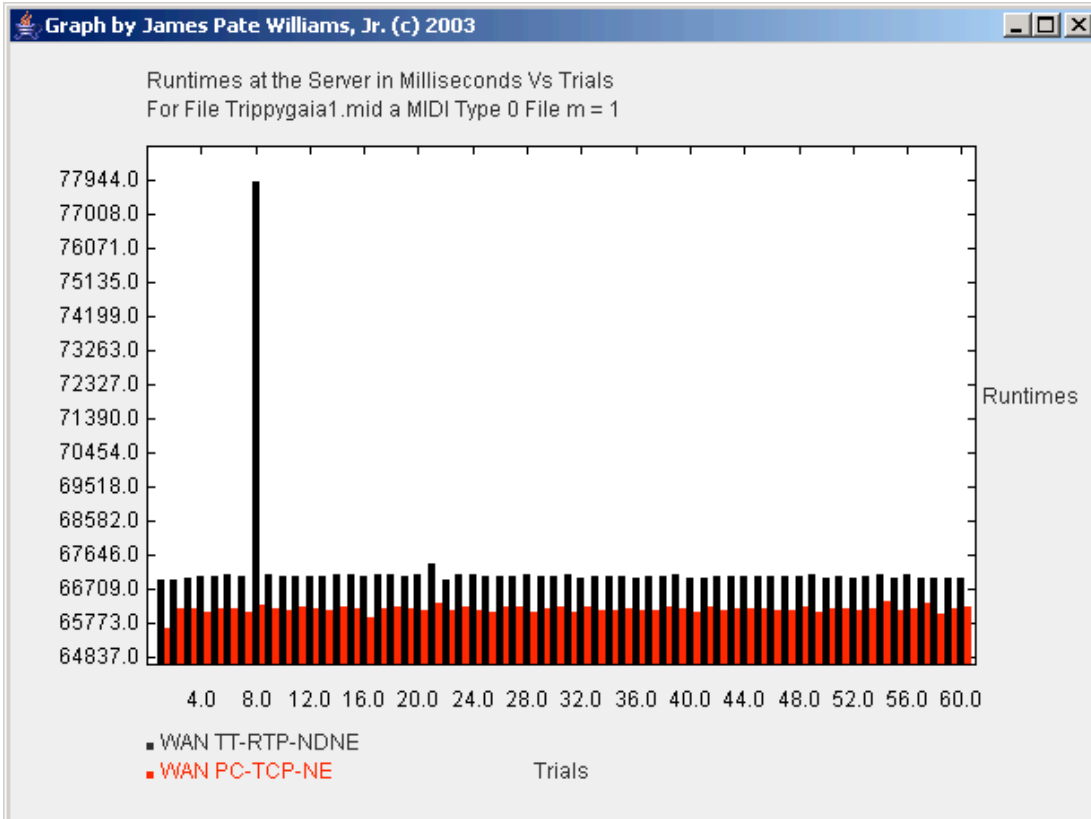


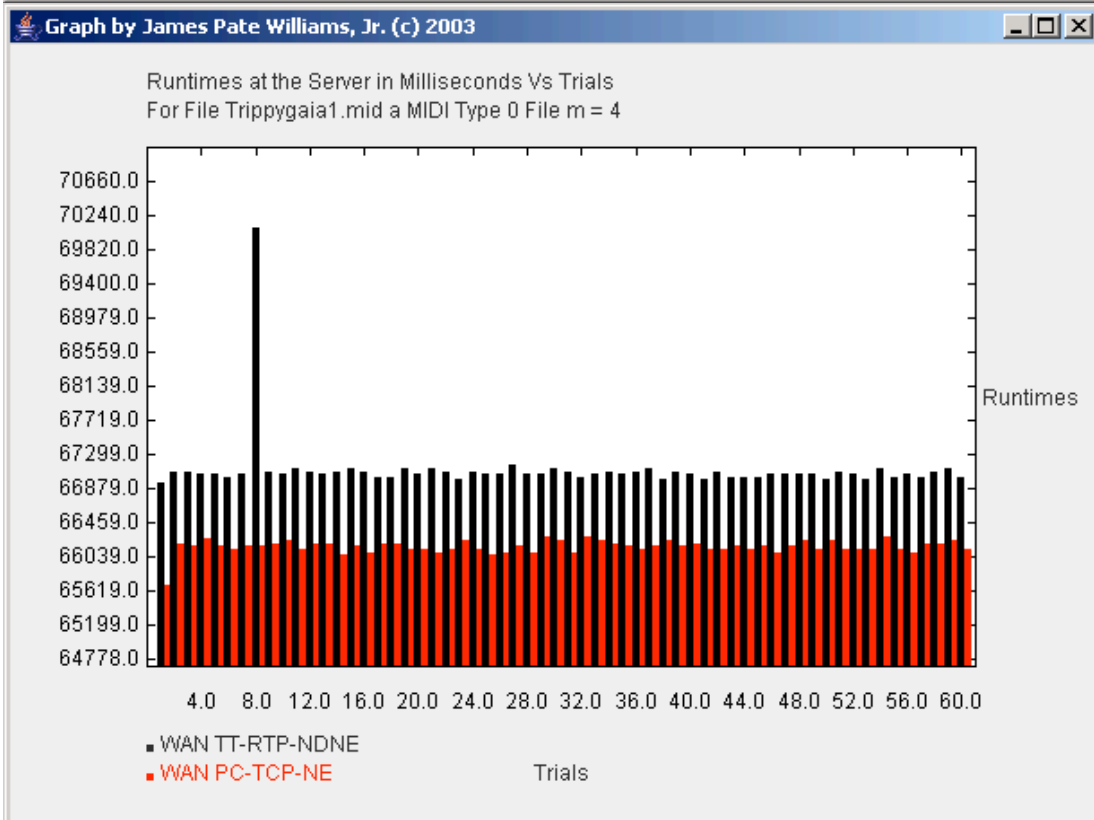
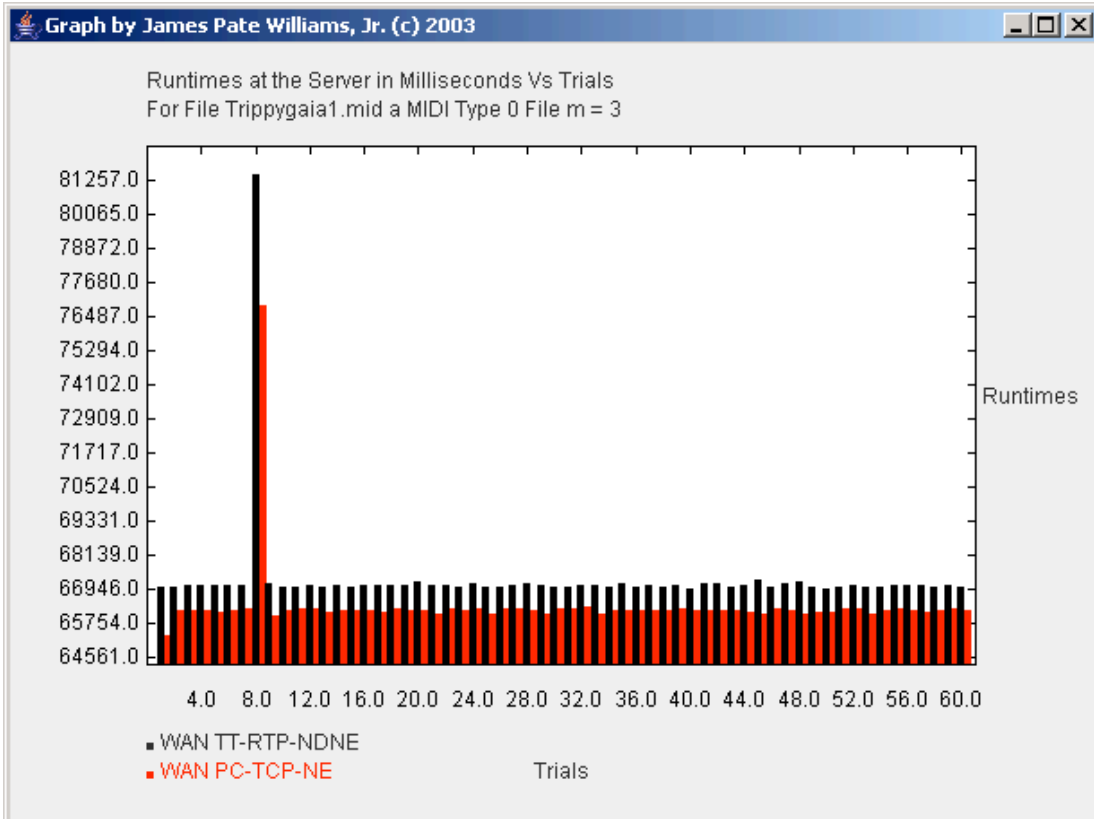


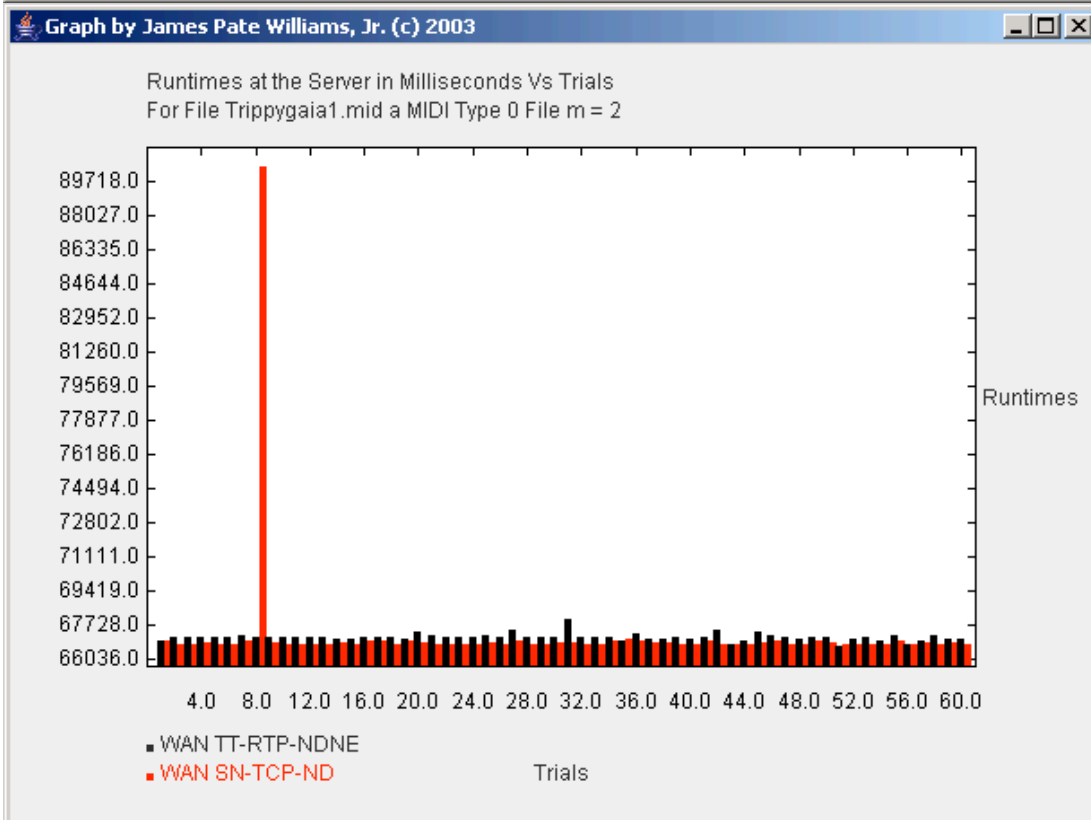
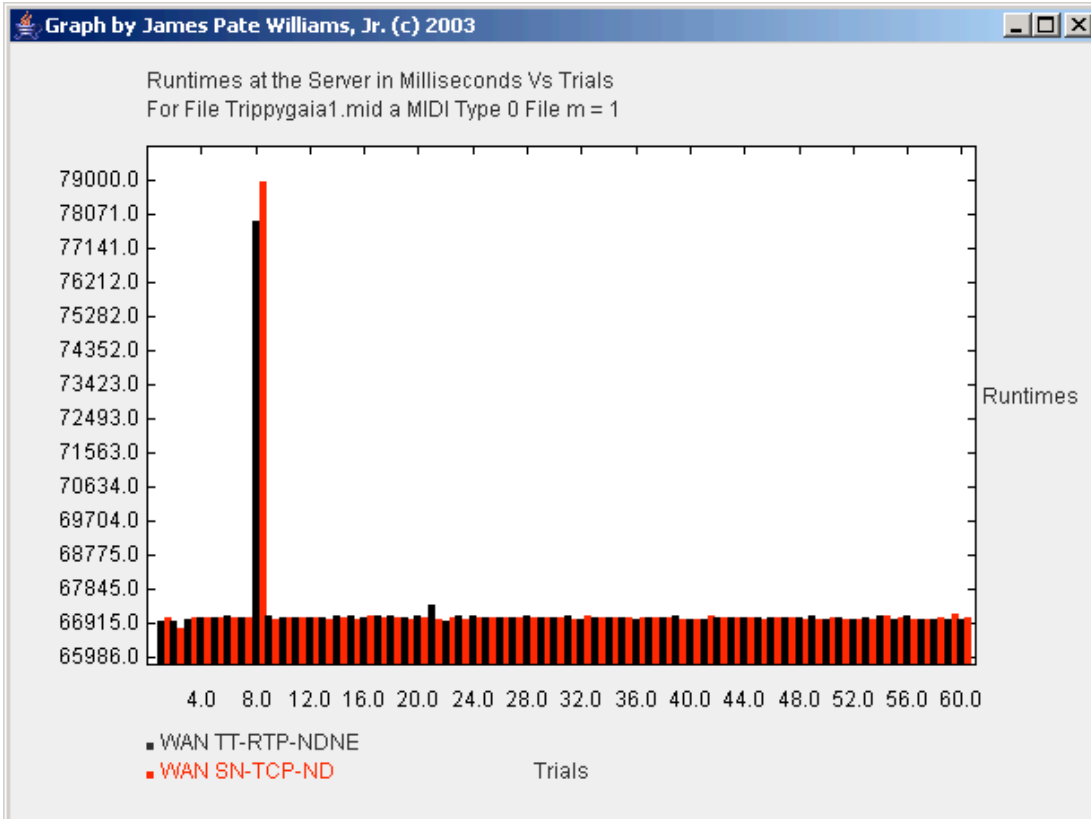


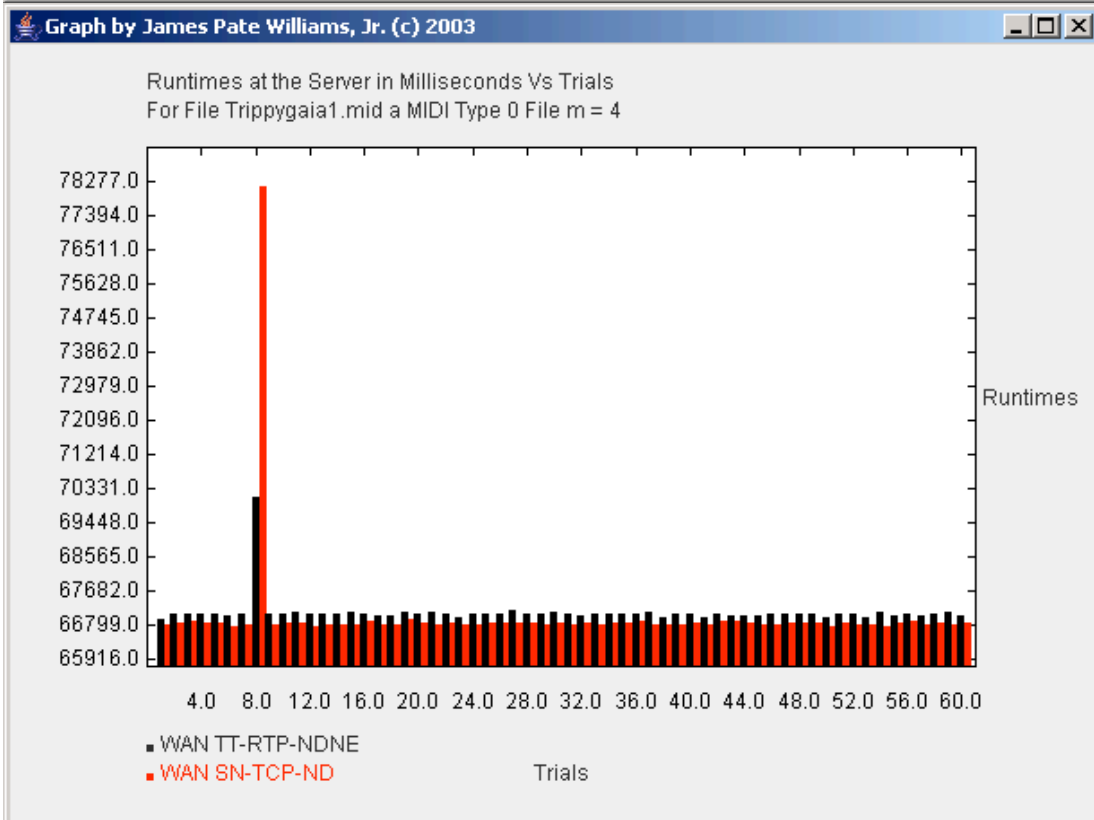
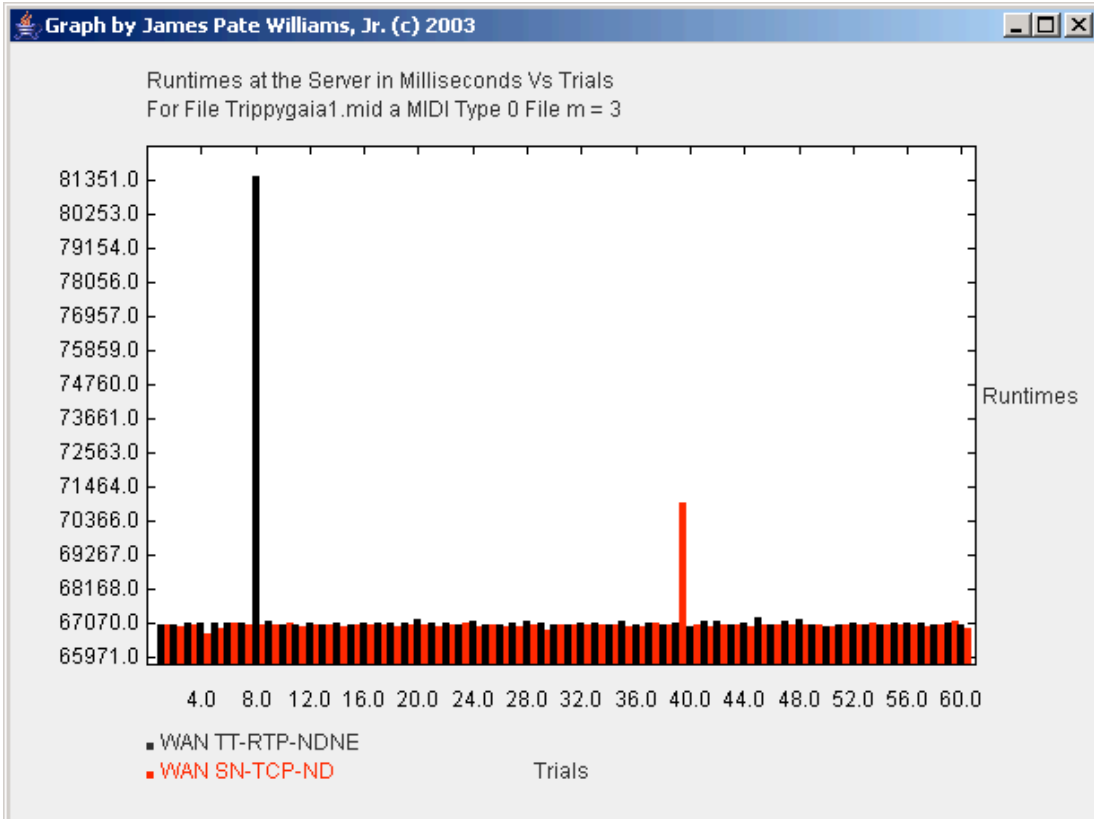


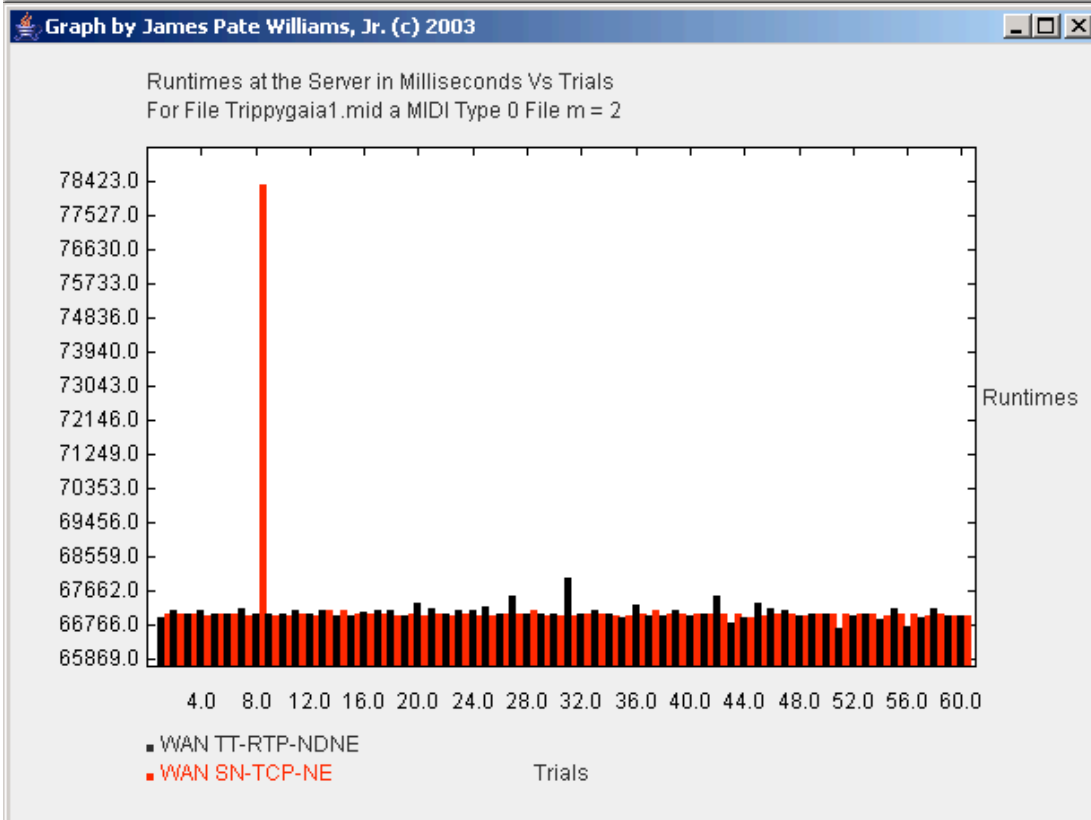
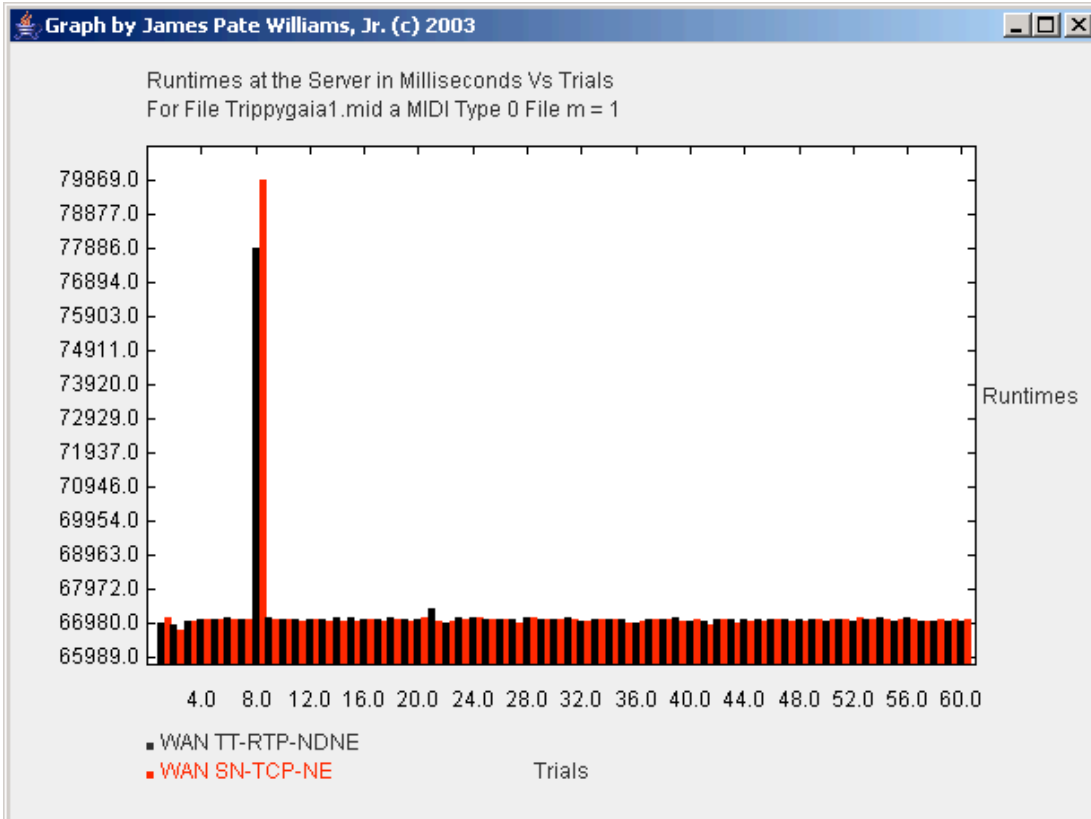


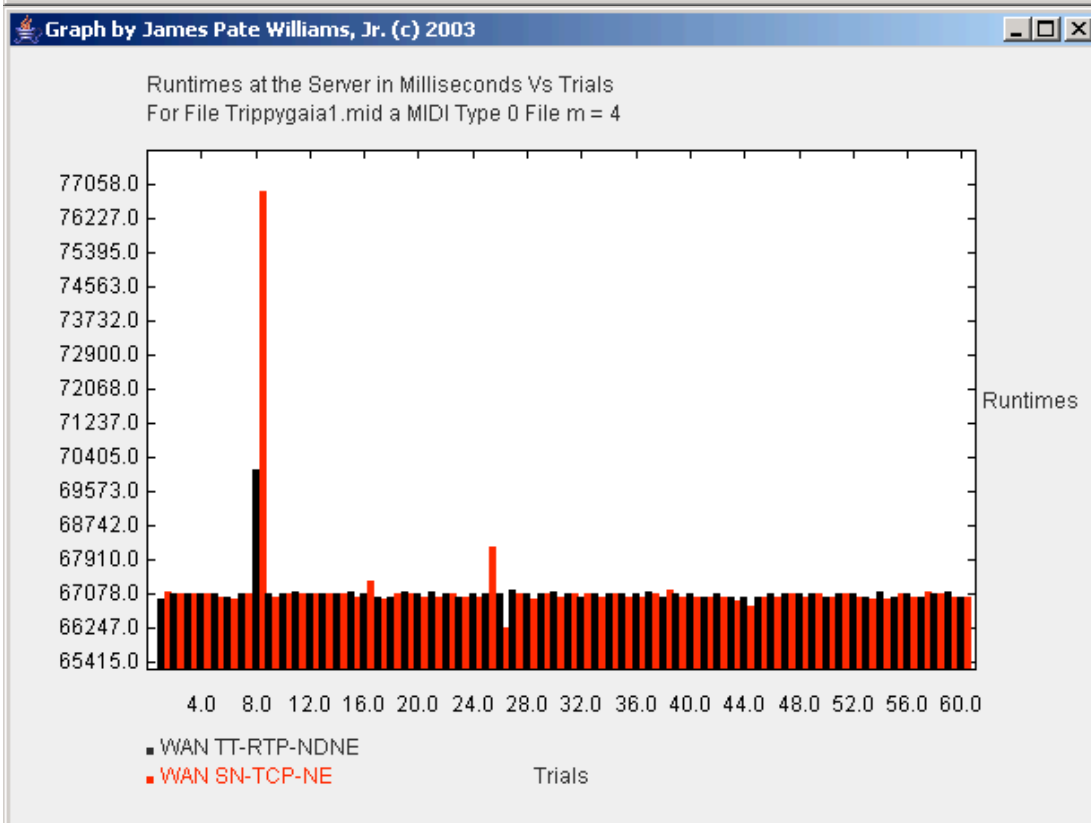
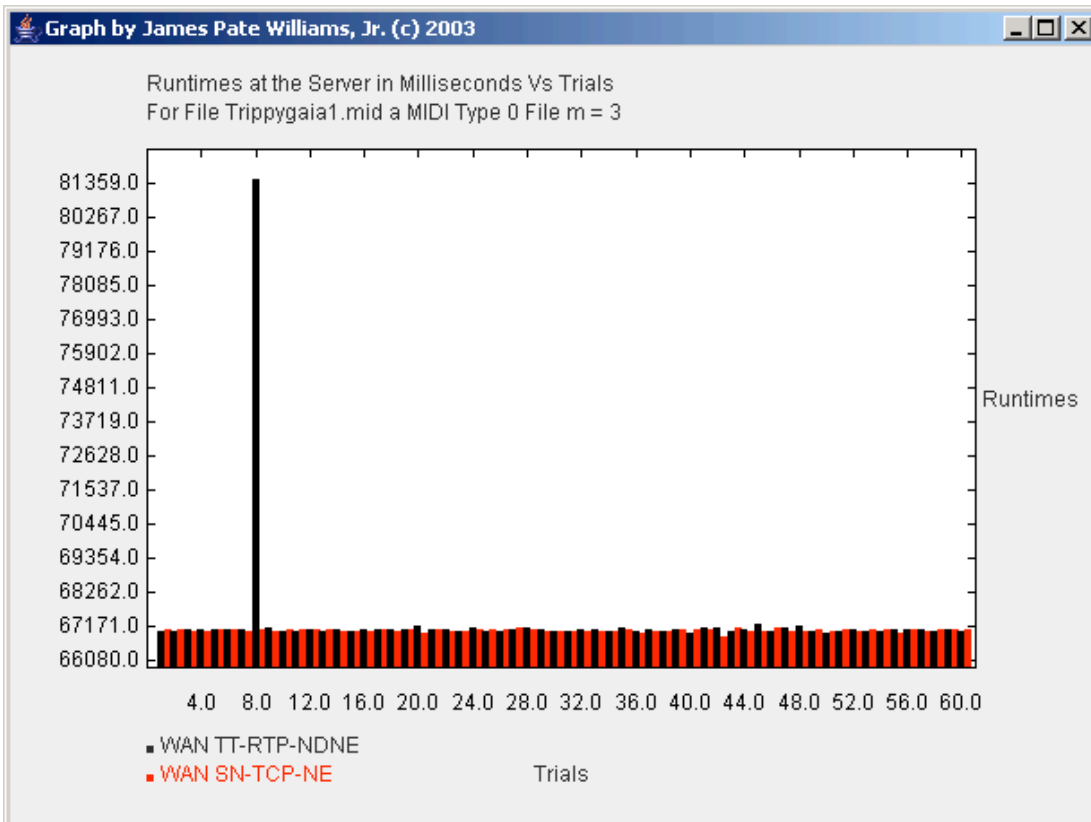


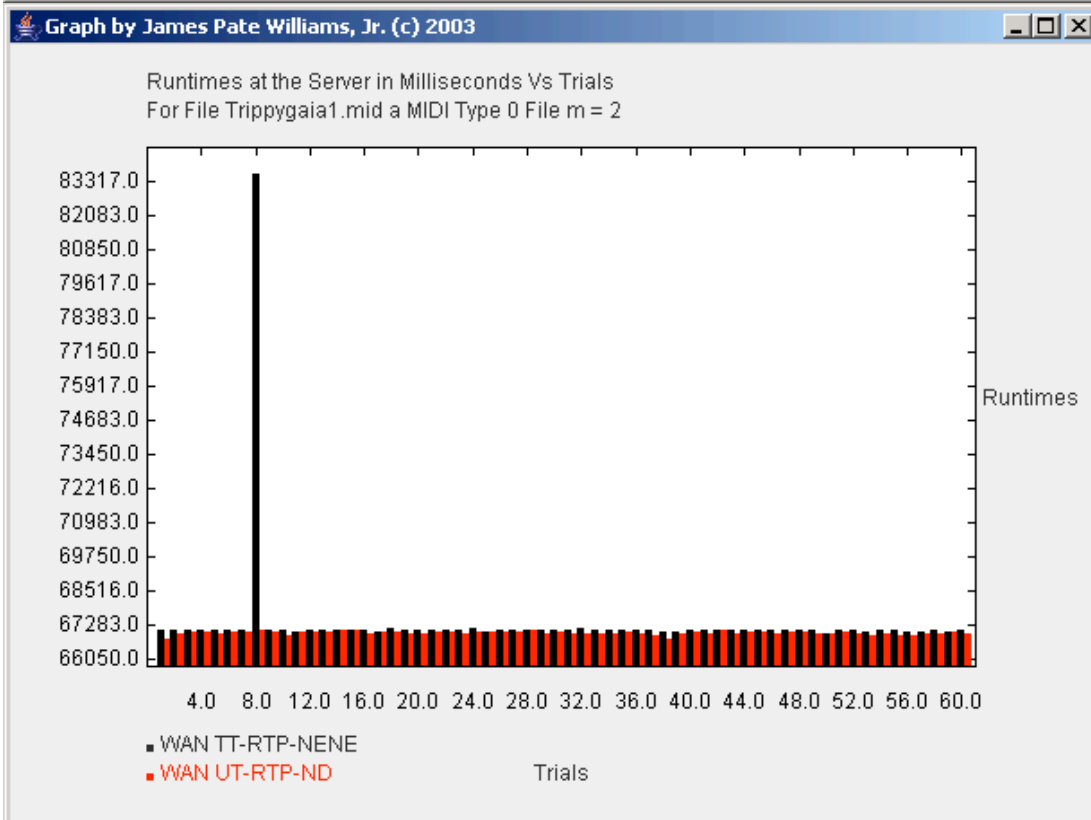
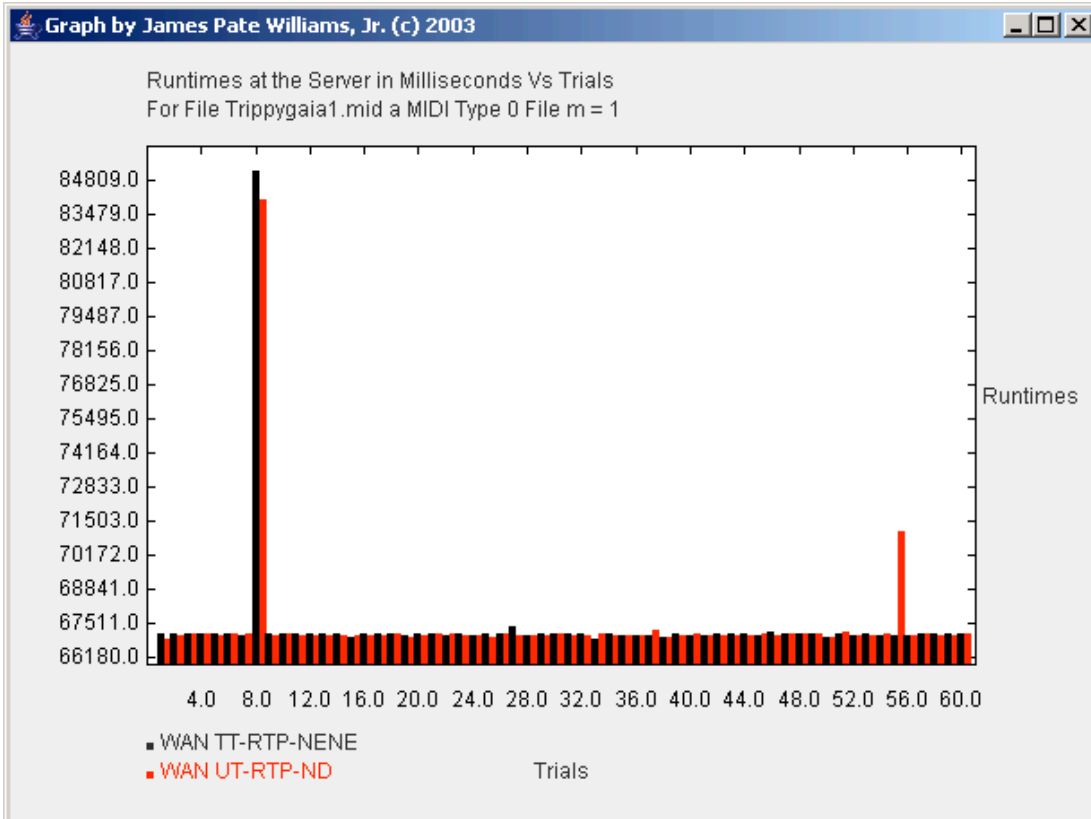


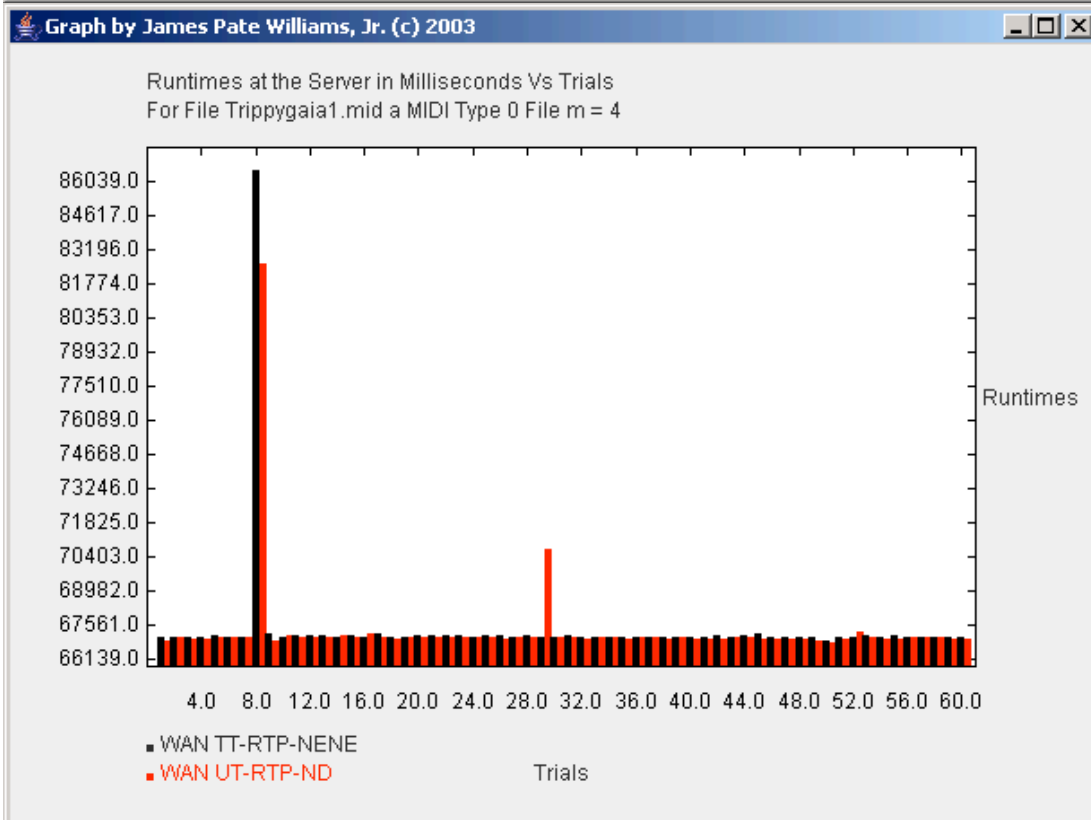
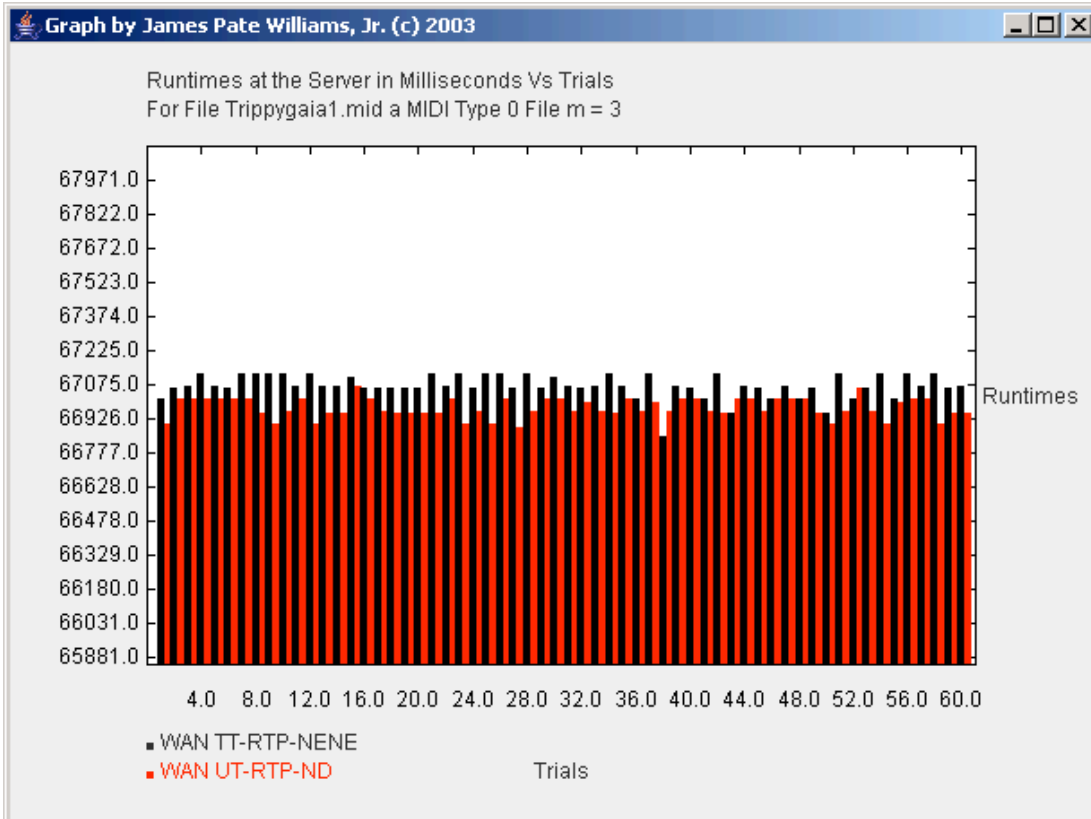




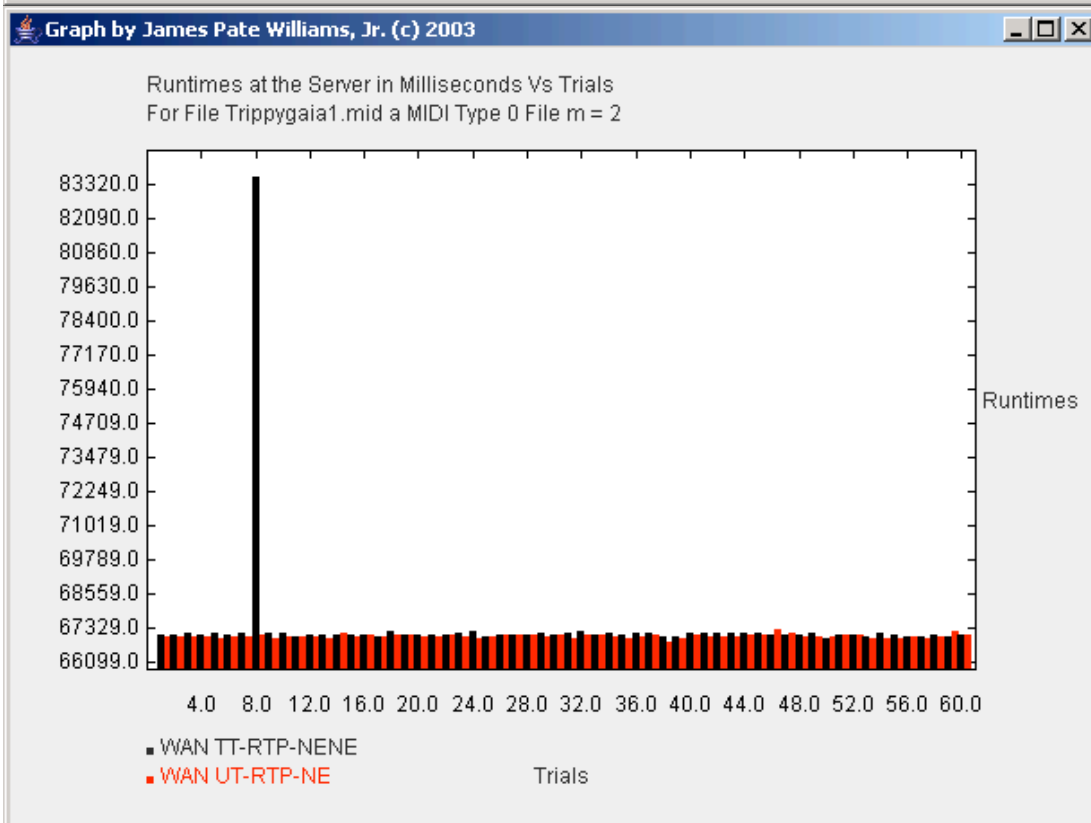
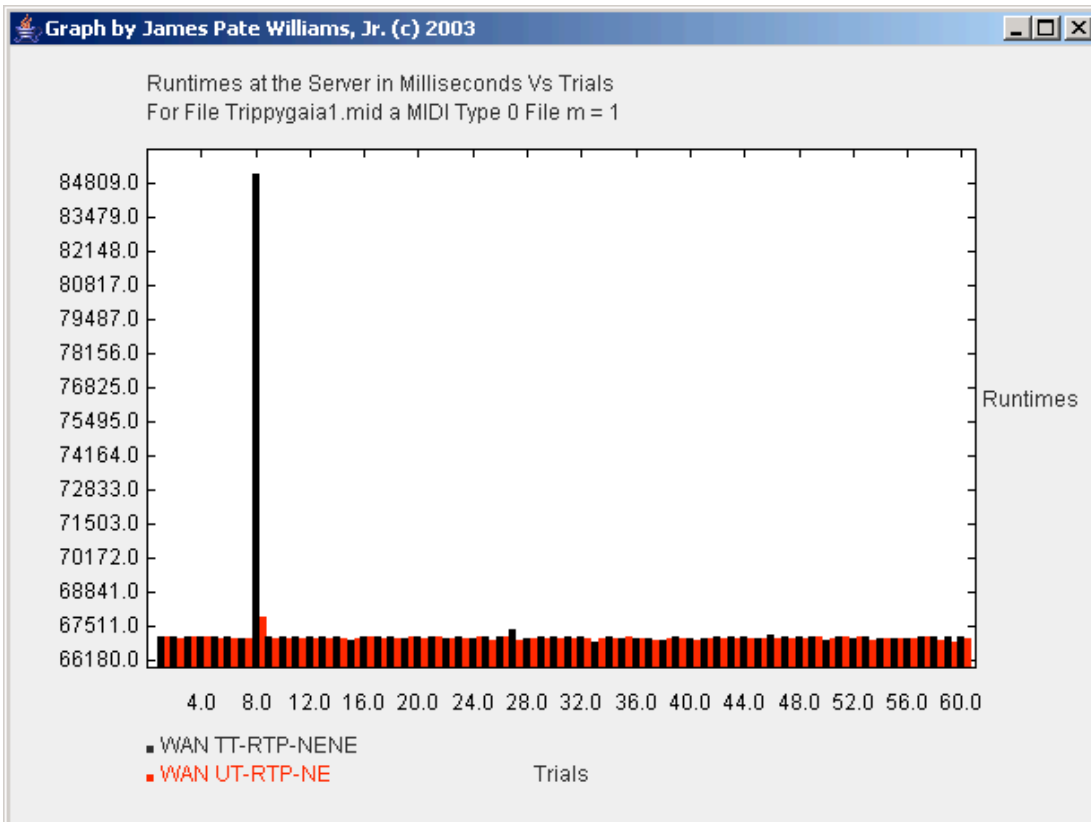


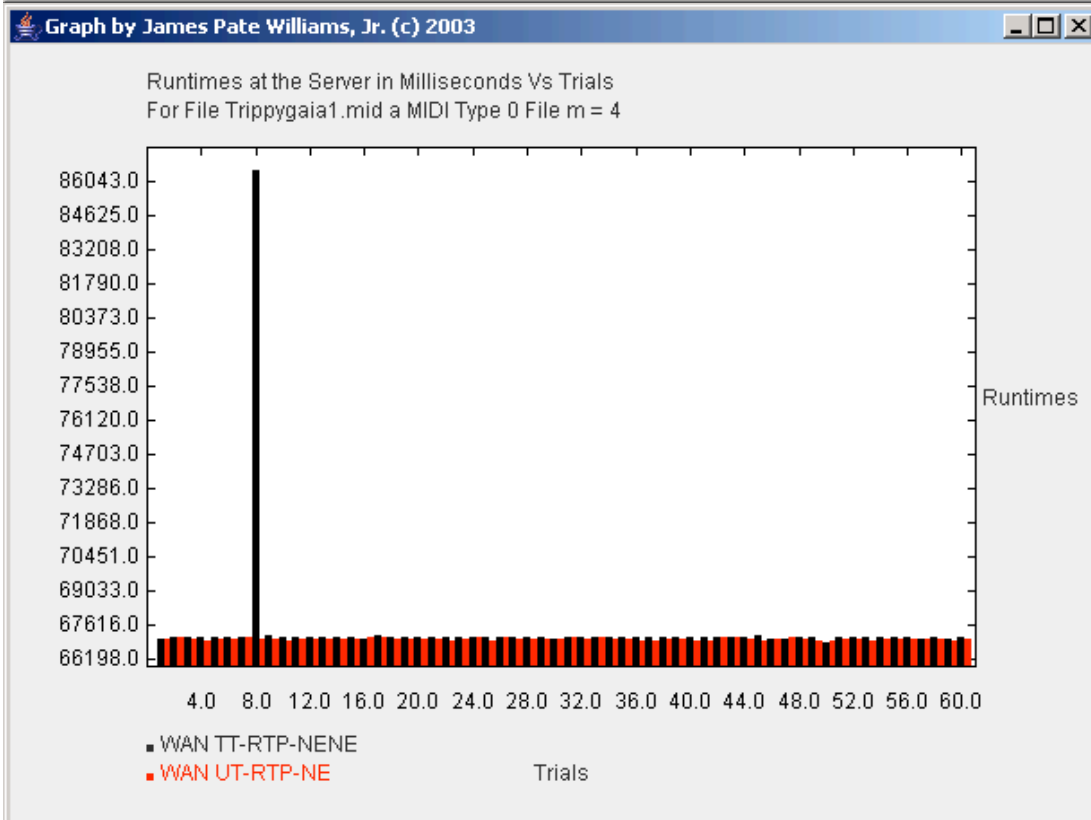
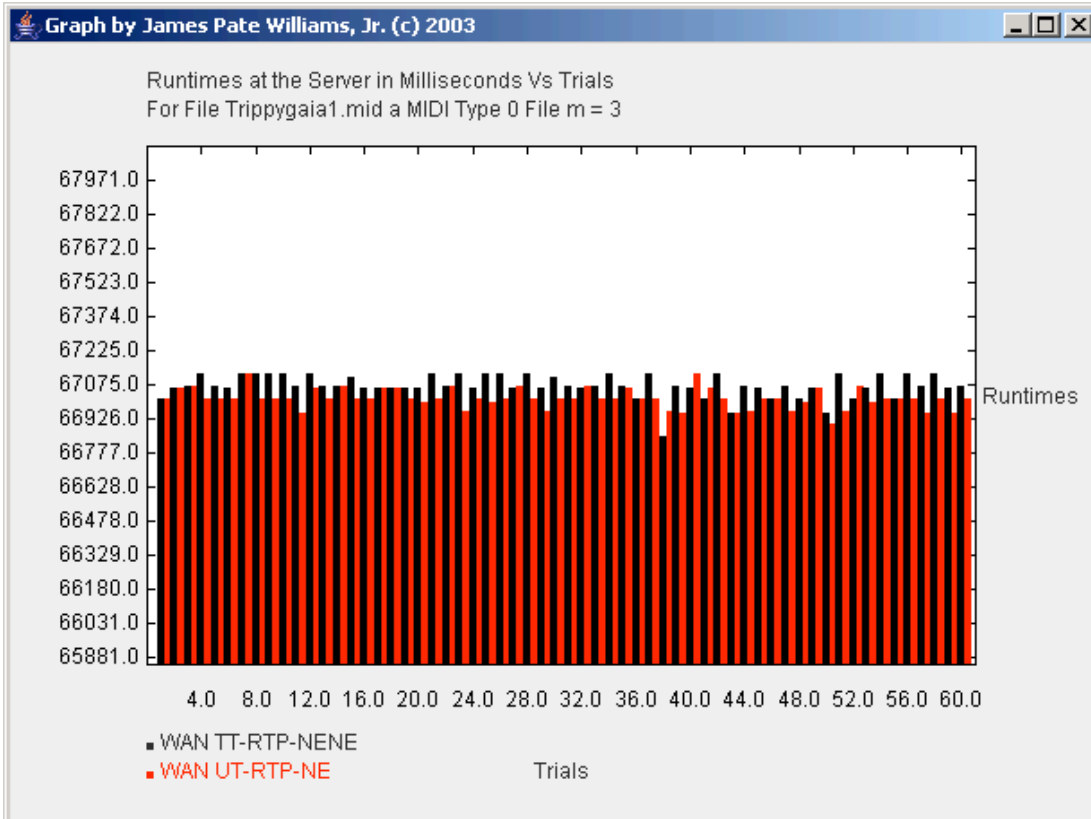


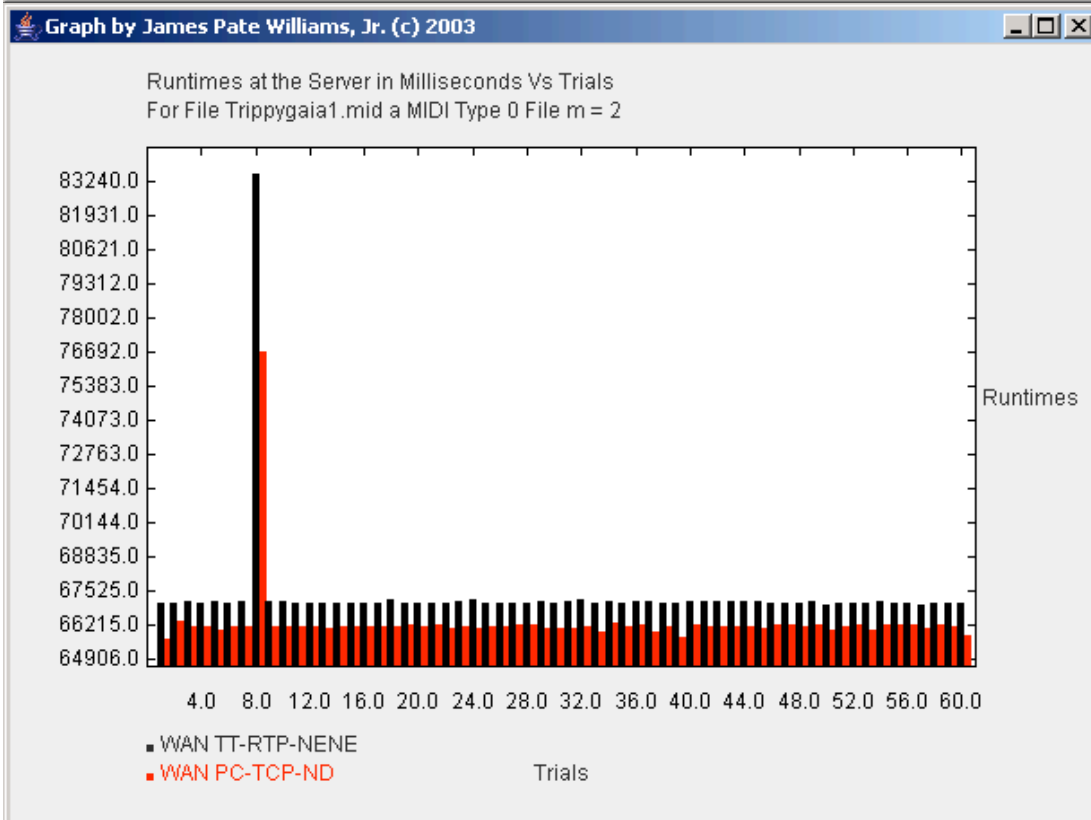
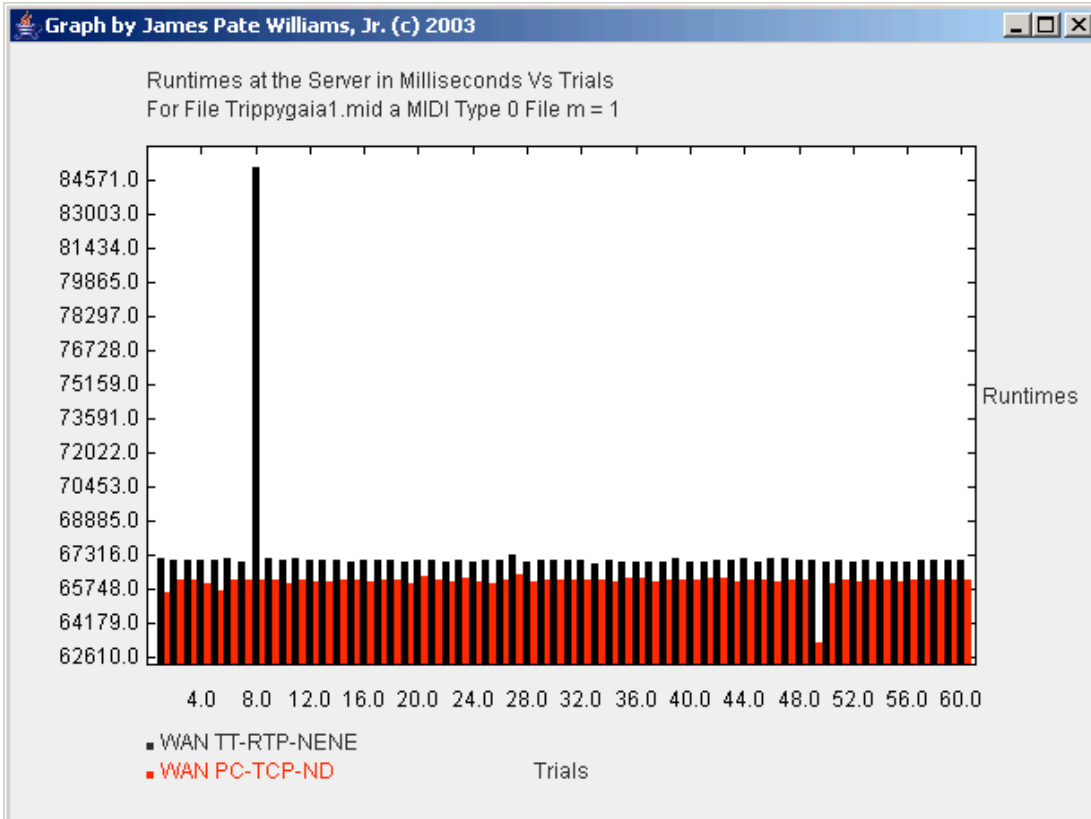


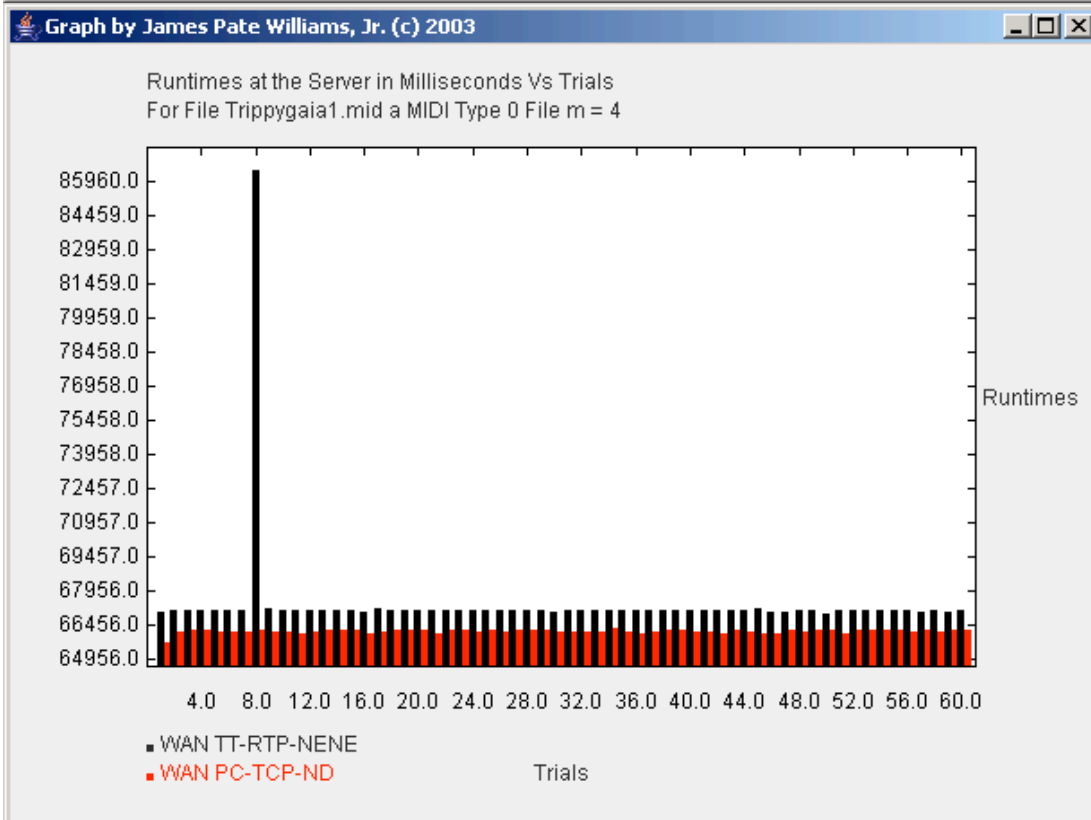
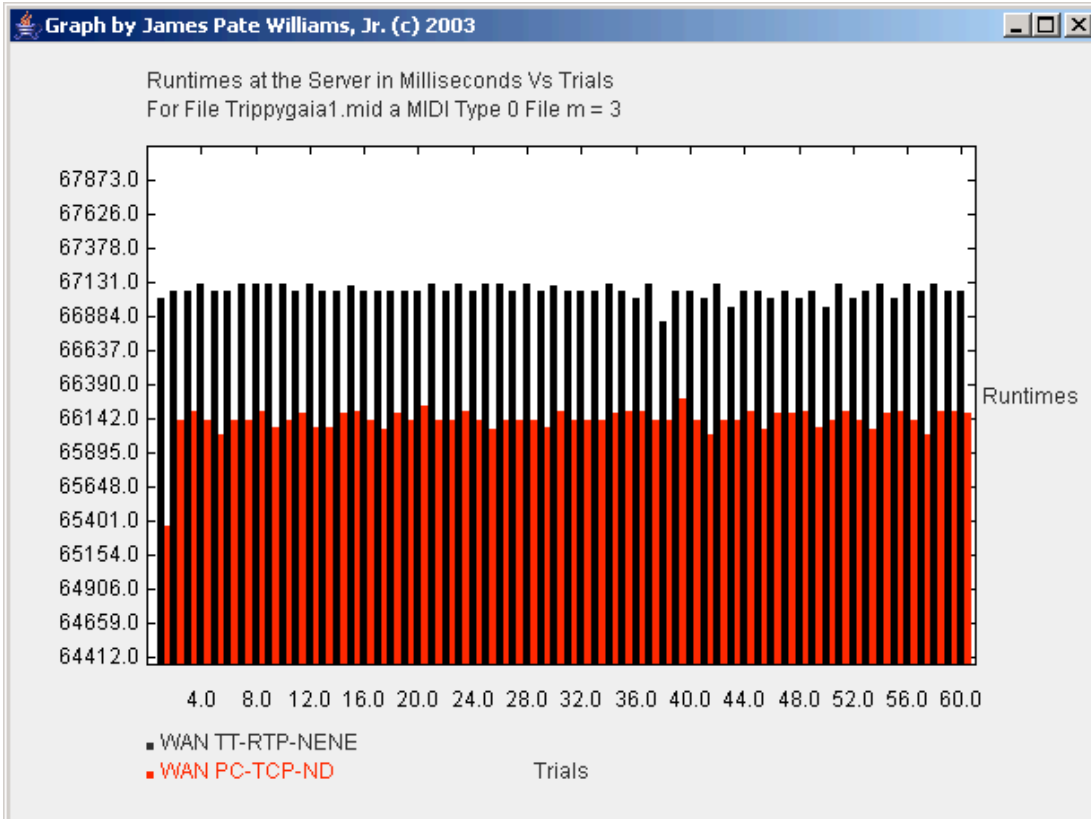


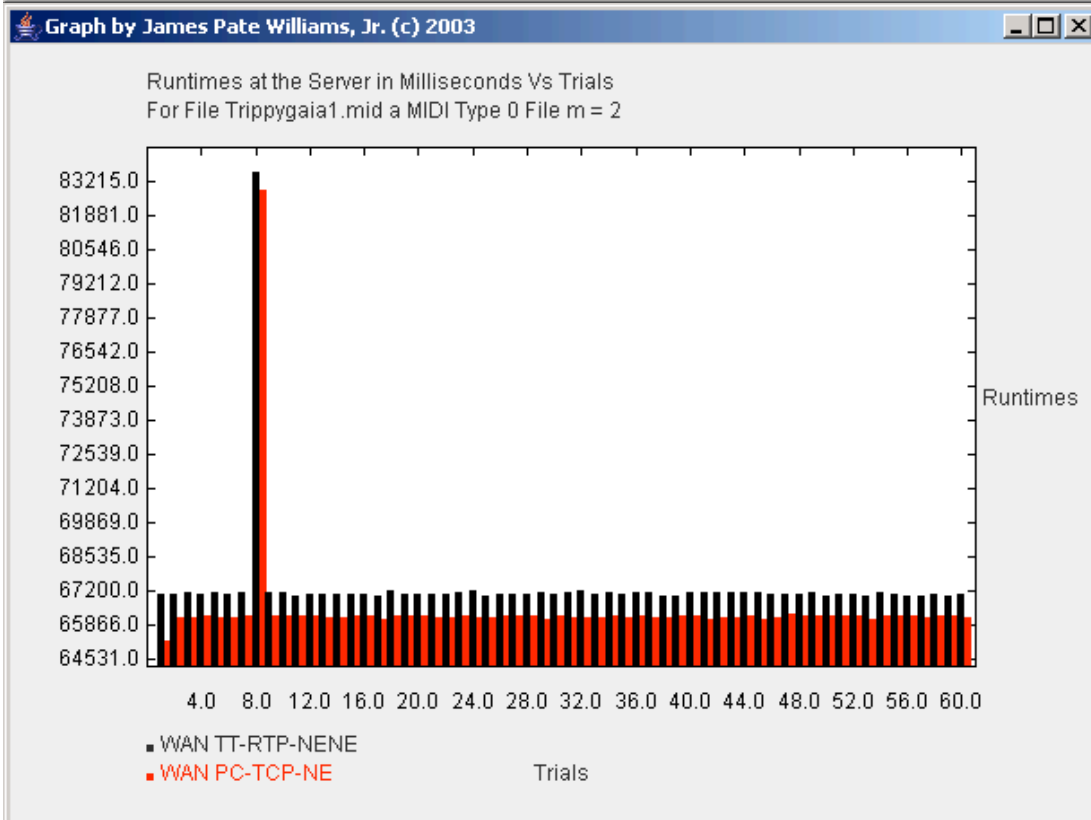
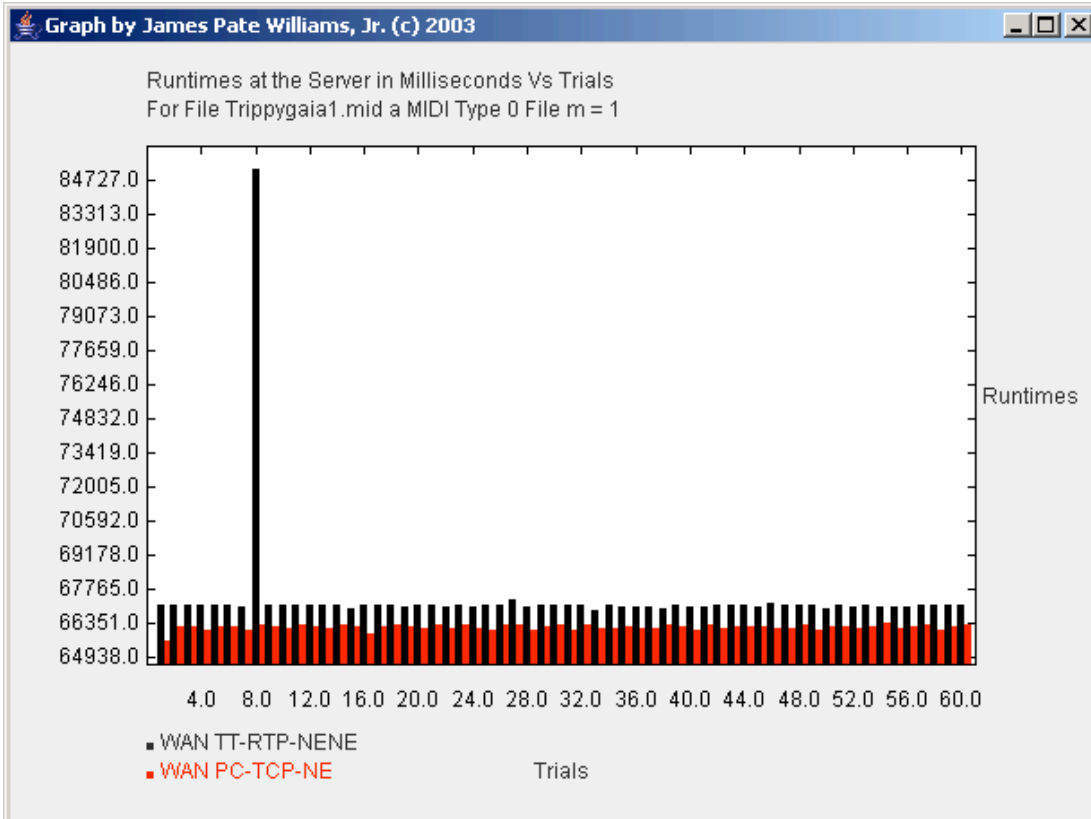


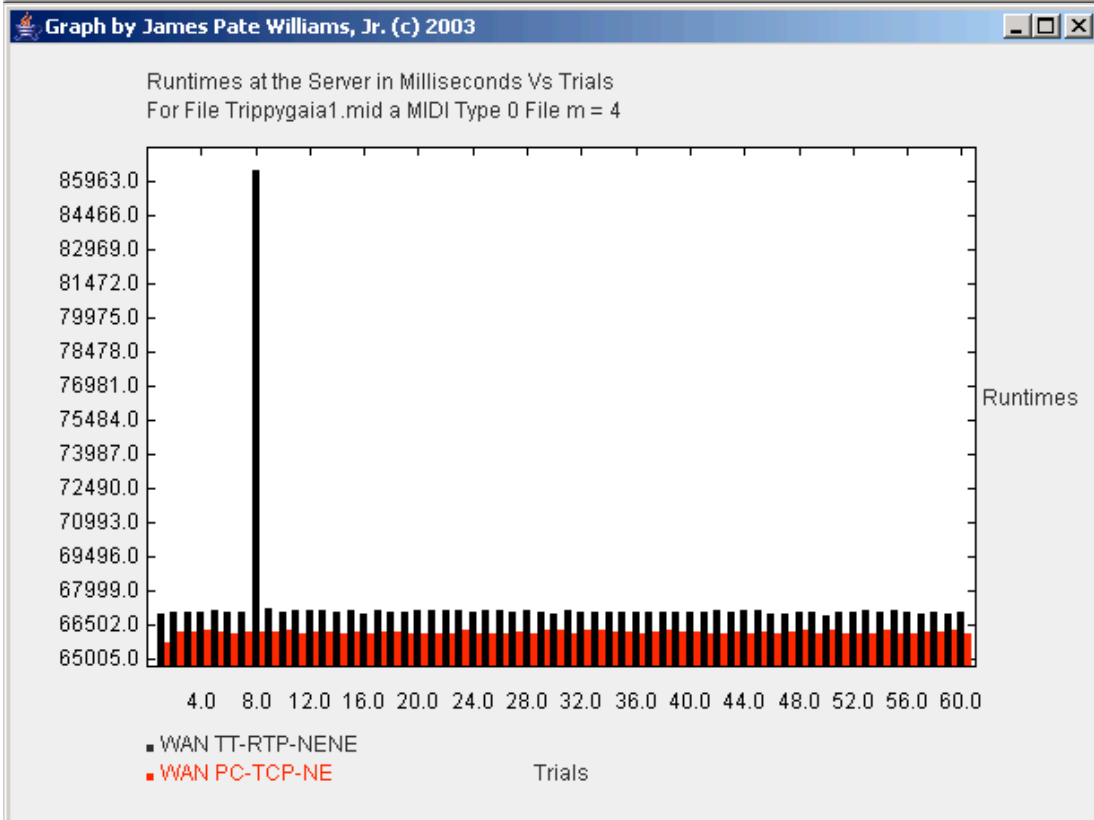
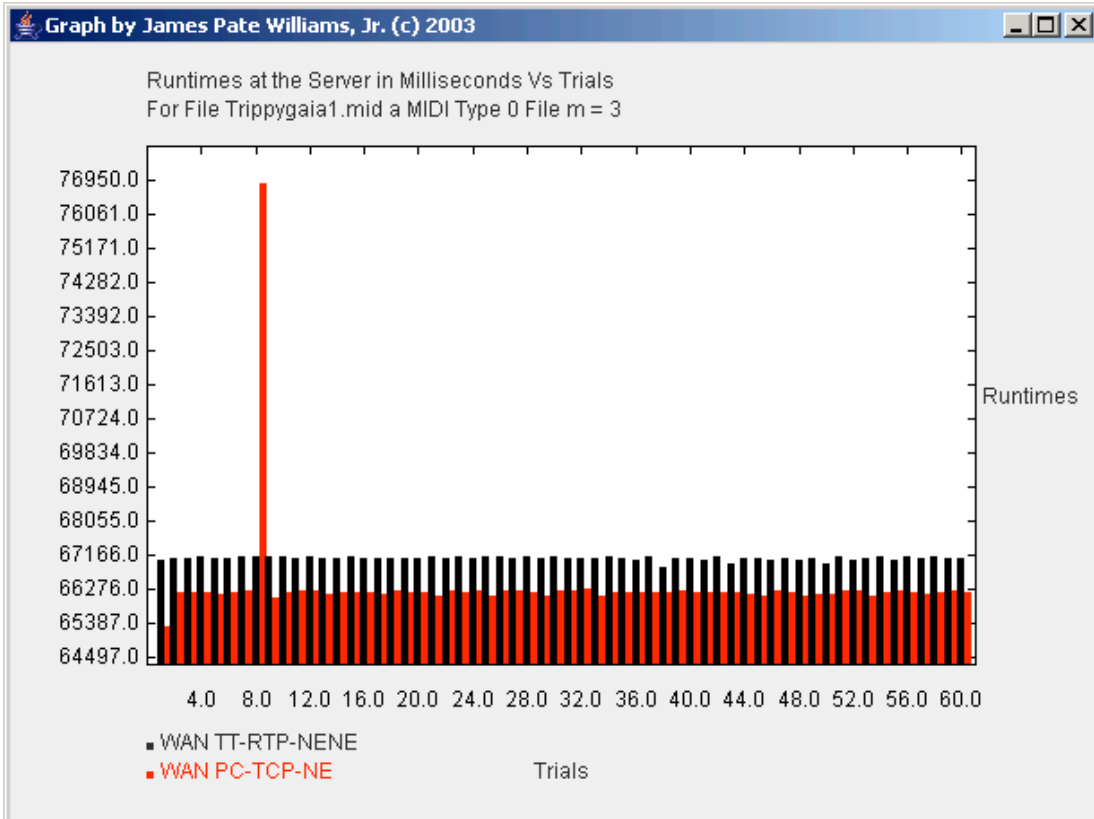


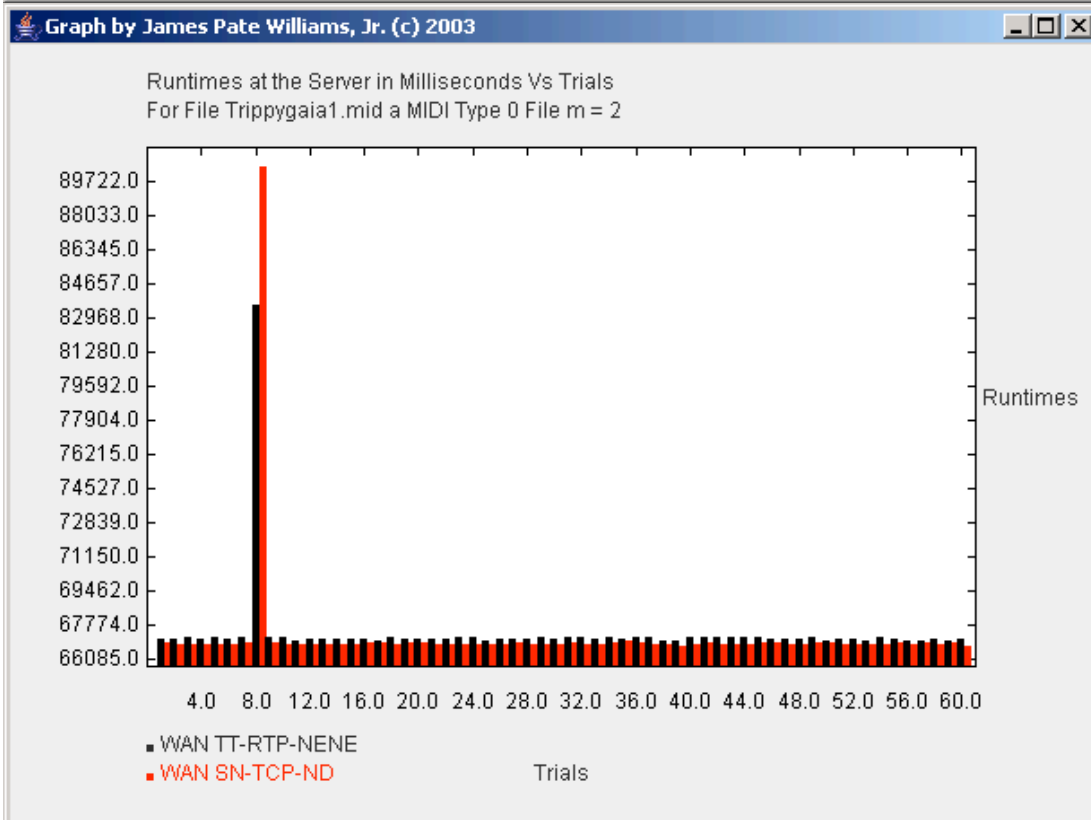
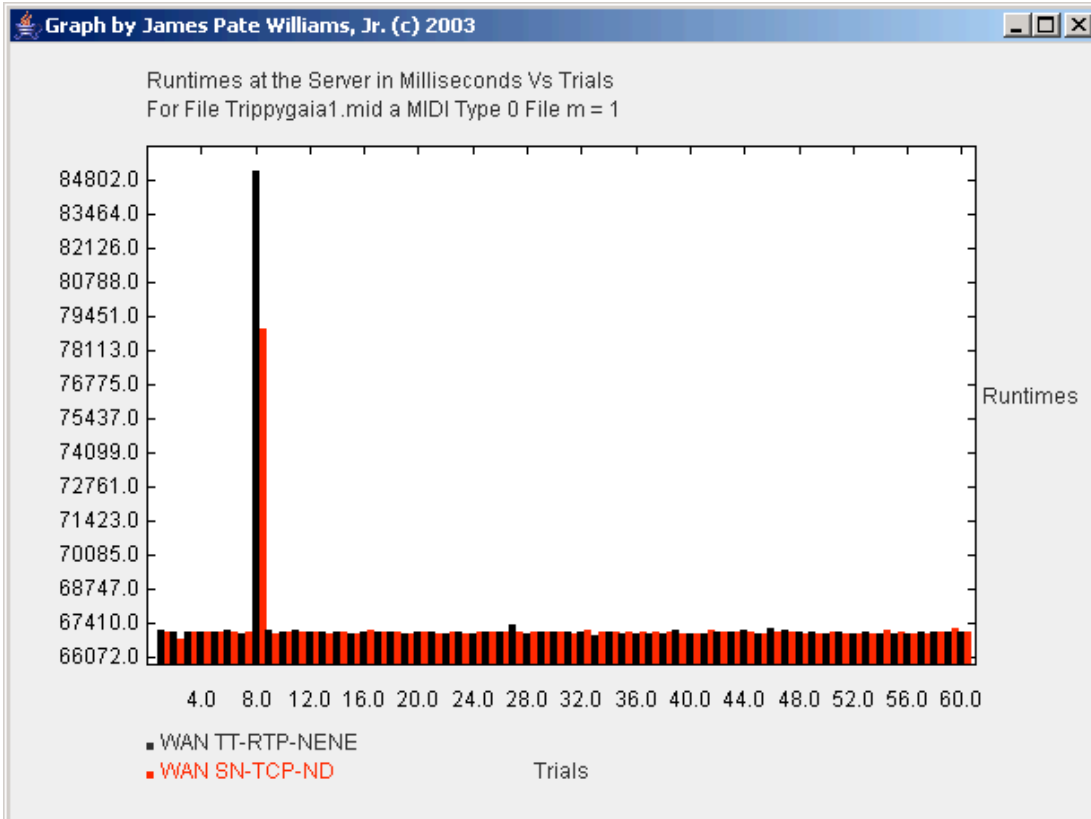


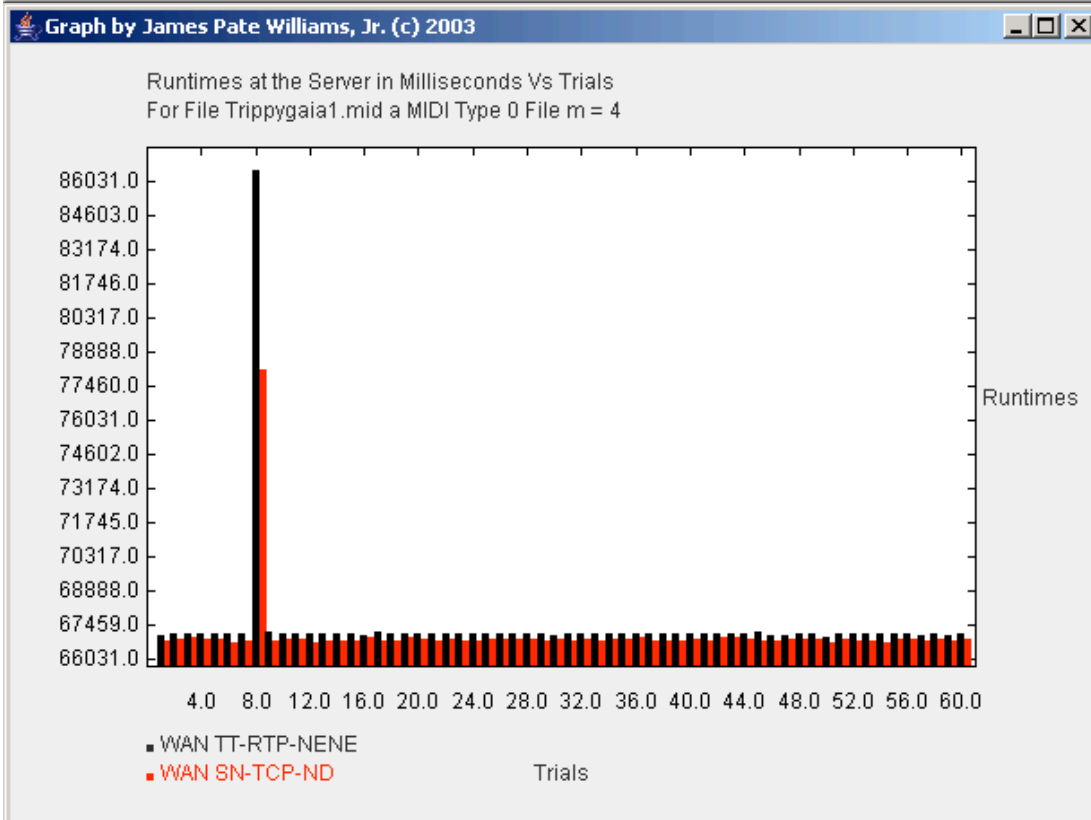
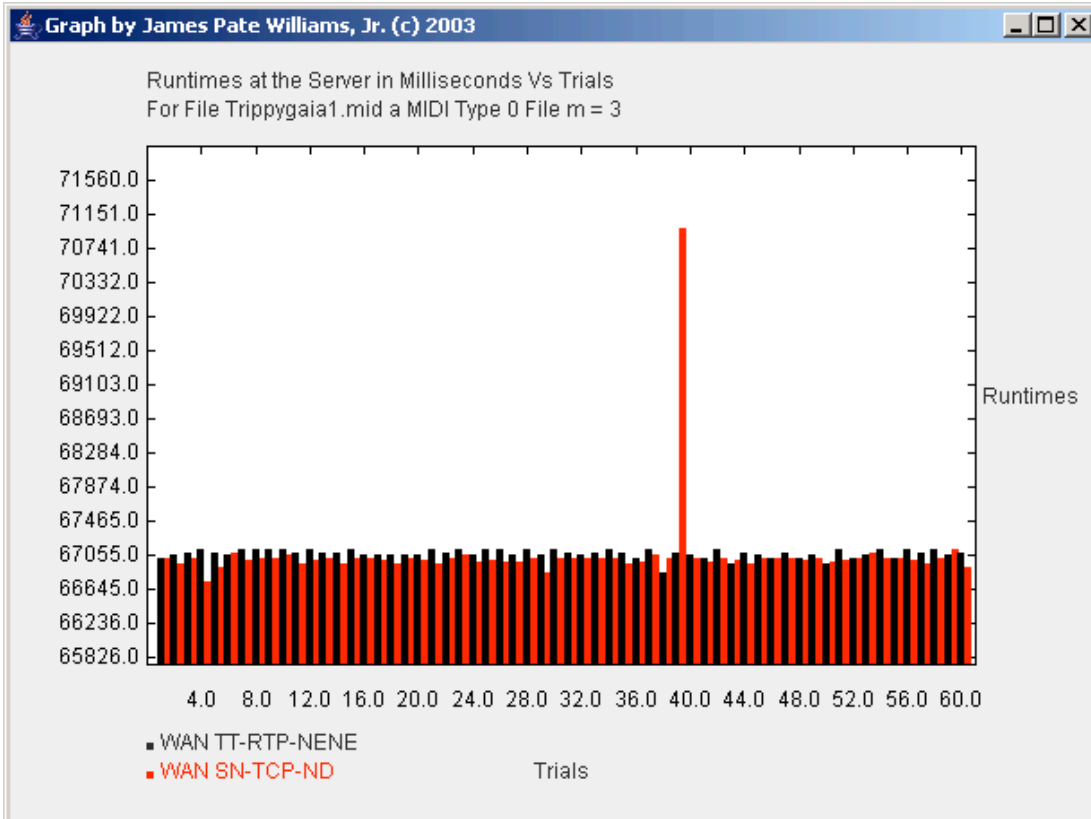




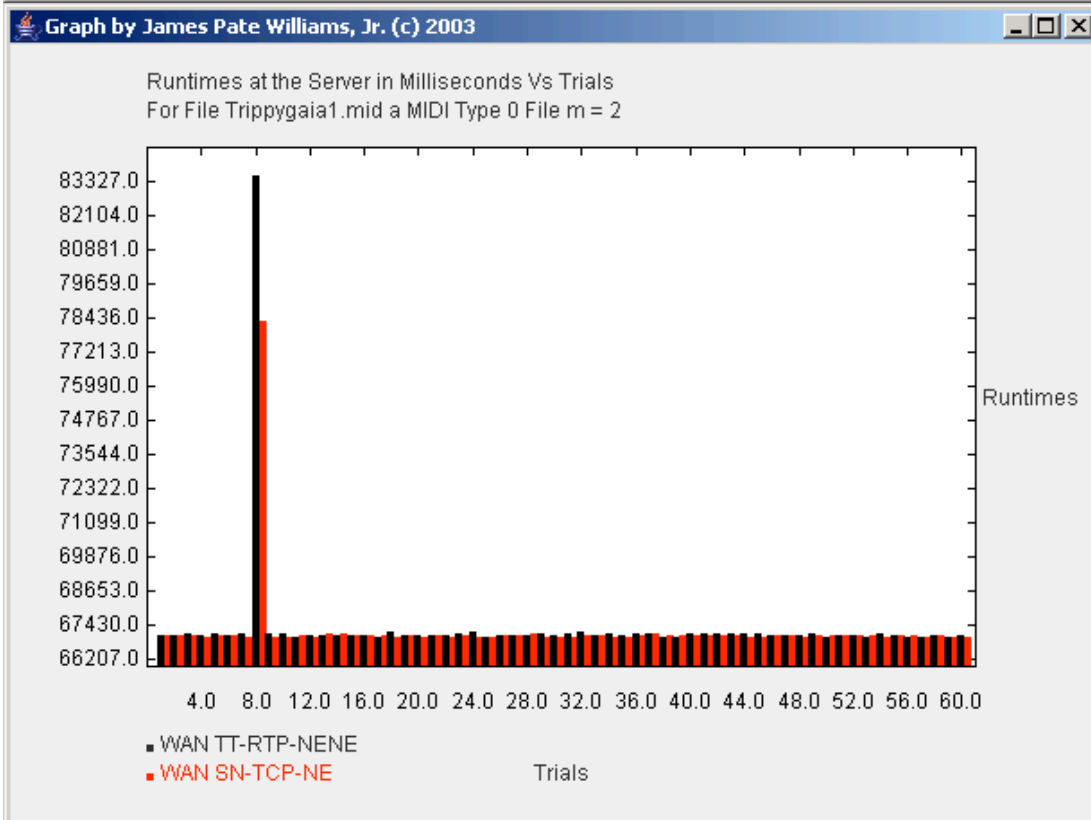
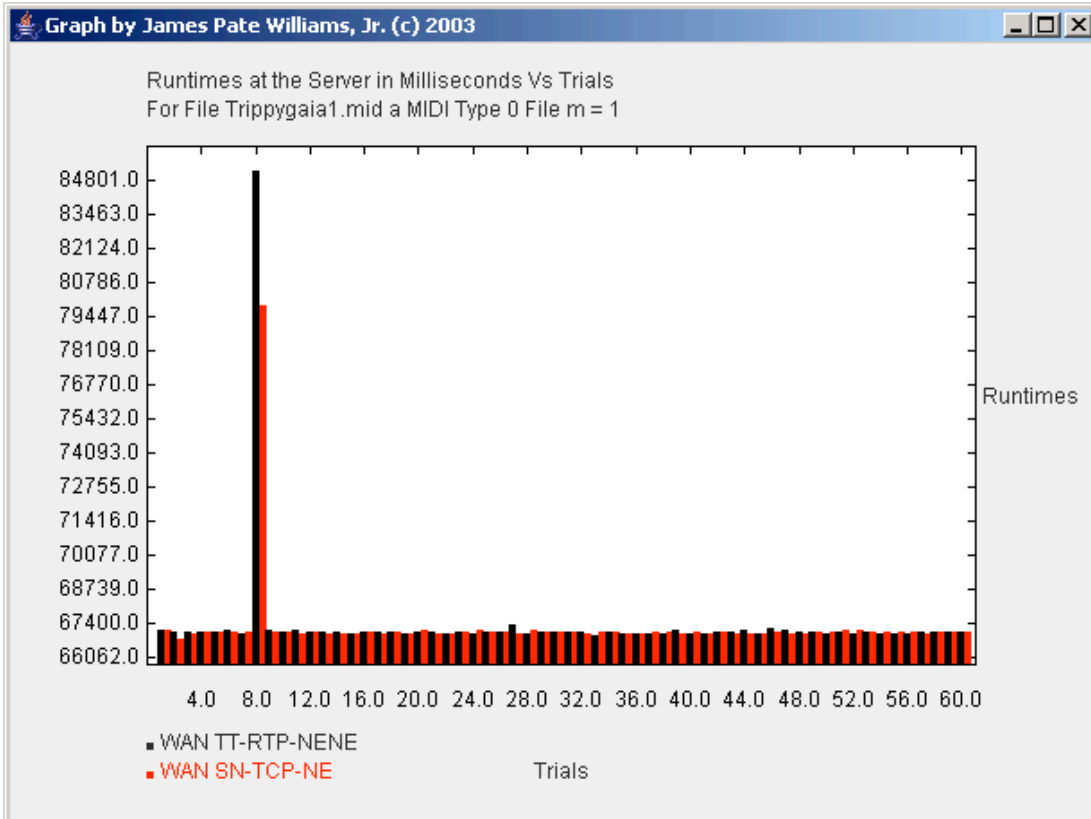


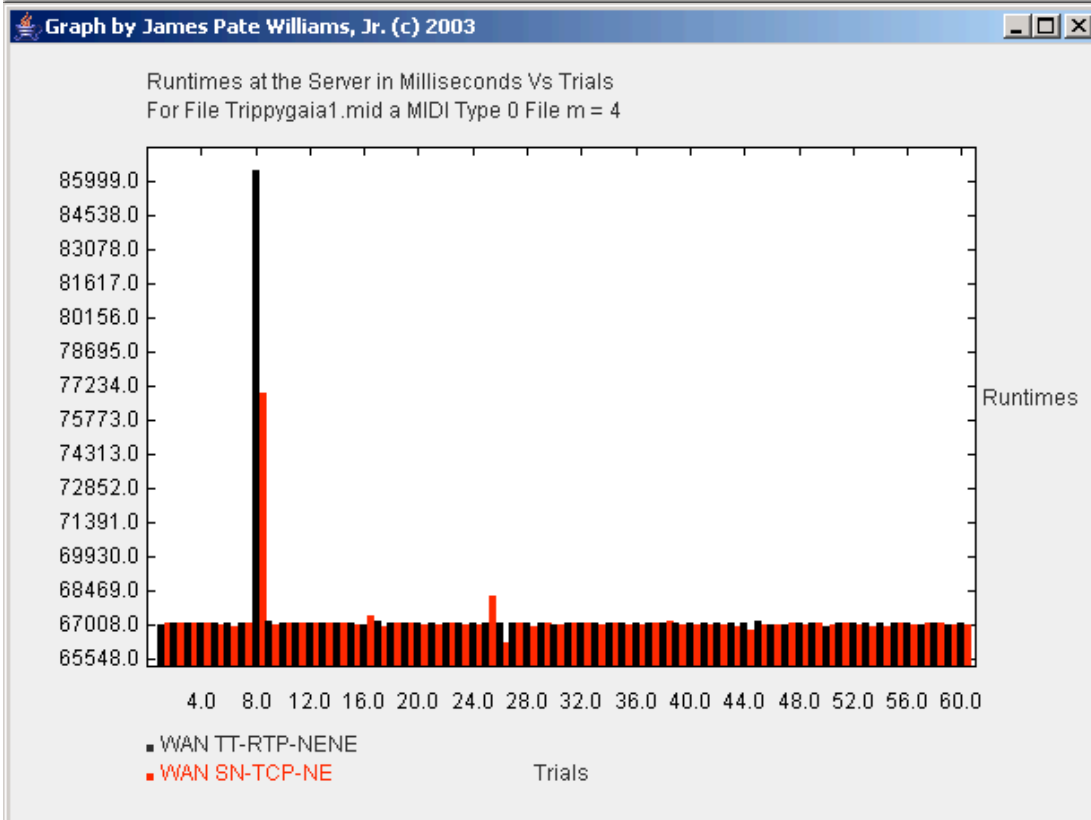
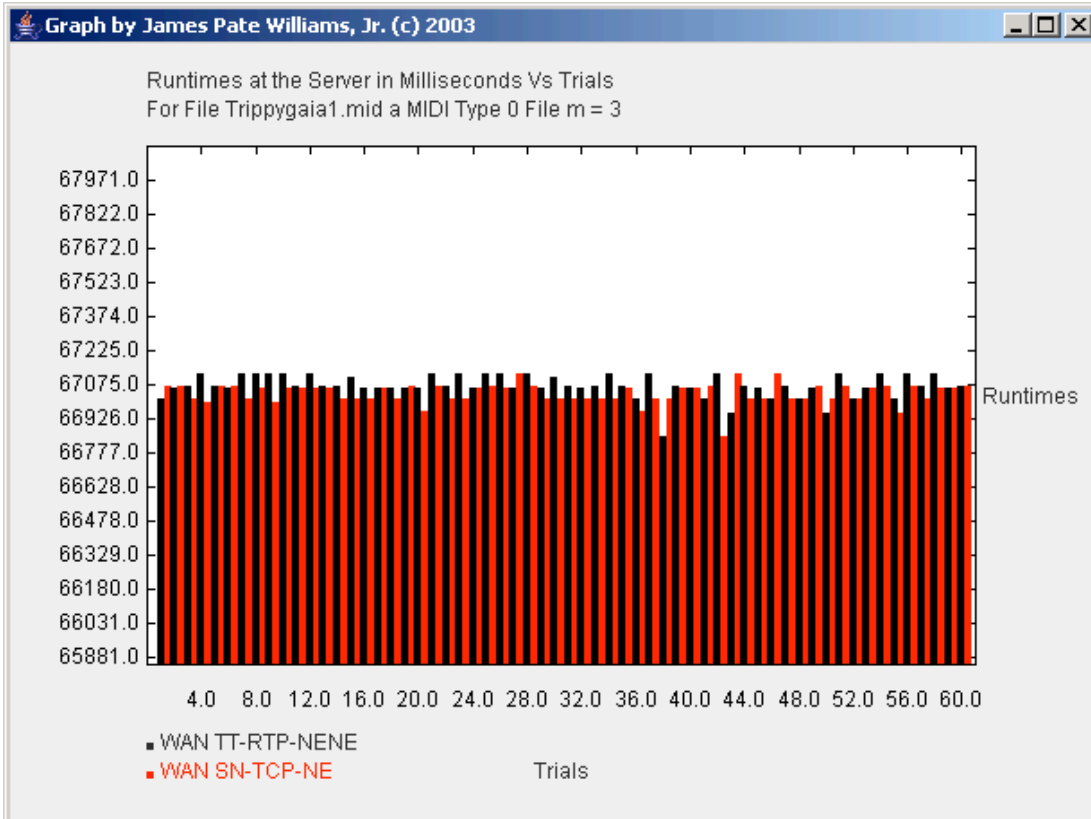


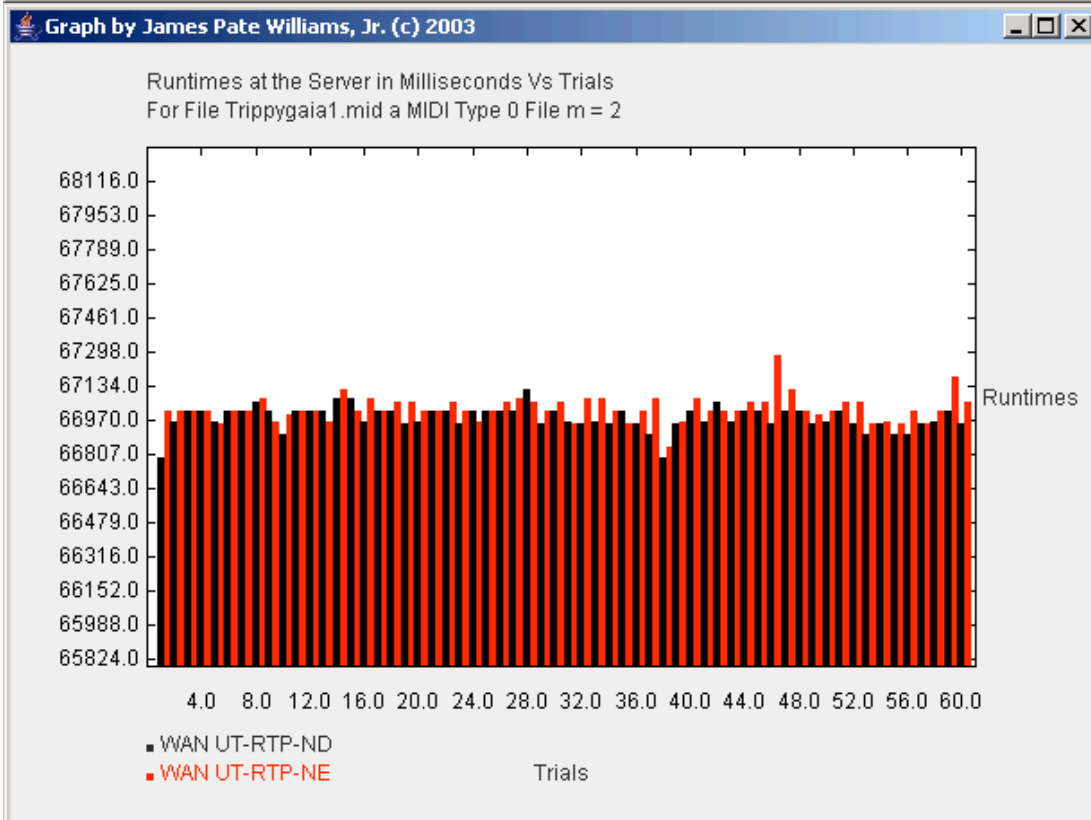
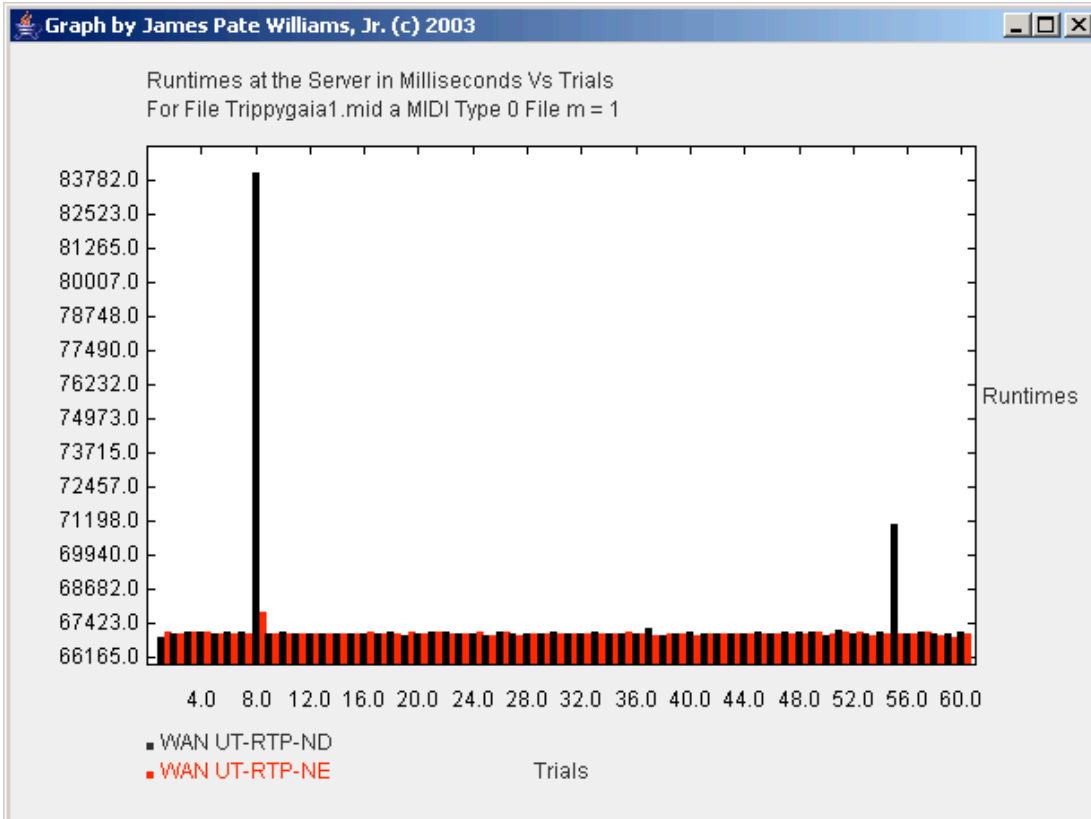


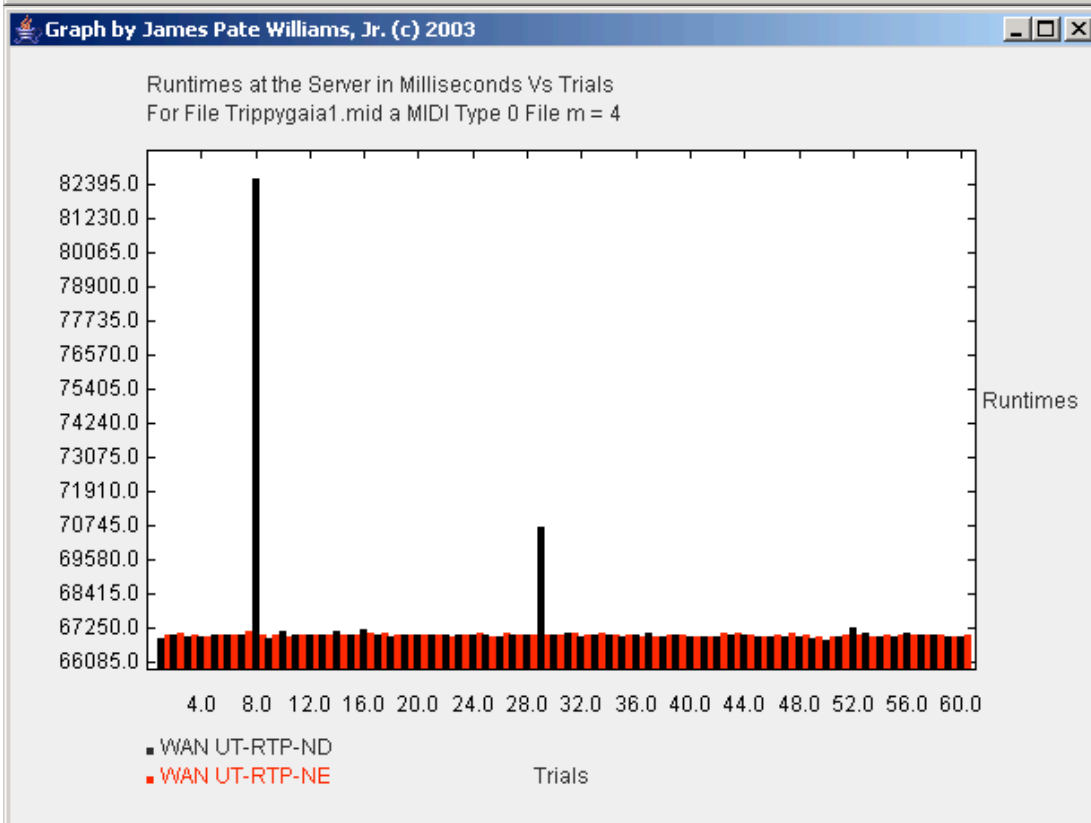
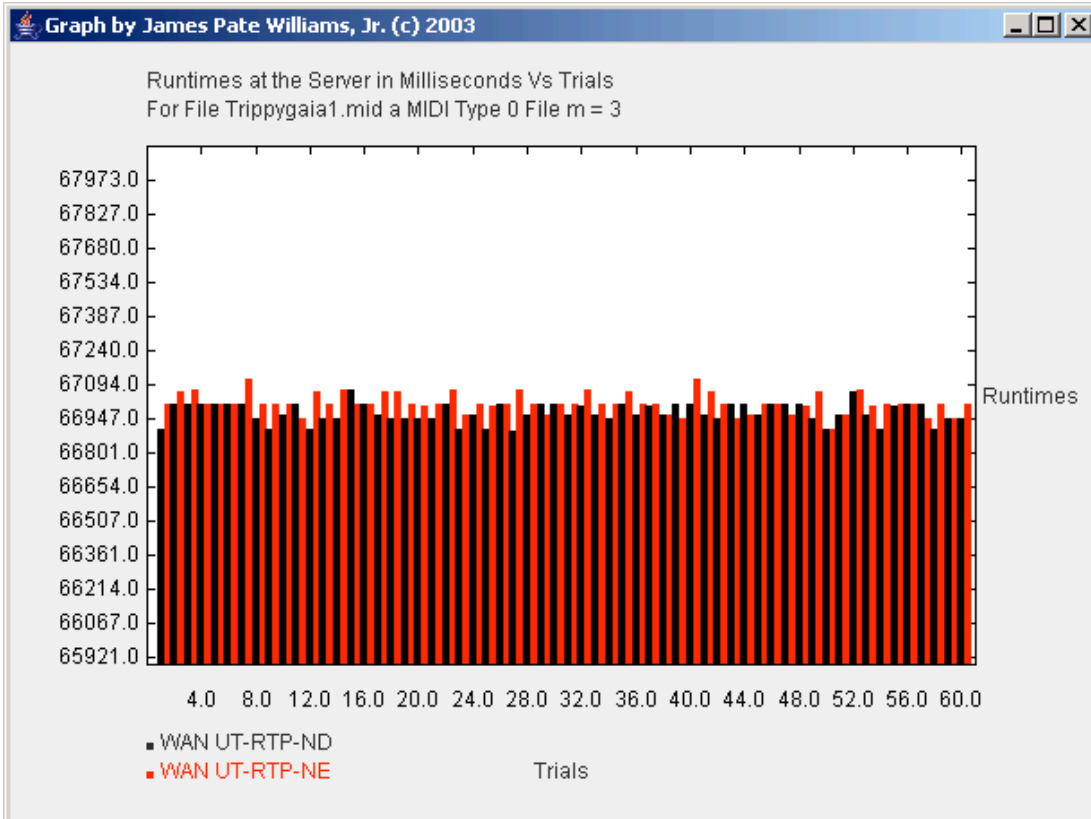


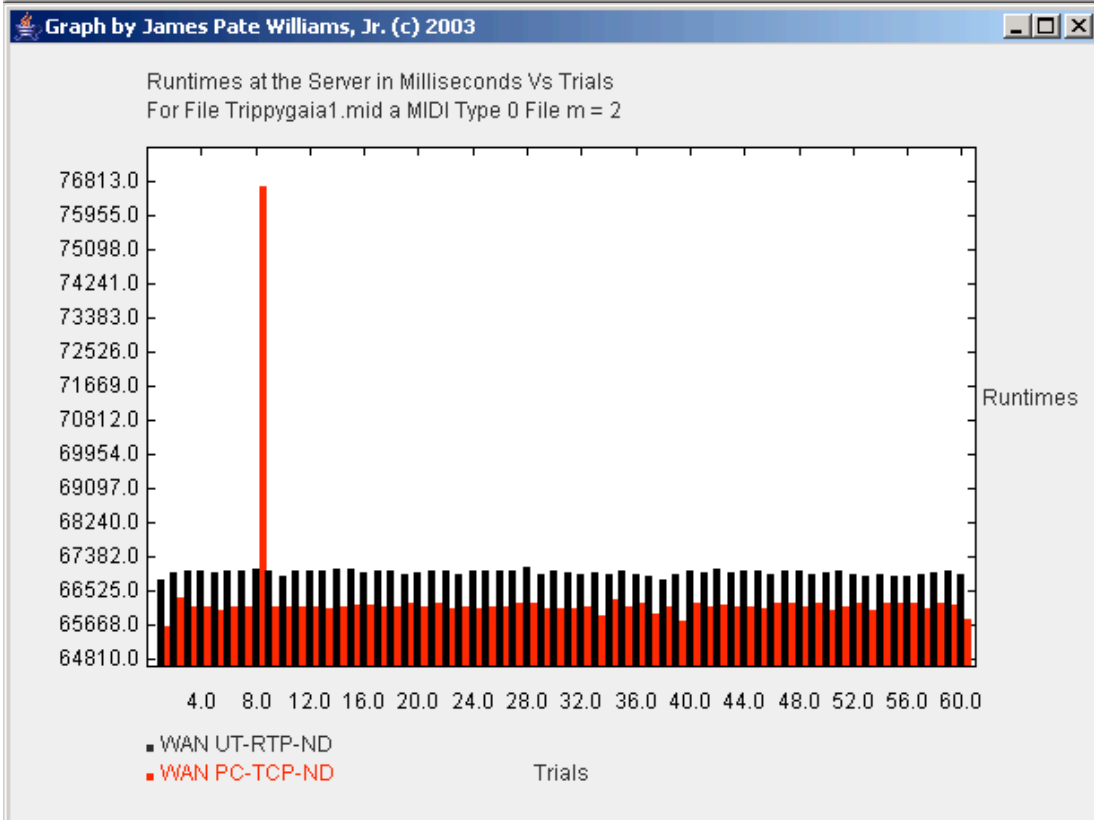
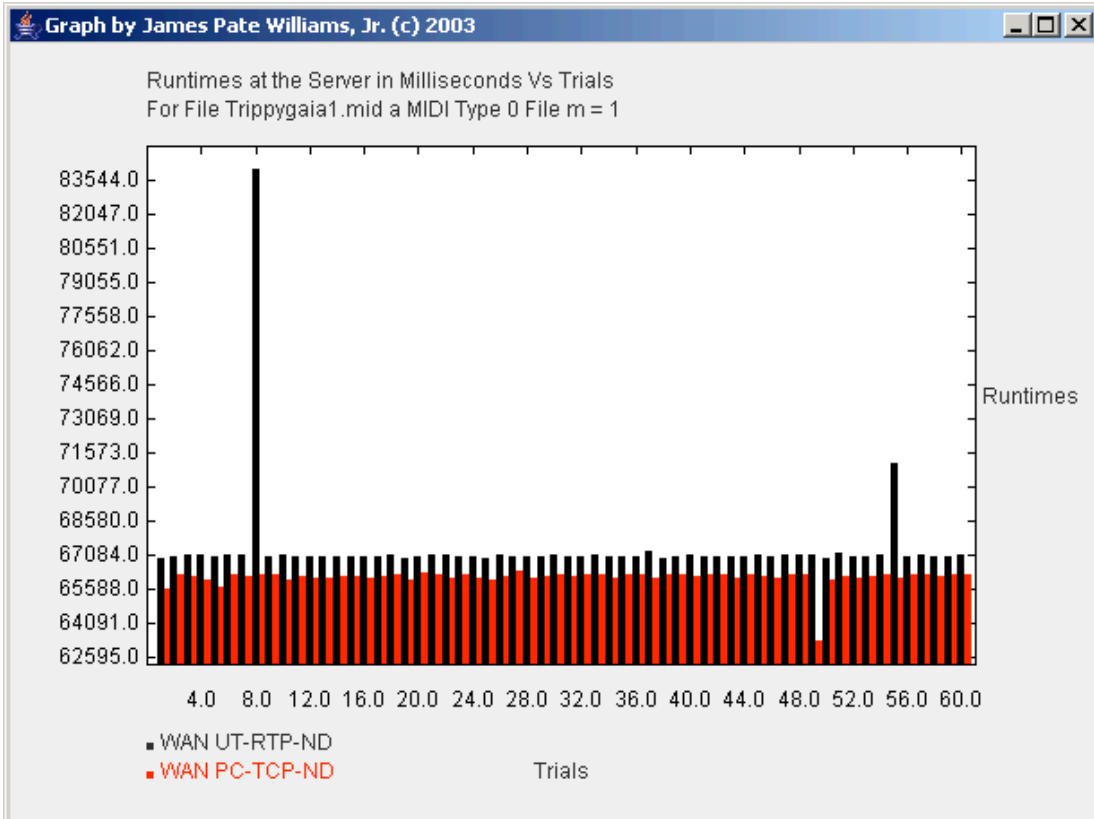


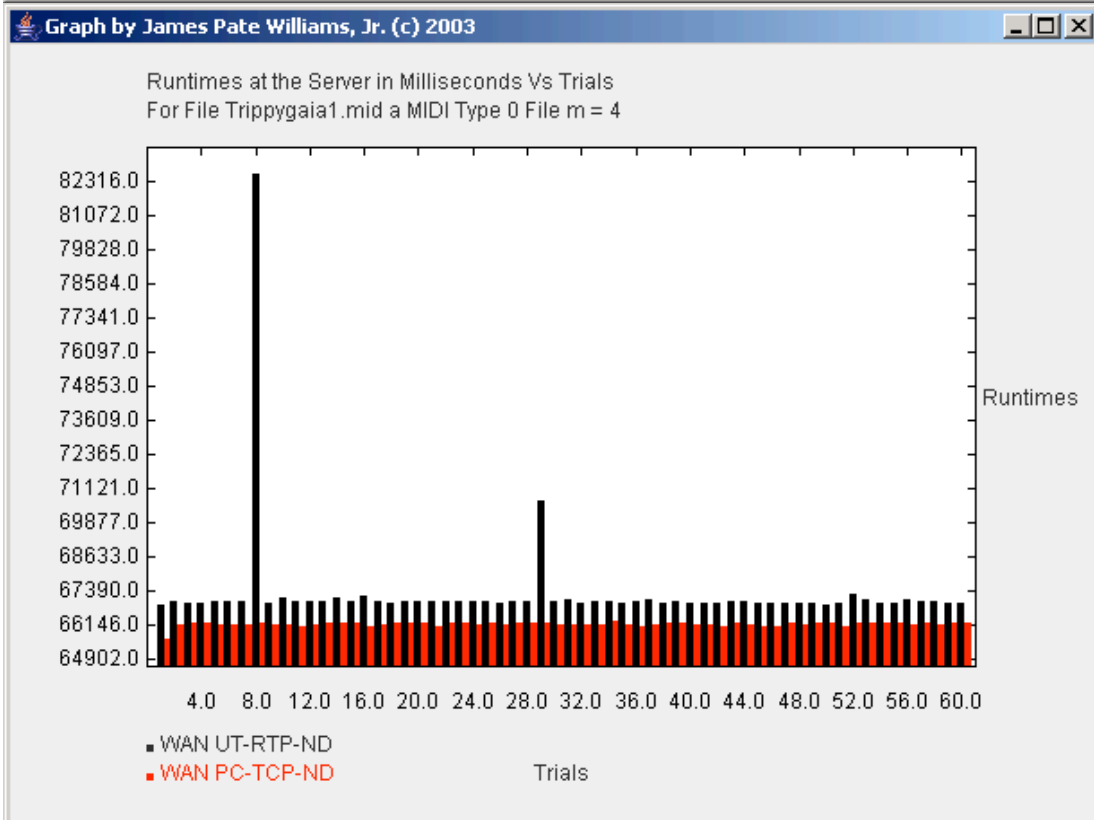
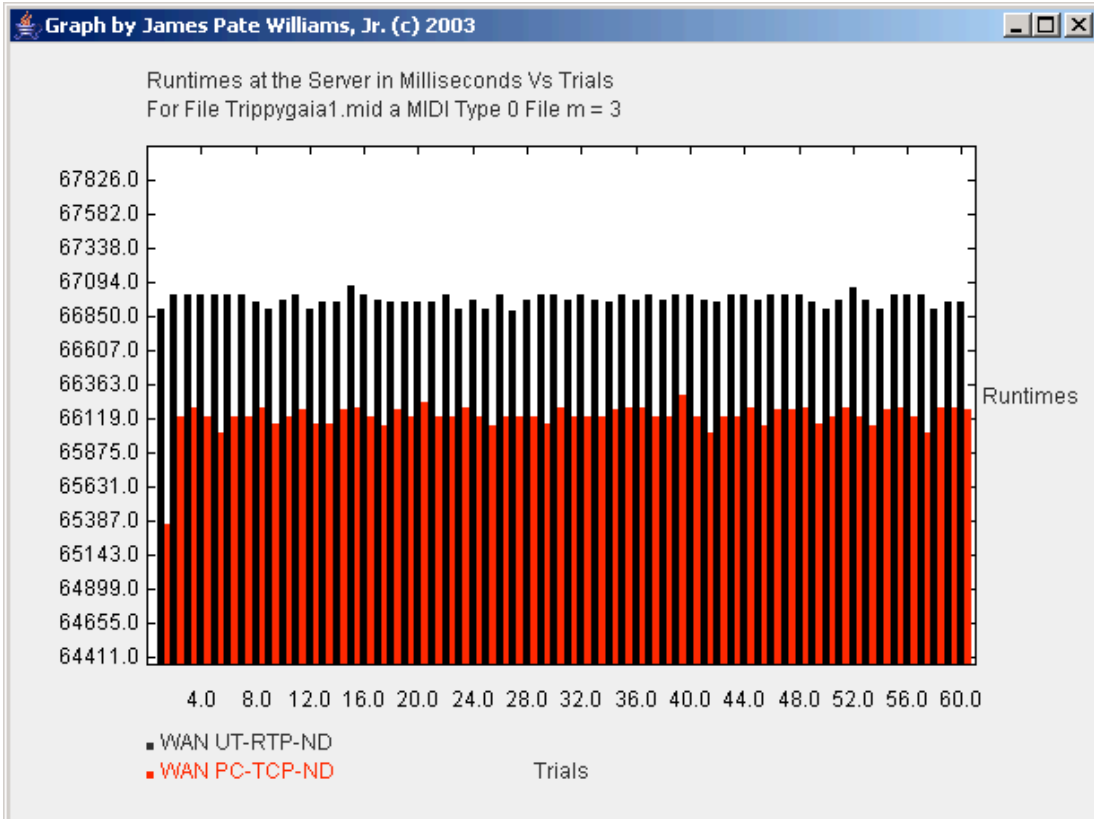


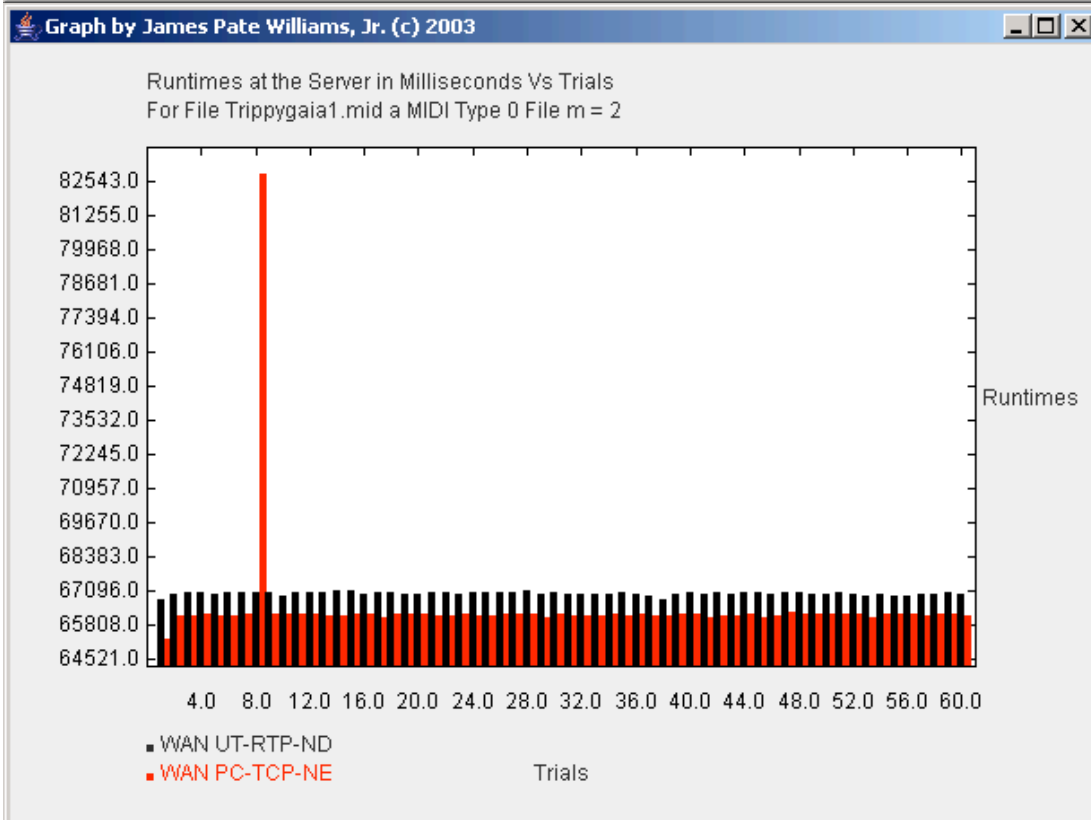
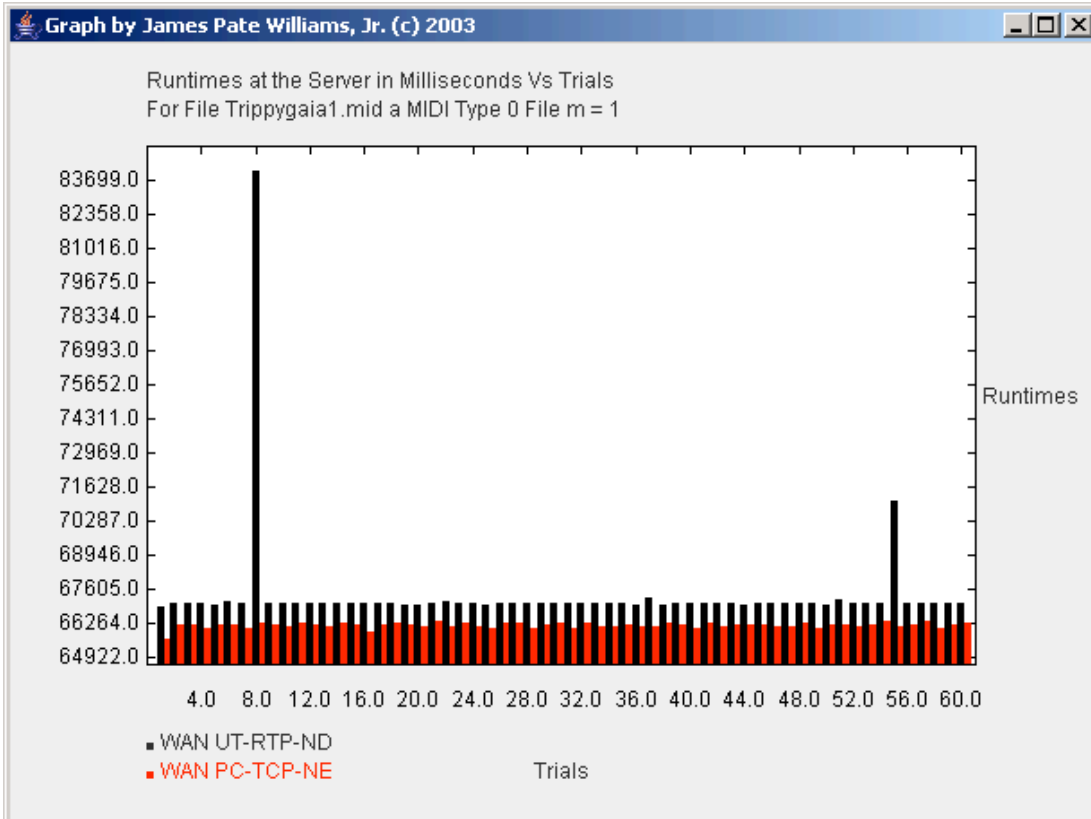


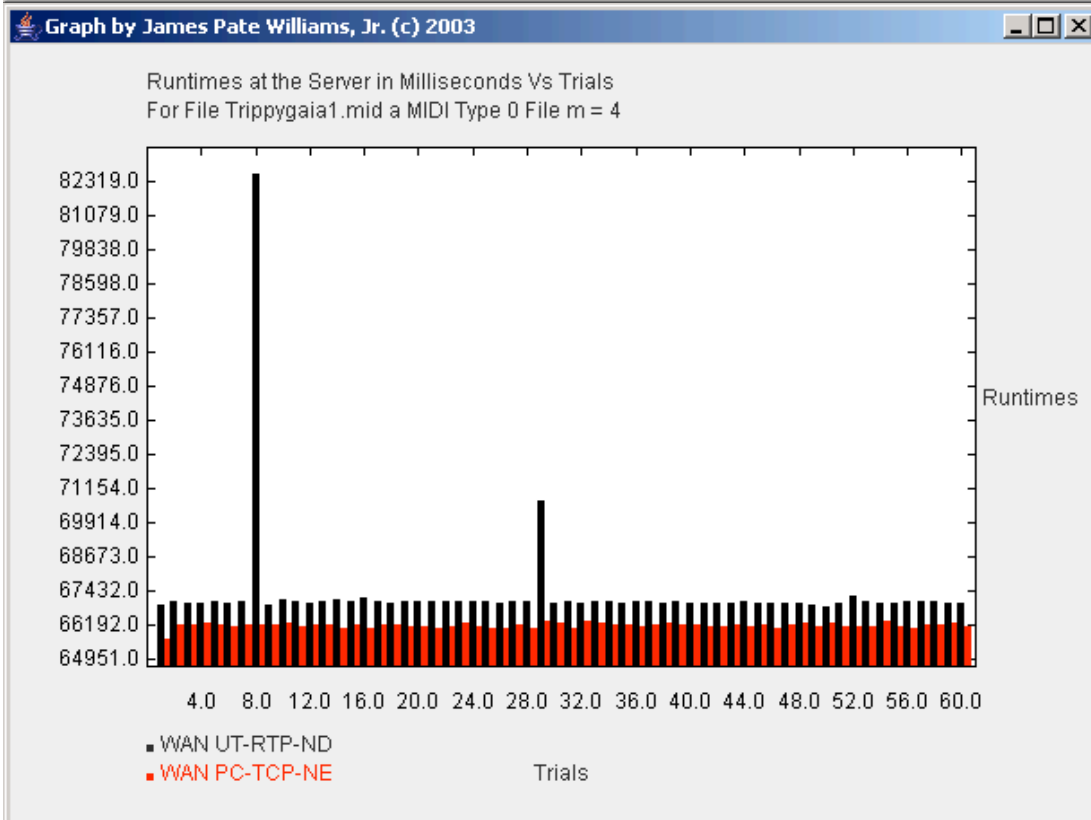
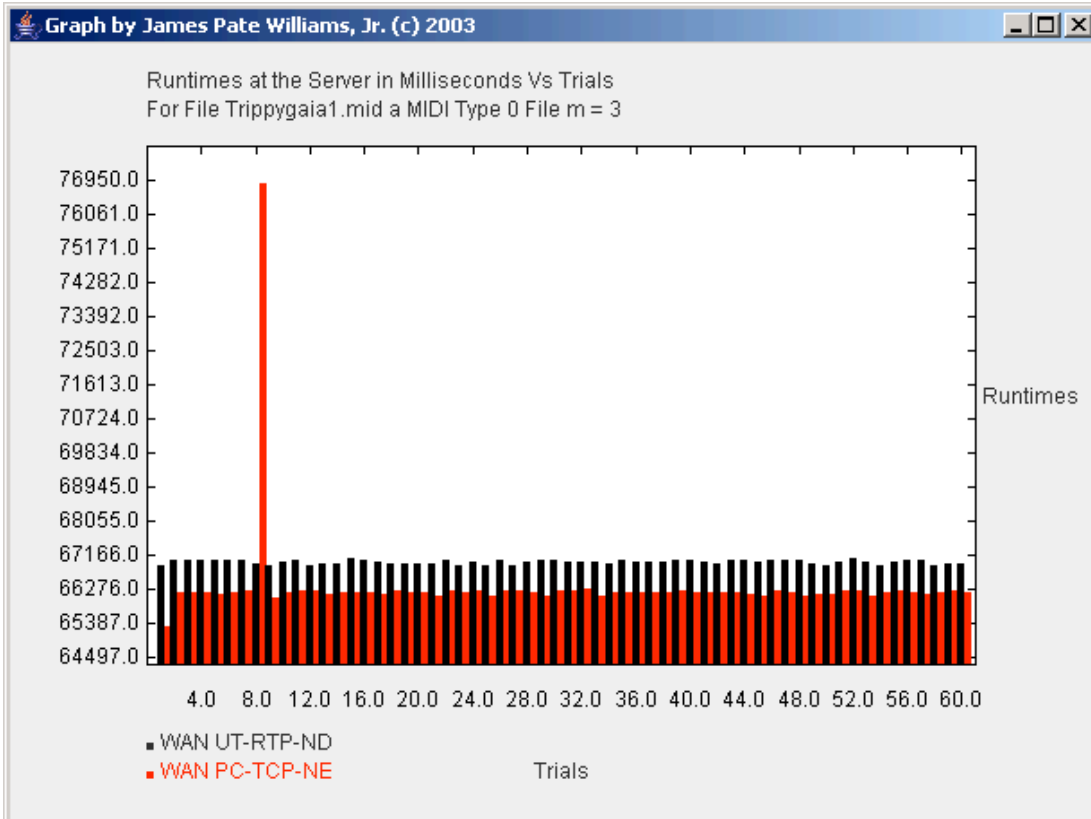




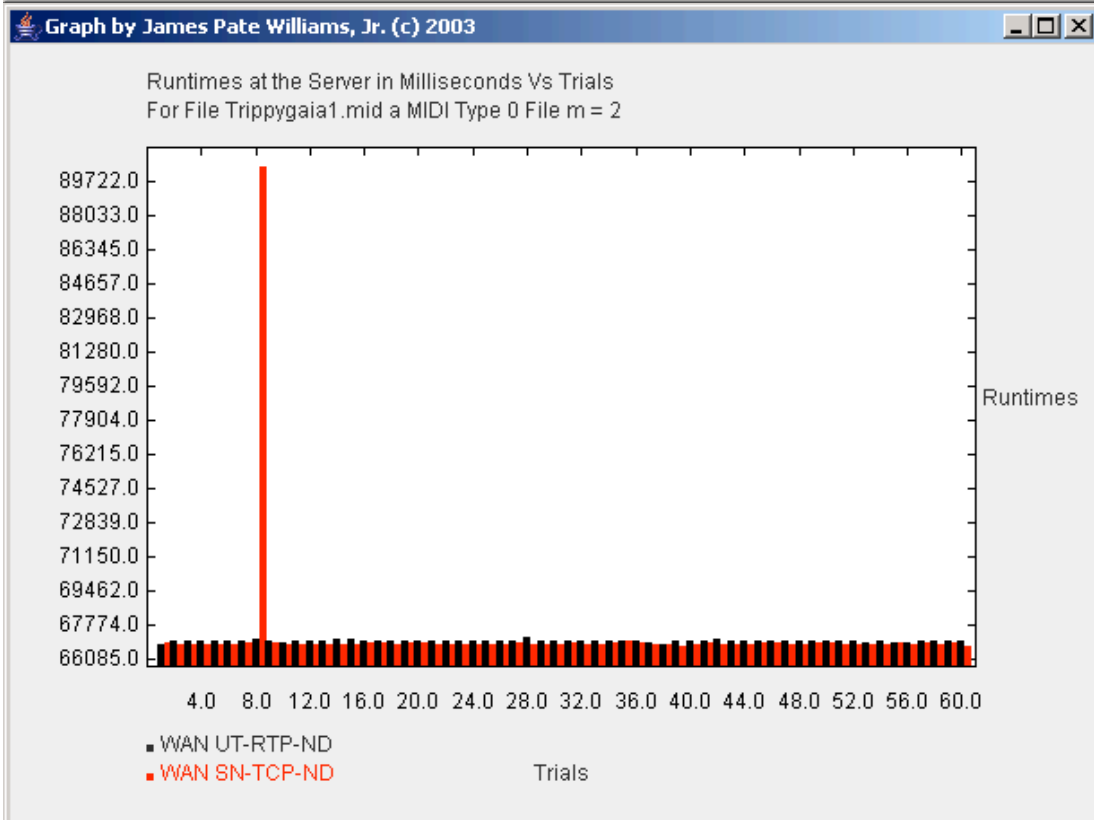
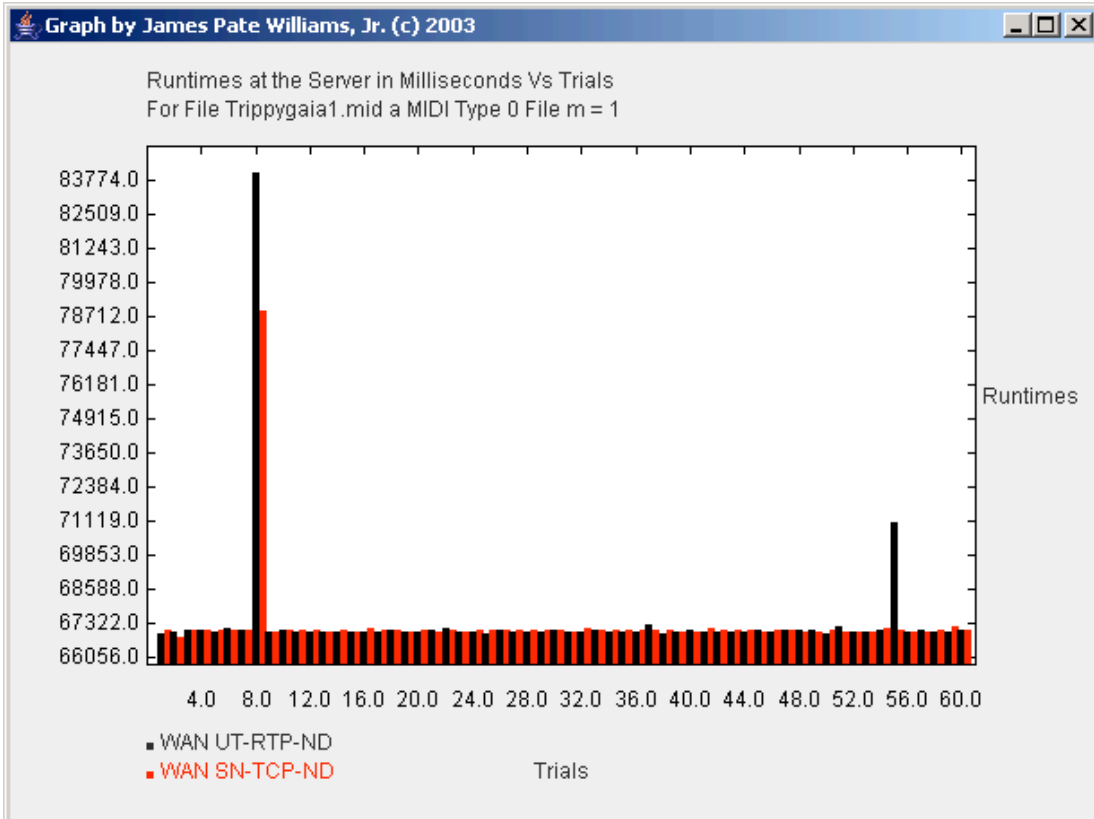


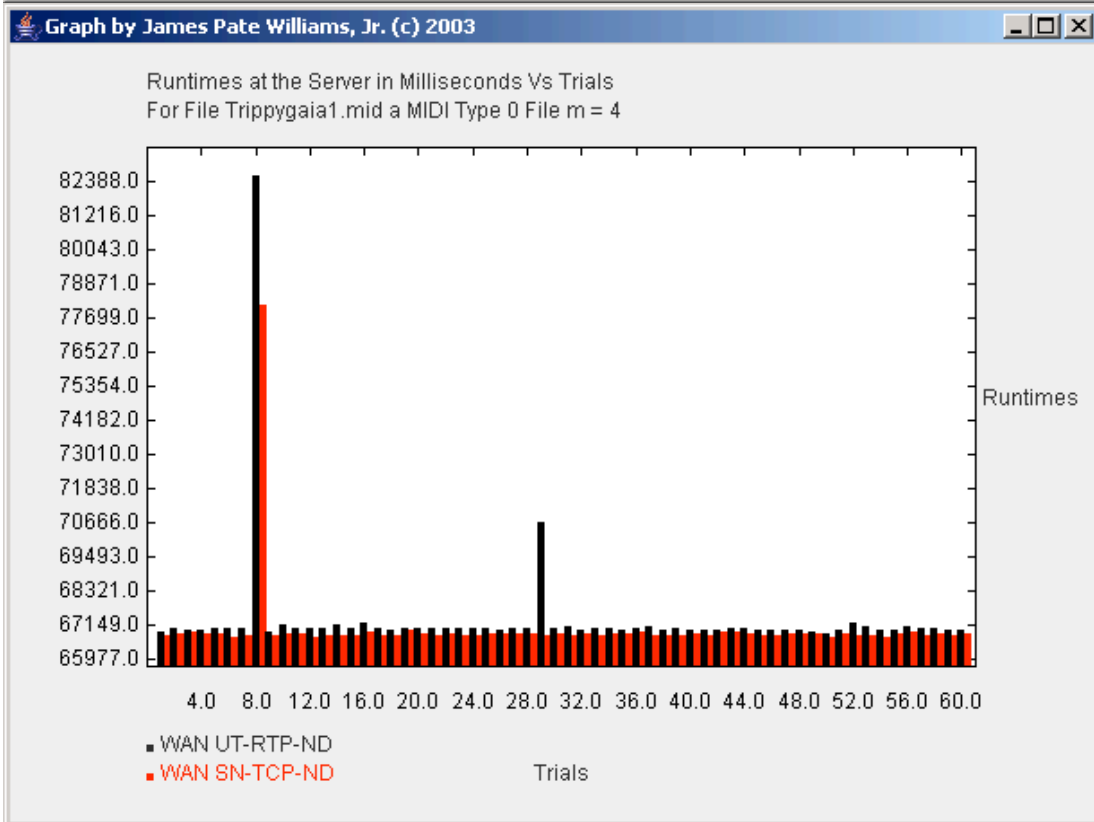
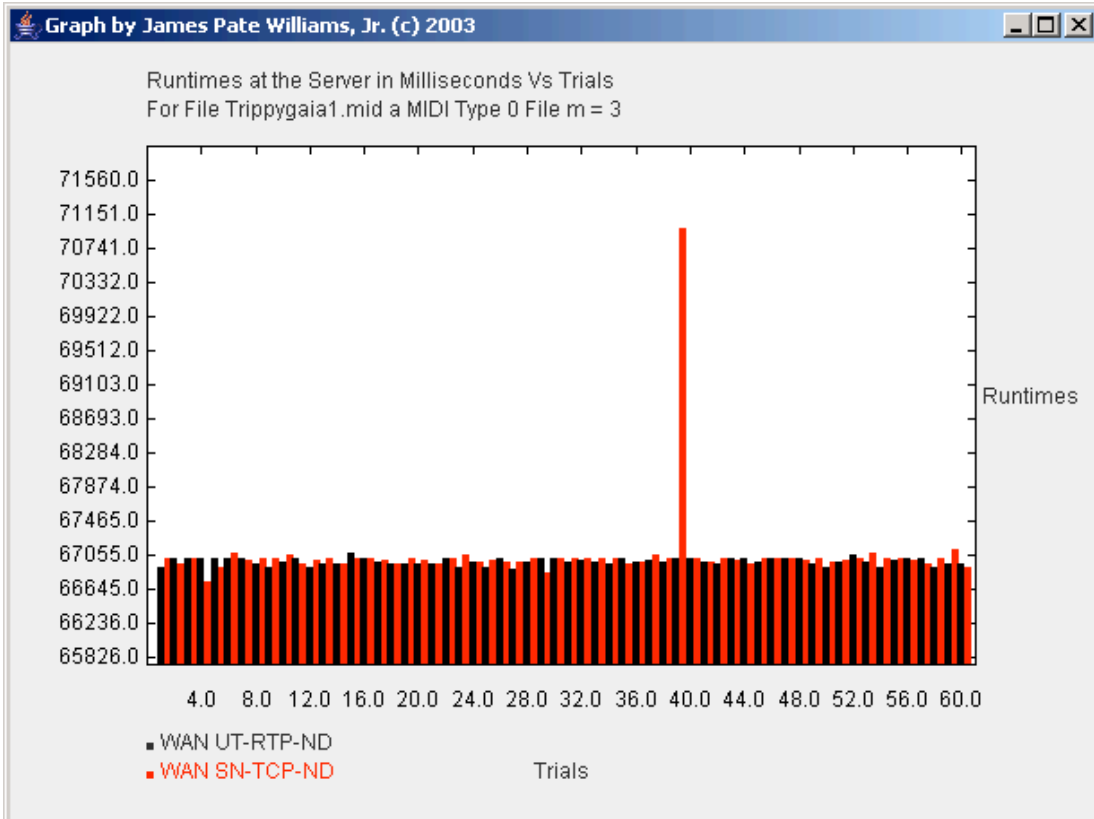


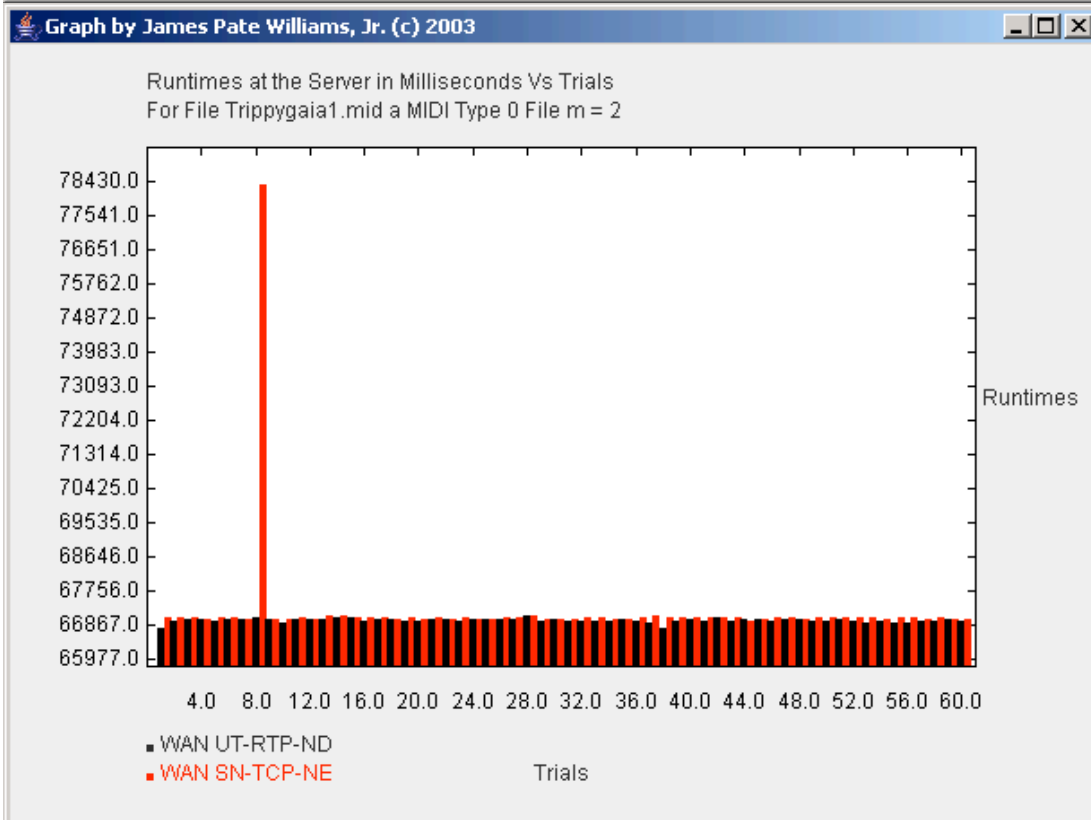
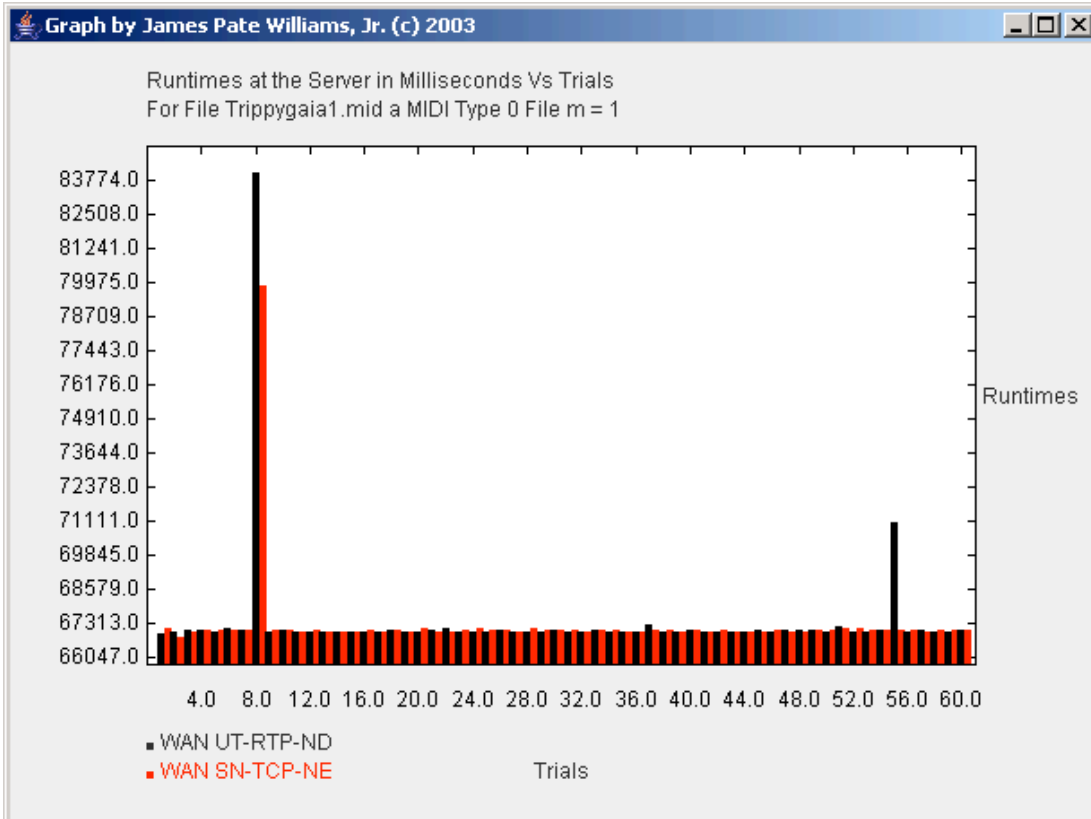


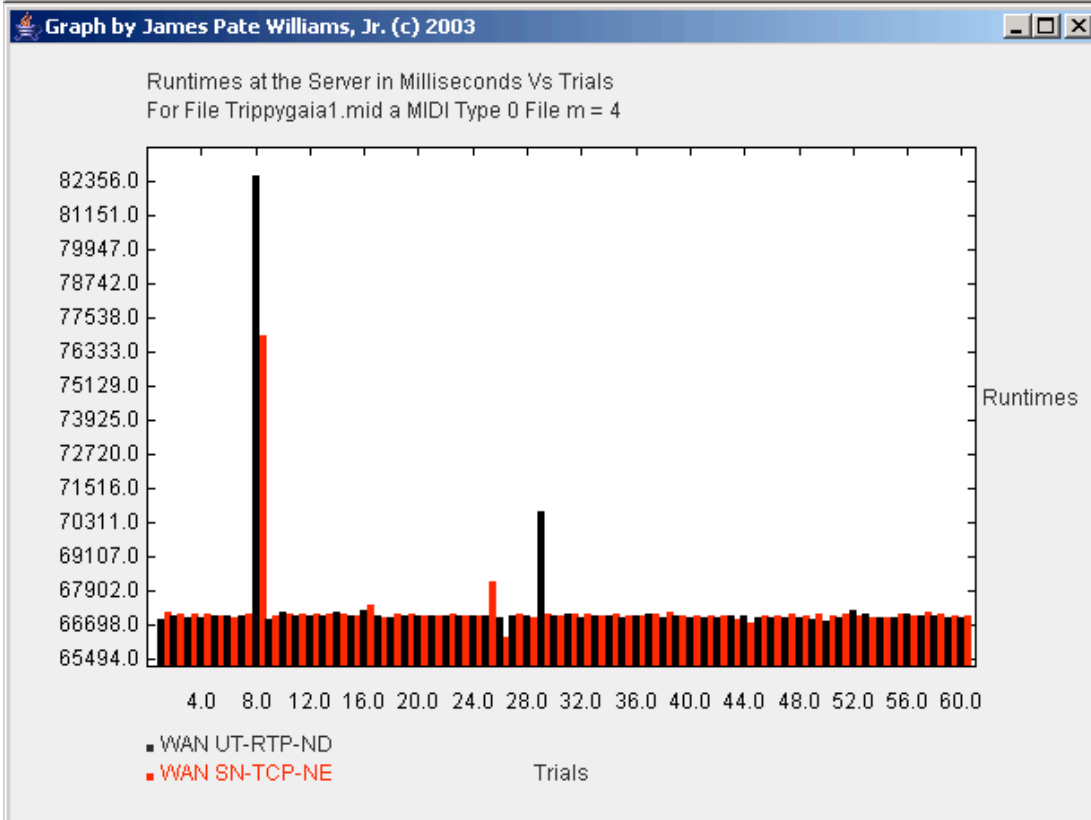
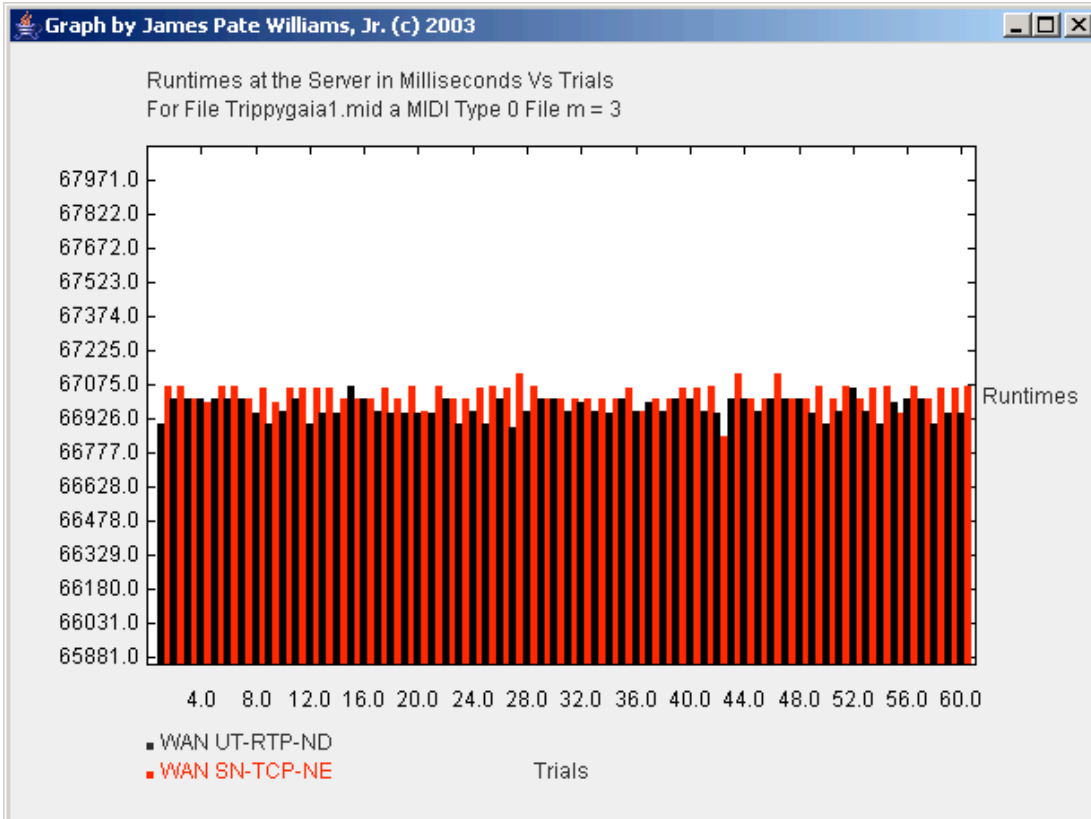


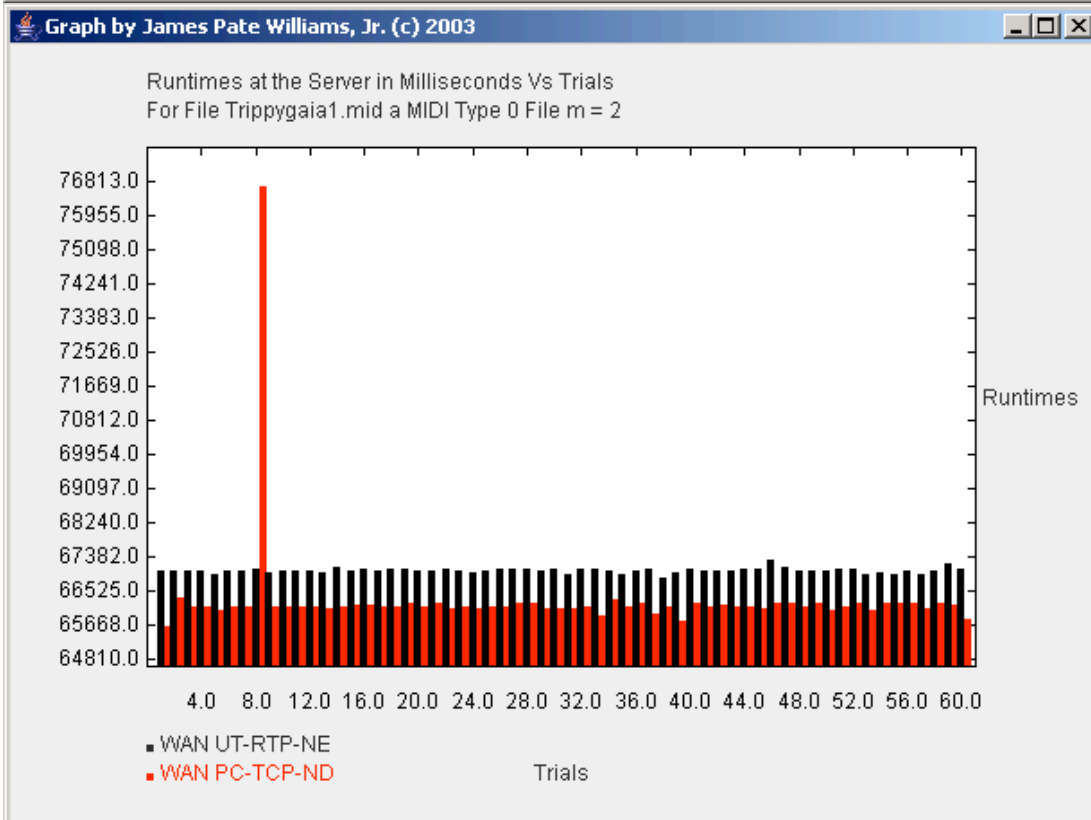
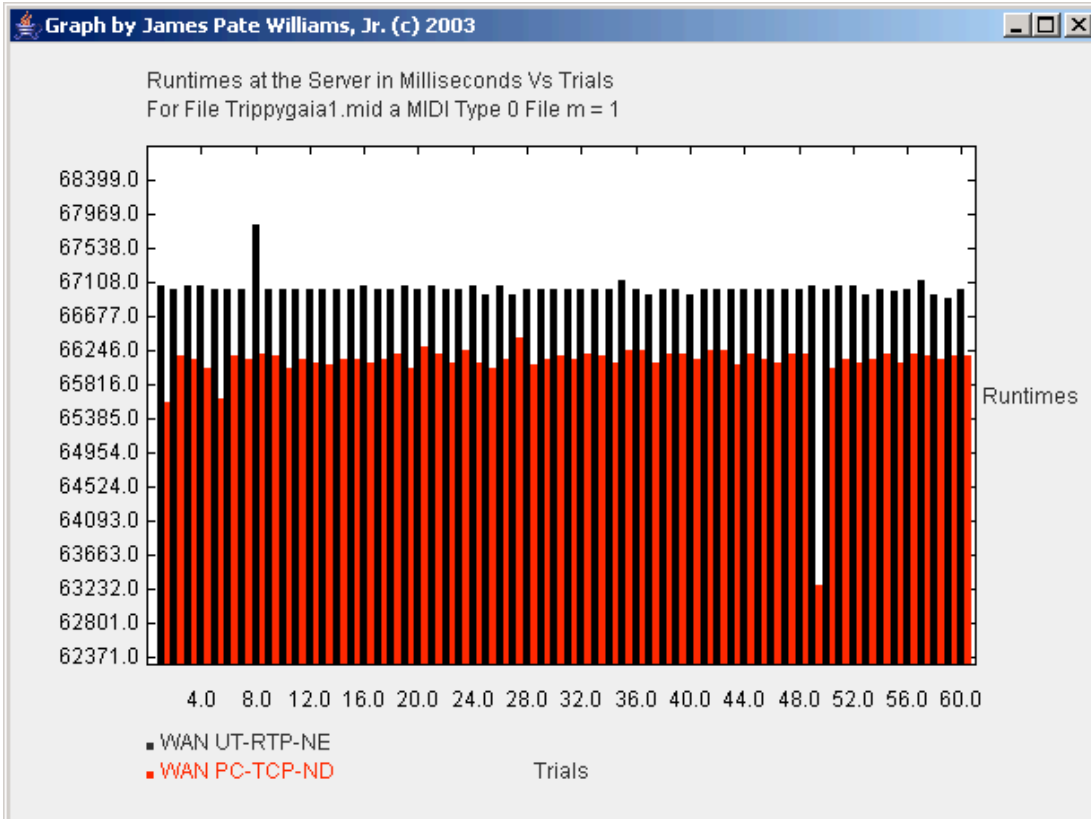


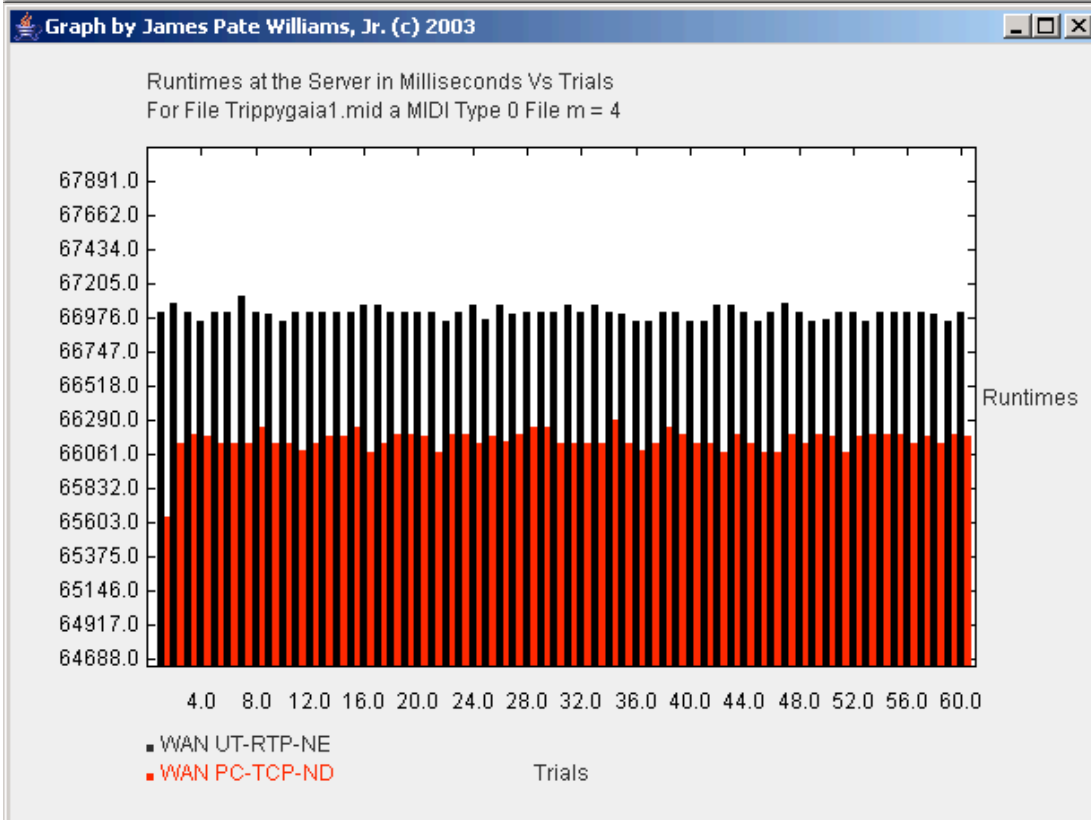
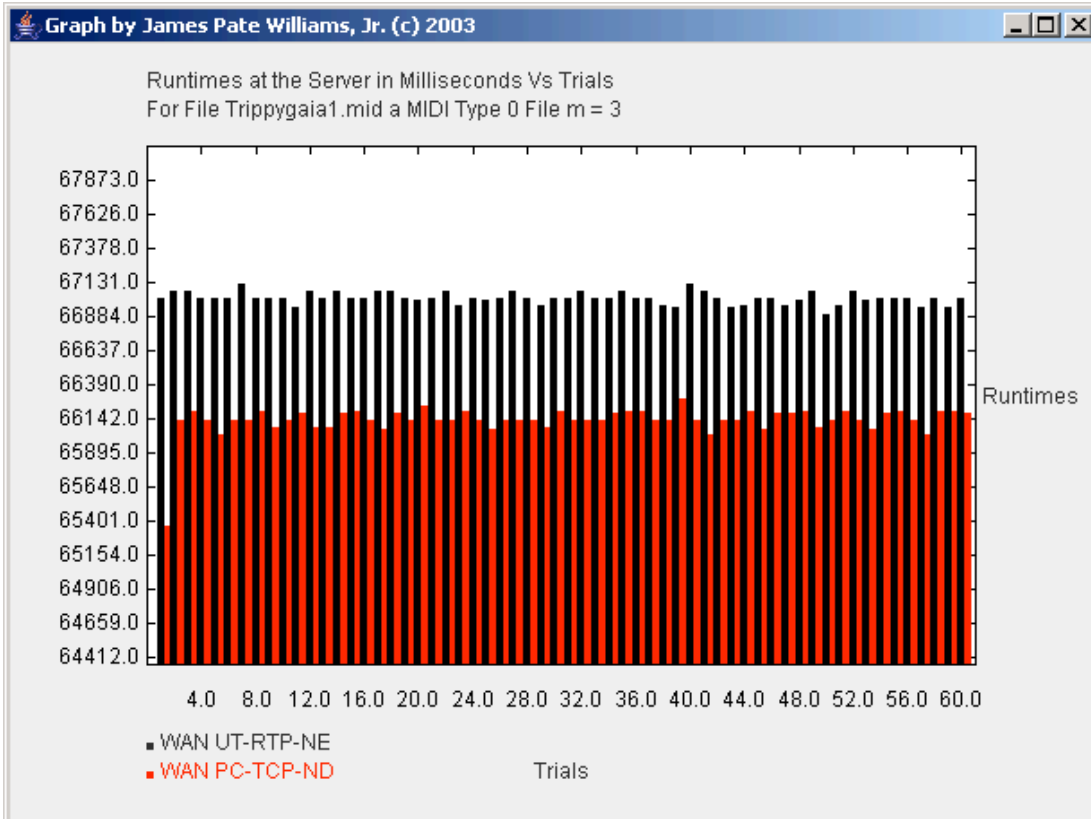


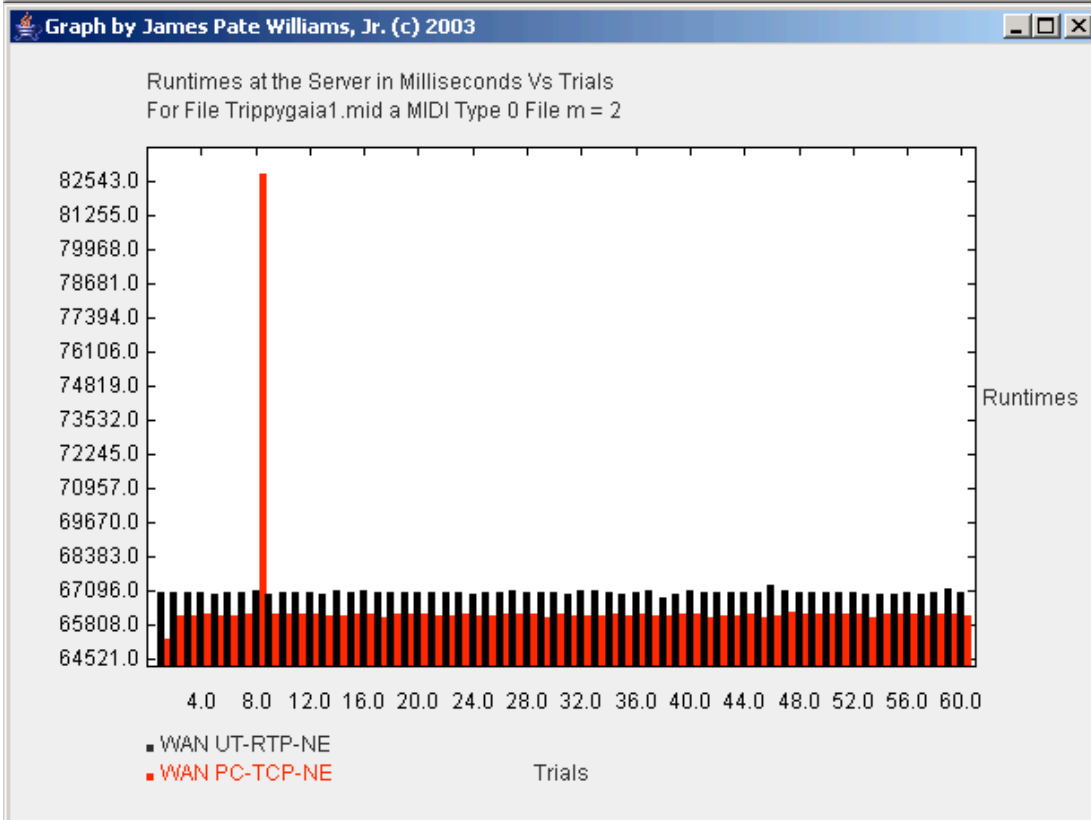
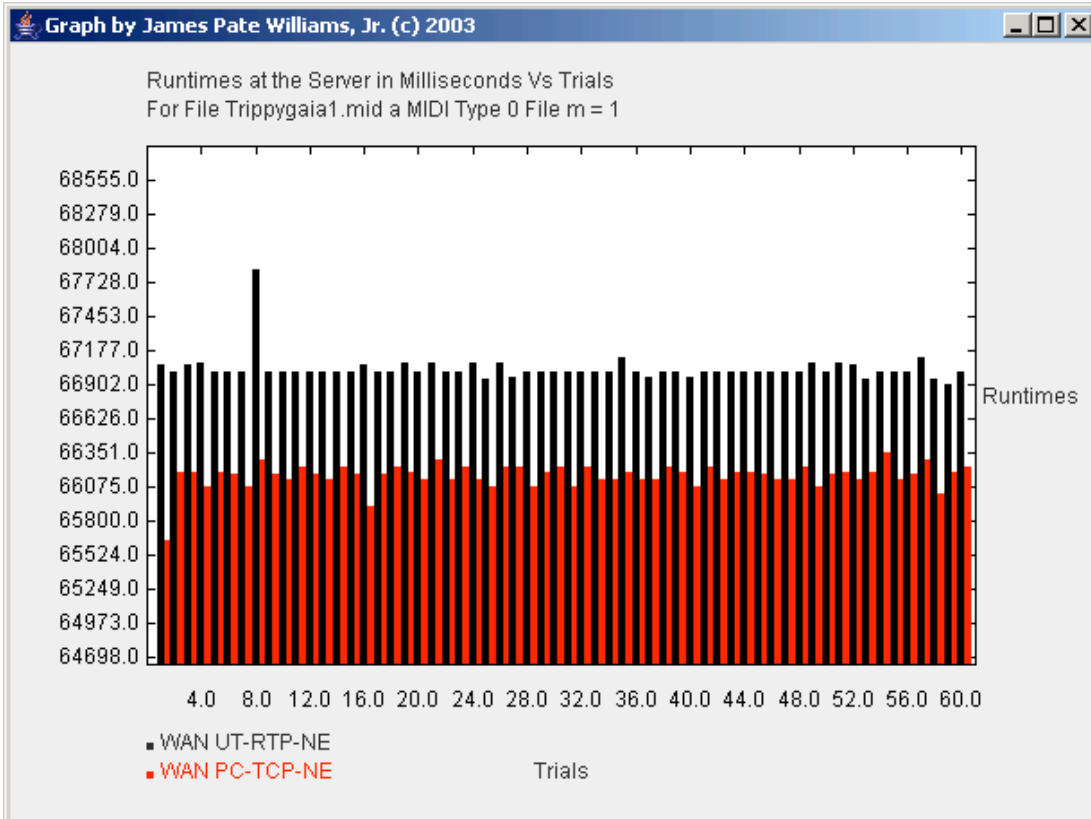


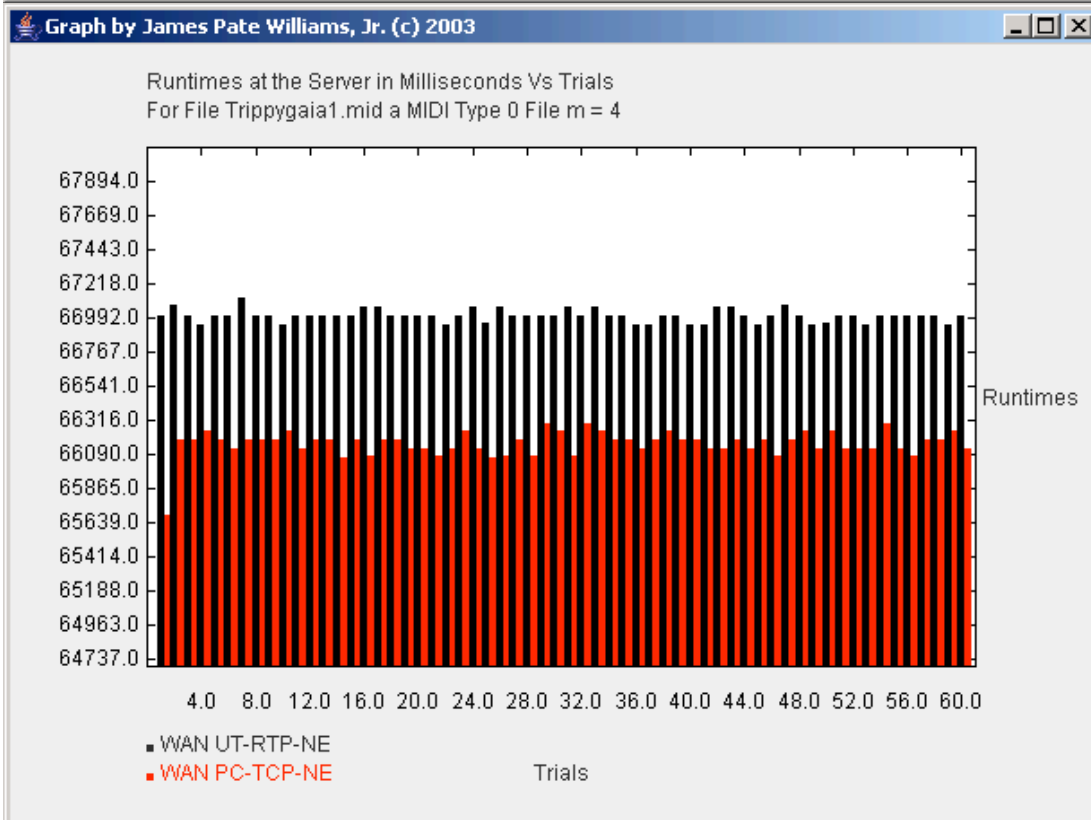
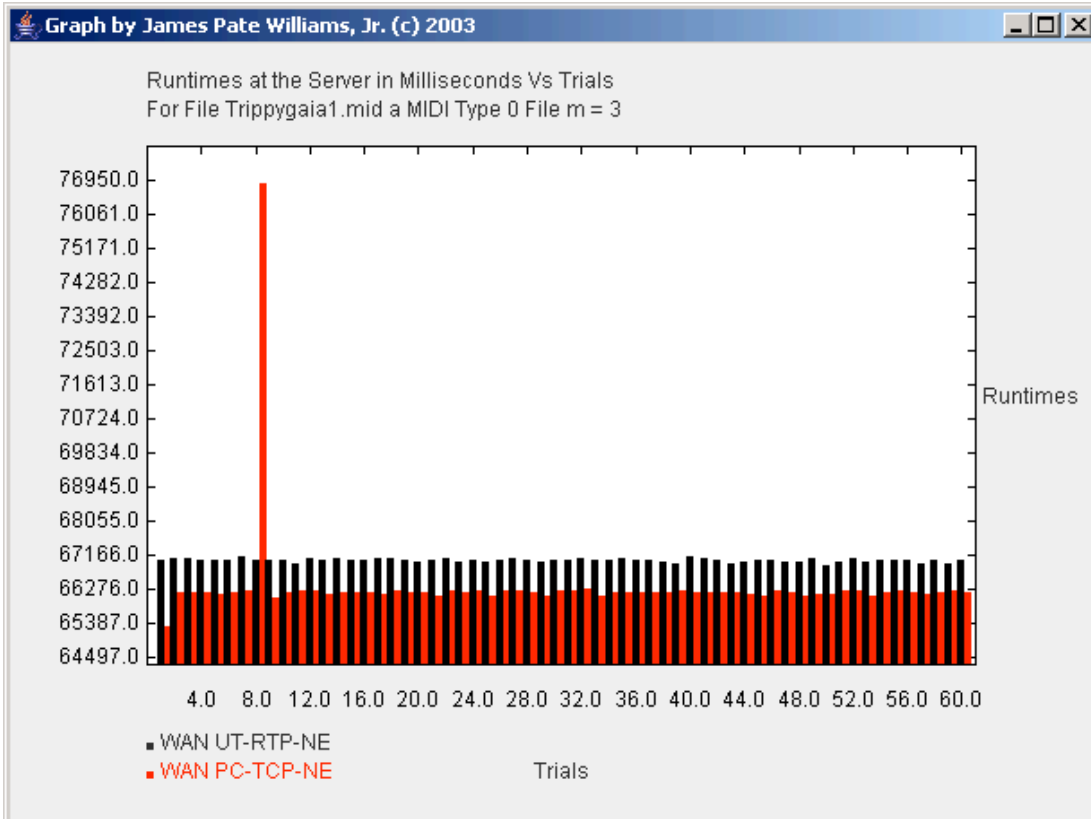




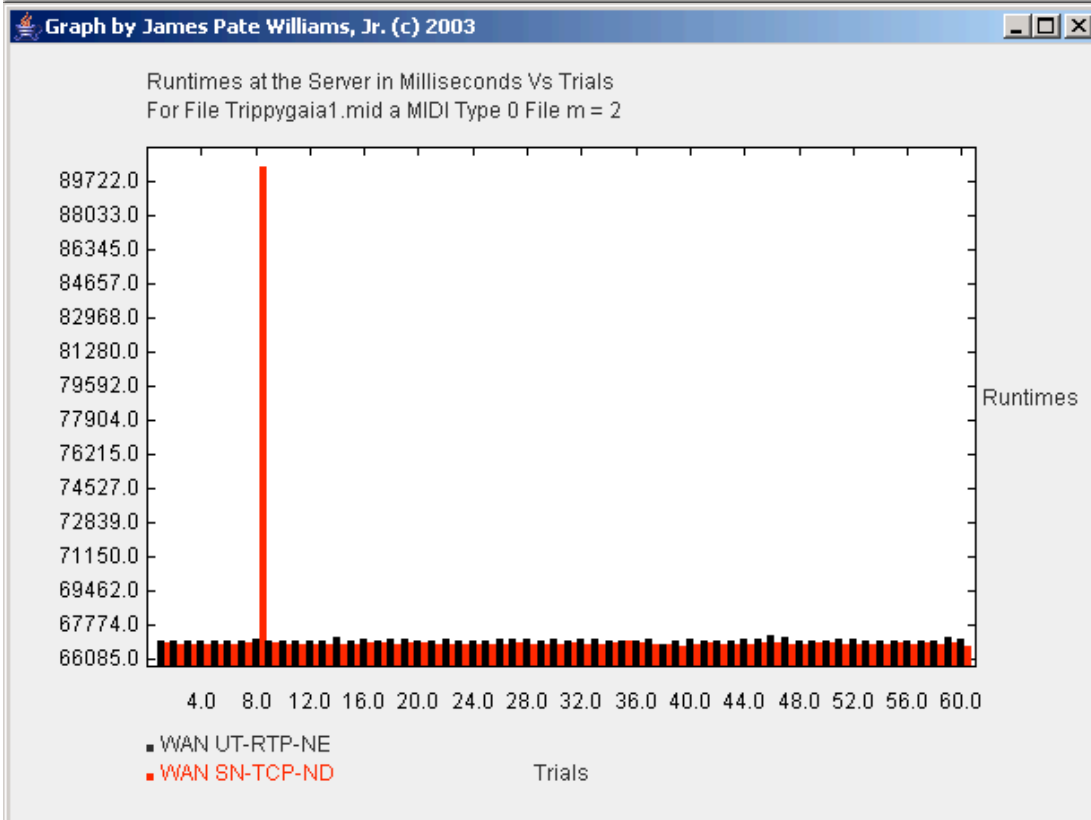
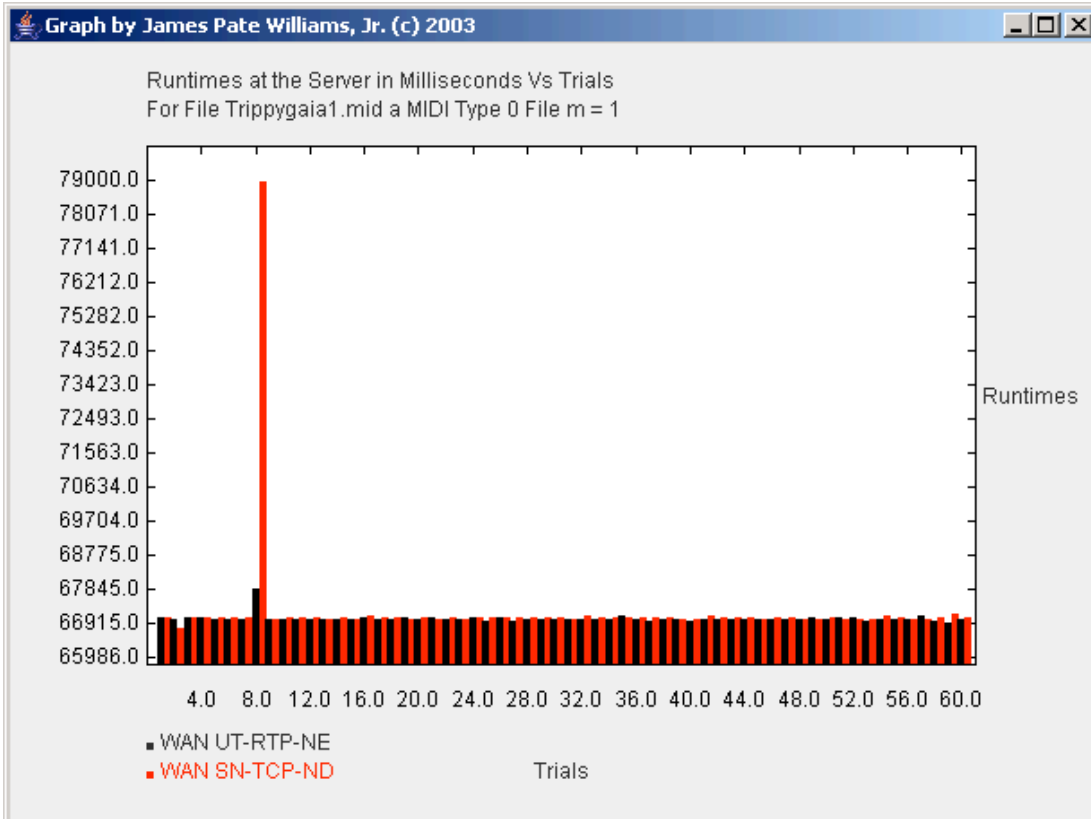


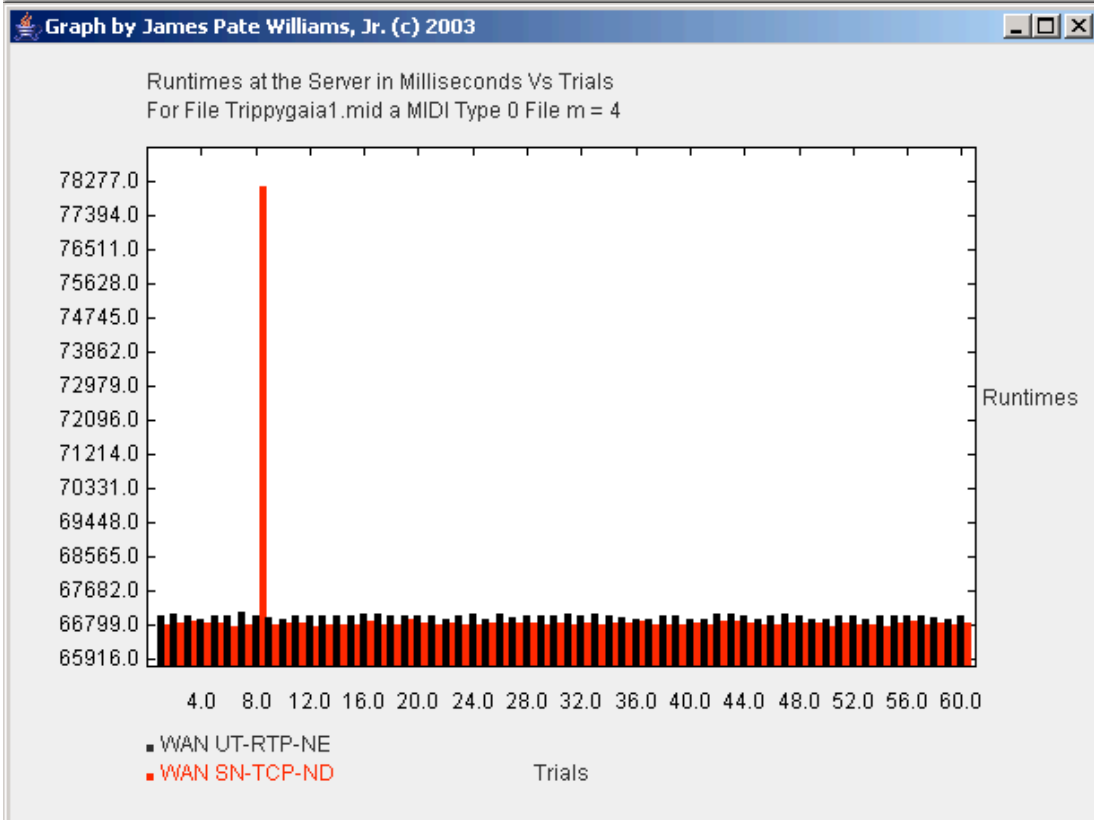
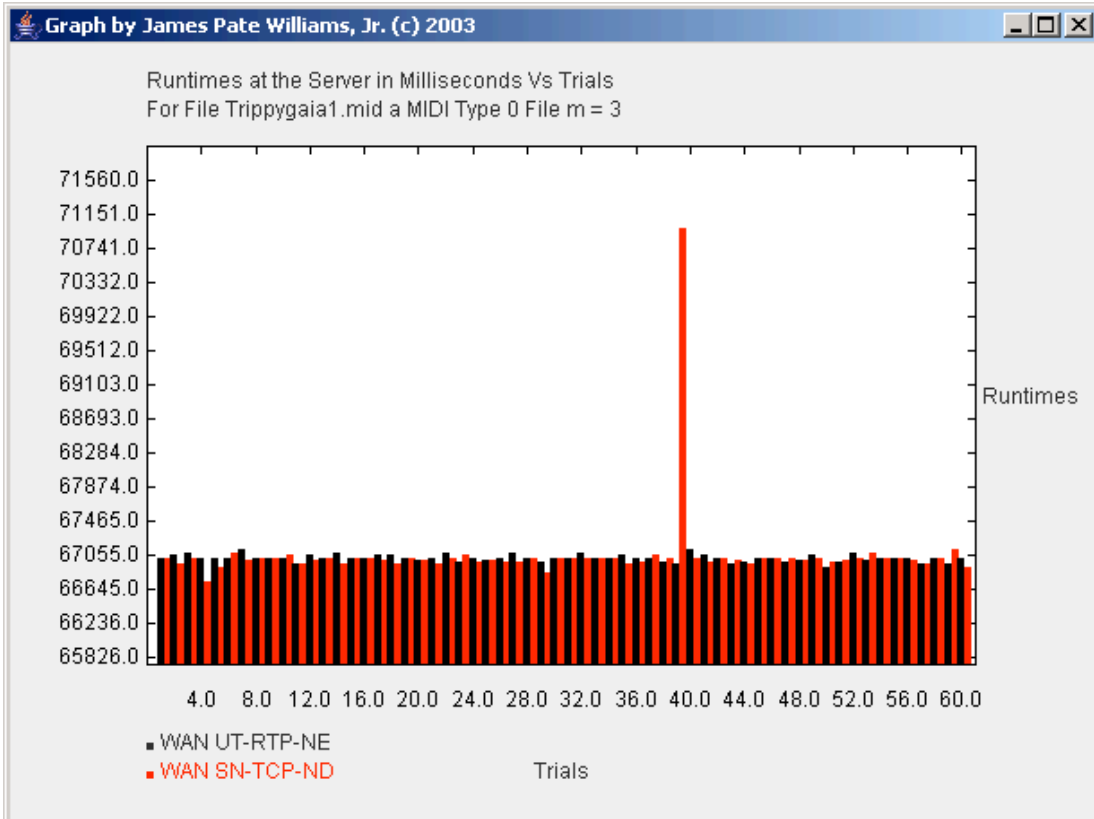


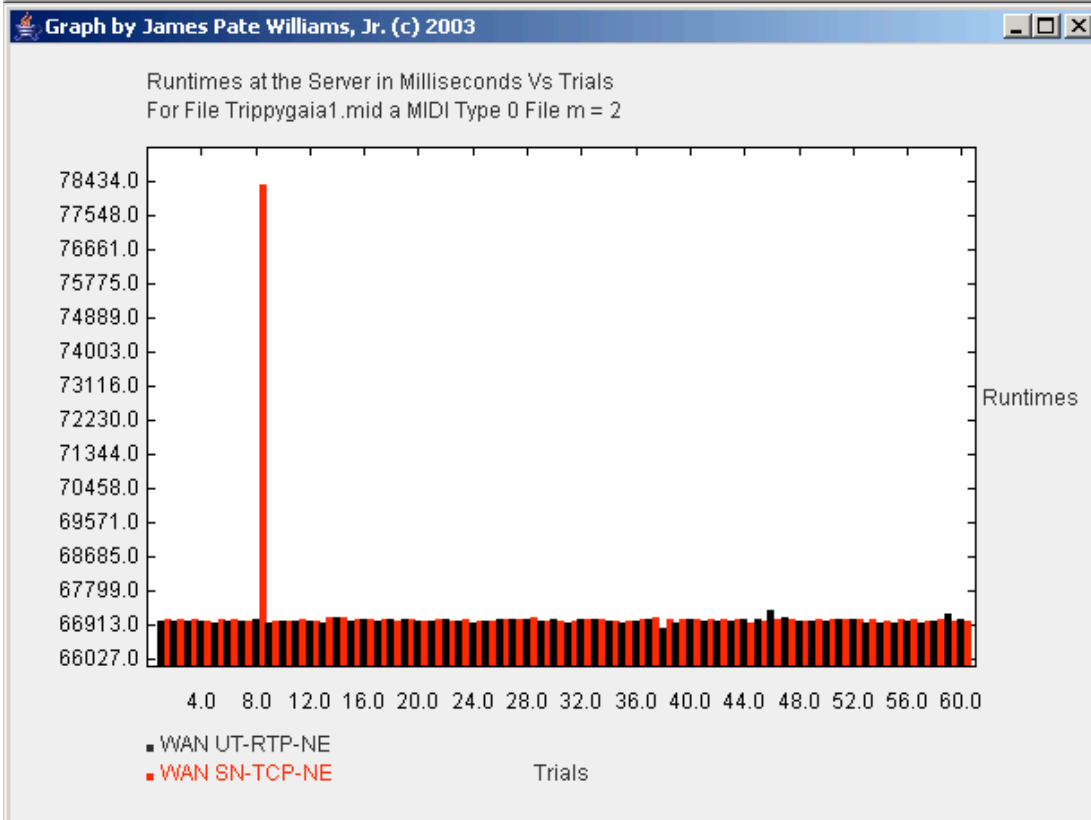
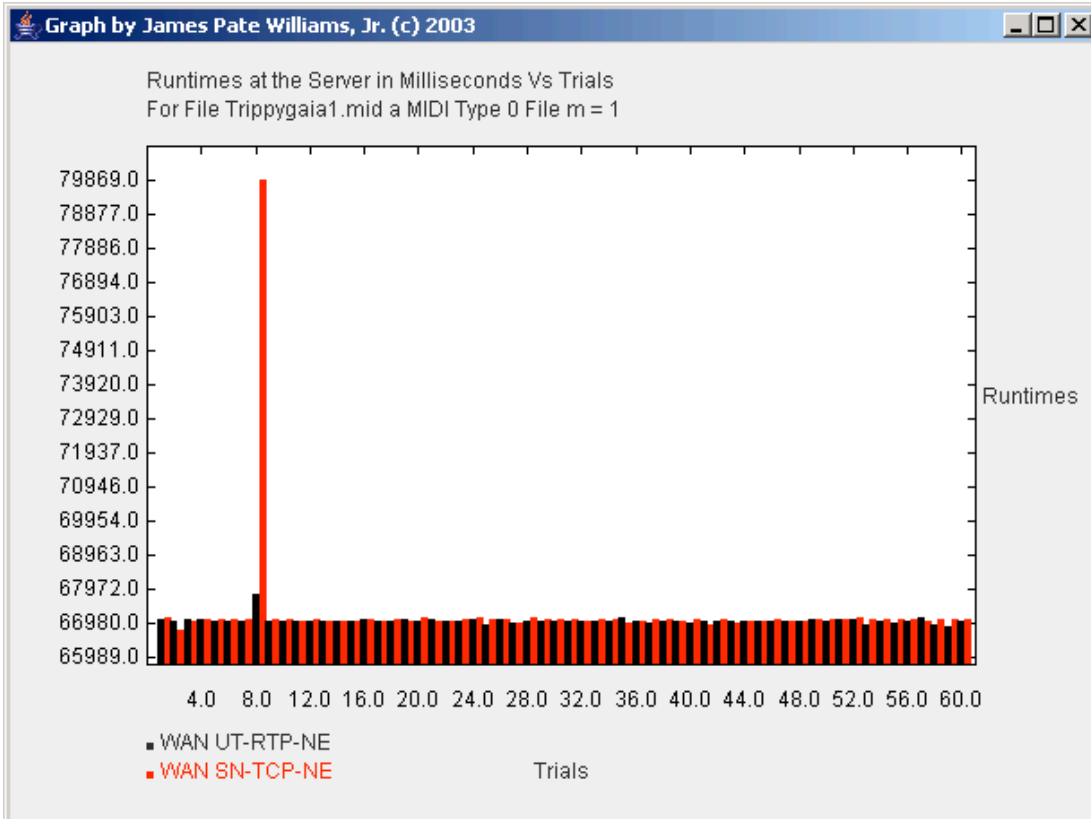


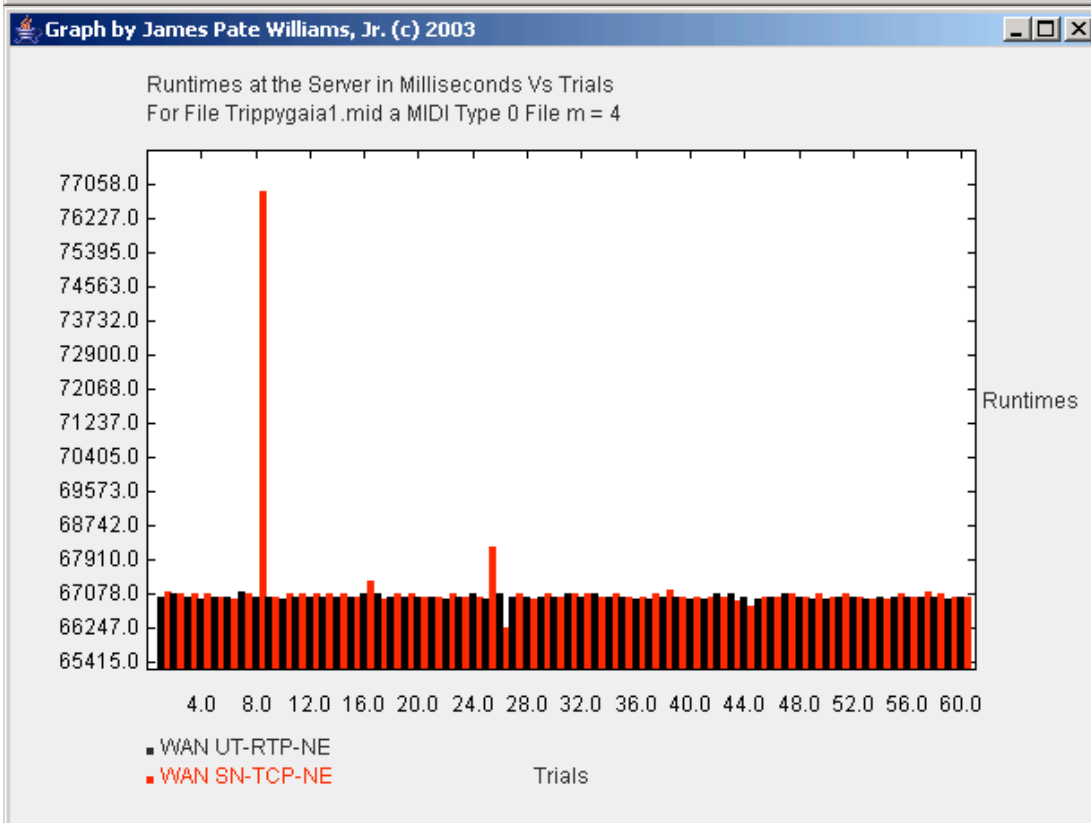
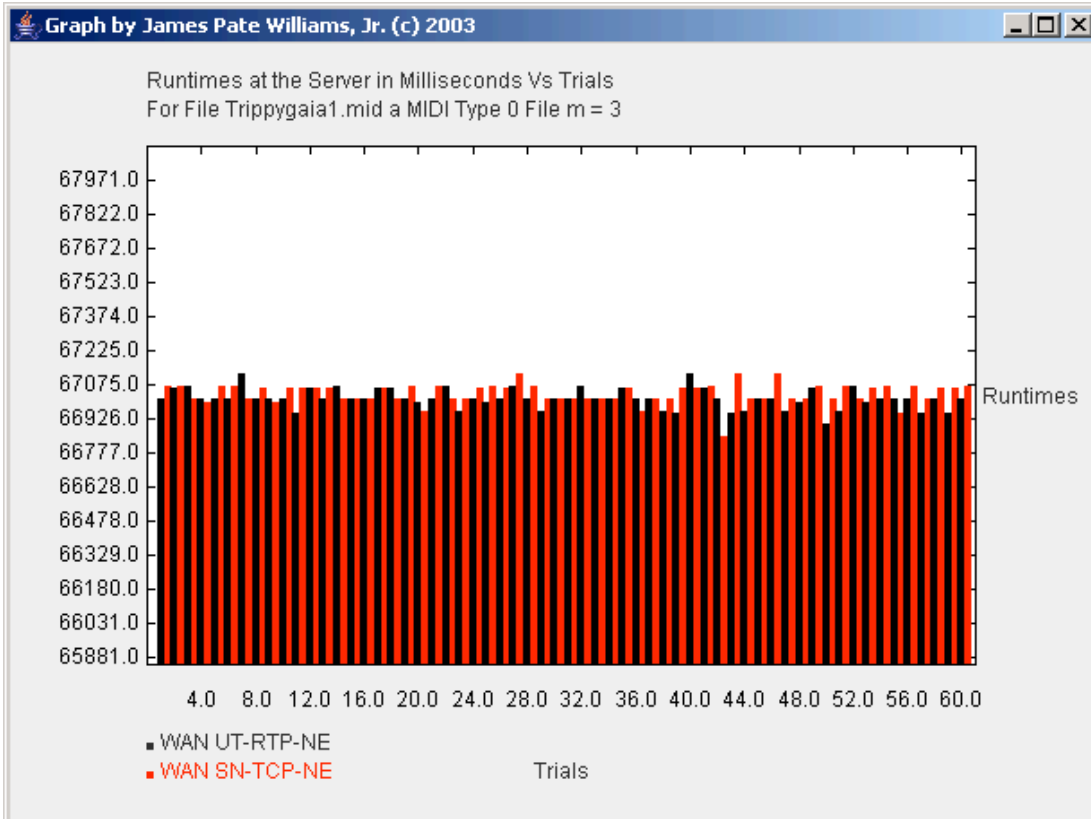


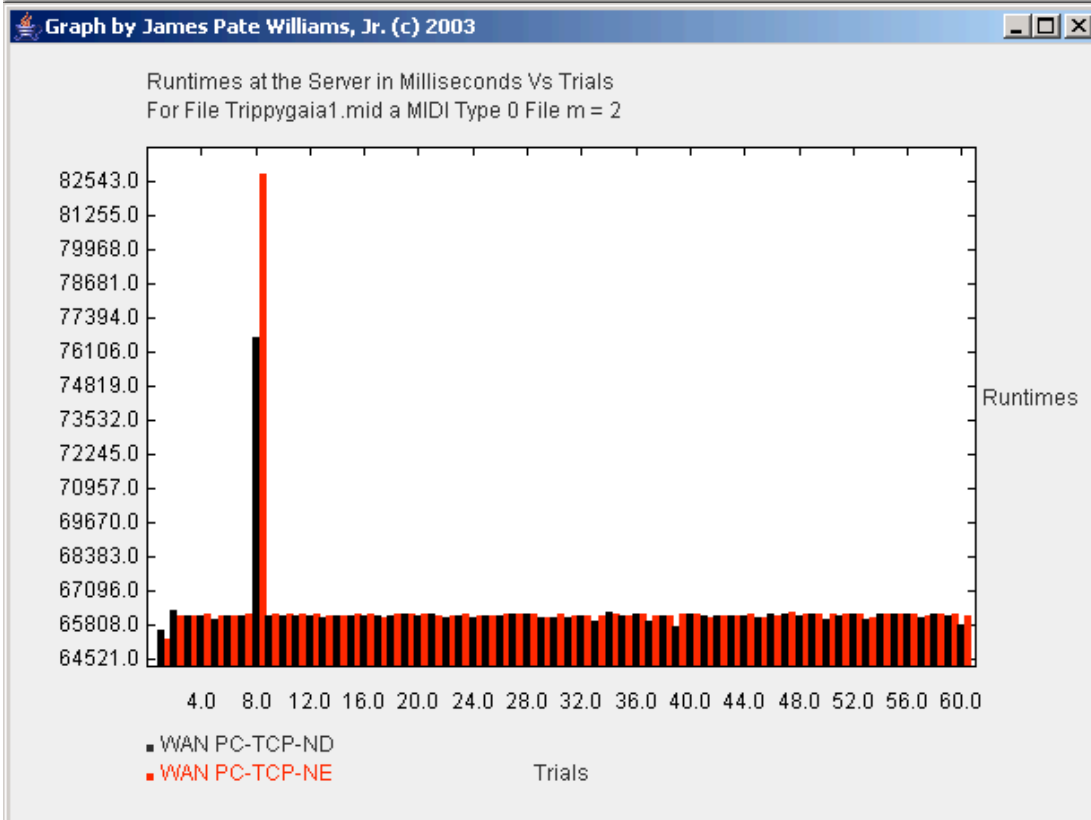


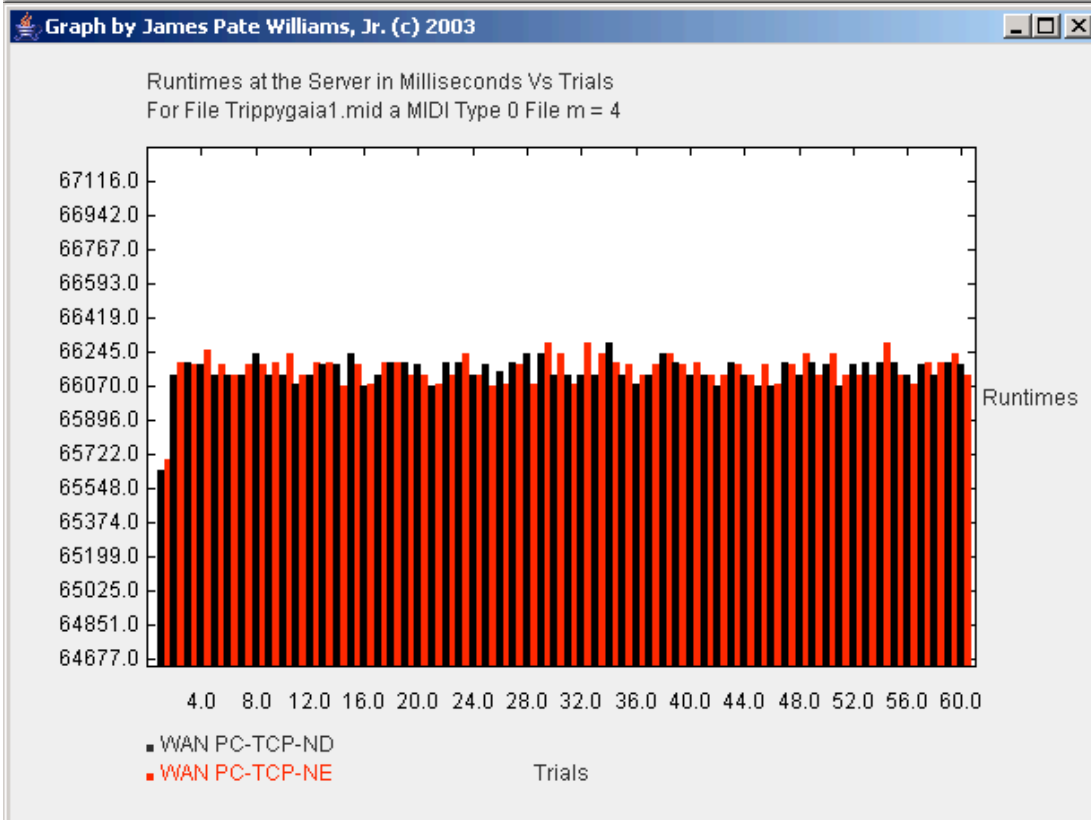
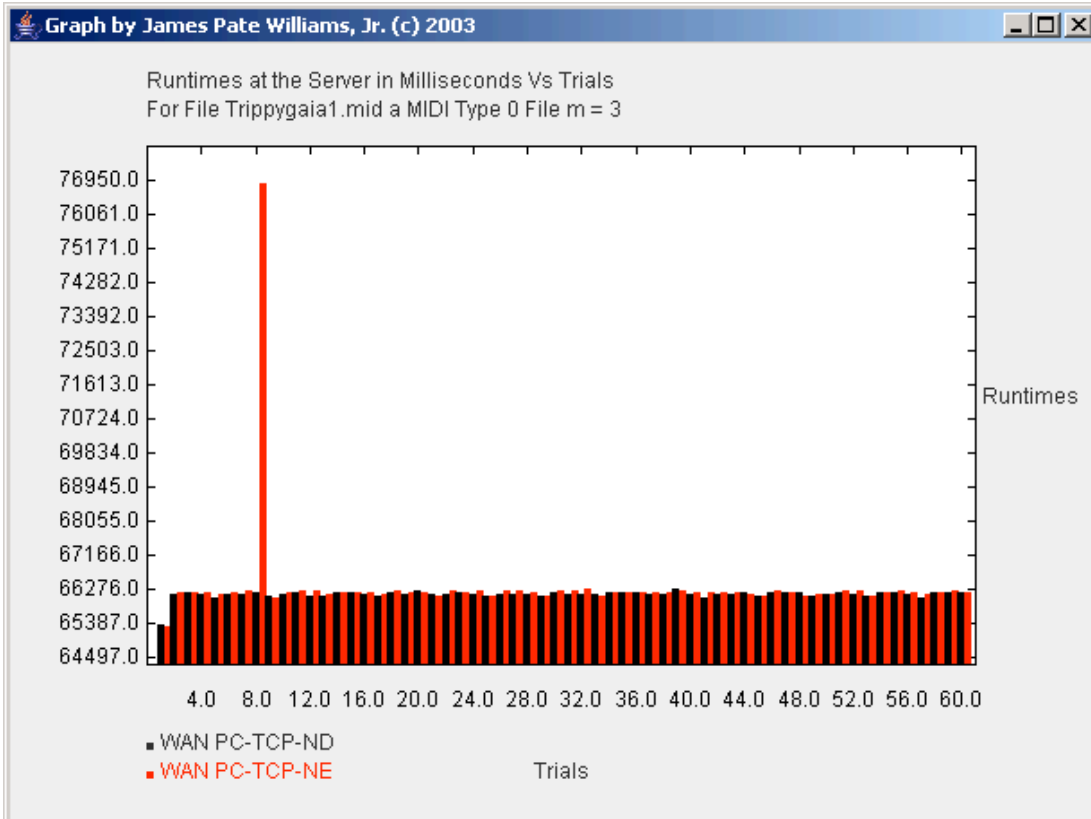


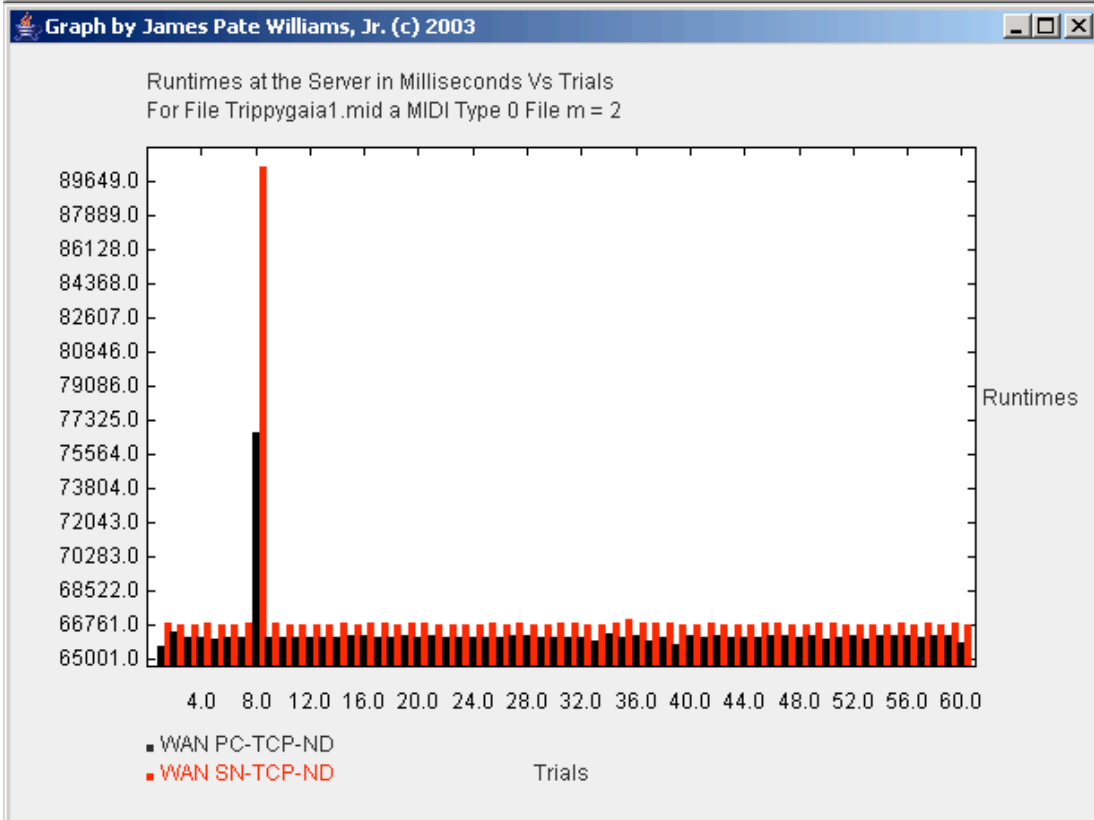
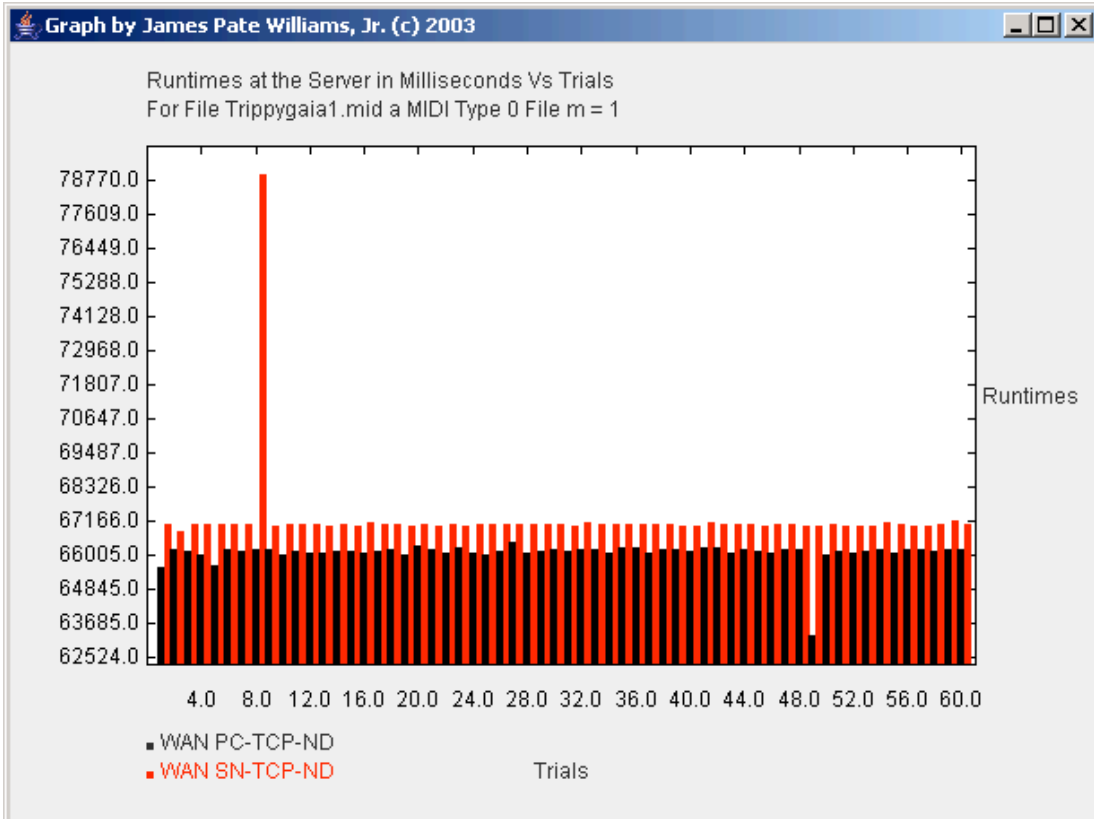


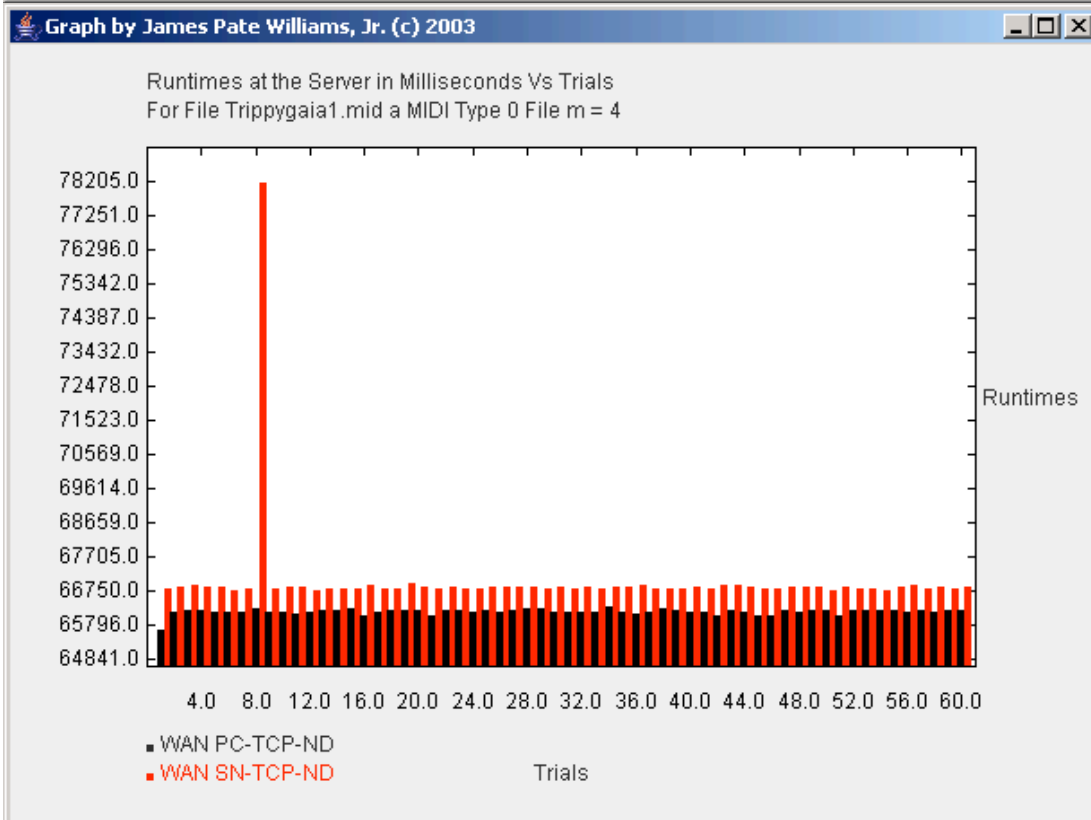
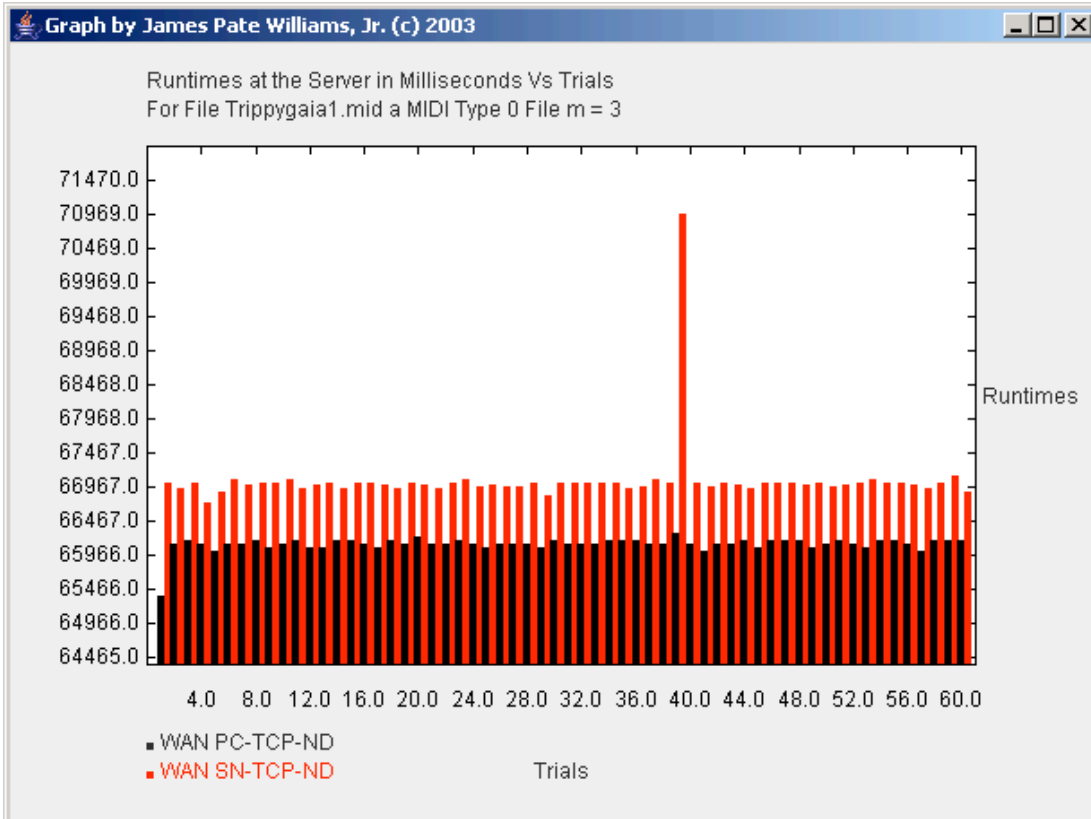




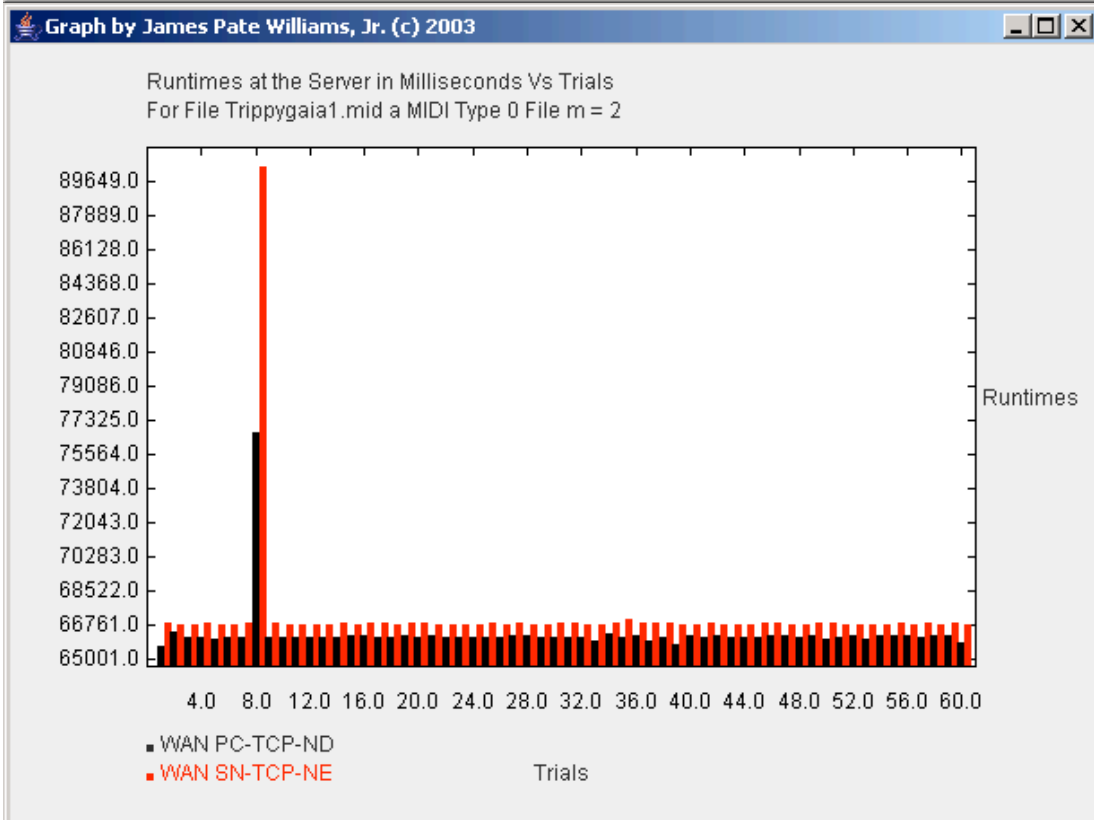
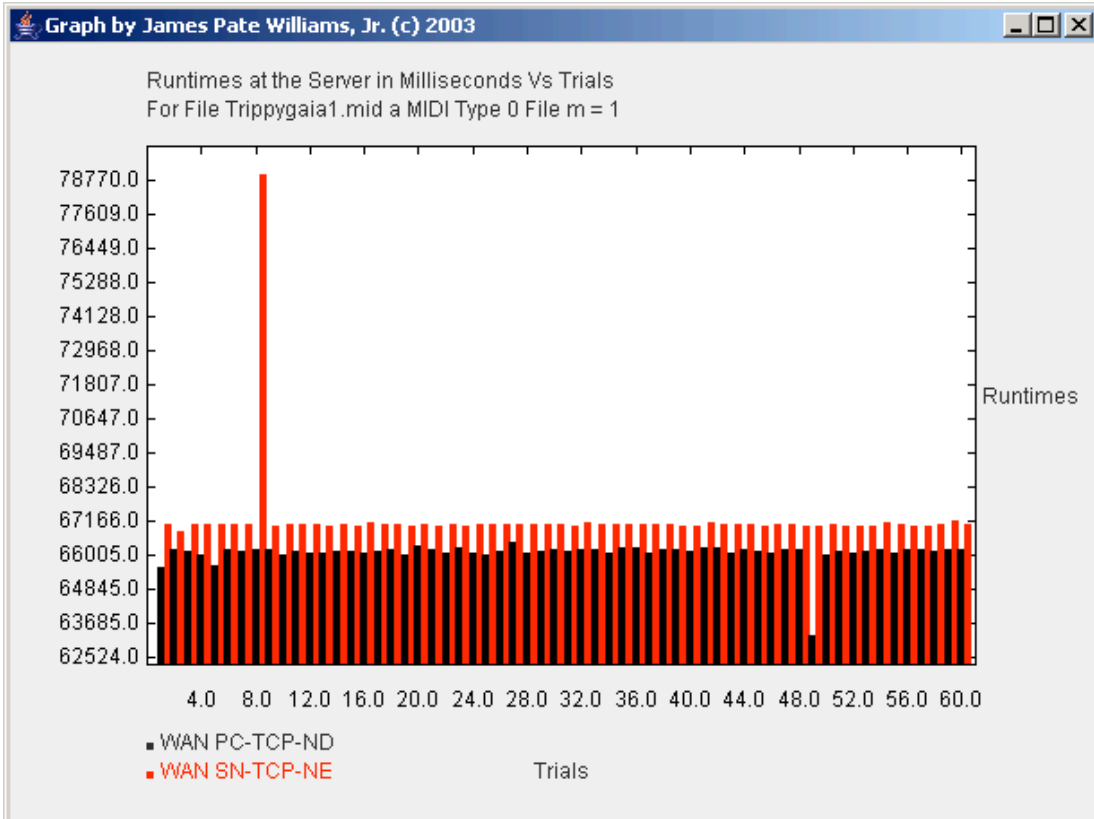


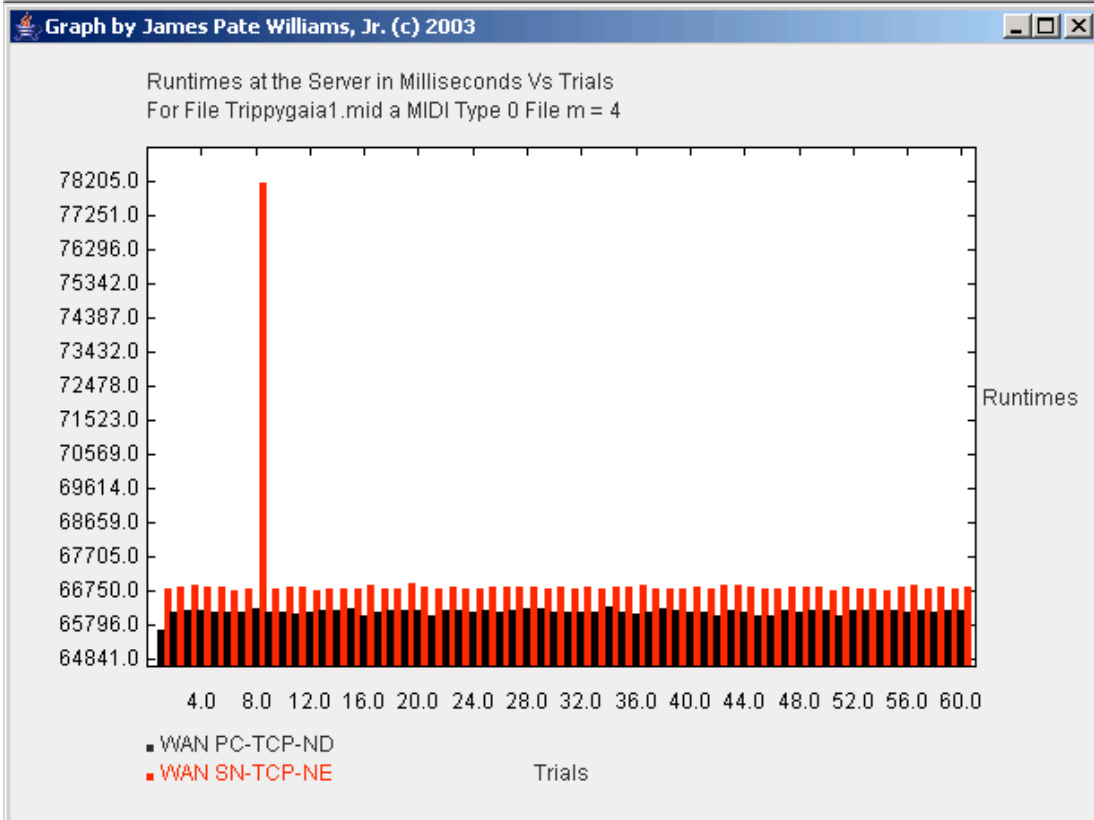
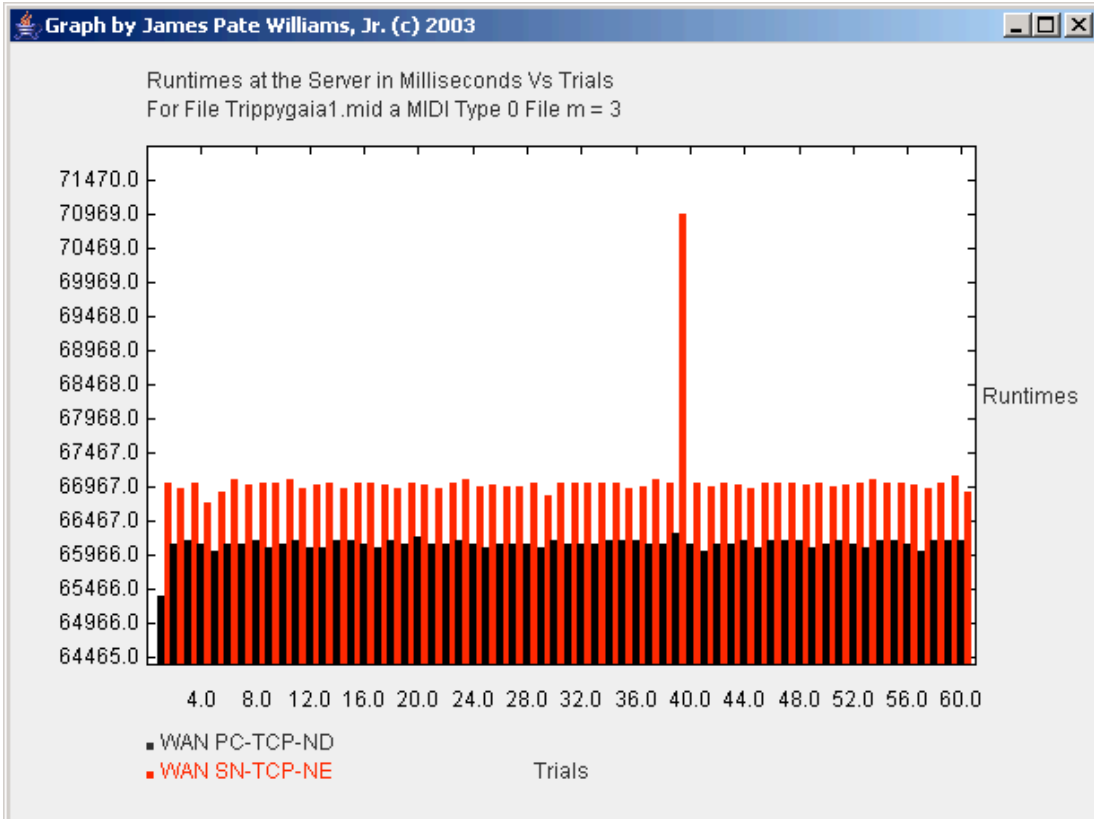


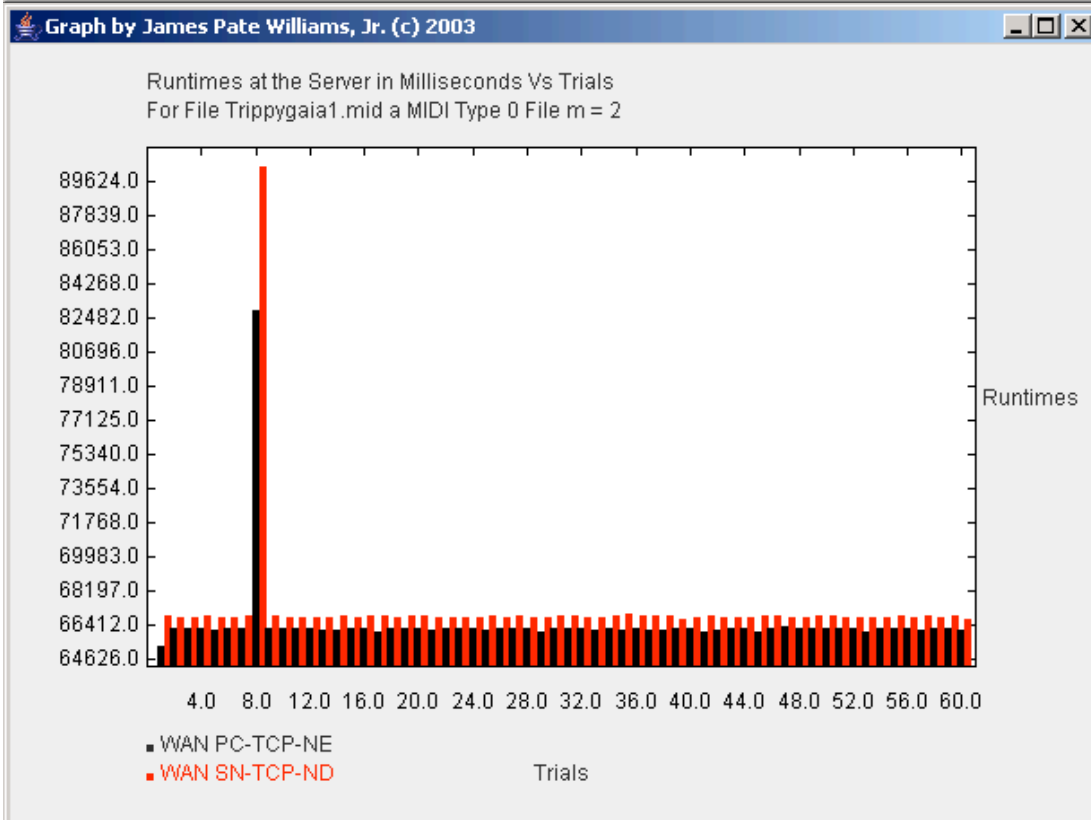
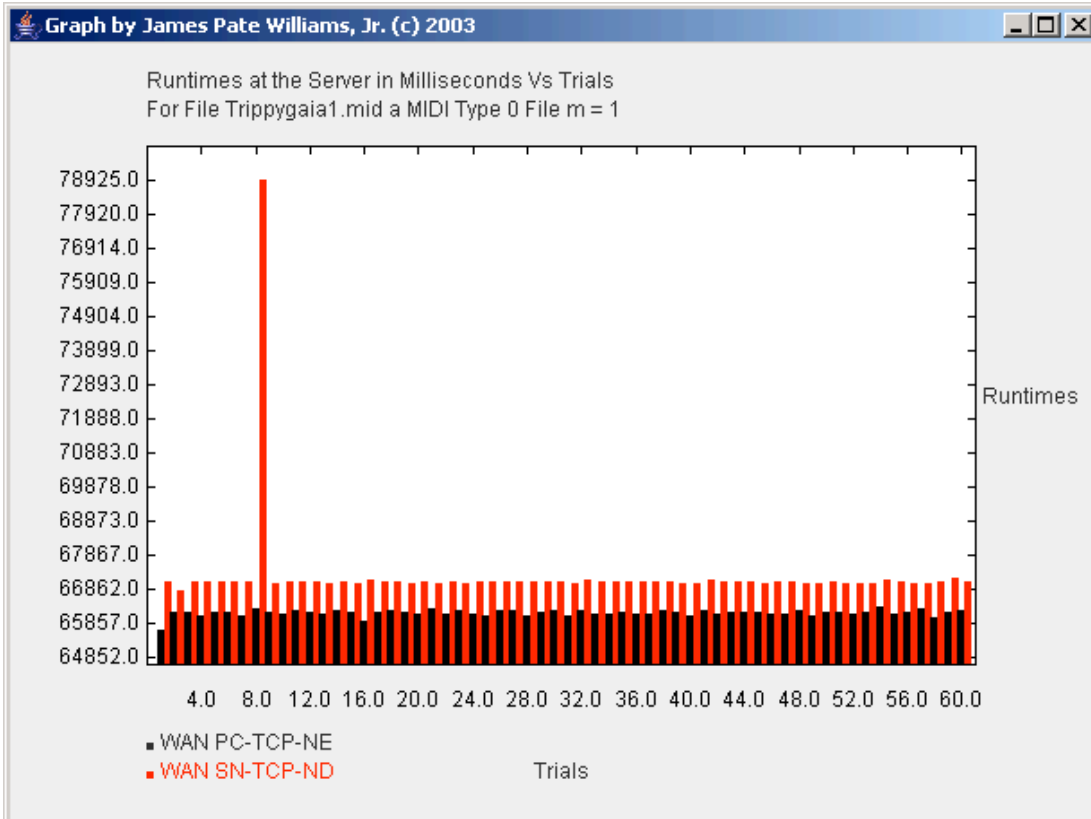


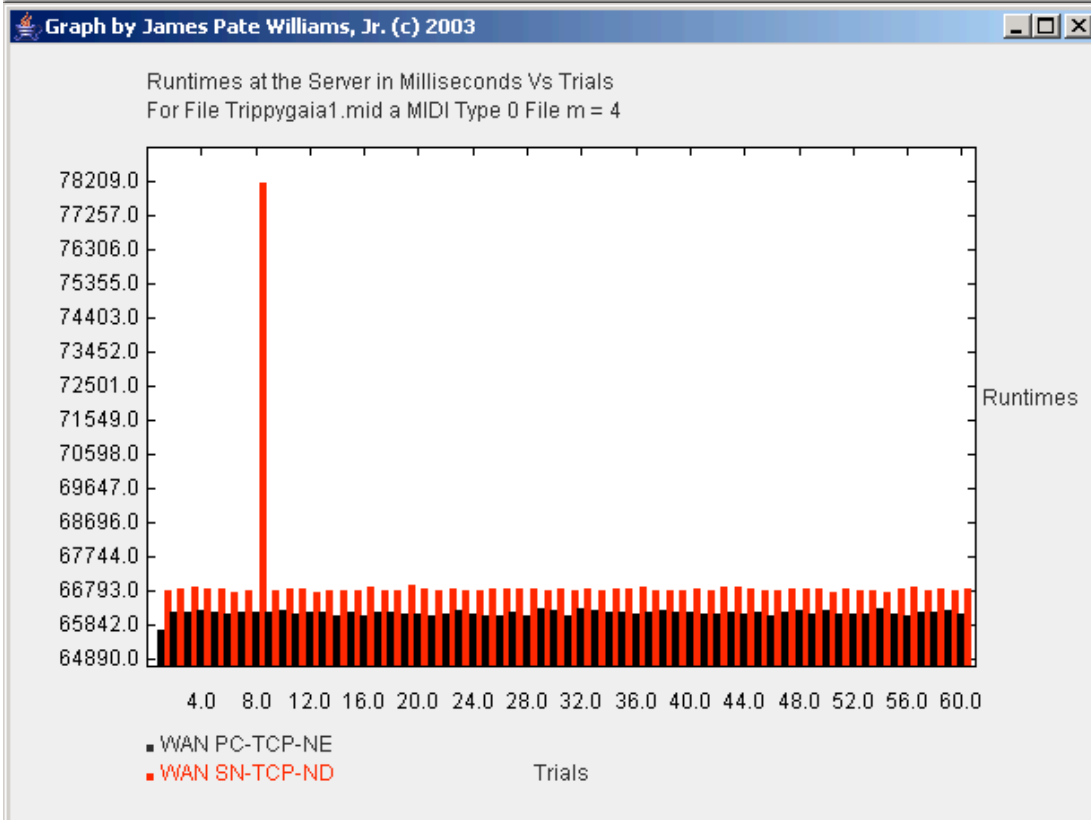
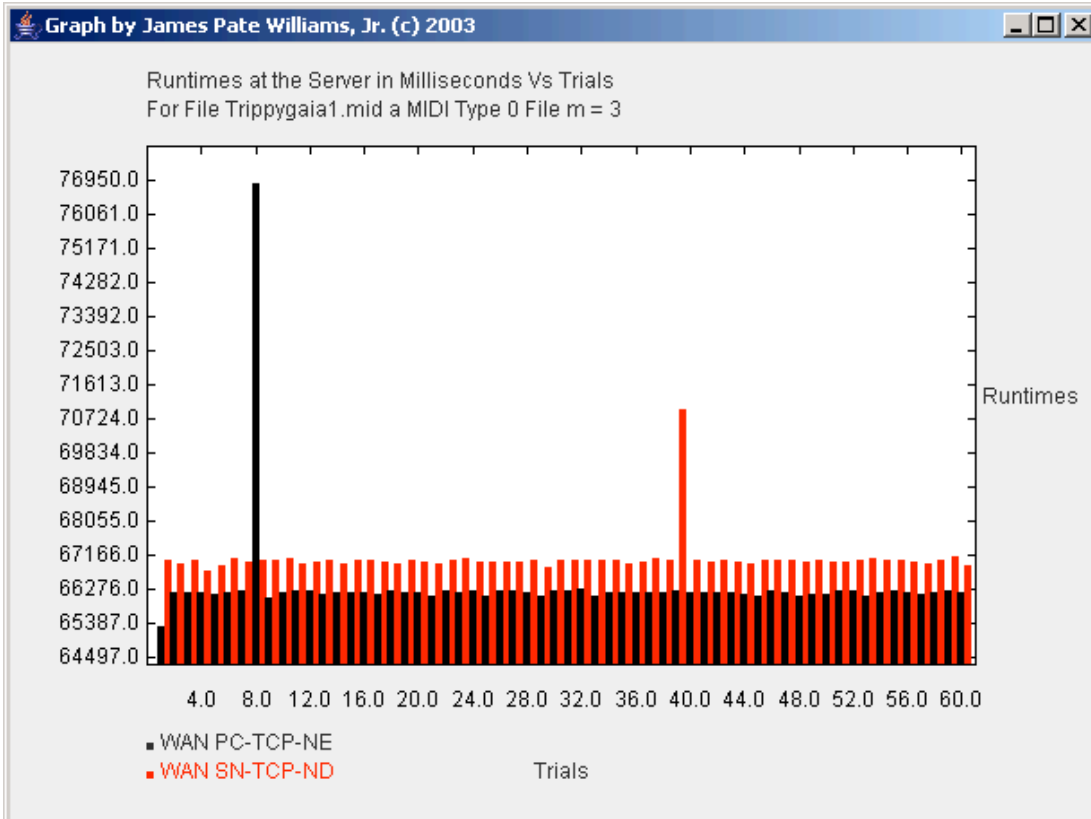


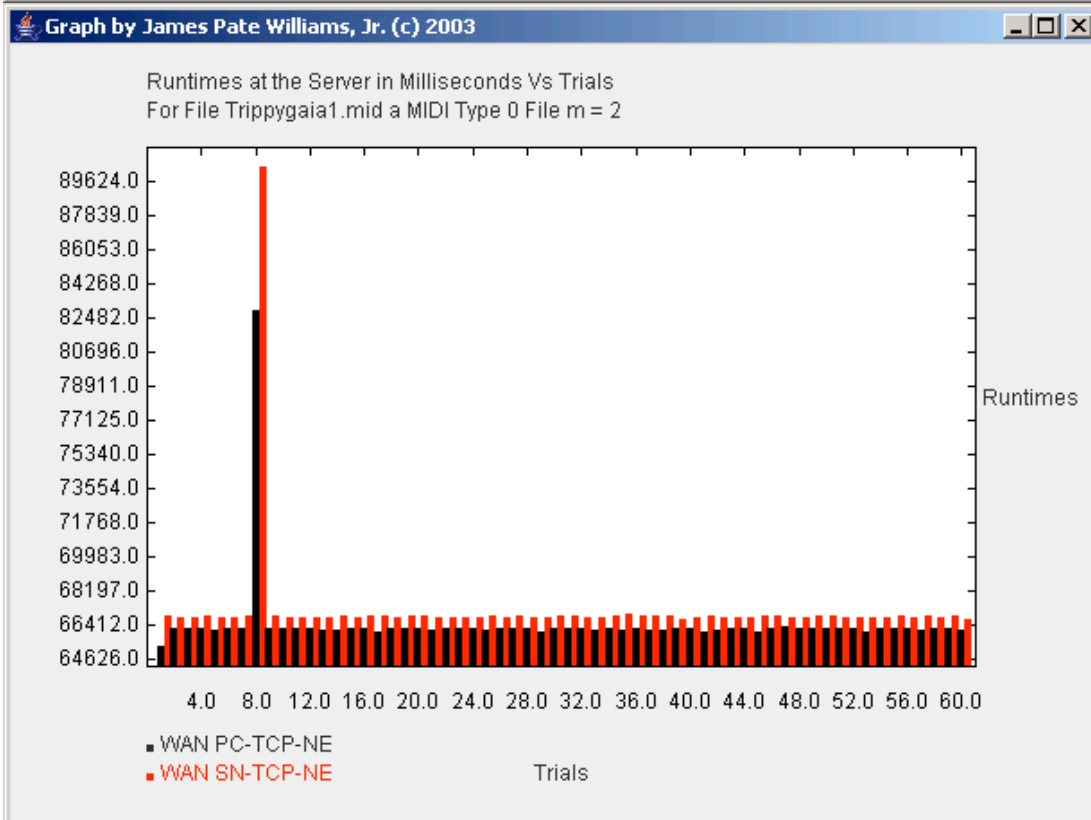
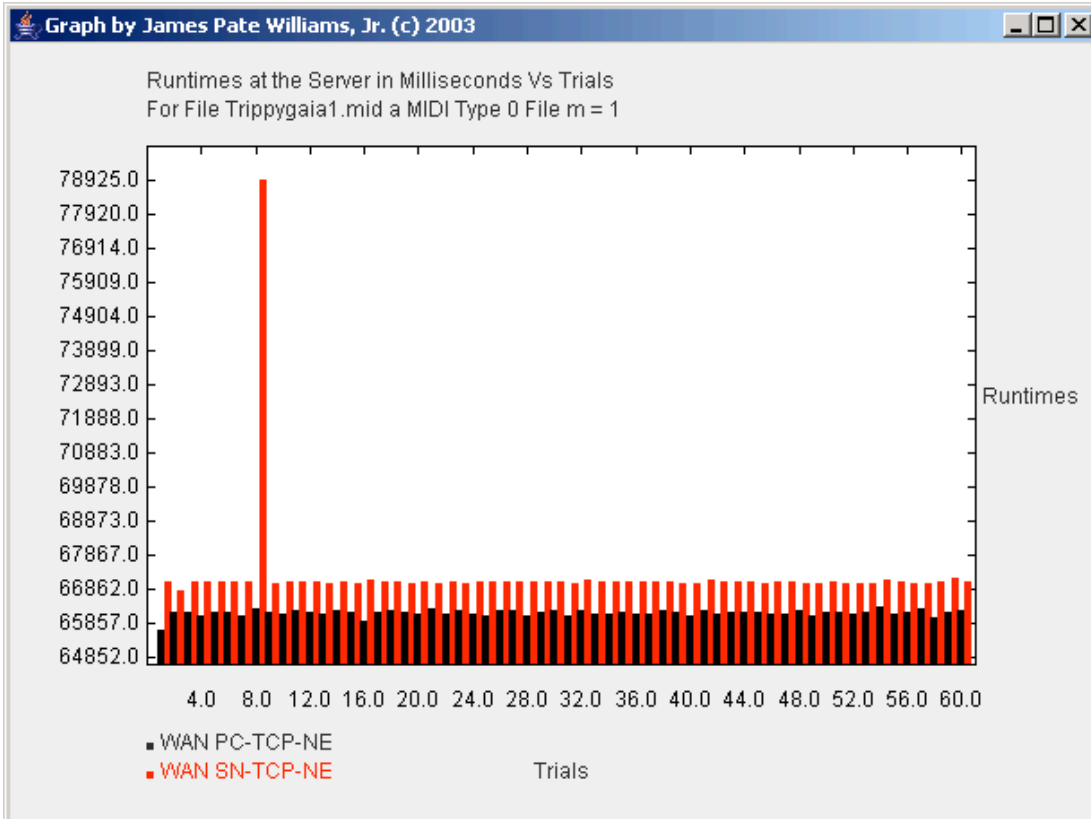


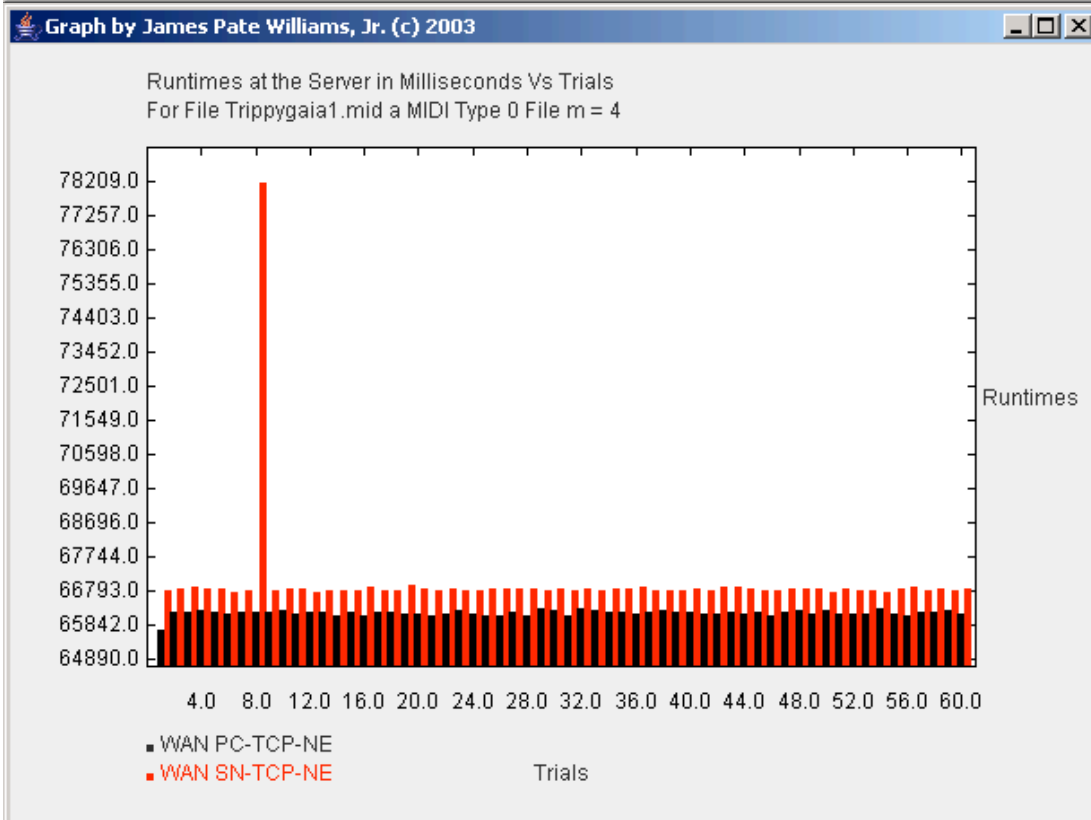
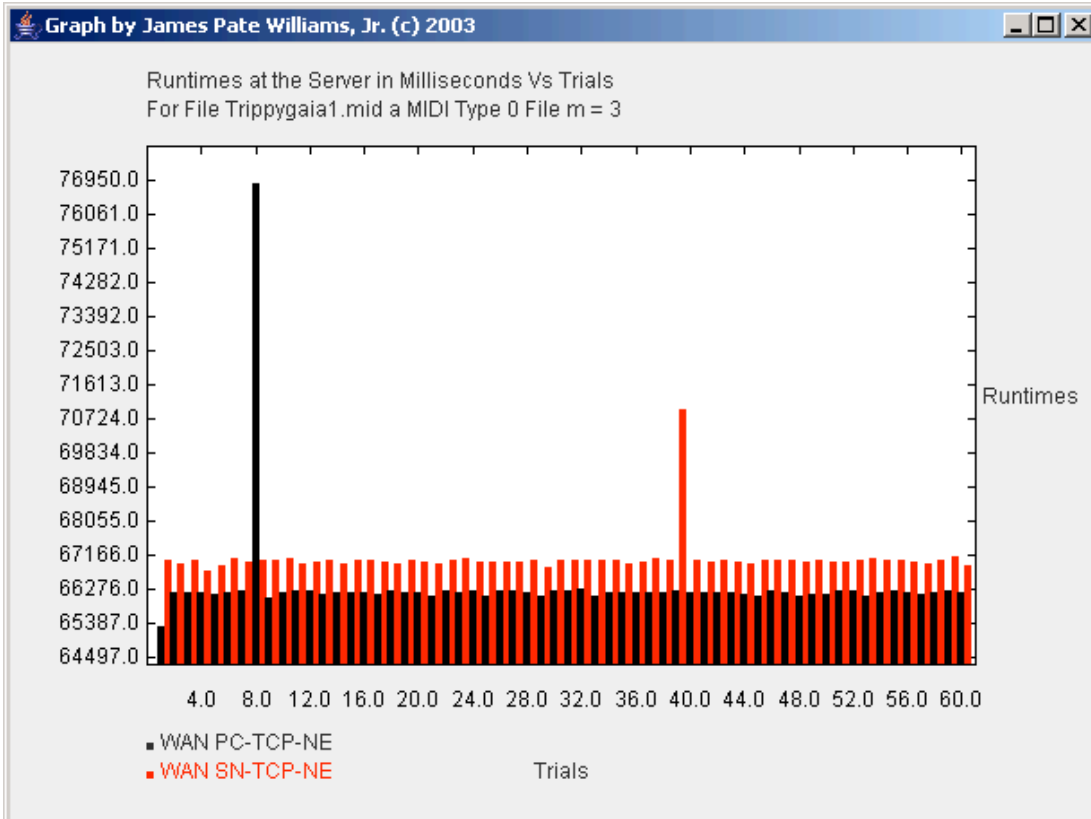


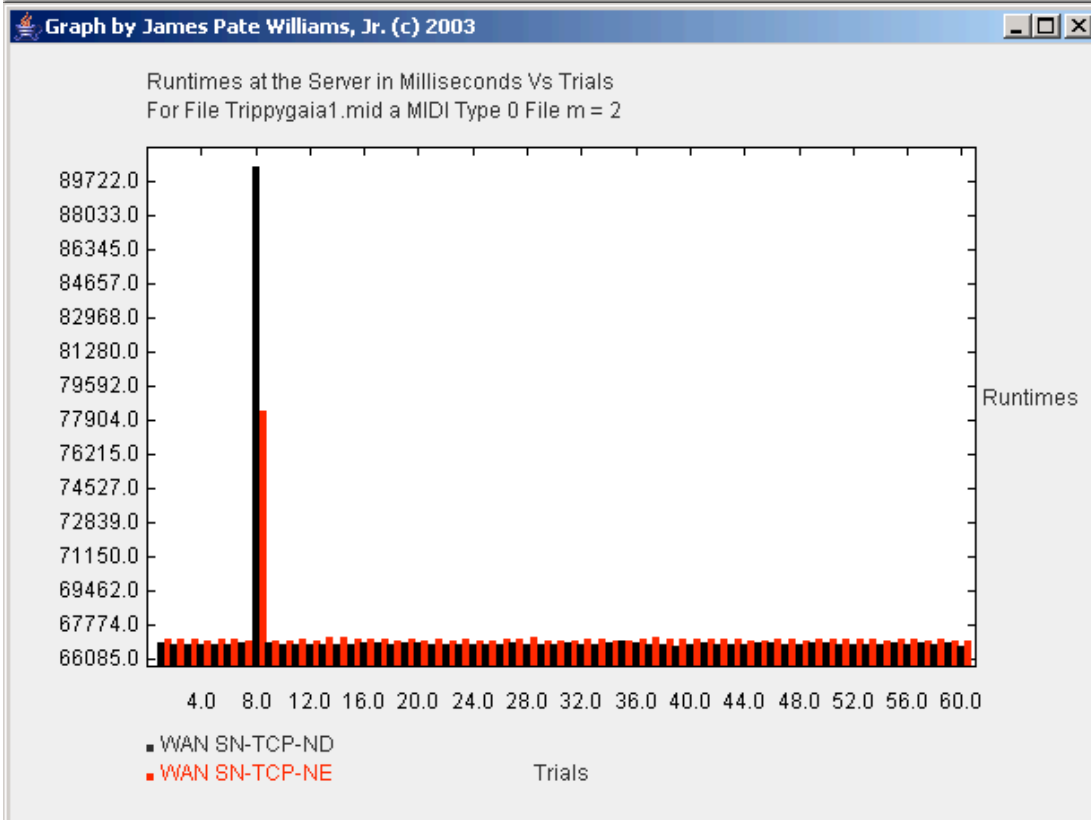
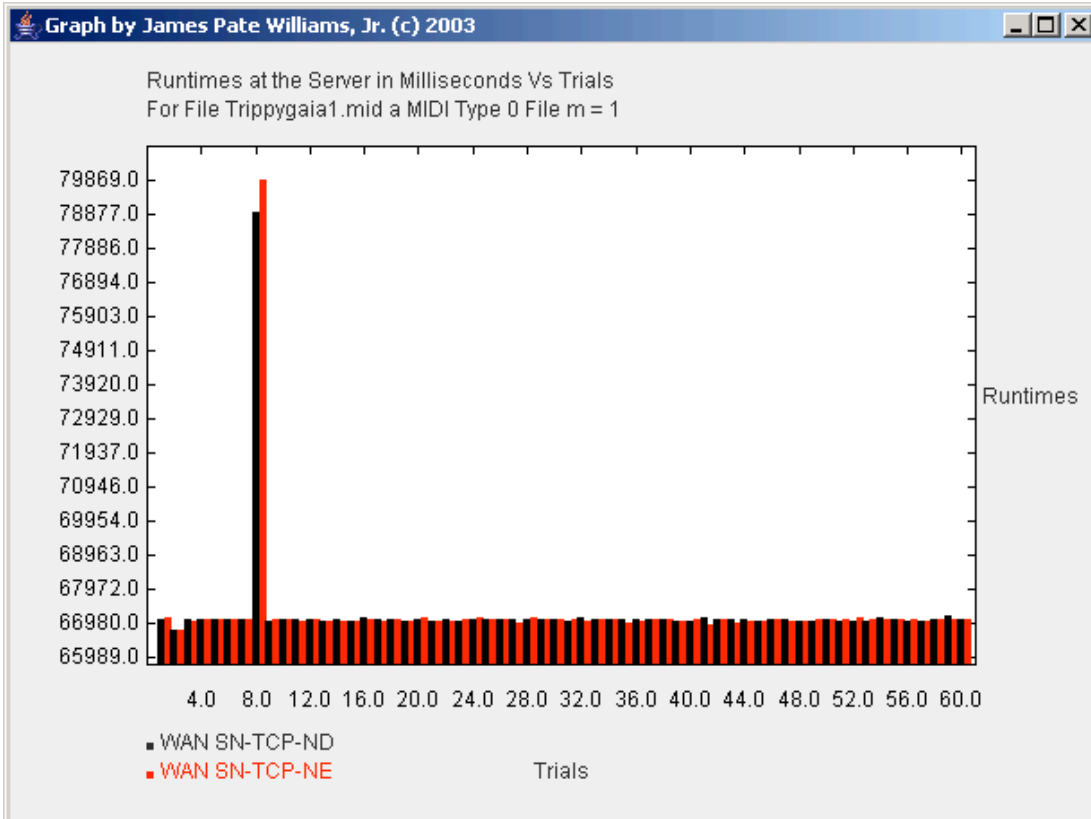


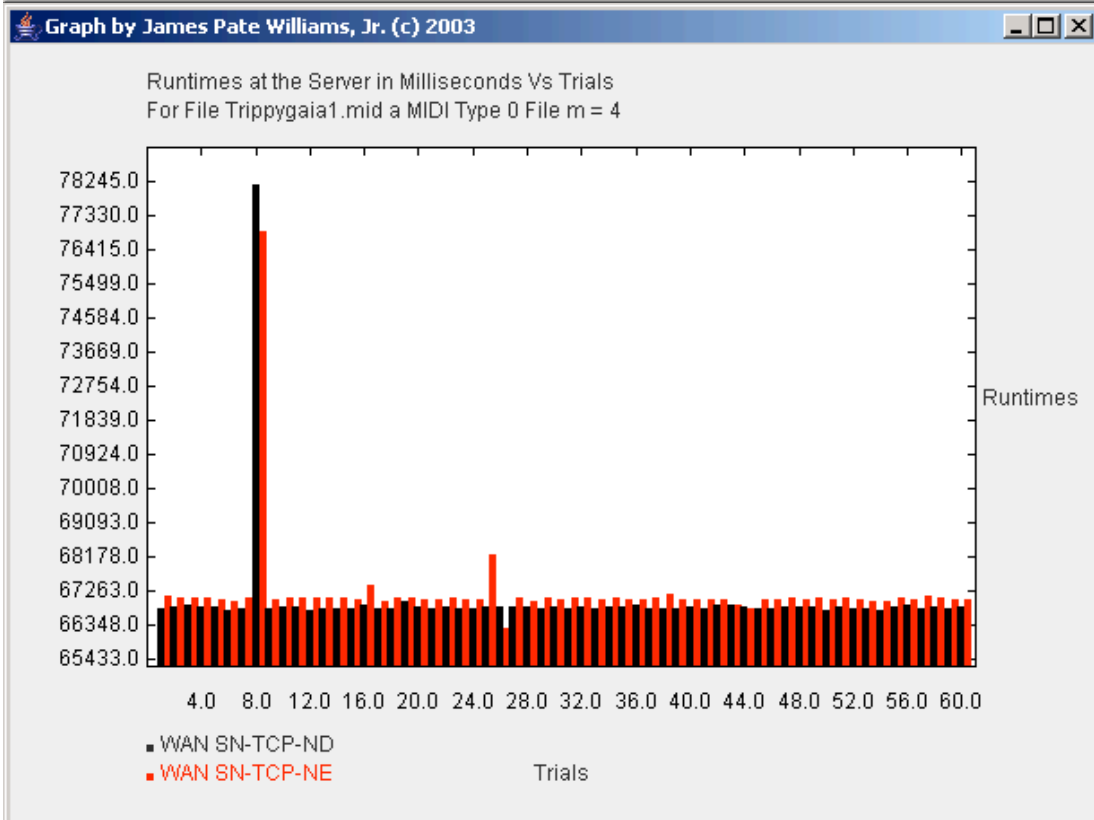
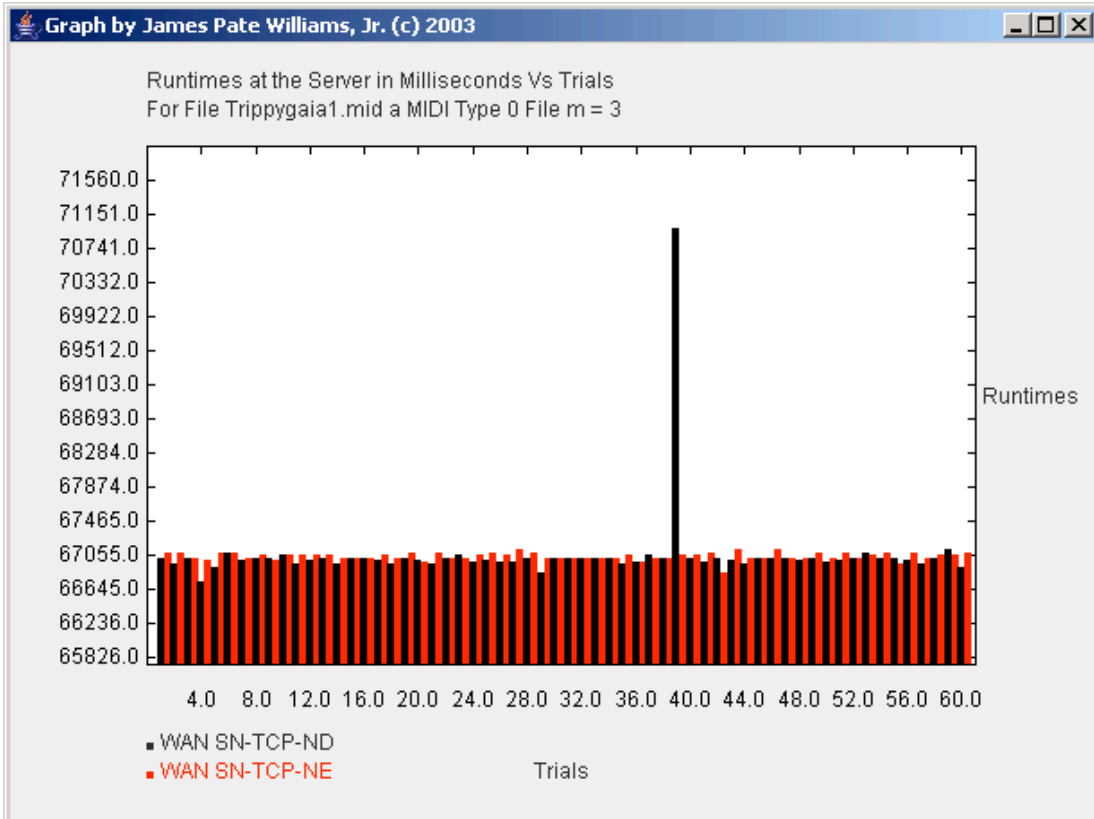














## APPENDIX E LAN PAIRED MEANS COMPARISON STATISTICS

This appendix consists of tables of data that summarize the LAN paired means Student's t-tests. The paired means were dependent since they were measuring the same experimental metric that is run-time of a MIDI sequence at the ultimate destination. The first data in the tables are the values of  $m$ , the number of MIDI short messages per TCP packet. This value varied from 1 to 4. The next two data items are the protocol mnemonics involved in the paired comparison. Then the measured means are given along with their differences. The final three data items are the standard deviation, the Student's t-value, and the Student's t-value significance. A negative t-value meant that the first protocol (protocol #1) was potentially the best protocol in the pair. A positive t-value meant that the second protocol (protocol #2) was potentially statistically superior to the first protocol. If the value of the Student's t-value significance was less than or equal 0.05 then one of the protocols statistically outperformed the other protocol in the pairing.

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NDND	TT-RTP-NDNE	66967.83	67174.67	-206.83	1553.33	-1.0314	0.3066
2	TT-RTP-NDND	TT-RTP-NDNE	66977.50	66970.00	+7.50	41.77	+1.3909	0.1695
3	TT-RTP-NDND	TT-RTP-NDNE	67037.17	67198.00	-160.83	1825.76	-0.6824	0.4977
4	TT-RTP-NDND	TT-RTP-NDNE	66965.00	67333.67	-368.67	2820.63	-1.0124	0.3155
1	TT-RTP-NDND	TT-RTP-NEND	66967.83	66970.83	-3.00	54.81	-0.4240	0.6731
2	TT-RTP-NDND	TT-RTP-NEND	66977.50	66984.33	-6.83	42.96	-1.2320	0.2228
3	TT-RTP-NDND	TT-RTP-NEND	67037.17	67096.67	-59.50	1087.14	-0.4239	0.6731
4	TT-RTP-NDND	TT-RTP-NEND	66965.00	66976.33	-11.33	51.99	-1.6884	0.0966
1	TT-RTP-NDND	TT-RTP-NENE	66967.83	67069.17	-101.33	732.62	-1.0714	0.2884
2	TT-RTP-NDND	TT-RTP-NENE	66977.50	66968.67	+8.83	37.83	+1.8089	0.0756
3	TT-RTP-NDND	TT-RTP-NENE	67037.17	66981.67	+55.50	502.58	+0.8554	0.3958
4	TT-RTP-NDND	TT-RTP-NENE	66965.00	66969.50	-4.50	46.34	-0.7522	0.4549
1	TT-RTP-NDND	UT-RTP-ND	66967.83	66930.17	+37.67	145.23	+2.0089	0.0491
2	TT-RTP-NDND	UT-RTP-ND	66977.50	67093.50	-116.00	1226.80	-0.7324	0.4668
3	TT-RTP-NDND	UT-RTP-ND	67037.17	66984.00	+53.17	605.99	+0.6796	0.4994
4	TT-RTP-NDND	UT-RTP-ND	66965.00	67018.83	-53.83	641.14	-0.6504	0.5180
1	TT-RTP-NDND	UT-RTP-NE	66967.83	67175.17	-207.33	1768.18	-0.9083	0.3674
2	TT-RTP-NDND	UT-RTP-NE	66977.50	66960.83	+16.67	45.05	+2.8658	0.0058
3	TT-RTP-NDND	UT-RTP-NE	67037.17	66959.50	+77.67	525.71	+1.1444	0.2571
4	TT-RTP-NDND	UT-RTP-NE	66965.00	66903.83	+61.17	82.61	+5.7355	0.0000
1	TT-RTP-NDND	PC-TCP-ND	66967.83	66345.33	+622.50	1001.45	+4.8149	0.0000
2	TT-RTP-NDND	PC-TCP-ND	66977.50	66395.33	+582.17	637.34	+7.0754	0.0000
3	TT-RTP-NDND	PC-TCP-ND	67037.17	66472.17	+565.00	1479.38	+2.9583	0.0044
4	TT-RTP-NDND	PC-TCP-ND	66965.00	66069.67	+895.33	387.33	+17.9054	0.0000
1	TT-RTP-NDND	PC-TCP-NE	66967.83	66449.83	+518.00	1158.05	+3.4648	0.0010
2	TT-RTP-NDND	PC-TCP-NE	66977.50	66302.67	+674.83	70.70	+73.9370	0.0000
3	TT-RTP-NDND	PC-TCP-NE	67037.17	66418.17	+619.00	1042.99	+4.5971	0.0000
4	TT-RTP-NDND	PC-TCP-NE	66965.00	66293.83	+671.17	60.51	+85.9185	0.0000
1	TT-RTP-NDND	SN-TCP-ND	66967.83	67034.67	-66.83	747.96	-0.6921	0.4916
2	TT-RTP-NDND	SN-TCP-ND	66977.50	67089.83	-112.33	780.99	-1.1141	0.2697
3	TT-RTP-NDND	SN-TCP-ND	67037.17	67155.67	-118.50	1358.57	-0.6756	0.5019
4	TT-RTP-NDND	SN-TCP-ND	66965.00	66988.67	-23.67	57.99	-3.1614	0.0025
1	TT-RTP-NDND	SN-TCP-NE	66967.83	67080.33	-112.50	1178.54	-0.7394	0.4626
2	TT-RTP-NDND	SN-TCP-NE	66977.50	67095.67	-118.17	829.07	-1.1040	0.2741
3	TT-RTP-NDND	SN-TCP-NE	67037.17	66991.67	+45.50	487.73	+0.7226	0.4728
4	TT-RTP-NDND	SN-TCP-NE	66965.00	66989.83	-24.83	56.49	-3.4050	0.0012
1	TT-RTP-NEND	TT-RTP-NDNE	66970.83	67174.67	-203.83	1561.21	-1.0113	0.3160
2	TT-RTP-NEND	TT-RTP-NDNE	66984.33	66970.00	+14.33	50.57	+2.1955	0.0321
3	TT-RTP-NEND	TT-RTP-NDNE	67096.67	67198.00	-101.33	789.42	-0.9943	0.3241
4	TT-RTP-NEND	TT-RTP-NDNE	66976.33	67333.67	-357.33	2815.87	-0.9830	0.3296
1	TT-RTP-NEND	TT-RTP-NENE	66970.83	67069.17	-98.33	739.44	-1.0301	0.3072
2	TT-RTP-NEND	TT-RTP-NENE	66984.33	66968.67	+15.67	50.90	+2.3840	0.0204
3	TT-RTP-NEND	TT-RTP-NENE	67096.67	66981.67	+115.00	859.34	+1.0366	0.3042
4	TT-RTP-NEND	TT-RTP-NENE	66976.33	66969.50	+6.83	44.63	+1.1861	0.2403
1	TT-RTP-NEND	UT-RTP-ND	66970.83	66930.17	+40.67	179.02	+1.7596	0.0837
2	TT-RTP-NEND	UT-RTP-ND	66984.33	67093.50	-109.17	1228.31	-0.6884	0.4939
3	TT-RTP-NEND	UT-RTP-ND	67096.67	66984.00	+112.67	614.60	+1.4200	0.1609
4	TT-RTP-NEND	UT-RTP-ND	66976.33	67018.83	-42.50	646.88	-0.5089	0.6127
1	TT-RTP-NEND	UT-RTP-NE	66970.83	67175.17	-204.33	1776.28	-0.8911	0.3765
2	TT-RTP-NEND	UT-RTP-NE	66984.33	66960.83	+23.50	50.58	+3.5986	0.0007
3	TT-RTP-NEND	UT-RTP-NE	67096.67	66959.50	+137.17	777.17	+1.3671	0.1768
4	TT-RTP-NEND	UT-RTP-NE	66976.33	66903.83	+72.50	82.06	+6.8433	0.0000
1	TT-RTP-NEND	PC-TCP-ND	66970.83	66345.33	+625.50	1010.40	+4.7952	0.0000
2	TT-RTP-NEND	PC-TCP-ND	66984.33	66395.33	+589.00	640.02	+7.1285	0.0000
3	TT-RTP-NEND	PC-TCP-ND	67096.67	66472.17	+624.50	434.69	+11.1283	0.0000
4	TT-RTP-NEND	PC-TCP-ND	66976.33	66069.67	+906.67	377.59	+18.5997	0.0000

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NEND	PC-TCP-NE	66970.83	66449.83	+521.00	1167.22	+3.4575	0.0010
2	TT-RTP-NEND	PC-TCP-NE	66984.33	66302.67	+681.67	83.65	+63.1228	0.0000
3	TT-RTP-NEND	PC-TCP-NE	67096.67	66418.17	+678.50	119.40	+44.0160	0.0000
4	TT-RTP-NEND	PC-TCP-NE	66976.33	66293.83	+682.50	73.61	+71.8151	0.0000
1	TT-RTP-NEND	SN-TCP-ND	66970.83	67034.67	-63.83	754.24	-0.6556	0.5147
2	TT-RTP-NEND	SN-TCP-ND	66984.33	67089.83	-105.50	782.22	-1.0447	0.3004
3	TT-RTP-NEND	SN-TCP-ND	67096.67	67155.67	-59.00	299.39	-1.5265	0.1322
4	TT-RTP-NEND	SN-TCP-ND	66976.33	66988.67	-12.33	48.38	-1.9748	0.0530
1	TT-RTP-NEND	SN-TCP-NE	66970.83	67080.33	-109.50	1185.04	-0.7157	0.4770
2	TT-RTP-NEND	SN-TCP-NE	66984.33	67095.67	-111.33	830.20	-1.0388	0.3032
3	TT-RTP-NEND	SN-TCP-NE	67096.67	66991.67	+105.00	953.98	+0.8526	0.3973
4	TT-RTP-NEND	SN-TCP-NE	66976.33	66989.83	-13.50	49.40	-2.1169	0.0385
1	TT-RTP-NDNE	TT-RTP-NENE	67174.67	67069.17	+105.50	823.50	+0.9924	0.3251
2	TT-RTP-NDNE	TT-RTP-NENE	66970.00	66968.67	+1.33	43.43	+0.2378	0.8129
3	TT-RTP-NDNE	TT-RTP-NENE	67198.00	66981.67	+216.33	1646.41	+1.0178	0.3129
4	TT-RTP-NDNE	TT-RTP-NENE	67333.67	66969.50	+364.17	2821.54	+0.9997	0.3215
1	TT-RTP-NDNE	UT-RTP-ND	67174.67	66930.17	+244.50	1572.81	+1.2041	0.2333
2	TT-RTP-NDNE	UT-RTP-ND	66970.00	67093.50	-123.50	1225.79	-0.7804	0.4383
3	TT-RTP-NDNE	UT-RTP-ND	67198.00	66984.00	+214.00	1401.68	+1.1826	0.2417
4	TT-RTP-NDNE	UT-RTP-ND	67333.67	67018.83	+314.83	2897.36	+0.8417	0.4034
1	TT-RTP-NDNE	UT-RTP-NE	67174.67	67175.17	-0.50	221.61	-0.0175	0.9861
2	TT-RTP-NDNE	UT-RTP-NE	66970.00	66960.83	+9.17	43.46	+1.6336	0.1077
3	TT-RTP-NDNE	UT-RTP-NE	67198.00	66959.50	+238.50	1563.96	+1.1812	0.2422
4	TT-RTP-NDNE	UT-RTP-NE	67333.67	66903.83	+429.83	2850.34	+1.1681	0.2475
1	TT-RTP-NDNE	PC-TCP-ND	67174.67	66345.33	+829.33	566.62	+11.3374	0.0000
2	TT-RTP-NDNE	PC-TCP-ND	66970.00	66395.33	+574.67	636.09	+6.9980	0.0000
3	TT-RTP-NDNE	PC-TCP-ND	67198.00	66472.17	+725.83	385.76	+14.5745	0.0000
4	TT-RTP-NDNE	PC-TCP-ND	67333.67	66069.67	+1264.00	2817.63	+3.4749	0.0010
1	TT-RTP-NDNE	PC-TCP-NE	67174.67	66449.83	+724.83	404.72	+13.8728	0.0000
2	TT-RTP-NDNE	PC-TCP-NE	66970.00	66302.67	+667.33	71.66	+72.1344	0.0000
3	TT-RTP-NDNE	PC-TCP-NE	67198.00	66418.17	+779.83	851.47	+7.0943	0.0000
4	TT-RTP-NDNE	PC-TCP-NE	67333.67	66293.83	+1039.83	2819.46	+2.8568	0.0059
1	TT-RTP-NDNE	SN-TCP-ND	67174.67	67034.67	+140.00	813.53	+1.3330	0.1877
2	TT-RTP-NDNE	SN-TCP-ND	66970.00	67089.83	-119.83	780.61	-1.1891	0.2392
3	TT-RTP-NDNE	SN-TCP-ND	67198.00	67155.67	+42.33	494.38	+0.6633	0.5097
4	TT-RTP-NDNE	SN-TCP-ND	67333.67	66988.67	+345.00	2817.71	+0.9484	0.3468
1	TT-RTP-NDNE	SN-TCP-NE	67174.67	67080.33	+94.33	385.22	+1.8968	0.0627
2	TT-RTP-NDNE	SN-TCP-NE	66970.00	67095.67	-125.67	828.71	-1.1746	0.2449
3	TT-RTP-NDNE	SN-TCP-NE	67198.00	66991.67	+206.33	1741.25	+0.9179	0.3624
4	TT-RTP-NDNE	SN-TCP-NE	67333.67	66989.83	+343.83	2817.80	+0.9452	0.3484
1	TT-RTP-NENE	UT-RTP-ND	67069.17	66930.17	+139.00	764.42	+1.4085	0.1642
2	TT-RTP-NENE	UT-RTP-ND	66968.67	67093.50	-124.83	1225.45	-0.7891	0.4332
3	TT-RTP-NENE	UT-RTP-ND	66981.67	66984.00	-2.33	250.48	-0.0722	0.9427
4	TT-RTP-NENE	UT-RTP-ND	66969.50	67018.83	-49.33	646.20	-0.5914	0.5565
1	TT-RTP-NENE	UT-RTP-NE	67069.17	67175.17	-106.00	1038.96	-0.7903	0.4325
2	TT-RTP-NENE	UT-RTP-NE	66968.67	66960.83	+7.83	49.51	+1.2256	0.2252
3	TT-RTP-NENE	UT-RTP-NE	66981.67	66959.50	+22.17	96.97	+1.7706	0.0818
4	TT-RTP-NENE	UT-RTP-NE	66969.50	66903.83	+65.67	79.56	+6.3932	0.0000
1	TT-RTP-NENE	PC-TCP-ND	67069.17	66345.33	+723.83	289.14	+19.3911	0.0000
2	TT-RTP-NENE	PC-TCP-ND	66968.67	66395.33	+573.33	635.81	+6.9848	0.0000
3	TT-RTP-NENE	PC-TCP-ND	66981.67	66472.17	+509.50	1282.39	+3.0775	0.0032
4	TT-RTP-NENE	PC-TCP-ND	66969.50	66069.67	+899.83	378.01	+18.4388	0.0000
1	TT-RTP-NENE	PC-TCP-NE	67069.17	66449.83	+619.33	434.97	+11.0290	0.0000
2	TT-RTP-NENE	PC-TCP-NE	66968.67	66302.67	+666.00	70.11	+73.5778	0.0000
3	TT-RTP-NENE	PC-TCP-NE	66981.67	66418.17	+563.50	808.19	+5.4008	0.0000
4	TT-RTP-NENE	PC-TCP-NE	66969.50	66293.83	+675.67	77.45	+67.5788	0.0000

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NENE	SN-TCP-ND	67069.17	67034.67	+34.50	77.01	+3.4702	0.0010
2	TT-RTP-NENE	SN-TCP-ND	66968.67	67089.83	-121.17	780.28	-1.2028	0.2338
3	TT-RTP-NENE	SN-TCP-ND	66981.67	67155.67	-174.00	1154.34	-1.1676	0.2477
4	TT-RTP-NENE	SN-TCP-ND	66969.50	66988.67	-19.17	52.57	-2.8241	0.0065
1	TT-RTP-NENE	SN-TCP-NE	67069.17	67080.33	-11.17	450.30	-0.1921	0.8483
2	TT-RTP-NENE	SN-TCP-NE	66968.67	67095.67	-127.00	828.29	-1.1877	0.2397
3	TT-RTP-NENE	SN-TCP-NE	66981.67	66991.67	-10.00	107.86	-0.7181	0.4755
4	TT-RTP-NENE	SN-TCP-NE	66969.50	66989.83	-20.33	52.88	-2.9784	0.0042
1	UT-RTP-ND	UT-RTP-NE	66930.17	67175.17	-245.00	1785.02	-1.0632	0.2920
2	UT-RTP-ND	UT-RTP-NE	67093.50	66960.83	+132.67	1230.84	+0.8349	0.4071
3	UT-RTP-ND	UT-RTP-NE	66984.00	66959.50	+24.50	169.59	+1.1190	0.2677
4	UT-RTP-ND	UT-RTP-NE	67018.83	66903.83	+115.00	646.82	+1.3772	0.1737
1	UT-RTP-ND	PC-RTP-ND	66930.17	66345.33	+584.83	1019.30	+4.4443	0.0000
2	UT-RTP-ND	PC-RTP-ND	67093.50	66395.33	+698.17	597.86	+9.0456	0.0000
3	UT-RTP-ND	PC-RTP-ND	66984.00	66472.17	+511.83	1037.44	+3.8216	0.0003
4	UT-RTP-ND	PC-RTP-ND	67018.83	66069.67	+949.17	720.09	+10.2101	0.0000
1	UT-RTP-ND	PC-RTP-NE	66930.17	66449.83	+480.33	1174.08	+3.1690	0.0024
2	UT-RTP-ND	PC-RTP-NE	67093.50	66302.67	+790.83	1233.12	+4.9677	0.0000
3	UT-RTP-ND	PC-RTP-NE	66984.00	66418.17	+565.83	564.77	+7.7606	0.0000
4	UT-RTP-ND	PC-RTP-NE	67018.83	66293.83	+725.00	631.80	+8.8886	0.0000
1	UT-RTP-ND	SN-RTP-ND	66930.17	67034.67	-104.50	780.09	-1.0376	0.3037
2	UT-RTP-ND	SN-RTP-ND	67093.50	67089.83	+3.67	452.15	+0.0628	0.9501
3	UT-RTP-ND	SN-RTP-ND	66984.00	67155.67	-171.67	910.52	-1.4604	0.1495
4	UT-RTP-ND	SN-RTP-ND	67018.83	66988.67	+30.17	645.33	+0.3621	0.7186
1	UT-RTP-ND	SN-RTP-NE	66930.17	67080.33	-150.17	1203.75	-0.9663	0.3378
2	UT-RTP-ND	SN-RTP-NE	67093.50	67095.67	-2.17	405.25	-0.0414	0.9671
3	UT-RTP-ND	SN-RTP-NE	66984.00	66991.67	-7.67	346.18	-0.1715	0.8644
4	UT-RTP-ND	SN-RTP-NE	67018.83	66989.83	+29.00	642.98	+0.3494	0.7281
1	UT-RTP-NE	PC-RTP-ND	67175.17	66345.33	+829.83	777.83	+8.2638	0.0000
2	UT-RTP-NE	PC-RTP-ND	66960.83	66395.33	+565.50	641.15	+6.8320	0.0000
3	UT-RTP-NE	PC-RTP-ND	66959.50	66472.17	+487.33	1199.07	+3.1482	0.0026
4	UT-RTP-NE	PC-RTP-ND	66903.83	66069.67	+834.17	415.40	+15.5547	0.0000
1	UT-RTP-NE	PC-RTP-NE	67175.17	66449.83	+725.33	615.47	+9.1286	0.0000
2	UT-RTP-NE	PC-RTP-NE	66960.83	66302.67	+658.17	70.10	+72.7302	0.0000
3	UT-RTP-NE	PC-RTP-NE	66959.50	66418.17	+541.33	724.61	+5.7868	0.0000
4	UT-RTP-NE	PC-RTP-NE	66903.83	66293.83	+610.00	105.20	+44.9133	0.0000
1	UT-RTP-NE	SN-RTP-ND	67175.17	67034.67	+140.50	1028.45	+1.0582	0.2943
2	UT-RTP-NE	SN-RTP-ND	66960.83	67089.83	-129.00	786.19	-1.2710	0.2087
3	UT-RTP-NE	SN-RTP-ND	66959.50	67155.67	-196.17	1072.41	-1.4169	0.1618
4	UT-RTP-NE	SN-RTP-ND	66903.83	66988.67	-84.83	80.24	-8.1891	0.0000
1	UT-RTP-NE	SN-RTP-NE	67175.17	67080.33	+94.83	598.51	+1.2274	0.2246
2	UT-RTP-NE	SN-RTP-NE	66960.83	67095.67	-134.83	834.07	-1.2522	0.2154
3	UT-RTP-NE	SN-RTP-NE	66959.50	66991.67	-32.17	188.94	-1.3187	0.1924
4	UT-RTP-NE	SN-RTP-NE	66903.83	66989.83	-86.00	72.47	-9.1925	0.0000
1	PC-TCP-ND	PC-TCP-NE	66345.33	66449.83	-104.50	185.76	-4.3575	0.0000
2	PC-TCP-ND	PC-TCP-NE	66395.33	66302.67	+92.67	639.22	+1.1229	0.2660
3	PC-TCP-ND	PC-TCP-NE	66472.17	66418.17	+54.00	478.66	+0.8739	0.3857
4	PC-TCP-ND	PC-TCP-NE	66069.67	66293.83	-224.17	393.01	-4.4182	0.0000
1	PC-TCP-ND	SN-RTP-ND	66345.33	67034.67	-689.33	289.01	-18.4755	0.0000
2	PC-TCP-ND	SN-RTP-ND	66395.33	67089.83	-694.50	178.66	-30.1104	0.0000
3	PC-TCP-ND	SN-RTP-ND	66472.17	67155.67	-683.50	169.09	-31.3103	0.0000
4	PC-TCP-ND	SN-RTP-ND	66069.67	66988.67	-919.00	393.89	-18.0723	0.0000
1	PC-TCP-ND	SN-RTP-NE	66345.33	67034.67	-689.33	289.01	-18.4755	0.0000
2	PC-TCP-ND	SN-RTP-NE	66395.33	67089.83	-694.50	178.66	-30.1104	0.0000
3	PC-TCP-ND	SN-RTP-NE	66472.17	67155.67	-683.50	169.09	-31.3103	0.0000
4	PC-TCP-ND	SN-RTP-NE	66069.67	66988.67	-919.00	393.89	-18.0723	0.0000

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	PC-TCP-NE	SN-RTP-ND	66449.83	67034.67	-584.83	428.70	-10.5671	0.0000
2	PC-TCP-NE	SN-RTP-ND	66302.67	67089.83	-787.17	790.34	-7.7149	0.0000
3	PC-TCP-NE	SN-RTP-ND	66418.17	67155.67	-737.50	368.06	-15.5209	0.0000
4	PC-TCP-NE	SN-RTP-ND	66293.83	66988.67	-694.83	85.99	-62.5879	0.0000
1	PC-TCP-NE	SN-RTP-NE	66449.83	67034.67	-584.83	428.70	-10.5671	0.0000
2	PC-TCP-NE	SN-RTP-NE	66302.67	67089.83	-787.17	790.34	-7.7149	0.0000
3	PC-TCP-NE	SN-RTP-NE	66418.17	67155.67	-737.50	368.06	-15.5209	0.0000
4	PC-TCP-NE	SN-RTP-NE	66293.83	66988.67	-694.83	85.99	-62.5879	0.0000
1	SN-TCP-ND	SN-RTP-NE	67034.67	67080.33	-45.67	437.14	-0.8092	0.4217
2	SN-TCP-ND	SN-RTP-NE	67089.83	67095.67	-5.83	60.23	-0.7502	0.4561
3	SN-TCP-ND	SN-RTP-NE	67155.67	66991.67	+164.00	1249.07	+1.0170	0.3133
4	SN-TCP-ND	SN-RTP-NE	66988.67	66989.83	-1.17	38.09	-0.2372	0.8133

## APPENDIX F WAN PAIRED MEANS COMPARISON STATISTICS

This appendix consists of tables of data that summarize the WAN paired means Student's t-tests. The paired means were dependent since they were measuring the same experimental metric that is run-time of a MIDI sequence at the ultimate destination. The first data in the tables are the values of  $m$ , the number of MIDI short messages per TCP packet. This value varied from 1 to 4. The next two data items are the protocol mnemonics involved in the paired comparison. Then the measured means are given along with their differences. The final three data items are the standard deviation, the Student's t-value, and the Student's t-value significance. A negative t-value meant that the first protocol (protocol #1) was potentially the best protocol in the pair. A positive t-value meant that the second protocol (protocol #2) was potentially statistically superior to the first protocol. If the value of the Student's t-value significance was less than or equal 0.05 then one of the protocols statistically outperformed the other protocol in the pairing.

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NDND	TT-RTP-NDNE	67184.33	67246.67	-62.33	195.07	-2.4752	0.0162
2	TT-RTP-NDND	TT-RTP-NDNE	67086.00	67094.67	-8.67	761.38	-0.0882	0.9300
3	TT-RTP-NDND	TT-RTP-NDNE	67123.00	67292.50	-169.50	1890.02	-0.6947	0.4900
4	TT-RTP-NDND	TT-RTP-NDNE	66947.50	67107.50	-160.00	389.35	-3.1831	0.0023
1	TT-RTP-NDND	TT-RTP-NEND	67184.33	67023.33	+161.00	1221.11	+1.0213	0.3113
2	TT-RTP-NDND	TT-RTP-NEND	67086.00	67098.50	-12.50	1211.23	-0.0799	0.9366
3	TT-RTP-NDND	TT-RTP-NEND	67123.00	67240.17	-117.17	1996.88	-0.4545	0.6511
4	TT-RTP-NDND	TT-RTP-NEND	66947.50	66977.50	-30.00	354.70	-0.6551	0.5149
1	TT-RTP-NDND	TT-RTP-NENE	67184.33	67356.67	-172.33	1118.98	-1.1929	0.2377
2	TT-RTP-NDND	TT-RTP-NENE	67086.00	67352.00	-266.00	2264.49	-0.9099	0.3666
3	TT-RTP-NDND	TT-RTP-NENE	67123.00	67069.00	+54.00	779.99	+0.5363	0.5938
4	TT-RTP-NDND	TT-RTP-NENE	66947.50	67399.33	-451.83	2356.68	-1.4851	0.1428
1	TT-RTP-NDND	UT-RTP-ND	67184.33	67380.83	-196.50	1103.35	-1.3795	0.1729
2	TT-RTP-NDND	UT-RTP-ND	67086.00	66977.67	+108.33	752.51	+1.1151	0.2693
3	TT-RTP-NDND	UT-RTP-ND	67123.00	66972.67	+150.33	776.23	+1.5002	0.1389
4	TT-RTP-NDND	UT-RTP-ND	66947.50	67317.33	-369.83	1923.86	-1.4890	0.1418
1	TT-RTP-NDND	UT-RTP-NE	67184.33	67030.17	+154.17	1113.97	+1.0720	0.2881
2	TT-RTP-NDND	UT-RTP-NE	67086.00	67023.83	+62.17	752.78	+0.6397	0.5249
3	TT-RTP-NDND	UT-RTP-NE	67123.00	67013.00	+110.00	785.66	+1.0845	0.2825
4	TT-RTP-NDND	UT-RTP-NE	66947.50	67007.17	-59.67	359.69	-1.2849	0.2038
1	TT-RTP-NDND	PC-TCP-ND	67184.33	66079.83	+1104.50	1264.12	+6.7679	0.0000
2	TT-RTP-NDND	PC-TCP-ND	67086.00	66290.83	+795.17	1561.68	+3.9441	0.0002
3	TT-RTP-NDND	PC-TCP-ND	67123.00	66127.67	+995.33	790.00	+9.7592	0.0000
4	TT-RTP-NDND	PC-TCP-ND	66947.50	66147.67	+799.83	364.08	+17.0168	0.0000
1	TT-RTP-NDND	PC-TCP-NE	67184.33	66162.83	+1021.50	1206.69	+6.5572	0.0000
2	TT-RTP-NDND	PC-TCP-NE	67086.00	66448.67	+637.33	2286.90	+2.1587	0.0350
3	TT-RTP-NDND	PC-TCP-NE	67123.00	66343.17	+779.83	1473.51	+4.0994	0.0001
4	TT-RTP-NDND	PC-TCP-NE	66947.50	66159.50	+788.00	354.08	+17.2386	0.0000
1	TT-RTP-NDND	SN-TCP-ND	67184.33	67247.17	-62.83	322.18	-1.5107	0.1362
2	TT-RTP-NDND	SN-TCP-ND	67086.00	67219.17	-133.17	3146.71	-0.3278	0.7442
3	TT-RTP-NDND	SN-TCP-ND	67123.00	67056.83	+66.17	960.81	+0.5334	0.5957
4	TT-RTP-NDND	SN-TCP-ND	66947.50	67011.17	-63.67	1334.28	-0.3696	0.7130
1	TT-RTP-NDND	SN-TCP-NE	67184.33	67256.67	-72.33	441.27	-1.2697	0.2092
2	TT-RTP-NDND	SN-TCP-NE	67086.00	67233.83	-147.83	1644.50	-0.6963	0.4890
3	TT-RTP-NDND	SN-TCP-NE	67123.00	67035.50	+87.50	780.24	+0.8687	0.3885
4	TT-RTP-NDND	SN-TCP-NE	66947.50	67205.17	-257.67	1168.43	-1.7082	0.0929
1	TT-RTP-NEND	TT-RTP-NDNE	67023.33	67246.67	-223.33	1399.82	-1.2358	0.2214
2	TT-RTP-NEND	TT-RTP-NDNE	67098.50	67094.67	+3.83	962.69	+0.0308	0.9755
3	TT-RTP-NEND	TT-RTP-NDNE	67240.17	67292.50	-52.33	135.80	-2.9850	0.0041
4	TT-RTP-NEND	TT-RTP-NDNE	66977.50	67107.50	-130.00	389.56	-2.5849	0.0122
1	TT-RTP-NEND	TT-RTP-NENE	67023.33	67356.67	-333.33	2337.95	-1.1044	0.2739
2	TT-RTP-NEND	TT-RTP-NENE	67098.50	67352.00	-253.50	1187.11	-1.6541	0.1034
3	TT-RTP-NEND	TT-RTP-NENE	67240.17	67069.00	+171.17	1968.95	+0.6734	0.5033
4	TT-RTP-NEND	TT-RTP-NENE	66977.50	67399.33	-421.83	2498.75	-1.3077	0.1961
1	TT-RTP-NEND	UT-RTP-ND	67023.33	67380.83	-357.50	2251.36	-1.2300	0.2236
2	TT-RTP-NEND	UT-RTP-ND	67098.50	66977.67	+120.83	931.32	+1.0050	0.3190
3	TT-RTP-NEND	UT-RTP-ND	67240.17	66972.67	+267.50	1978.65	+1.0472	0.2993
4	TT-RTP-NEND	UT-RTP-ND	66977.50	67317.33	-339.83	2053.35	-1.2820	0.2049
1	TT-RTP-NEND	UT-RTP-NE	67023.33	67030.17	-6.83	123.47	-0.4287	0.6697
2	TT-RTP-NEND	UT-RTP-NE	67098.50	67023.83	+74.67	936.41	+0.6176	0.5392
3	TT-RTP-NEND	UT-RTP-NE	67240.17	67013.00	+227.17	1975.95	+0.8905	0.3768
4	TT-RTP-NEND	UT-RTP-NE	66977.50	67007.17	-29.67	63.65	-3.6106	0.0006
1	TT-RTP-NEND	PC-TCP-ND	67023.33	66079.83	+943.50	386.56	+18.9062	0.0000
2	TT-RTP-NEND	PC-TCP-ND	67098.50	66290.83	+807.67	437.97	+14.2846	0.0000
3	TT-RTP-NEND	PC-TCP-ND	67240.17	66127.67	+1112.50	1969.10	+4.3763	0.0000
4	TT-RTP-NEND	PC-TCP-ND	66977.50	66147.67	+829.83	85.34	+75.3201	0.0000

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NEND	PC-TCP-NE	67023.33	66162.83	+860.50	100.94	+66.0366	0.0000
2	TT-RTP-NEND	PC-TCP-NE	67098.50	66448.67	+649.83	1217.80	+4.1333	0.0001
3	TT-RTP-NEND	PC-TCP-NE	67240.17	66343.17	+897.00	607.96	+11.4286	0.0000
4	TT-RTP-NEND	PC-TCP-NE	66977.50	66159.50	+818.00	80.69	+78.5234	0.0000
1	TT-RTP-NEND	SN-TCP-ND	67023.33	67247.17	-223.83	1537.24	-1.1279	0.2639
2	TT-RTP-NEND	SN-TCP-ND	67098.50	67219.17	-120.67	2105.41	-0.4439	0.6587
3	TT-RTP-NEND	SN-TCP-ND	67240.17	67056.83	+183.33	2046.99	+0.6937	0.4906
4	TT-RTP-NEND	SN-TCP-ND	66977.50	67011.17	-33.67	1461.32	-0.1785	0.8590
1	TT-RTP-NEND	SN-TCP-NE	67023.33	67256.67	-233.33	1658.11	-1.0900	0.2801
2	TT-RTP-NEND	SN-TCP-NE	67098.50	67233.83	-135.33	517.69	-2.0249	0.0474
3	TT-RTP-NEND	SN-TCP-NE	67240.17	67035.50	+204.67	1972.43	+0.8037	0.4248
4	TT-RTP-NEND	SN-TCP-NE	66977.50	67205.17	-227.67	1284.78	-1.3726	0.1751
1	TT-RTP-NDNE	TT-RTP-NENE	67246.67	67356.67	-110.00	942.33	-0.9042	0.3696
2	TT-RTP-NDNE	TT-RTP-NENE	67094.67	67352.00	-257.33	2138.79	-0.9320	0.3551
3	TT-RTP-NDNE	TT-RTP-NENE	67292.50	67069.00	+223.50	1853.32	+0.9341	0.3540
4	TT-RTP-NDNE	TT-RTP-NENE	67107.50	67399.33	-291.83	2113.04	-1.0698	0.2891
1	TT-RTP-NDNE	UT-RTP-ND	67246.67	67380.83	-134.17	954.55	-1.0887	0.2807
2	TT-RTP-NDNE	UT-RTP-ND	67094.67	66977.67	+117.00	183.74	+4.9324	0.0000
3	TT-RTP-NDNE	UT-RTP-ND	67292.50	66972.67	+319.83	1862.79	+1.3299	0.1887
4	TT-RTP-NDNE	UT-RTP-ND	67107.50	67317.33	-209.83	1681.19	-0.9668	0.3376
1	TT-RTP-NDNE	UT-RTP-NE	67246.67	67030.17	+216.50	1292.77	+1.2972	0.1996
2	TT-RTP-NDNE	UT-RTP-NE	67094.67	67023.83	+70.83	195.71	+2.8035	0.0068
3	TT-RTP-NDNE	UT-RTP-NE	67292.50	67013.00	+279.50	1860.21	+1.1638	0.2492
4	TT-RTP-NDNE	UT-RTP-NE	67107.50	67007.17	+100.33	393.76	+1.9737	0.0531
1	TT-RTP-NDNE	PC-TCP-ND	67246.67	66079.83	+1166.83	1436.37	+6.2924	0.0000
2	TT-RTP-NDNE	PC-TCP-ND	67094.67	66290.83	+803.83	1383.05	+4.5020	0.0000
3	TT-RTP-NDNE	PC-TCP-ND	67292.50	66127.67	+1164.83	1854.42	+4.8655	0.0000
4	TT-RTP-NDNE	PC-TCP-ND	67107.50	66147.67	+959.83	386.99	+19.2122	0.0000
1	TT-RTP-NDNE	PC-TCP-NE	67246.67	66162.83	+1083.83	1382.92	+6.0707	0.0000
2	TT-RTP-NDNE	PC-TCP-NE	67094.67	66448.67	+646.00	2166.31	+2.3099	0.0244
3	TT-RTP-NDNE	PC-TCP-NE	67292.50	66343.17	+949.33	499.03	+14.7355	0.0000
4	TT-RTP-NDNE	PC-TCP-NE	67107.50	66159.50	+948.00	396.77	+18.5074	0.0000
1	TT-RTP-NDNE	SN-TCP-ND	67246.67	67247.17	-0.50	159.10	-0.0243	0.9807
2	TT-RTP-NDNE	SN-TCP-ND	67094.67	67219.17	-124.50	3054.55	-0.3157	0.7533
3	TT-RTP-NDNE	SN-TCP-ND	67292.50	67056.83	+235.67	1935.58	+0.9431	0.3495
4	TT-RTP-NDNE	SN-TCP-ND	67107.50	67011.17	+96.33	1074.85	+0.6942	0.4903
1	TT-RTP-NDNE	SN-TCP-NE	67246.67	67256.67	-10.00	268.64	-0.2883	0.7741
2	TT-RTP-NDNE	SN-TCP-NE	67094.67	67233.83	-139.17	1472.01	-0.7323	0.4669
3	TT-RTP-NDNE	SN-TCP-NE	67292.50	67035.50	+257.00	1857.13	+1.0719	0.2881
4	TT-RTP-NDNE	SN-TCP-NE	67107.50	67205.17	-97.67	904.07	-0.8368	0.4061
1	TT-RTP-NENE	UT-RTP-ND	67356.67	67380.83	-24.17	553.65	-0.3381	0.7365
2	TT-RTP-NENE	UT-RTP-ND	67352.00	66977.67	+374.33	2116.88	+1.3697	0.1760
3	TT-RTP-NENE	UT-RTP-ND	67069.00	66972.67	+96.33	69.33	+10.7629	0.0000
4	TT-RTP-NENE	UT-RTP-ND	67399.33	67317.33	+82.00	696.23	+0.9123	0.3653
1	TT-RTP-NENE	UT-RTP-NE	67356.67	67030.17	+326.50	2230.97	+1.1336	0.2615
2	TT-RTP-NENE	UT-RTP-NE	67352.00	67023.83	+328.17	2121.86	+1.1980	0.2357
3	TT-RTP-NENE	UT-RTP-NE	67069.00	67013.00	+56.00	59.69	+7.2666	0.0000
4	TT-RTP-NENE	UT-RTP-NE	67399.33	67007.17	+392.17	2502.74	+1.2138	0.2297
1	TT-RTP-NENE	PC-TCP-ND	67356.67	66079.83	+1276.83	2352.95	+4.2034	0.0000
2	TT-RTP-NENE	PC-TCP-ND	67352.00	66290.83	+1061.17	772.66	+10.6383	0.0000
3	TT-RTP-NENE	PC-TCP-ND	67069.00	66127.67	+941.33	116.87	+62.3888	0.0000
4	TT-RTP-NENE	PC-TCP-ND	67399.33	66147.67	+1251.67	2491.94	+3.8907	0.0003
1	TT-RTP-NENE	PC-TCP-NE	67356.67	66162.83	+1193.83	2320.97	+3.9843	0.0002
2	TT-RTP-NENE	PC-TCP-NE	67352.00	66448.67	+903.33	138.91	+50.3706	0.0000
3	TT-RTP-NENE	PC-TCP-NE	67069.00	66343.17	+725.83	1377.48	+4.0816	0.0001
4	TT-RTP-NENE	PC-TCP-NE	67399.33	66159.50	+1239.83	2501.61	+3.8390	0.0003



M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	TT-RTP-NENE	SN-TCP-ND	67356.67	67247.17	+109.50	804.64	+1.0541	0.2961
2	TT-RTP-NENE	SN-TCP-ND	67352.00	67219.17	+132.83	920.86	+1.1173	0.2684
3	TT-RTP-NENE	SN-TCP-ND	67069.00	67056.83	+12.17	518.98	+0.1816	0.8565
4	TT-RTP-NENE	SN-TCP-ND	67399.33	67011.17	+388.17	1040.70	+2.8891	0.0054
1	TT-RTP-NENE	SN-TCP-NE	67356.67	67256.67	+100.00	685.05	+1.1307	0.2628
2	TT-RTP-NENE	SN-TCP-NE	67352.00	67233.83	+118.17	673.86	+1.3583	0.1795
3	TT-RTP-NENE	SN-TCP-NE	67069.00	67035.50	+33.50	68.27	+3.8011	0.0003
4	TT-RTP-NENE	SN-TCP-NE	67399.33	67205.17	+194.17	1248.48	+1.2047	0.2331
1	UT-RTP-ND	UT-RTP-NE	67380.83	67030.17	+350.67	2147.89	+1.2646	0.2110
2	UT-RTP-ND	UT-RTP-NE	66977.67	67023.83	-46.17	69.41	-5.1523	0.0000
3	UT-RTP-ND	UT-RTP-NE	66972.67	67013.00	-40.33	57.49	-5.4345	0.0000
4	UT-RTP-ND	UT-RTP-NE	67317.33	67007.17	+310.17	2057.85	+1.1675	0.2477
1	UT-RTP-ND	PC-TCP-ND	67380.83	66079.83	+1301.00	2268.32	+4.4427	0.0000
2	UT-RTP-ND	PC-TCP-ND	66977.67	66290.83	+686.83	1356.09	+3.9232	0.0002
3	UT-RTP-ND	PC-TCP-ND	66972.67	66127.67	+845.00	111.03	+58.9524	0.0000
4	UT-RTP-ND	PC-TCP-ND	67317.33	66147.67	+1169.67	2044.74	+4.4310	0.0000
1	UT-RTP-ND	PC-TCP-NE	67380.83	66162.83	+1218.00	2235.36	+4.2206	0.0000
2	UT-RTP-ND	PC-TCP-NE	66977.67	66448.67	+529.00	2143.08	+1.9120	0.0607
3	UT-RTP-ND	PC-TCP-NE	66972.67	66343.17	+629.50	1386.30	+3.5173	0.0008
4	UT-RTP-ND	PC-TCP-NE	67317.33	66159.50	+1157.83	2053.38	+4.3677	0.0000
1	UT-RTP-ND	SN-TCP-ND	67380.83	67247.17	+133.67	838.12	+1.2354	0.2216
2	UT-RTP-ND	SN-TCP-ND	66977.67	67219.17	-241.50	3034.98	-0.6164	0.5400
3	UT-RTP-ND	SN-TCP-ND	66972.67	67056.83	-84.17	514.22	-1.2679	0.2098
4	UT-RTP-ND	SN-TCP-ND	67317.33	67011.17	+306.17	724.88	+3.2716	0.0018
1	UT-RTP-ND	SN-TCP-NE	67380.83	67256.67	+124.17	747.10	+1.2874	0.2030
2	UT-RTP-ND	SN-TCP-NE	66977.67	67233.83	-256.17	1446.18	-1.3721	0.1752
3	UT-RTP-ND	SN-TCP-NE	66972.67	67035.50	-62.83	64.99	-7.4886	0.0000
4	UT-RTP-ND	SN-TCP-NE	67317.33	67205.17	+112.17	892.17	+0.9739	0.3341
1	UT-RTP-NE	PC-TCP-ND	67030.17	66079.83	+950.33	406.87	+18.0924	0.0000
2	UT-RTP-NE	PC-TCP-ND	67023.83	66290.83	+733.00	1362.14	+4.1683	0.0001
3	UT-RTP-NE	PC-TCP-ND	67013.00	66127.67	+885.33	124.10	+55.2586	0.0000
4	UT-RTP-NE	PC-TCP-ND	67007.17	66147.67	+859.50	93.47	+71.2262	0.0000
1	UT-RTP-NE	PC-TCP-NE	67030.17	66162.83	+867.33	138.86	+48.3836	0.0000
2	UT-RTP-NE	PC-TCP-NE	67023.83	66448.67	+575.17	2149.14	+2.0730	0.0425
3	UT-RTP-NE	PC-TCP-NE	67013.00	66343.17	+669.83	1384.47	+3.7476	0.0004
4	UT-RTP-NE	PC-TCP-NE	67007.17	66159.50	+847.67	94.96	+69.1418	0.0000
1	UT-RTP-NE	SN-TCP-ND	67030.17	67247.17	-217.00	1430.35	-1.1751	0.2447
2	UT-RTP-NE	SN-TCP-ND	67023.83	67219.17	-195.33	3039.68	-0.4978	0.6205
3	UT-RTP-NE	SN-TCP-ND	67013.00	67056.83	-43.83	526.52	-0.6449	0.5215
4	UT-RTP-NE	SN-TCP-ND	67007.17	67011.17	-4.00	1464.42	-0.0212	0.9832
1	UT-RTP-NE	SN-TCP-NE	67030.17	67256.67	-226.50	1550.92	-1.1312	0.2625
2	UT-RTP-NE	SN-TCP-NE	67023.83	67233.83	-210.00	1450.99	-1.1211	0.2668
3	UT-RTP-NE	SN-TCP-NE	67013.00	67035.50	-22.50	60.47	-2.8823	0.0055
4	UT-RTP-NE	SN-TCP-NE	67007.17	67205.17	-198.00	1288.23	-1.1906	0.2386
1	PC-TCP-ND	PC-TCP-NE	66079.83	66162.83	-83.00	371.06	-1.7326	0.0884
2	PC-TCP-ND	PC-TCP-NE	66290.83	66448.67	-157.83	795.53	-1.5368	0.1297
3	PC-TCP-ND	PC-TCP-NE	66127.67	66343.17	-215.50	1371.18	-1.2174	0.2283
4	PC-TCP-ND	PC-TCP-NE	66147.67	66159.50	-11.83	65.37	-1.4022	0.1661
1	PC-TCP-ND	SN-TCP-ND	66079.83	67247.17	-1167.33	1567.51	-5.7685	0.0000
2	PC-TCP-ND	SN-TCP-ND	66290.83	67219.17	-928.33	1686.06	-4.2649	0.0000
3	PC-TCP-ND	SN-TCP-ND	66127.67	67056.83	-929.17	507.58	-14.1796	0.0000
4	PC-TCP-ND	SN-TCP-ND	66147.67	67011.17	-863.50	1454.49	-4.5986	0.0000
1	PC-TCP-ND	SN-TCP-NE	66079.83	67247.17	-1167.33	1567.51	-5.7685	0.0000
2	PC-TCP-ND	SN-TCP-NE	66290.83	67219.17	-928.33	1686.06	-4.2649	0.0000
3	PC-TCP-ND	SN-TCP-NE	66127.67	67056.83	-929.17	507.58	-14.1796	0.0000
4	PC-TCP-ND	SN-TCP-NE	66147.67	67011.17	-863.50	1454.49	-4.5986	0.0000

M	PROTOCOL #1	PROTOCOL #2	MEAN #1	MEAN #2	#1 - #2	STD DEV	T VALUE	T SIGN
1	PC-TCP-NE	SN-TCP-ND	66162.83	67247.17	-1084.33	1521.33	-5.5210	0.0000
2	PC-TCP-NE	SN-TCP-ND	66448.67	67219.17	-770.50	906.93	-6.5808	0.0000
3	PC-TCP-NE	SN-TCP-ND	66343.17	67056.83	-713.67	1478.98	-3.7377	0.0004
4	PC-TCP-NE	SN-TCP-ND	66159.50	67011.17	-851.67	1464.36	-4.5050	0.0000
1	PC-TCP-NE	SN-TCP-NE	66162.83	67247.17	-1084.33	1521.33	-5.5210	0.0000
2	PC-TCP-NE	SN-TCP-NE	66448.67	67219.17	-770.50	906.93	-6.5808	0.0000
3	PC-TCP-NE	SN-TCP-NE	66343.17	67056.83	-713.67	1478.98	-3.7377	0.0004
4	PC-TCP-NE	SN-TCP-NE	66159.50	67011.17	-851.67	1464.36	-4.5050	0.0000
1	SN-TCP-ND	SN-TCP-NE	67247.17	67256.67	-9.50	130.70	-0.5630	0.5756
2	SN-TCP-ND	SN-TCP-NE	67219.17	67233.83	-14.67	1590.49	-0.0714	0.9433
3	SN-TCP-ND	SN-TCP-NE	67056.83	67035.50	+21.33	515.17	+0.3208	0.7495
4	SN-TCP-ND	SN-TCP-NE	67011.17	67205.17	-194.00	278.26	-5.4004	0.0000

## APPENDIX G MIDI INSTRUMENTS

1	Acoustic Grand Piano	33	Acoustic Bass	65	Soprano Sax	97	FX 1 (Rain)
2	Bright Acoustic Piano	34	Electric Bass (Finger)	66	Alto Sax	98	FX 2 (Sound Track)
3	Electric Grand Piano	35	Electric Bass (Pick)	67	Tenor Sax	99	FX 3 (Crystal)
4	Honky-tonk Piano	36	Fretless Bass	68	Baritone Sax	100	FX 4 (Atmosphere)
5	Electric Piano 1	37	Slap Bass 1	69	Oboe	101	FX 5 (Brightness)
6	Electric Piano 2	38	Slap Bass 2	70	English Horn	102	FX 6 (Goblins)
7	Harpichord	39	Synth Bass 1	71	Bassoon	103	FX 7 (Echoes)
8	Clavichord	40	Synth Bass 2	72	Clarinet	104	FX 8 (Sci-Fi)
9	Celesta	41	Violin	73	Piccolo	105	Sitar
10	Glockenspiel	42	Viola	74	Flute	106	Banjo
11	Music Box	43	Cello	75	Recorder	107	Shamisen
12	Vibraphone	44	Contrabass	76	Pan Flute	108	Koto
13	Marimba	45	Tremolo Strings	77	Blown Bottle	109	Kalimba
14	Xylophone	46	Pizzicato Strings	78	Shakuhachi	110	Bag Pipe
15	Tubular Bells	47	Orchestral Harp	79	Whistle	111	Fiddle
16	Dulcimer	48	Timpani	80	Ocarina	112	Shanai
17	Drawbar Organ	49	String Ensemble 1	81	Lead 1 (Square)	113	Tinkle Bell
18	Percussive Organ	50	String Ensemble 2	82	Lead 2 (Sawtooth)	114	Agogo
19	Rock Organ	51	Synth Strings 1	83	Lead 3 (Calliope)	115	Steel Drums
20	Church Organ	52	Synth Strings 2	84	Lead 4 (Chiff)	116	Woodblock
21	Reed Organ	53	Choir Aahs	85	Lead 5 (Charang)	117	Tallo Drum
22	Accordion	54	Choir Oohs	86	Lead 6 (Voice)	118	Melodic Tom
23	Harmonica	55	Synth Voice	87	Lead 7 (Fifths)	119	Synth Drum
24	Tango Accordion	56	Orchestral Hit	88	Lead 8 (Bass + Lead)	120	Reverse Cymbal
25	Acoustic Guitar (Nylon)	57	Trumpet	89	Pad 1 (New Age)	121	Guitar Fret Noise
26	Acoustic Guitar (Steel)	58	Trombone	90	Pad 2 (Warm)	122	Breathe Noise
27	Electric Guitar (Jazz)	59	Tuba	91	Pad 3 (Polysynth)	123	Seashore
28	Electric Guitar (Clean)	60	Muted Trumpet	92	Pad 4 (Choir)	124	Bird Tweet
29	Electric Guitar (Muted)	61	French Horn	93	Pad 5 (Bowed)	125	Telephone Ring
30	Overdriven Guitar	62	Brass Section	94	Pad 6 (Metallic)	126	Helicopter
31	Distortion Guitar	63	Synth Brass 1	95	Pad 7 (Halo)	127	Applause
32	Guitar Harmonics	64	Synth Brass 2	96	Pad 8 (Sweep)	128	Gunshot

## APPENDIX H MIDI INSTRUMENT GROUPINGS

1 – 8	Piano
9 – 16	Chromatic Percussion
17 – 24	Organ
25 – 32	Guitar
33 – 40	Bass
41 – 48	Strings
49 – 56	Ensemble
57 – 64	Brass
65 – 72	Reed
73 – 80	Pipe
81 – 88	Synth Lead
89 – 96	Synth Pad
97 – 104	Synth Effects
105 – 112	Ethnic
113 – 120	Percussive
121 – 128	Sound Effects

## APPENDIX I MIDI META-MESSAGES AND MIDI CONTROLLERS

The general format of a MIDI meta-message is the octet 0xFF followed by a type 0x00 to 0x7F then a length, which is a variable length quantity one to four octets then length data octets. Not all the one hundred and twenty eight types are defined but a MIDI file reader should be able to ignore an undefined type [5].

0x00	0x02	#-hi	#-lo					Sequence Number
0x01	Length	Text						Text Event
0x02	Length	Text						Copyright Notice
0x03	Length	Text						Sequence/Track Name
0x04	Length	Text						Instrument Name
0x05	Length	Text						Lyric
0x06	Length	Text						Marker
0x07	Length	Text						Cue Point
0x20	0x01	Ch						MIDI Channel Prefix
0x2F	0x00							End of Track (Mandatory)
0x51	0x03	T1	T2	T3				Tempo T1 Highest Order Octet
0x54	0x05	Hrs	Min	Sec	Fr	FF		SMTPE Offset
0x58	0x04	Num	Den	MC	TS			Time Signature
0x59	0x02	#-s	MM					Key Signature
0x7F	Length	Id	Data					Sequencer-Specific Meta-Event

Length is a variable length quantity. Text is a series of Length octets. Fr is the number of frames, FF is the frame fraction, Num is the time signature numerator, Den is the time signature denominator exponent, MC is the MIDI clocks per metronome tick, TS is the number of 32<sup>nd</sup> notes per quarter note, #-s is the number of sharps or flats – 7 is 7 flats, + 7 is 7 sharps, 0 is the key of C, MM is 0 for a major key or 1 for a minor key, Id is 1 to 3 octets in length representing a manufacturer's id and is a variable length quantity, and Data is Length – length of id in length data octets [5]. The following two routines allow one to read and write variable length quantities [6].

APPENDIX J UT-RTP-ND & UT-RTP-NE VERSUS SN-TCP-ND & SN-TCP-NE

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66944.66	60	423.8622	54.7203
SN-TCP-ND	67183.00	60	765.3320	98.8039

Table AE-1 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-SN-TCP-ND	-238.3333	901.3420	116.3627	-2.0481	59	0.0449

Table J-2 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66944.66	60	423.8622	54.7203
SN-TCP-NE	67498.00	60	840.9998	108.5726

Table J-3 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-SN-TCP-NE	-553.3333	938.7598	121.1933	-4.5657	59	0.0000

Table J-4 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66944.66	60	423.8622	54.7203
UT-RTP-NE	67471.33	60	423.8622	251.2846

Table J-5 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-UT-RTP-NE	-526.6666	1997.1638	257.8327	-2.0426	59	0.0455

Table J-6 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67471.33	60	1946.4426	251.2846
SN-TCP-ND	67183.00	60	765.3320	98.8039

Table J-7 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-ND	288.3333	2107.9133	272.1304	1.0595	59	0.2936

Table J-8 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67471.33	60	1946.4426	251.2846
SN-TCP-NE	67498.00	60	840.9998	108.5726

Table J-9 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-NE	-26.6666	2140.4962	276.3368	-0.0965	59	0.9234

Table J-10 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67183.00	60	765.3320	98.8039
SN-TCP-NE	67498.00	60	840.9998	108.5726

Table J-11 Trippygaial.mid m = 1

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-315.0000	1158.5957	149.5740	-2.1059	59	0.0394

Table J-12 Trippygaial.mid m = 1

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66961.16	60	535.5658	69.1412
SN-TCP-ND	66952.66	60	49.3986	6.3773

Table J-13 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND- SN-TCP-ND	8.5000	545.3775	70.4079	0.1207	59	0.9043

Table J-14 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66961.16	60	535.5658	69.1412
SN-TCP-NE	67285.50	60	253.8211	32.7681

Table J-15 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND -SN-TCP-NE	-324.3333	586.3774	75.7010	-4.2843	59	0.0000

Table J-16 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	66961.16	60	535.5658	69.1412
UT-RTP-NE	67554.33	60	2134.2599	275.5317

Table J-17 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-UT-RTP-NE	-593.1666	2201.6784	284.2354	-2.0868	59	0.0412

Table J-18 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	66952.66	60	49.3986	6.3773
SN-TCP-NE	67285.50	60	253.8211	32.7681

Table J-19 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-332.8333	260.2807	33.6021	-9.9051	59	0.0000

Table J-20 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67554.33	60	2134.2599	275.5317
SN-TCP-ND	66952.66	60	49.3986	6.3773

Table J-21 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-ND	601.66	2131.8498	275.2206	2.1861	59	0.0327

Table J-22 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67554.33	60	2134.2599	275.5317
SN-TCP-NE	67285.50	60	253.8211	32.7681

Table J-23 Trippygaial.mid m = 2

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-NE	268.8333	2160.2660	278.8891	0.9639	59	0.3390

Table J-24 Trippygaial.mid m = 2

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67233.33	60	1467.7766	189.4891
SN-TCP-ND	67022.66	60	52.5894	6.7892

Table J-25 Trippygaial.mid m = 3

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND- SN-TCP-ND	210.6666	1457.4494	188.1559	1.1196	59	0.2674

Table J-26 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67233.33	60	1467.7766	189.4891
SN-TCP-NE	67624.66	60	2311.7206	298.4418

Table J-27 Trippygaial.mid m = 3

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND -SN-TCP-NE	-391.3333	1191.8742	153.8703	-2.5432	59	0.0136

Table J-28 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67233.33	60	1467.7766	189.4891
UT-RTP-NE	67285.66	60	331.2910	42.7694

Table J-29 Trippygaial.mid m = 3

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-UT-RTP-NE	-52.3333	1521.6531	196.4445	-0.2664	59	0.7908

Table J-30 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67022.66	60	52.5894	6.7892
SN-TCP-NE	67624.66	60	2311.7206	298.4418

Table J-31 Trippygaial.mid m = 3



Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-602.0000	2300.3034	296.9678	-2.0271	59	0.04717

Table J-32 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67285.66	60	331.2910	42.7694
SN-TCP-ND	67022.66	60	52.5894	6.7892

Table J-33 Trippygaial.mid m = 3

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-ND	263.0000	337.5851	43.5820	6.0345	59	0.0000

Table J-34 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67285.66	60	331.2910	42.7694
SN-TCP-NE	67624.66	60	2311.7206	298.4418

Table J-35 Trippygaial.mid m = 3

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-NE	-339.0000	2354.3920	303.9507	-1.1153	59	0.2692

Table J-36 Trippygaial.mid m = 3

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67299.83	60	1739.1099	224.5181
SN-TCP-ND	67068.00	60	355.1118	45.8447

Table J-37 Trippygaial.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND- SN-TCP-ND	231.8333	1788.3128	230.8701	1.0041	59	0.3193

Table J-38 Trippygaial.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67299.83	60	1739.1099	224.5181
SN-TCP-NE	67426.83	60	1448.4772	186.9976

Table J-39 Trippygaial.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND -SN-TCP-NE	-127.0000	2286.5230	295.1888	-0.4302	59	0.6685

Table J-40 Trippygaial.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-ND	67299.83	60	1739.1099	224.5181
UT-RTP-NE	67228.83	60	75.5565	9.7543

Table J-41 Trippygaial.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-ND-UT-RTP-NE	71.0000	1760.2471	227.2469	0.3124	59	0.7558

Table J-42 Trippygaial.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67068.00	60	355.1118	45.8447
SN-TCP-NE	67426.83	60	1448.4772	186.9976

Table J-43 Trippygaial.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-358.8333	1504.7203	194.2585	-1.8471	59	0.0697

Table J-44 Trippygaial.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67228.83	60	75.5565	9.7543
SN-TCP-ND	67068.00	60	355.1118	45.8447

Table J-45 Trippygaial.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-ND	160.8333	355.6387	45.9127	3.5030	59	0.0008

Table J-46 Trippygaial.mid m = 4

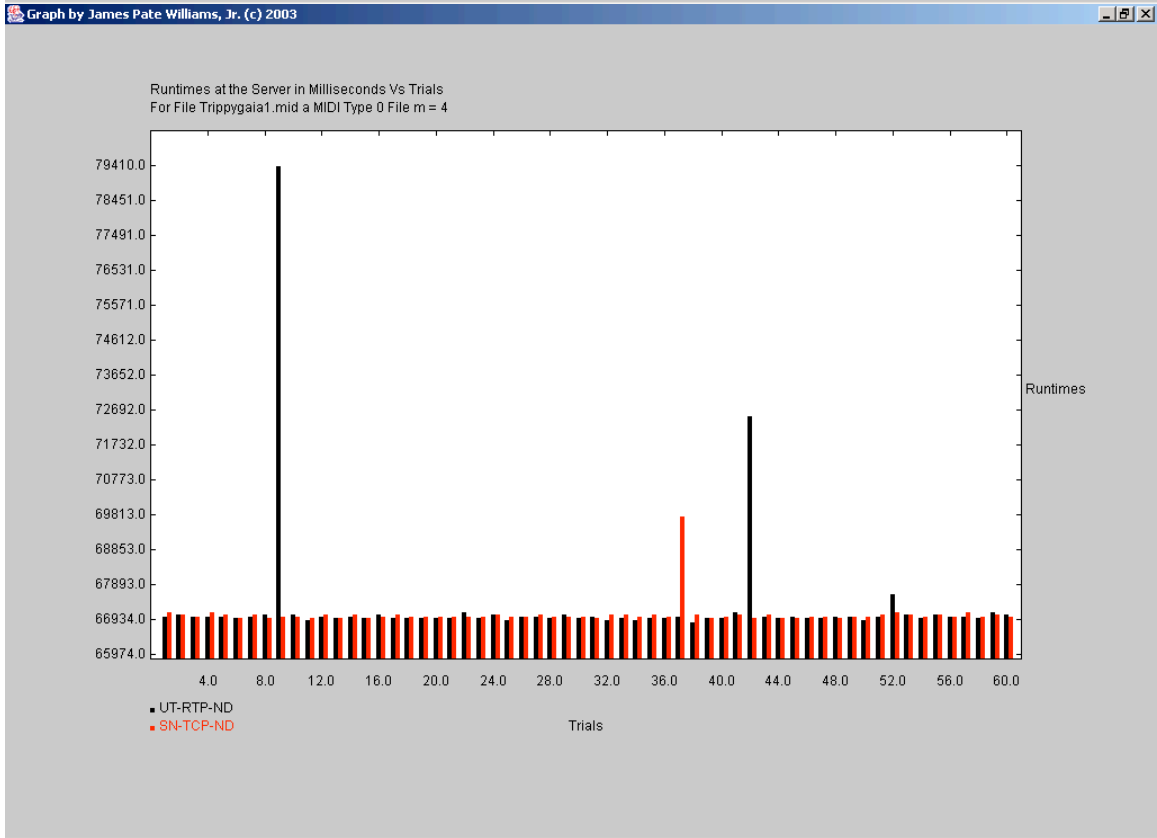
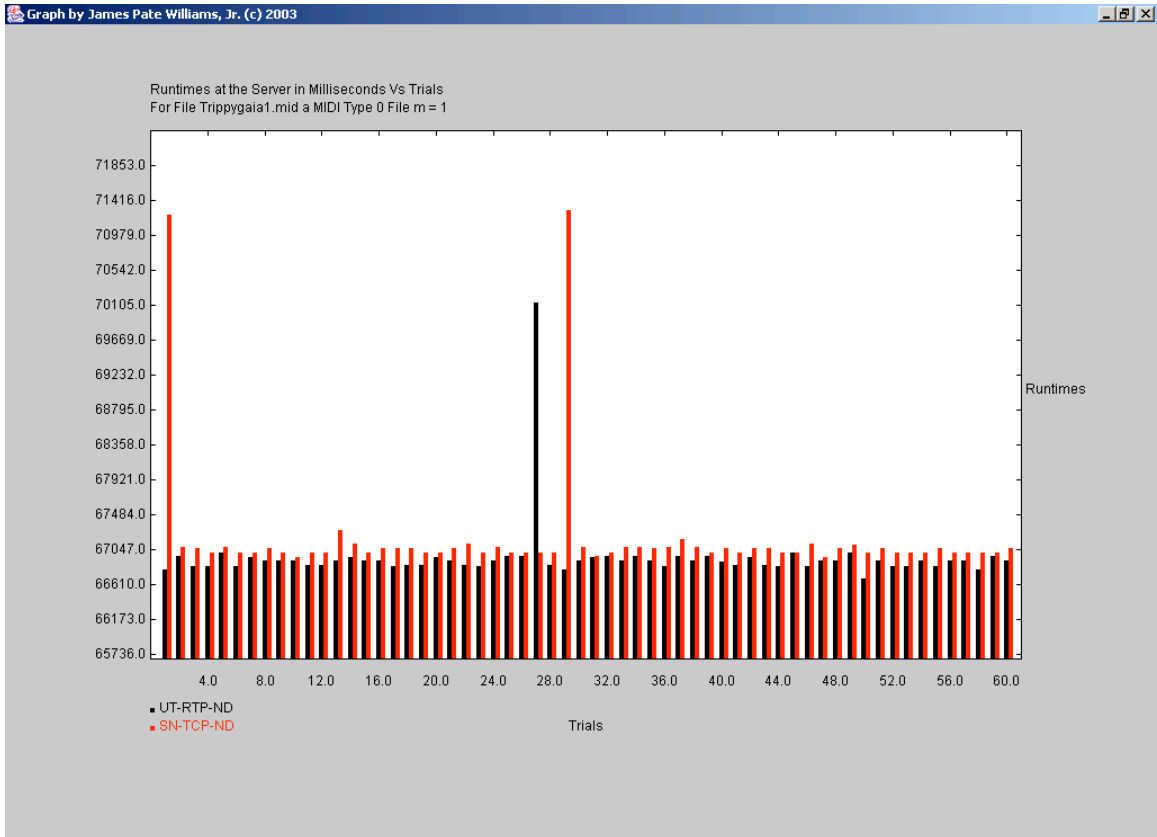
Protocol	Mean	N	Std. Dev.	Std. Error Mean
UT-RTP-NE	67228.83	60	75.5565	9.7543
SN-TCP-NE	67426.83	60	1448.4772	186.9976

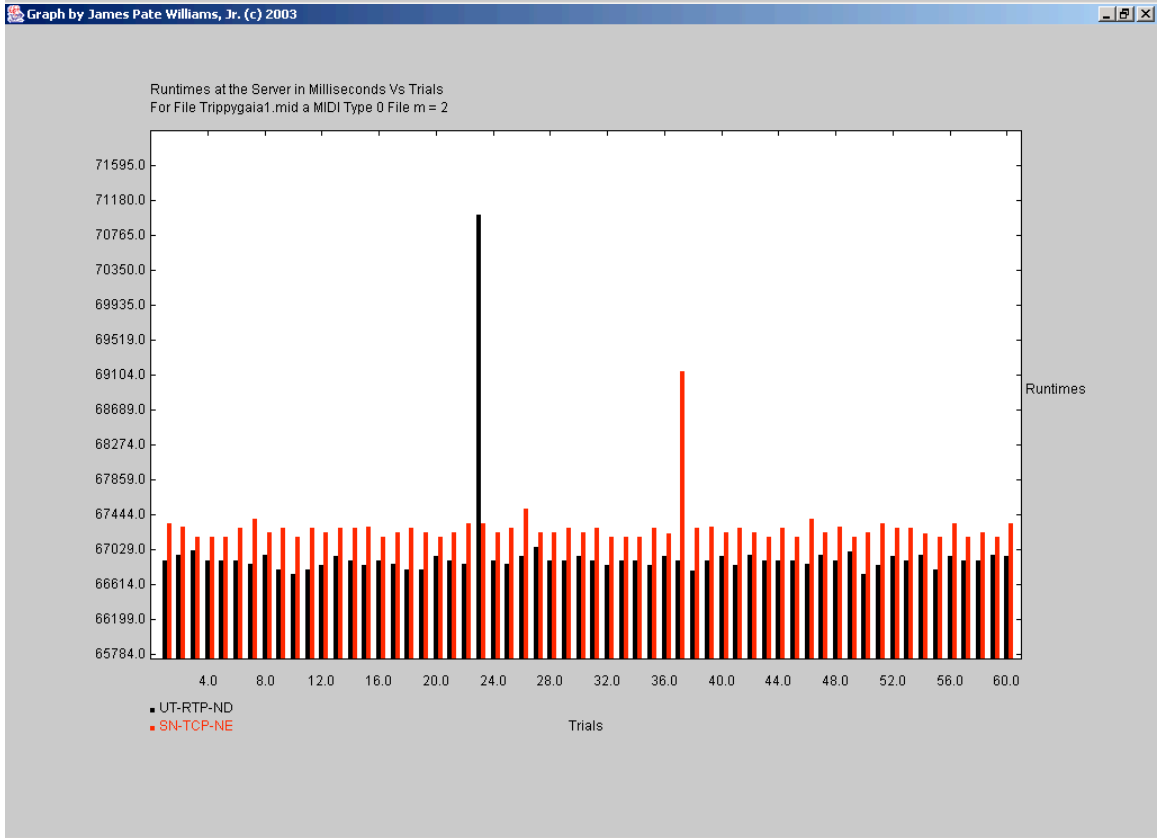
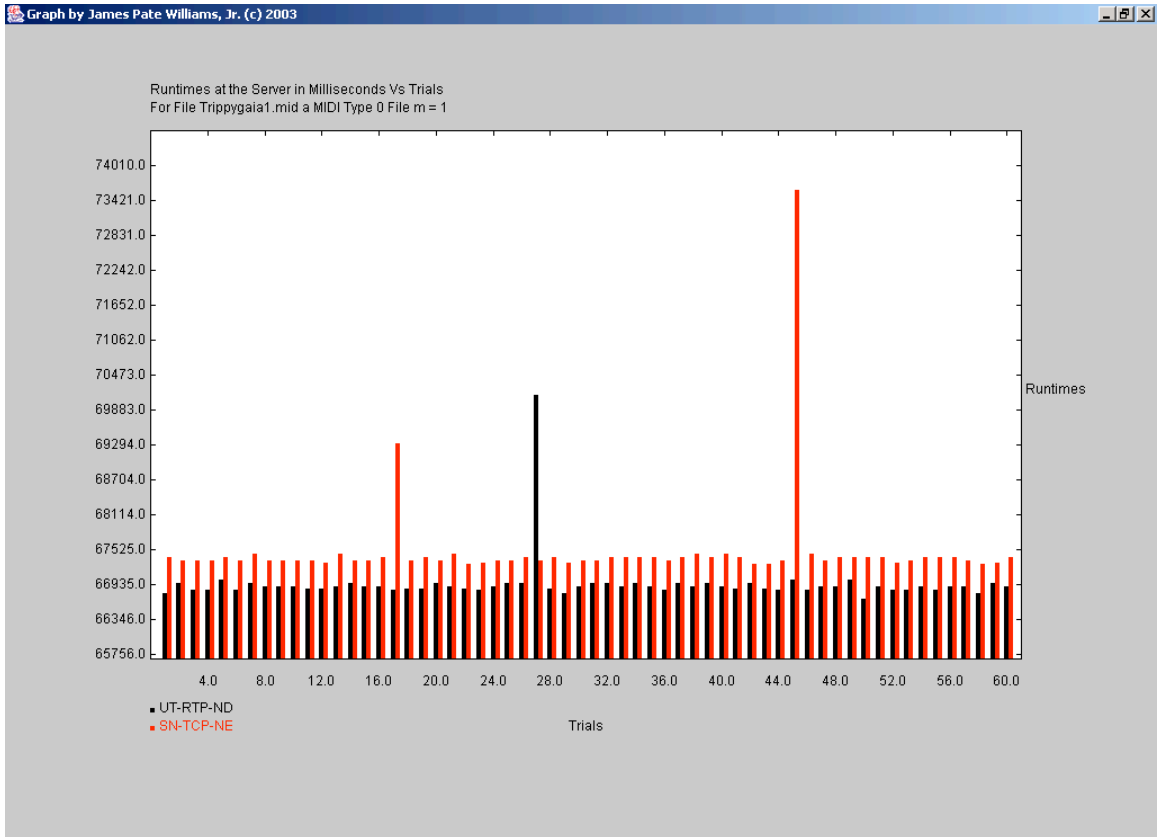
Table J-47 Trippygaial.mid m = 4

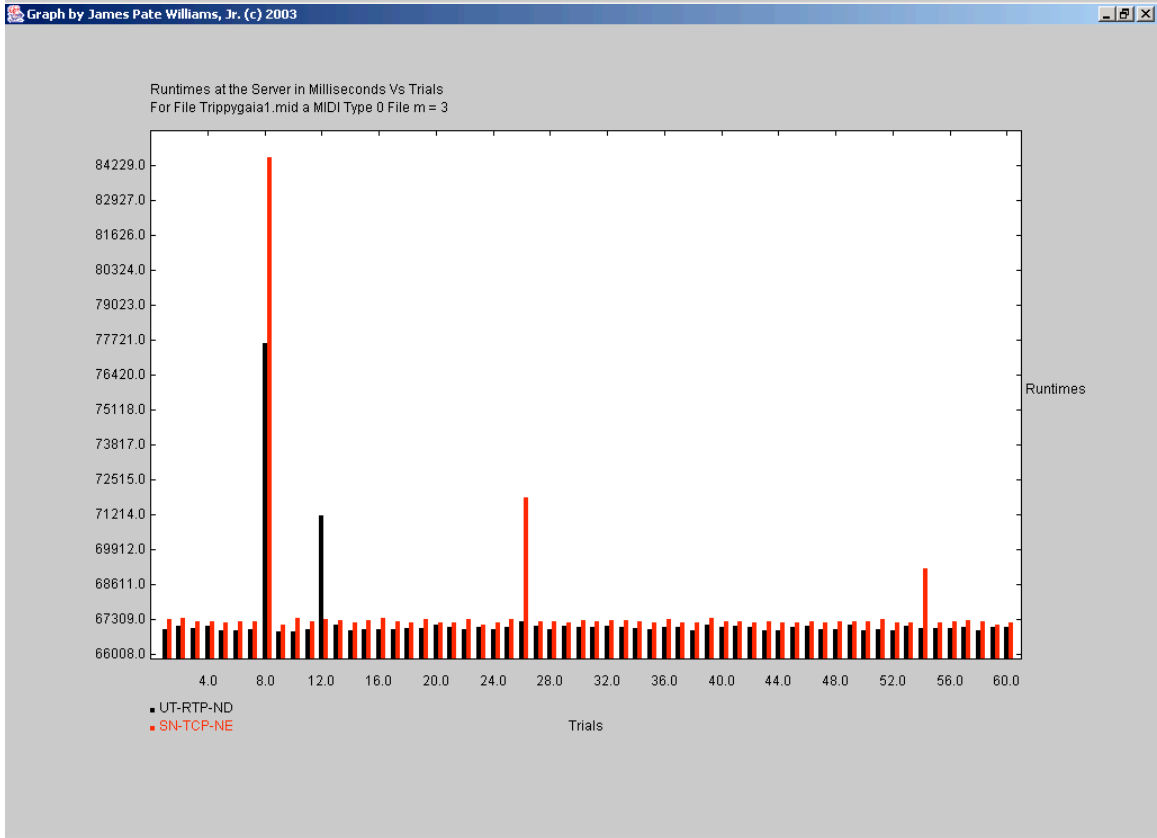
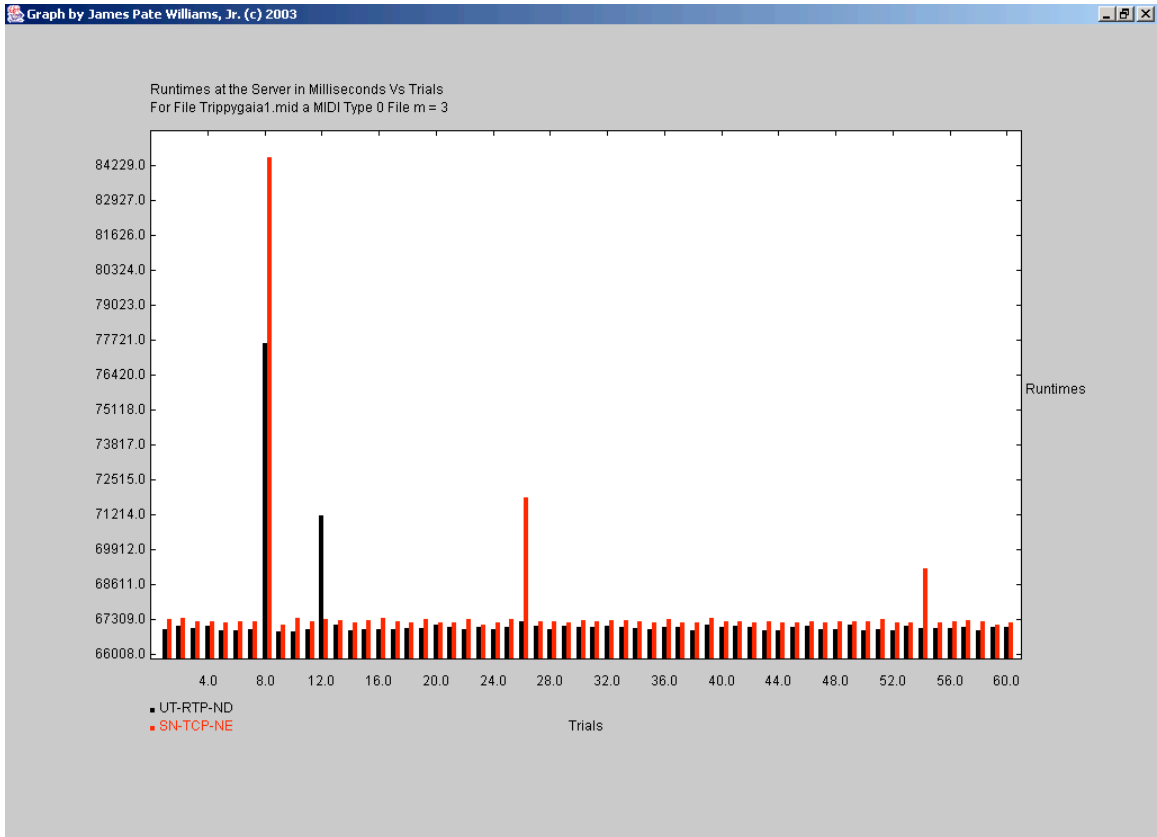
Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
UT-RTP-NE-SN-TCP-NE	-198.0000	1464.4057	189.0539	-1.0473	59	0.2992

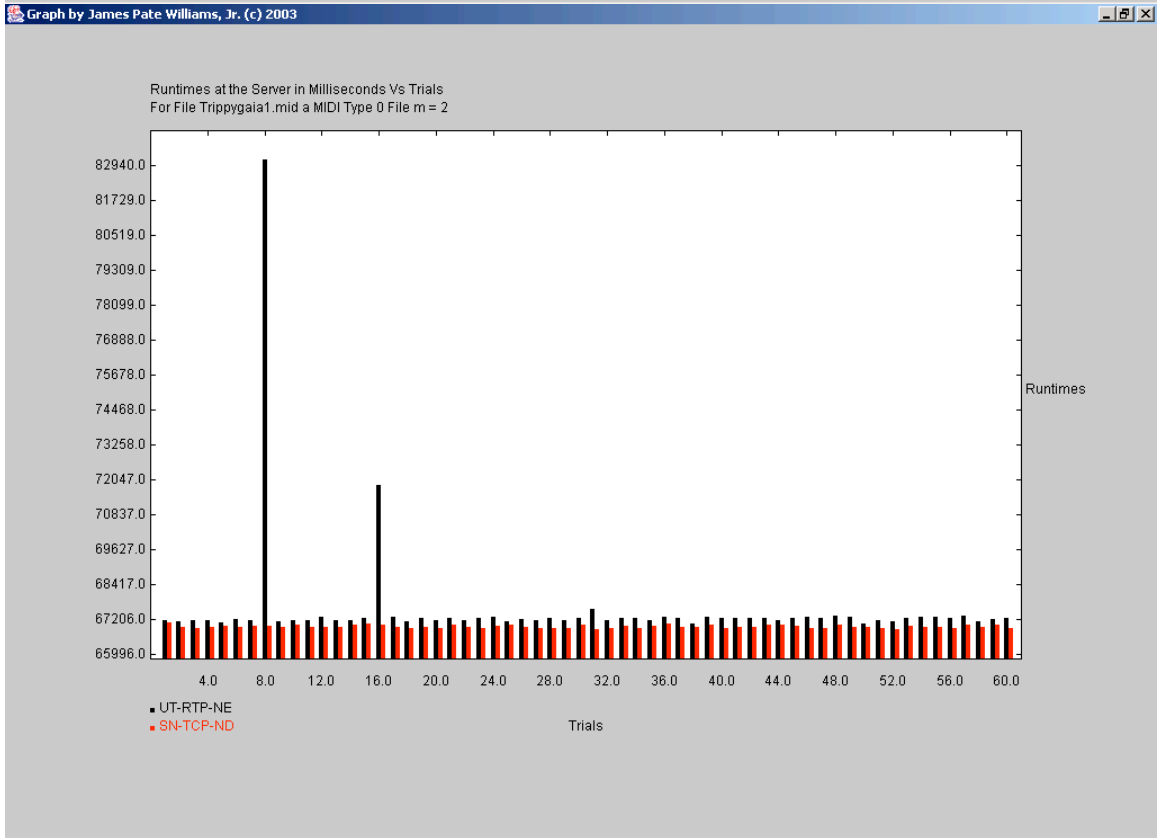
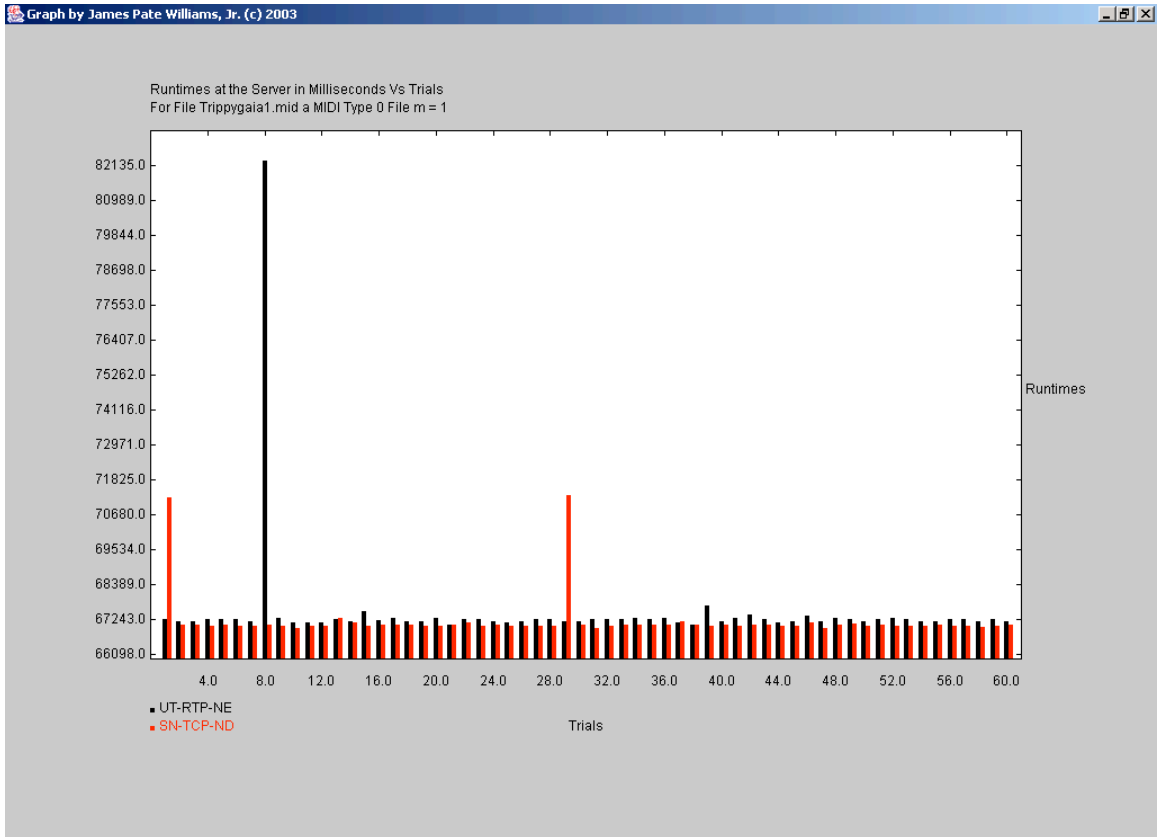
Table J-48 Trippygaial.mid m = 4

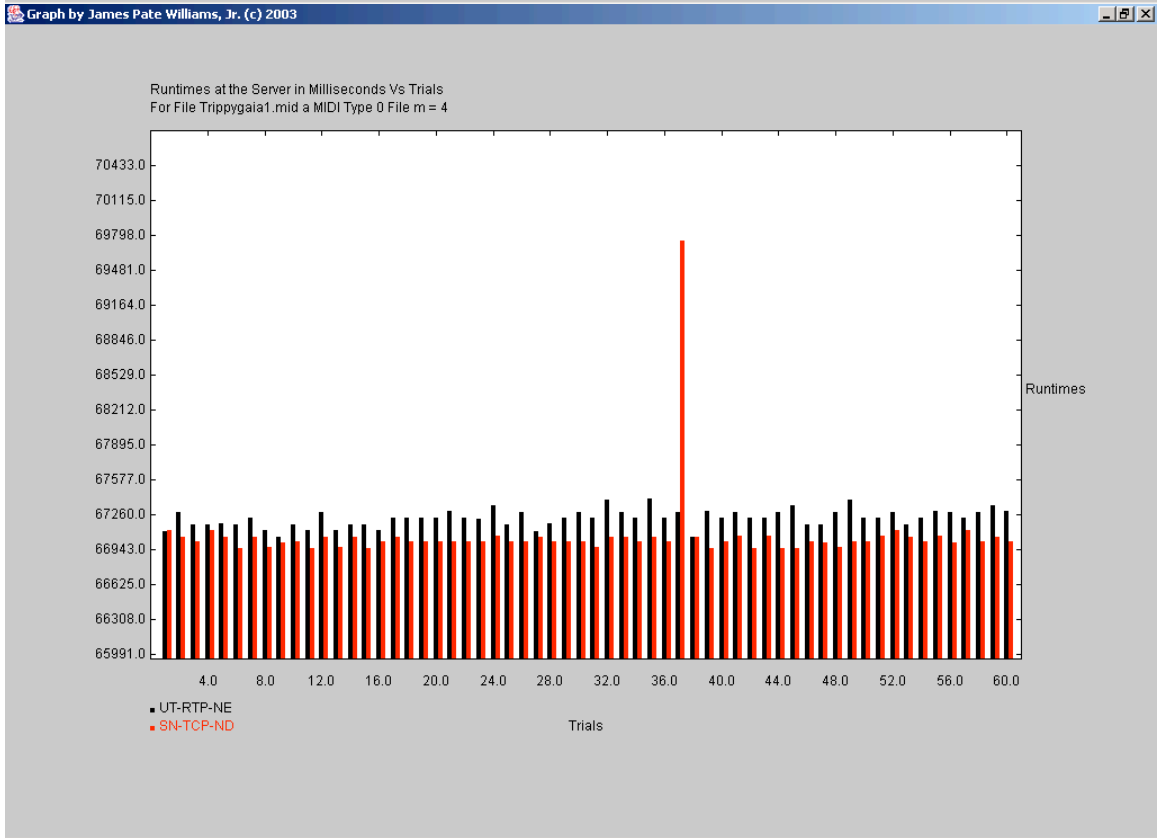
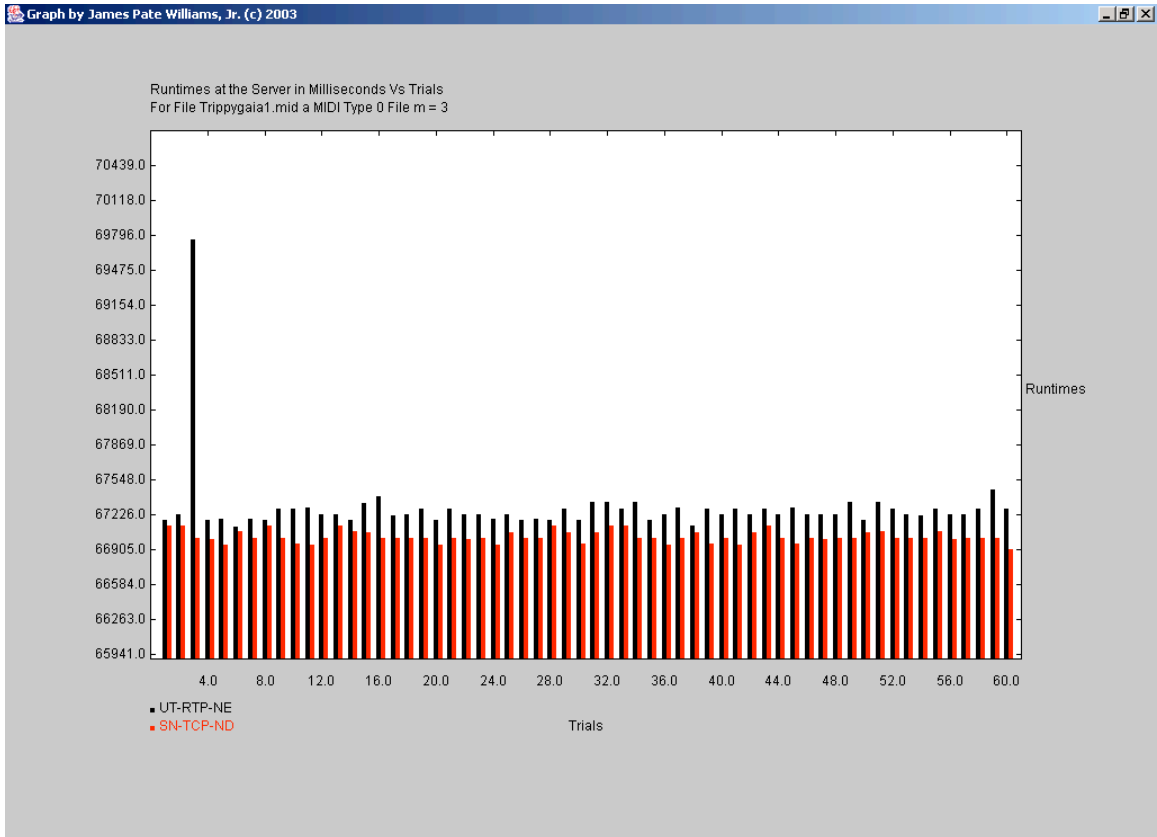
In the figures on the following pages the captions on the left are for the above figure and the captions on the left are for the below figure.

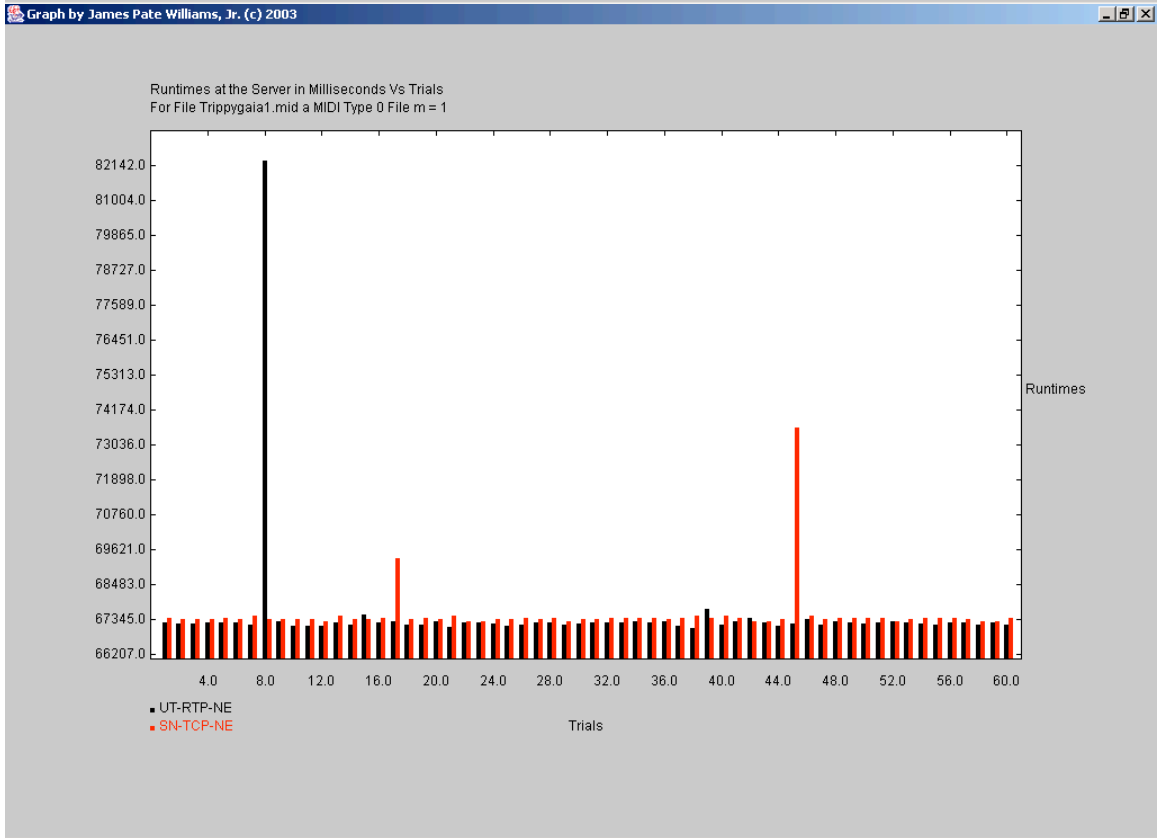
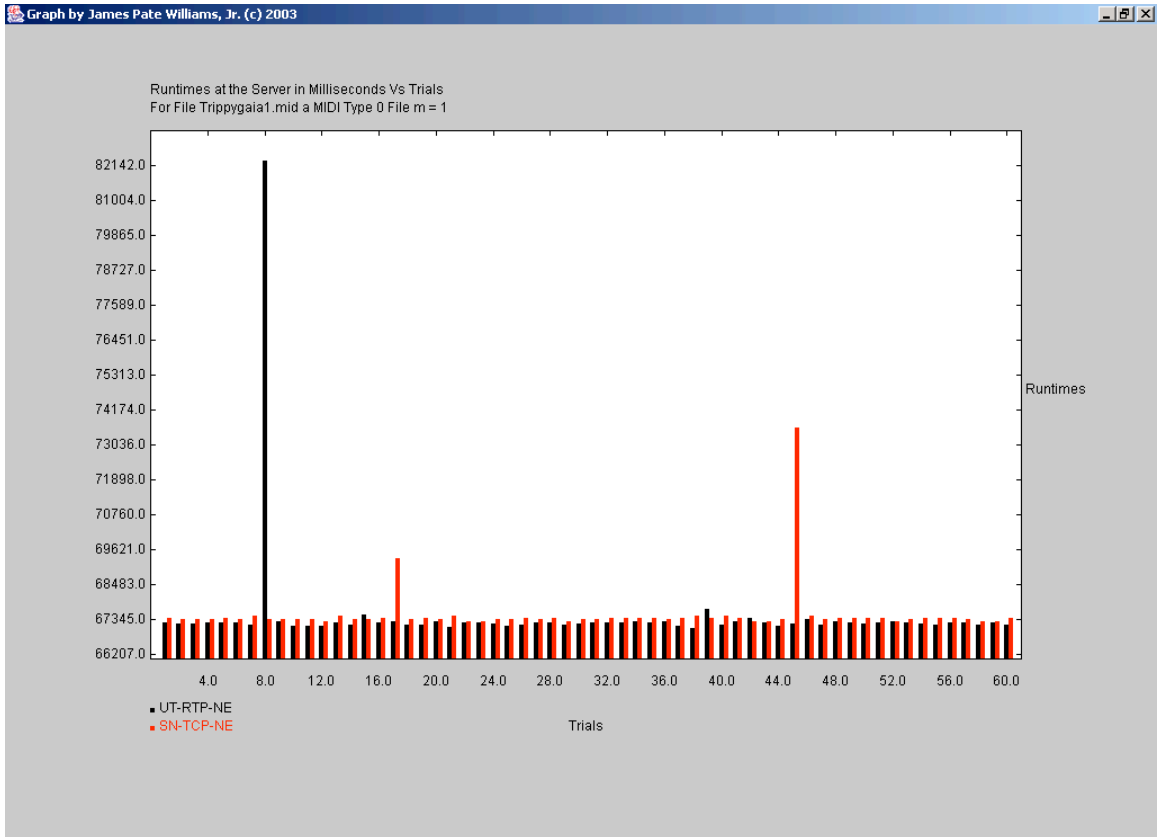




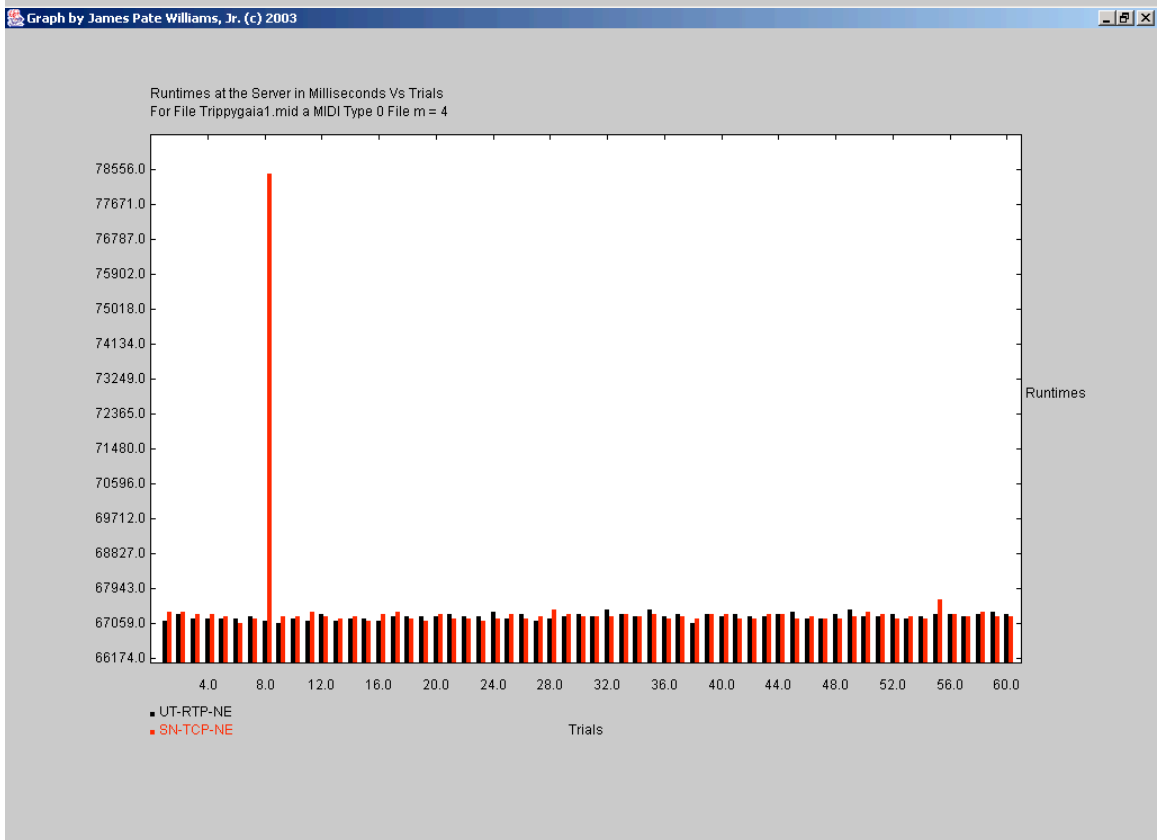
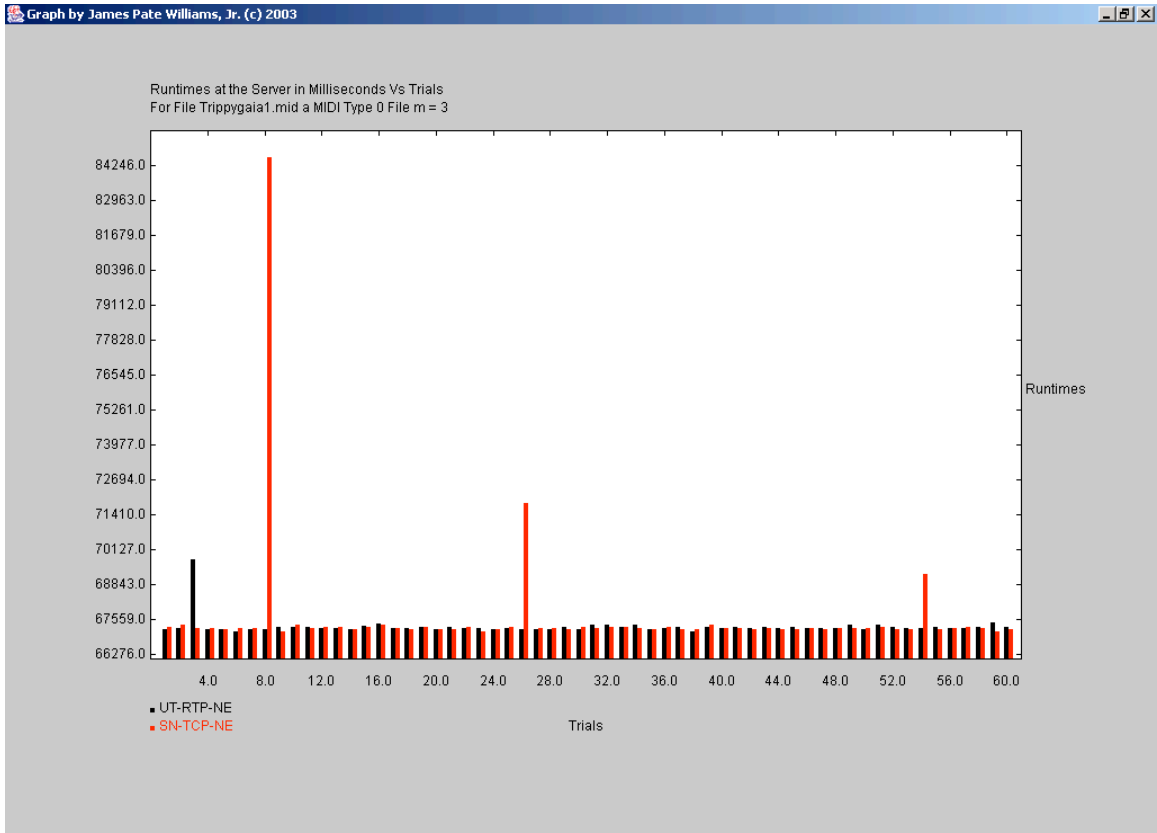


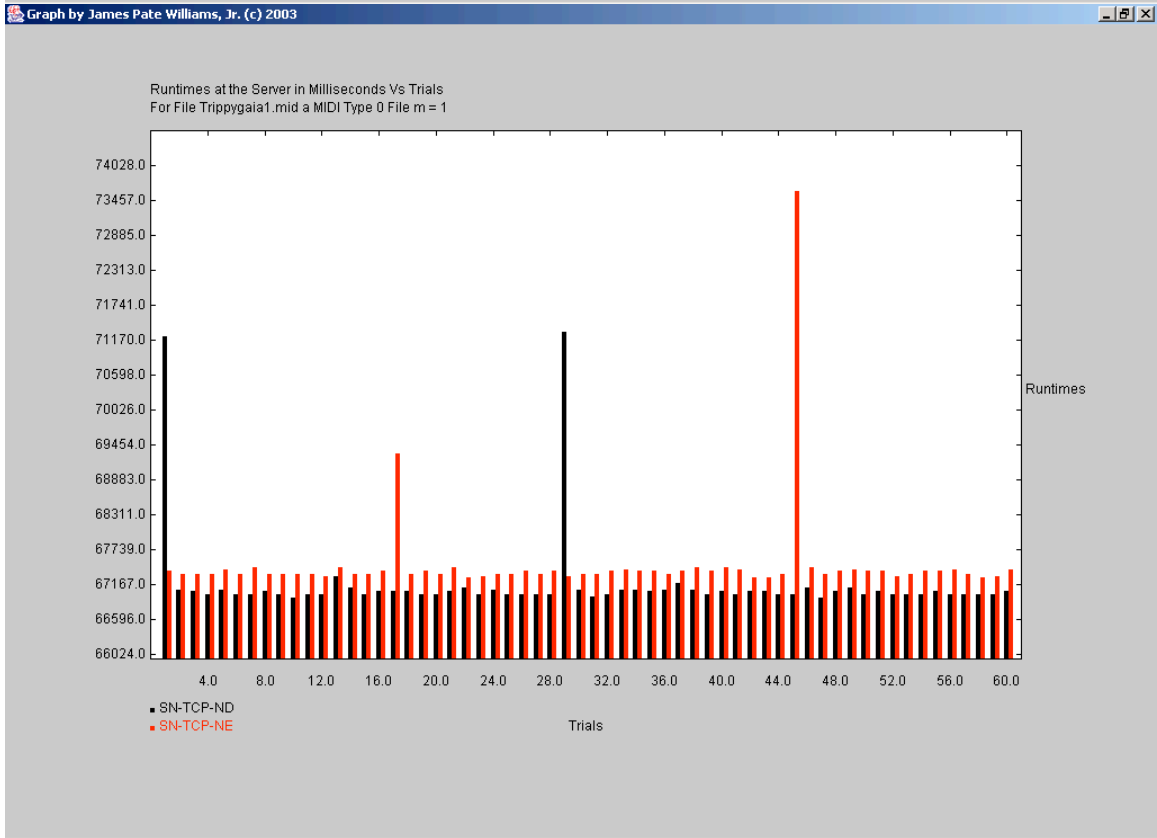
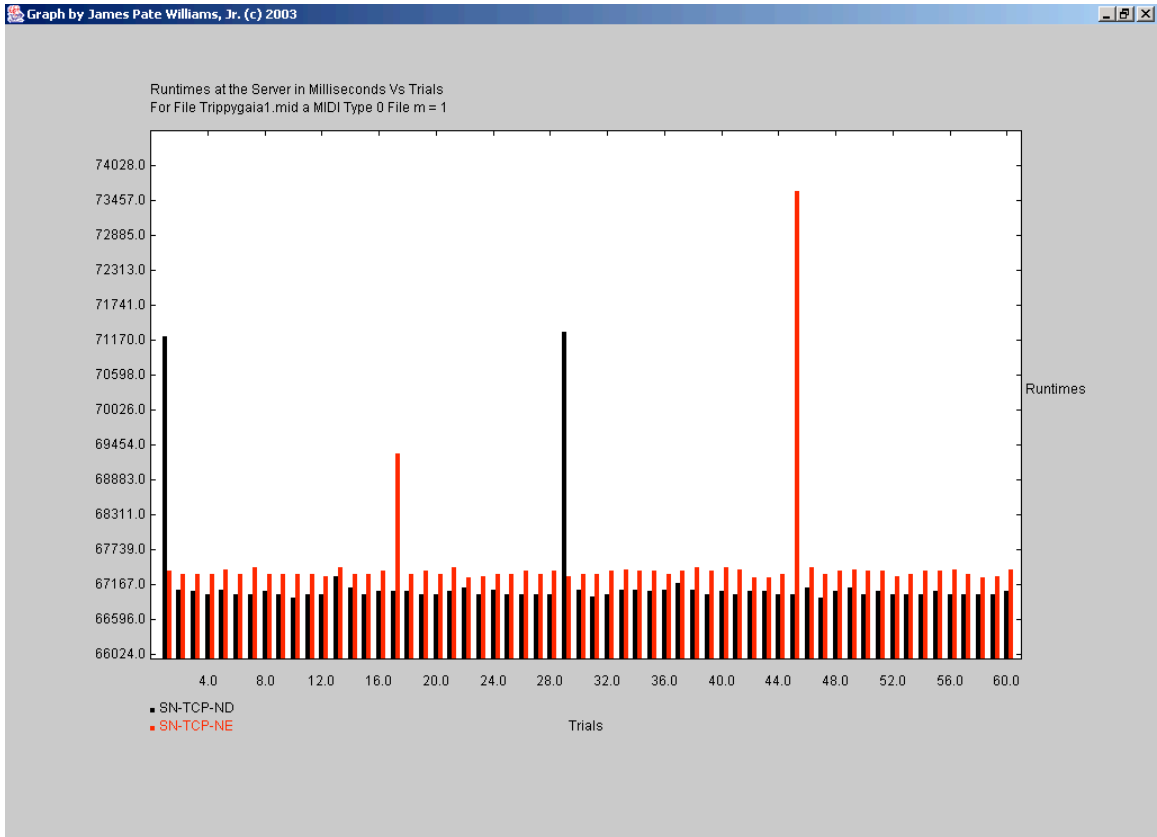


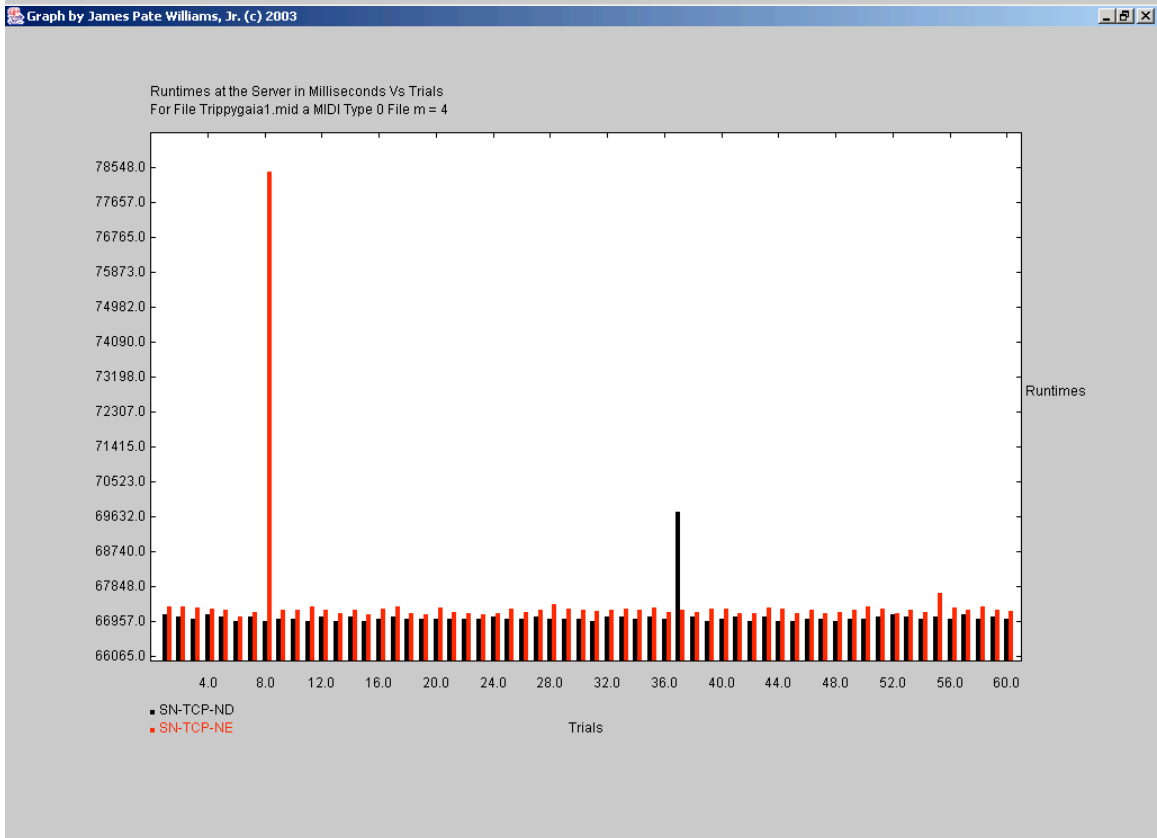
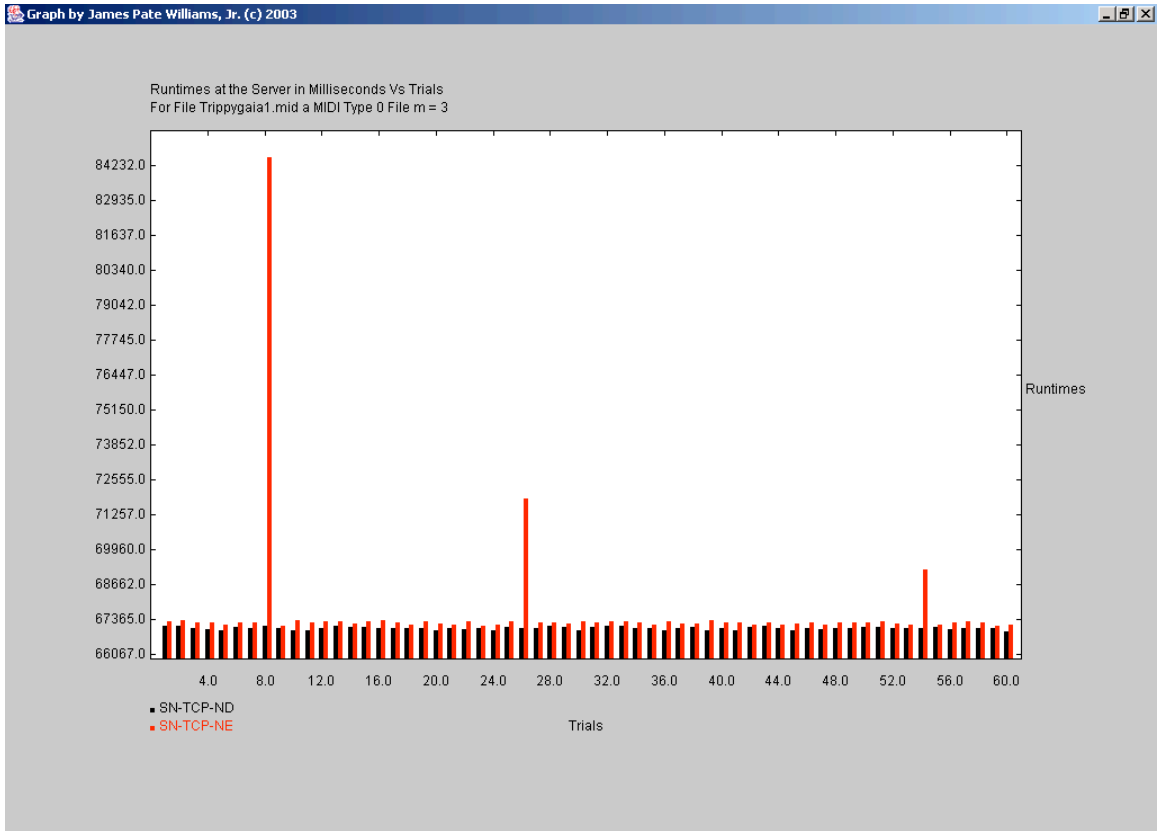


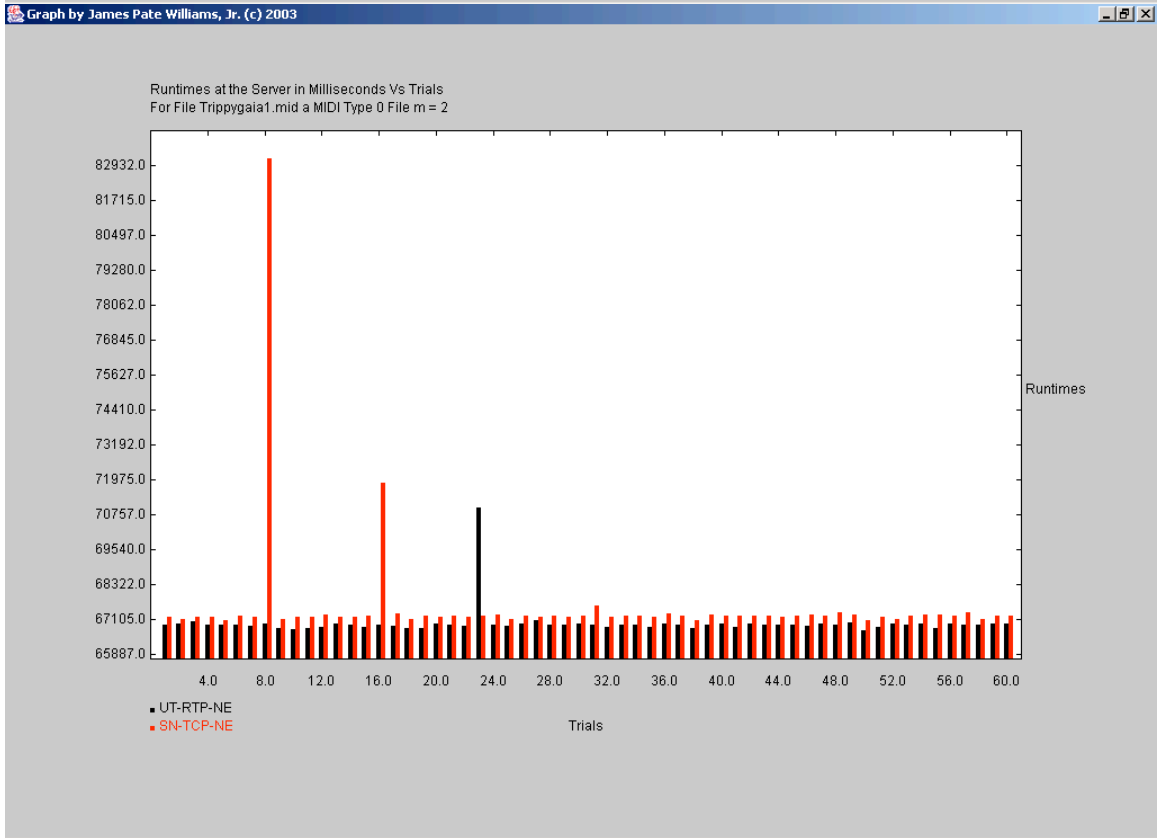
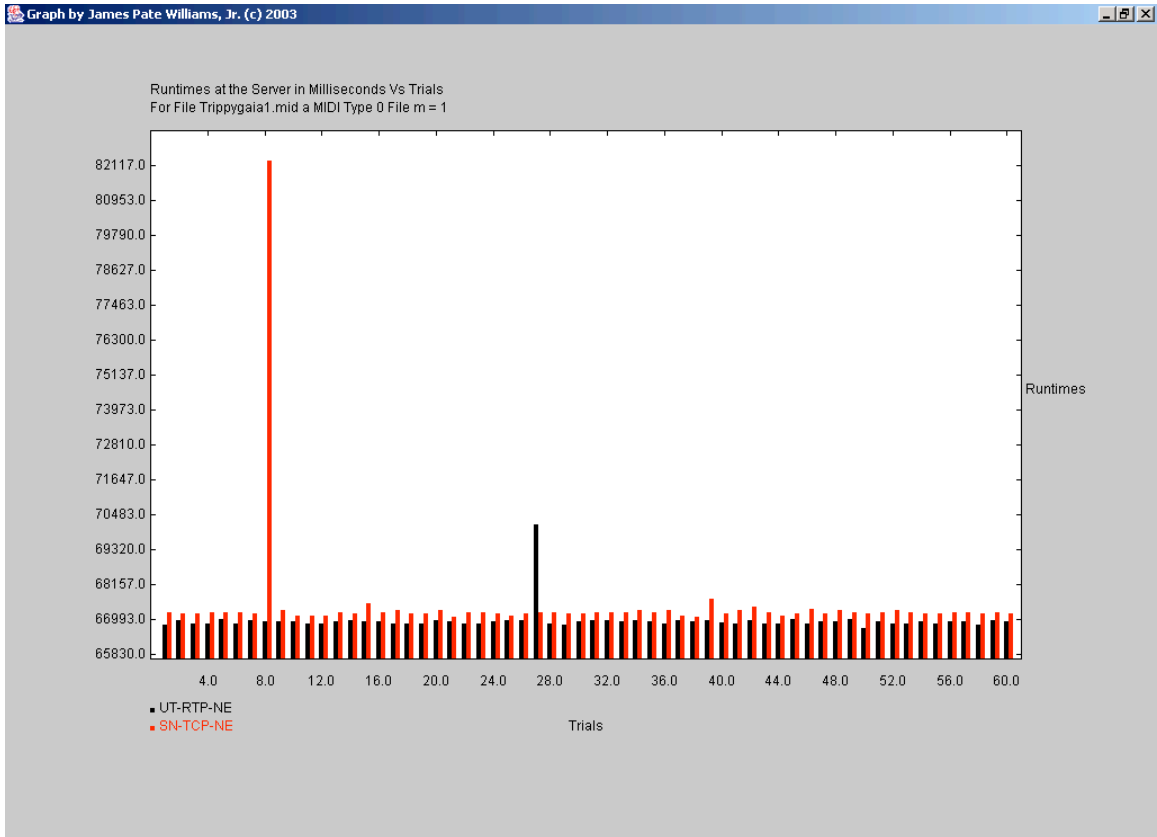


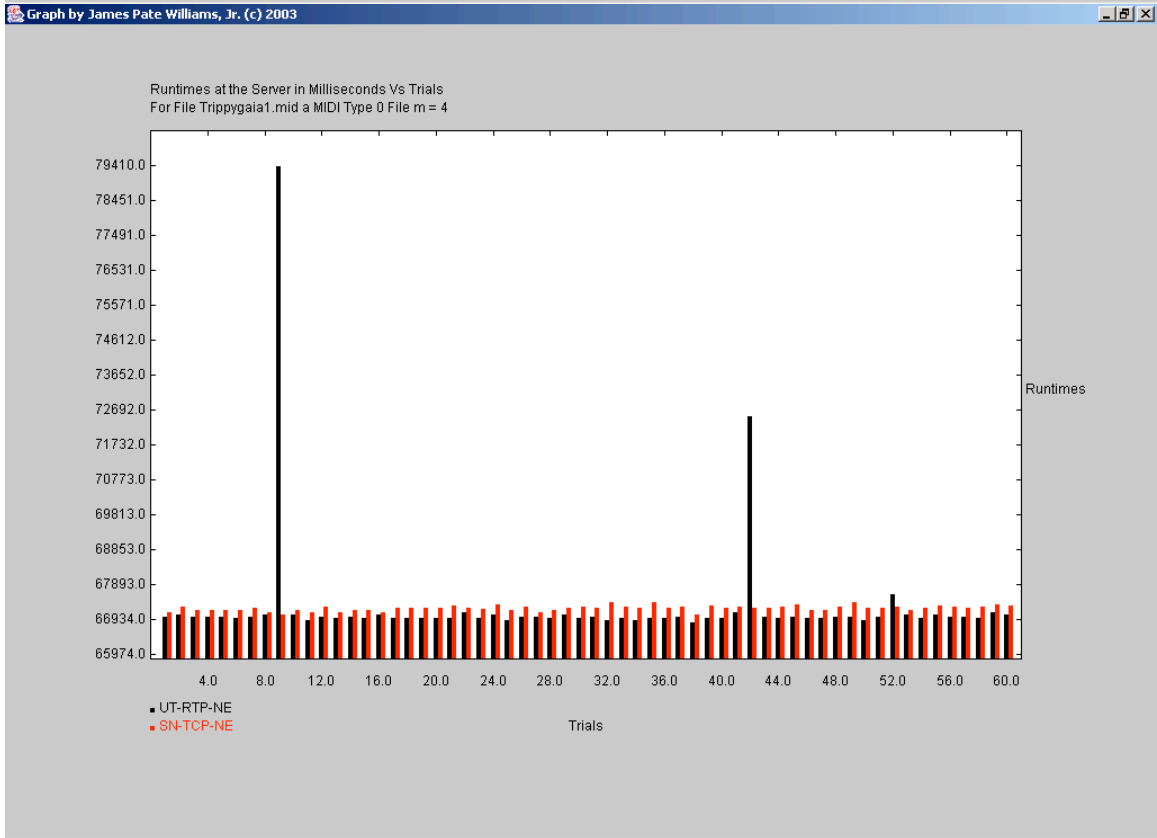
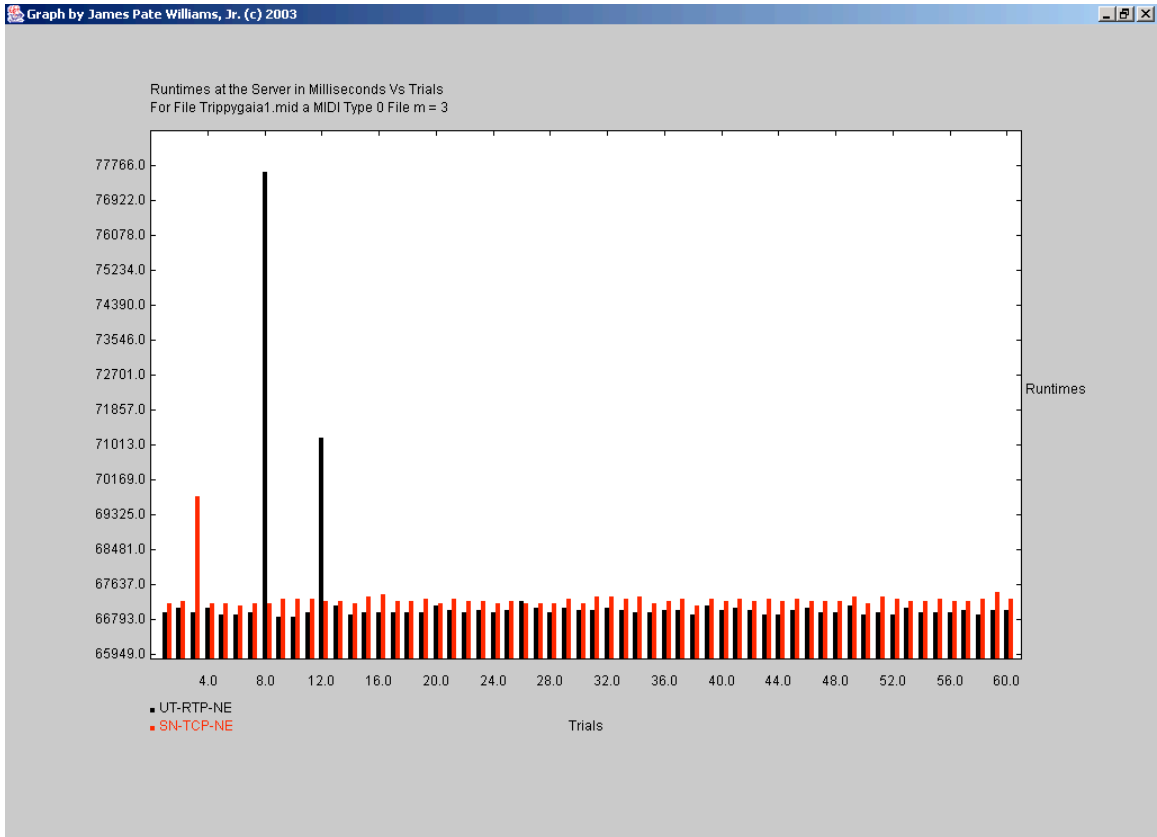












APPENDIX K ATCP-32 VERSUS SN-TCP-ND AND SN-TCP-NE

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	67104.00	60	2151.6688	277.7792
SN-TCP-ND	67119.83	60	229.6902	29.6528

Table K-1 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-ND	-15.8333	2163.0614	279.2500	-0.0566	59	0.9549

Table K-2 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	67104.00	60	2151.6688	277.7792
SN-TCP-NE	67490.33	60	1521.7139	196.4524

Table K-3 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-NE	-386.3333	2650.2637	342.1475	-1.1291	59	0.2634

Table K-4 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67119.83	60	229.6902	29.6528
SN-TCP-NE	67490.33	60	1521.7139	196.4524

Table K-5 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-370.5000	1539.9190	198.8026	-1.8636	59	0.0673

Table K-6 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	67066.00	60	2039.9494	263.3563
SN-TCP-ND	67100.16	60	532.2019	68.7069

Table K-7 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-ND	-34.1666	2111.7751	272.6289	-0.1253	59	0.9006

Table K-8 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	67066.00	60	2039.9494	263.3563
SN-TCP-NE	67230.83	60	82.8985	10.7021

Table K-9 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-NE	-164.8333	2048.1744	264.4181	-0.6233	59	0.5354

Table K-10 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67100.16	60	532.2019	68.7069
SN-TCP-NE	67230.83	60	82.8985	10.7021

Table K-11 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-130.66	551.8746	71.2467	-1.8340	59	0.0716

Table K-12 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	66782.83	60	53.5230	6.9097
SN-TCP-ND	67044.83	60	54.3838	7.0209

Table K-13 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-ND	-262.0000	72.7382	9.3904	-27.9006	59	0.0000

Table K-14 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	66782.83	60	53.5230	6.9097
SN-TCP-NE	67432.83	60	1596.0301	206.0466

Table K-15 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-NE	-650.0000	1601.9024	206.8047	-3.1430	59	0.0026

Table K-16 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67044.83	60	54.3838	7.0209
SN-TCP-NE	67432.83	60	1596.0301	206.0466

Table K-17 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-388.0000	1601.2736	206.7235	-1.8769	59	0.0654

Table K-18 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	66778.66	60	43.4708	5.6120
SN-TCP-ND	67371.16	60	2434.3969	314.2792

Table K-19 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-ND	-592.5000	2426.6071	313.2736	-1.8913	59	0.06349

Table K-20 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	66778.66	60	43.4708	5.6120
SN-TCP-NE	67442.50	60	1561.3256	201.5662

Table K-21 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-NE	-663.8333	1553.6889	200.5803	-3.3095	59	0.0015

Table K-22 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67371.16	60	2434.3969	314.2792
SN-TCP-NE	67442.50	60	1561.3256	201.5662

Table K-23 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-71.3333	879.0932	113.4904	-0.6285	59	0.5320

Table K-24 Trippygaial.mid m = 7

ATCP-32	66758.83	60	48.9583	6.3204
SN-TCP-ND	67220.00	60	1214.1189	156.7420

Table K-25 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-ND	-461.1666	1225.3966	158.1980	-2.9151	59	0.0050

Table K-26 Trippygaial.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-32	66758.83	60	48.9583	6.3204
SN-TCP-NE	67298.83	60	81.4922	10.5206

Table K-27 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-32-SN-TCP-NE	-540.0000	85.9838	11.1004	-48.6466	59	0.0000

Table K-28 Trippygaial.mid m = 8

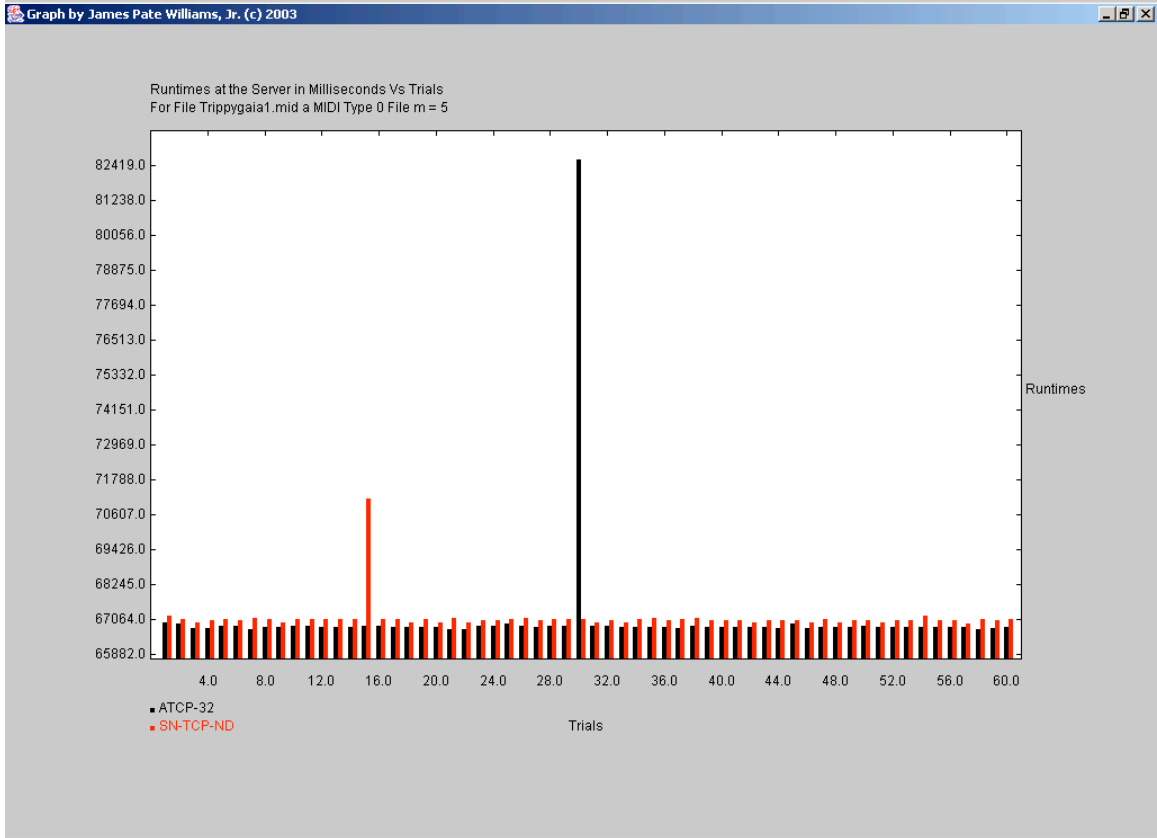
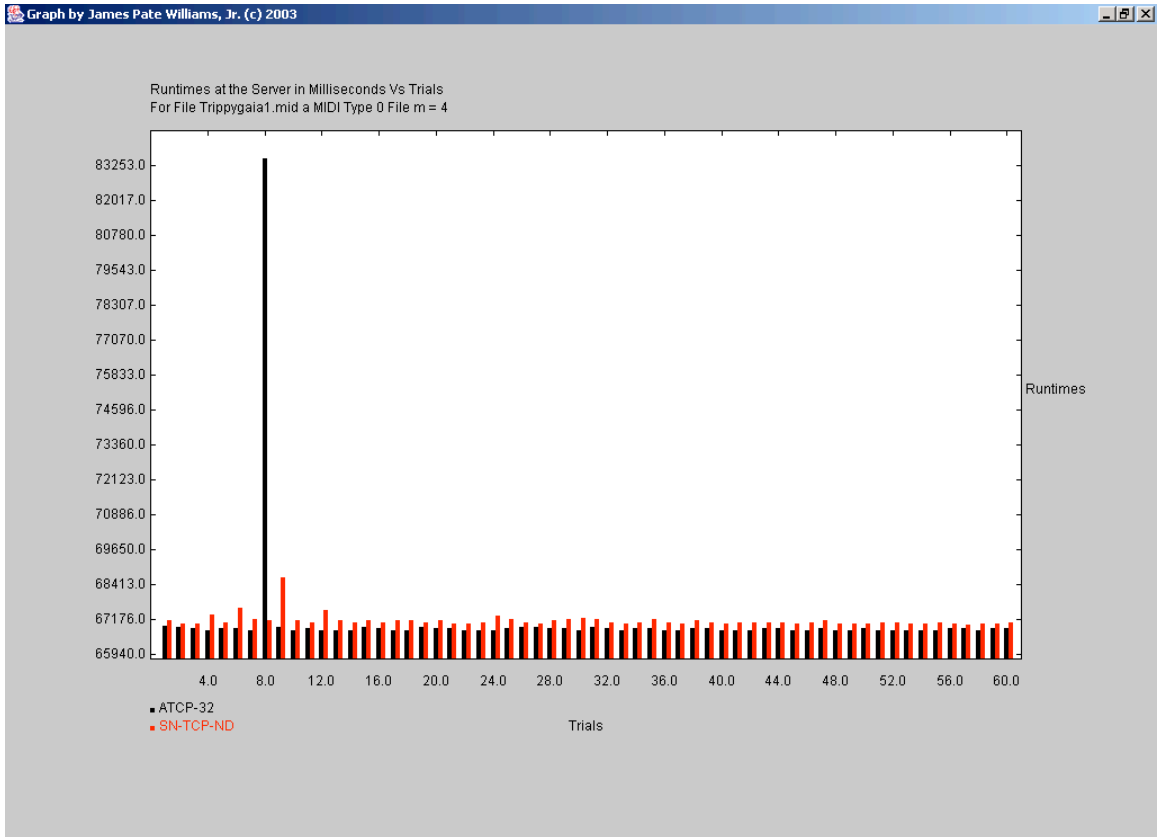
Protocol	Mean	N	Std. Dev.	Std. Error Mean
SN-TCP-ND	67220.00	60	1214.1189	156.7420
SN-TCP-NE	67298.83	60	81.4922	10.5206

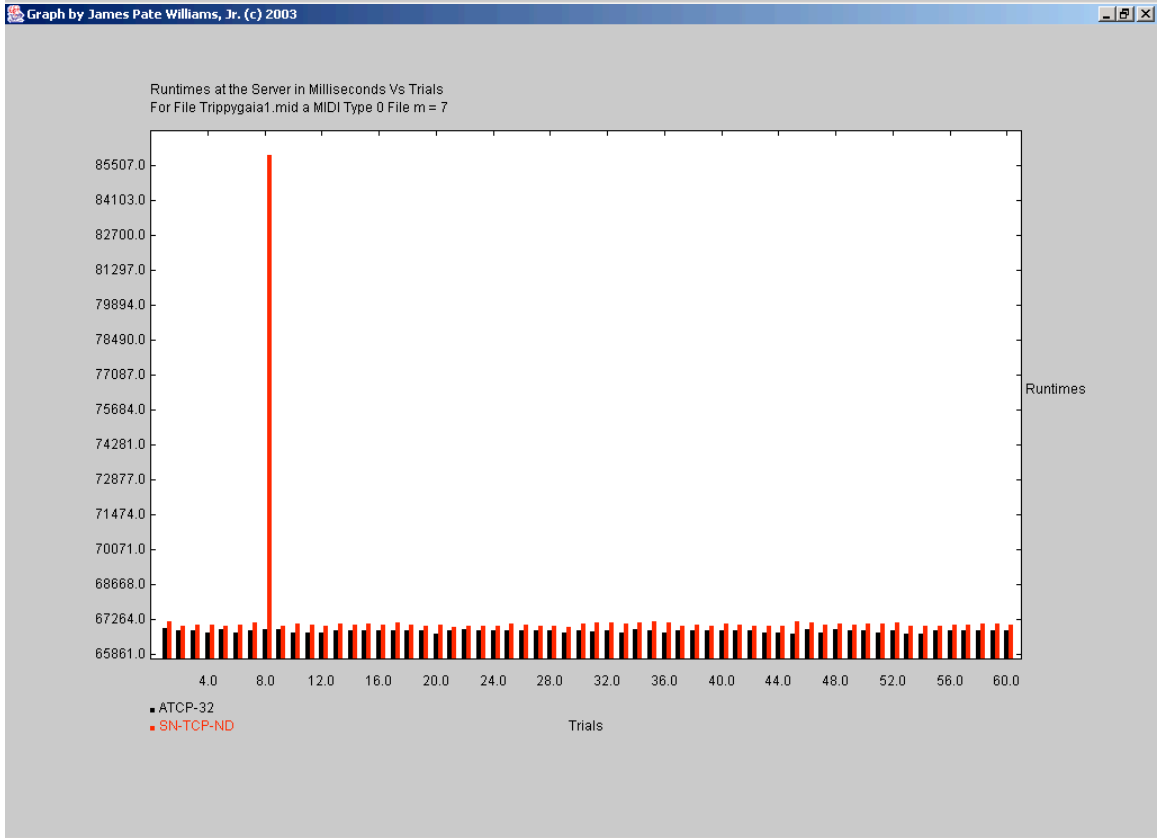
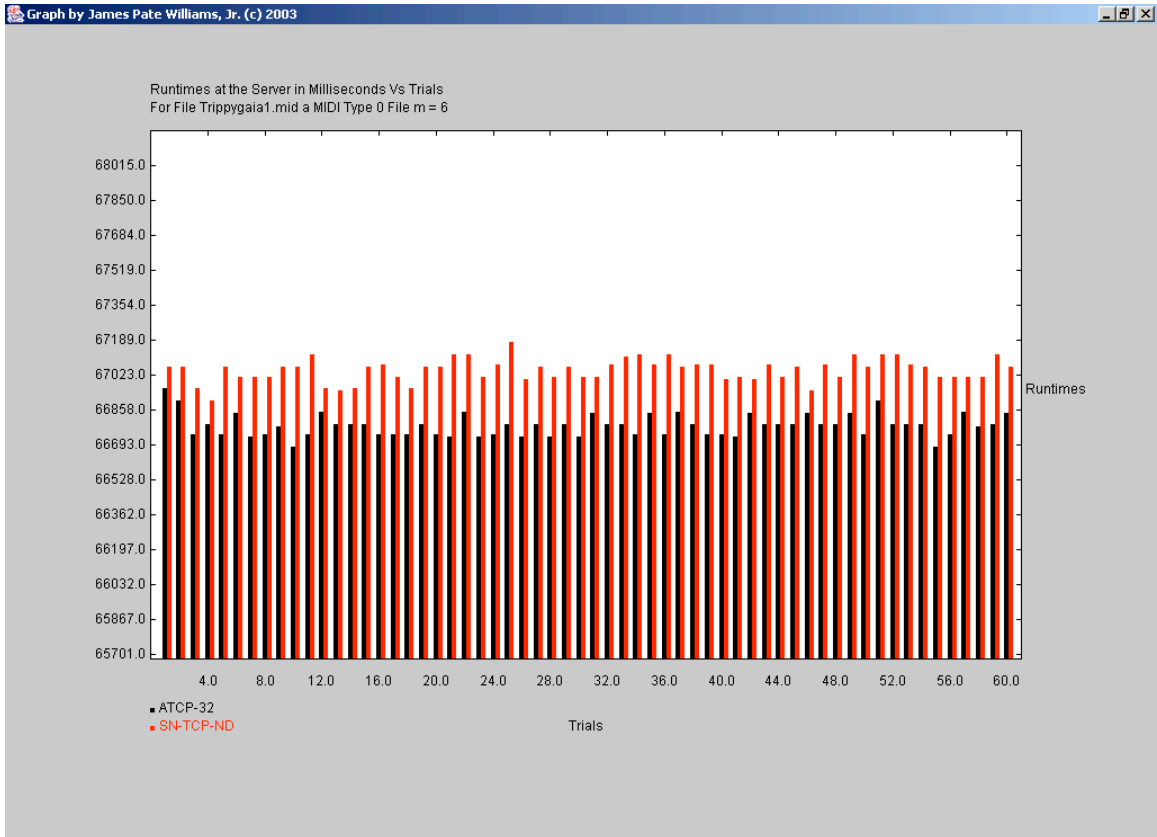
Table K-29 Trippygaial.mid m = 8

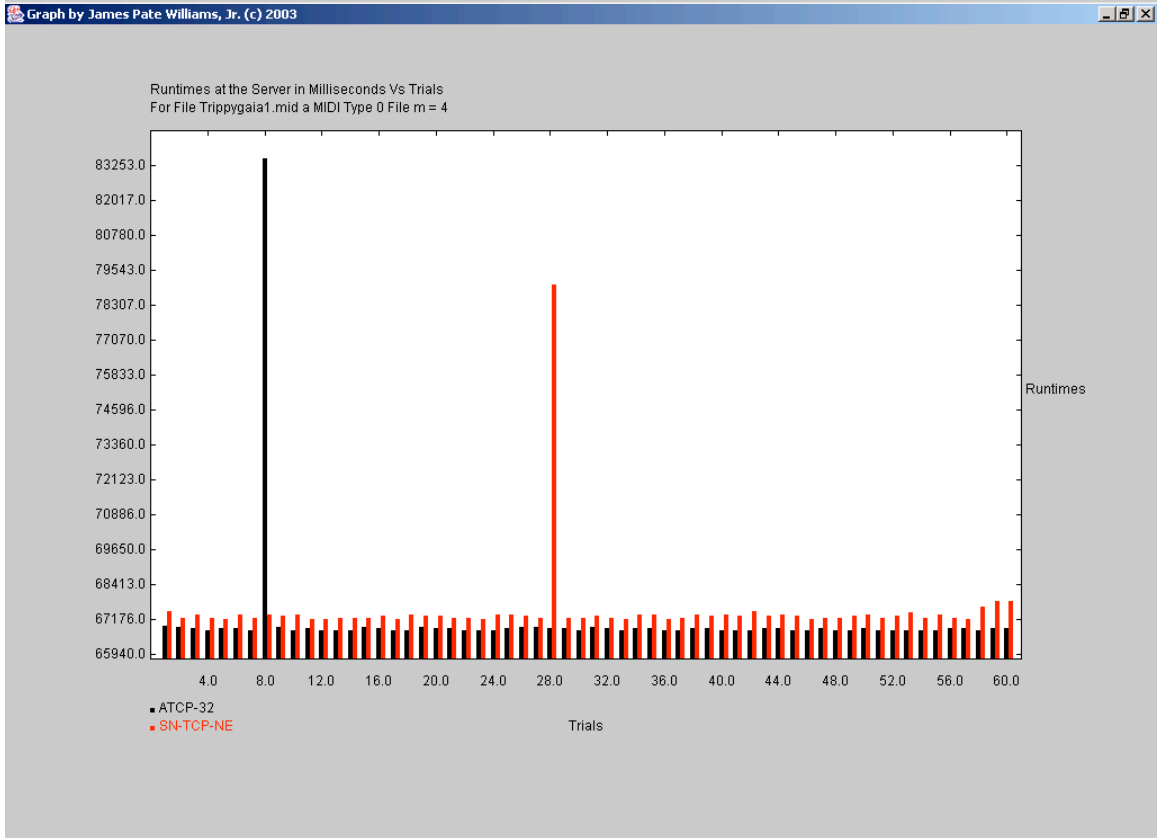
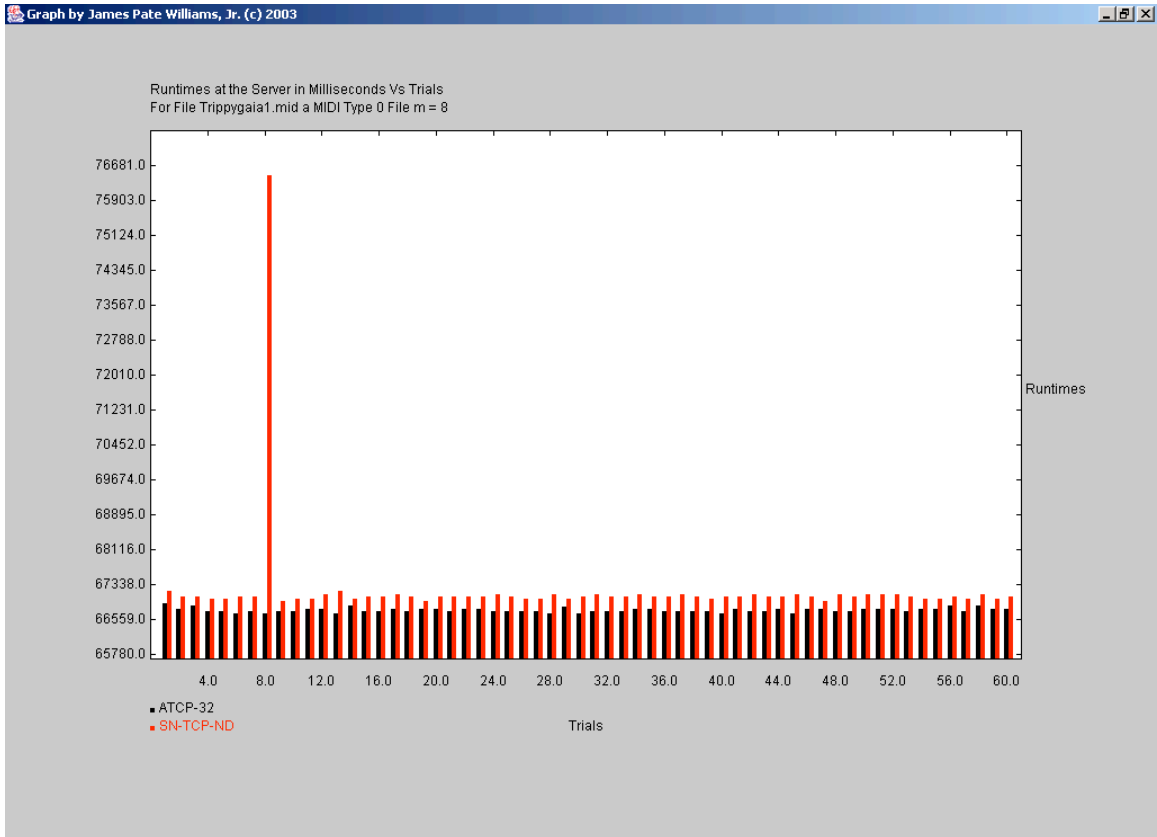
Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
SN-TCP-ND-SN-TCP-NE	-78.8333	1217.7543	157.2114	-0.5014	59	0.6179

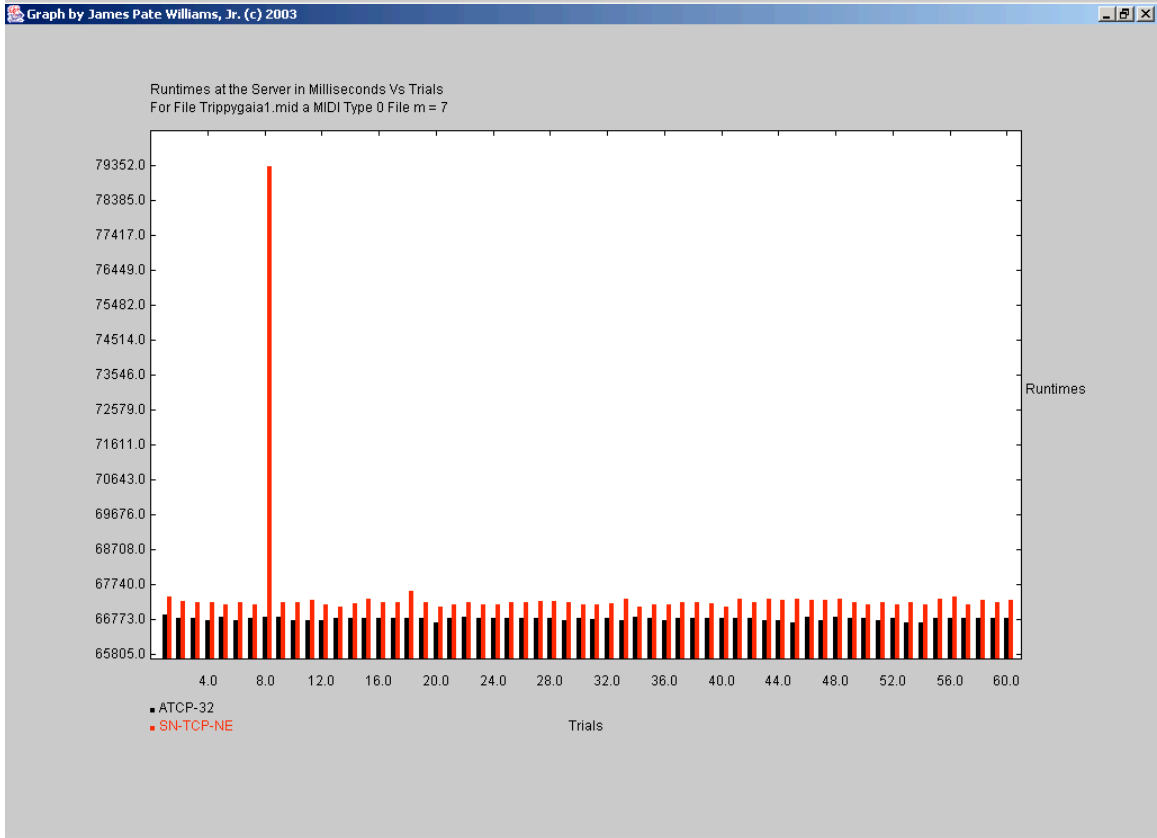
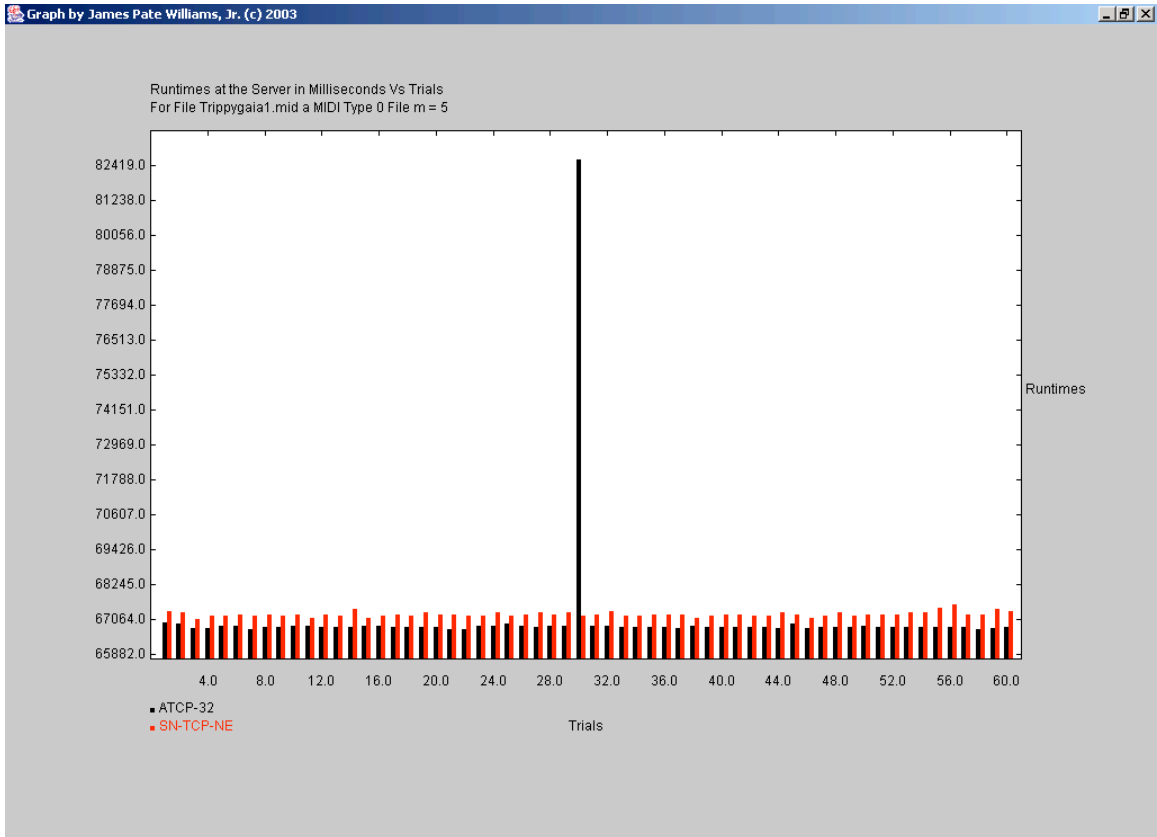
Table K-30 Trippygaial.mid m = 8

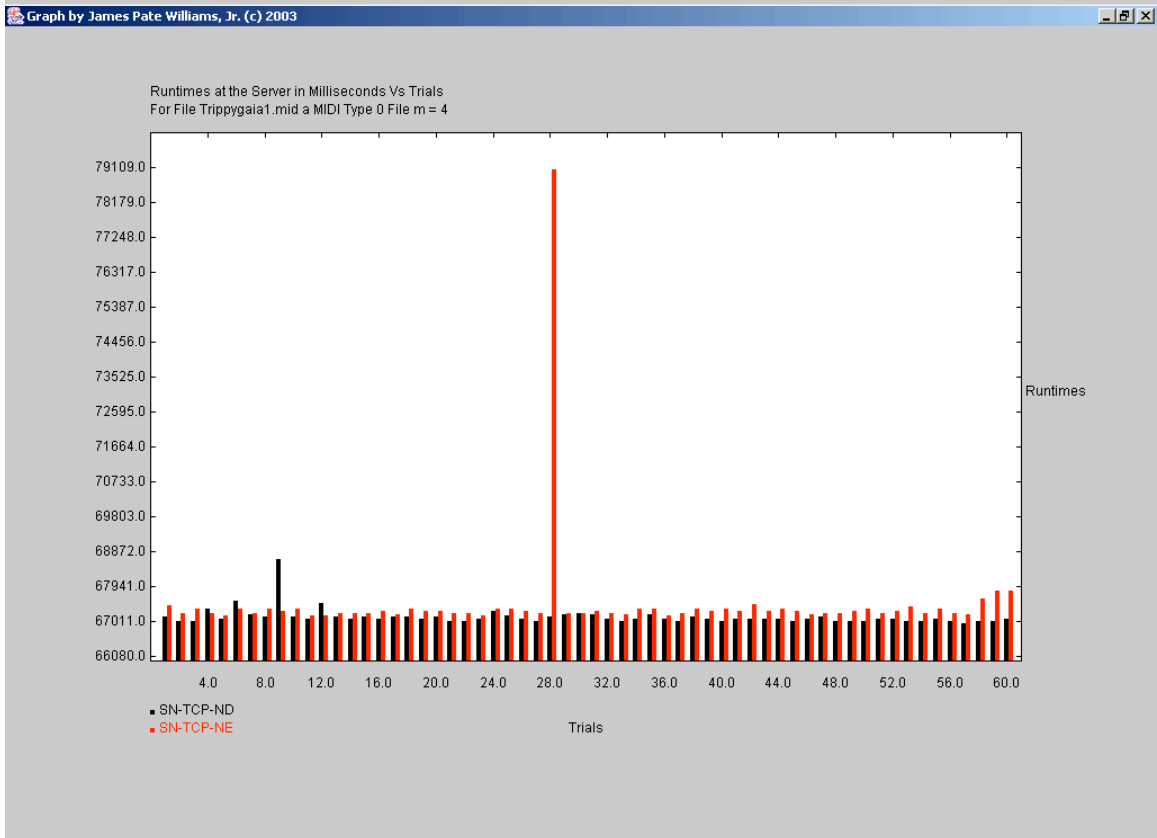
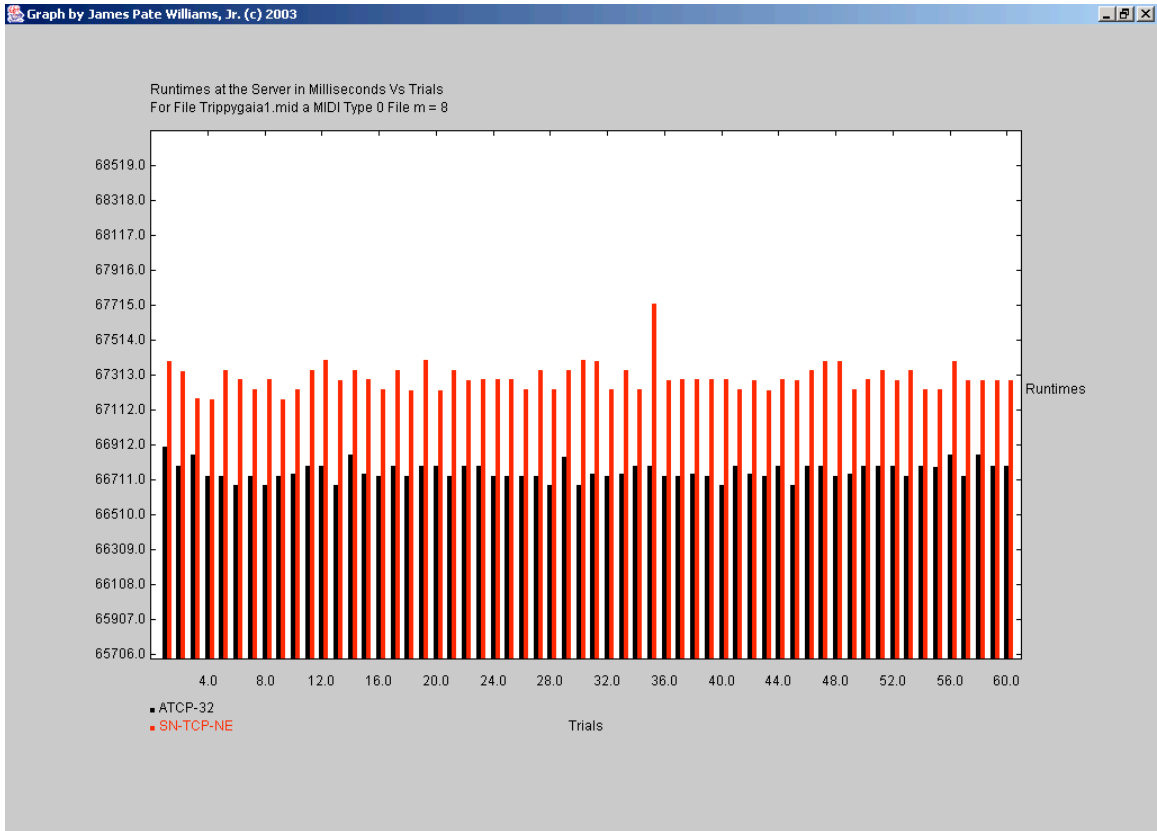


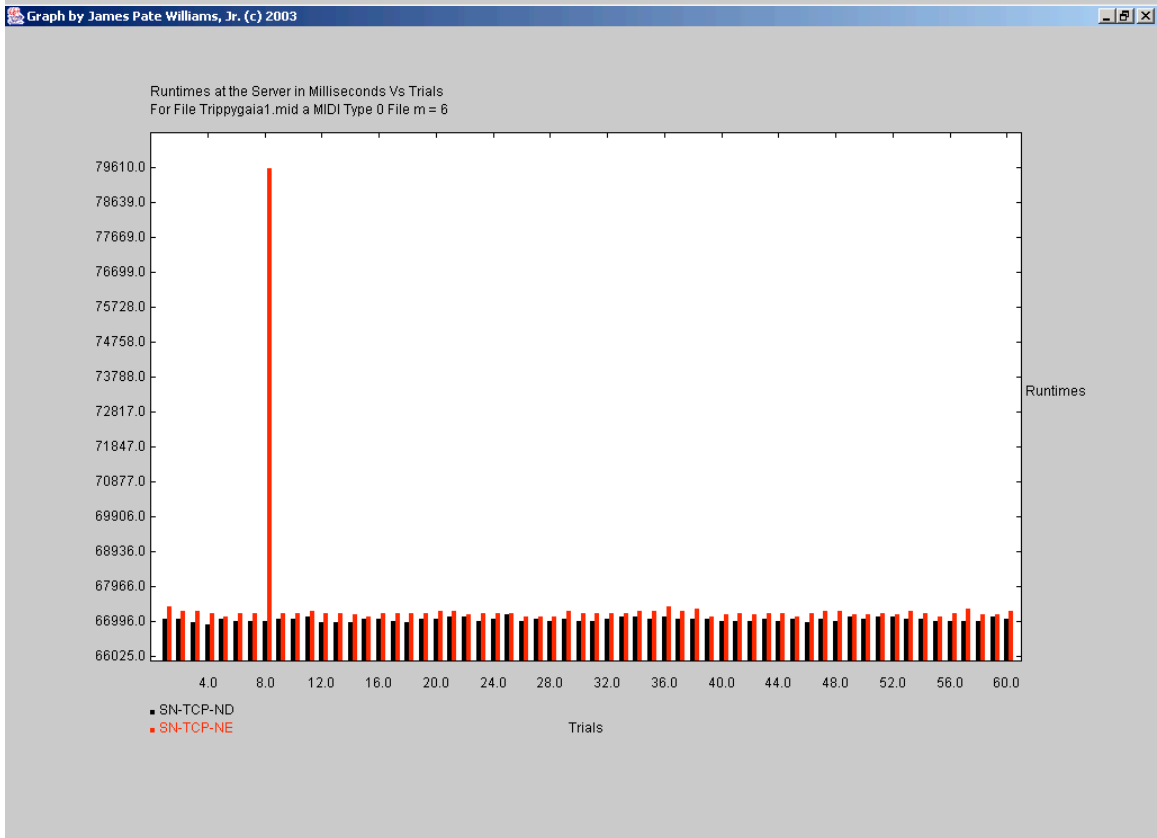
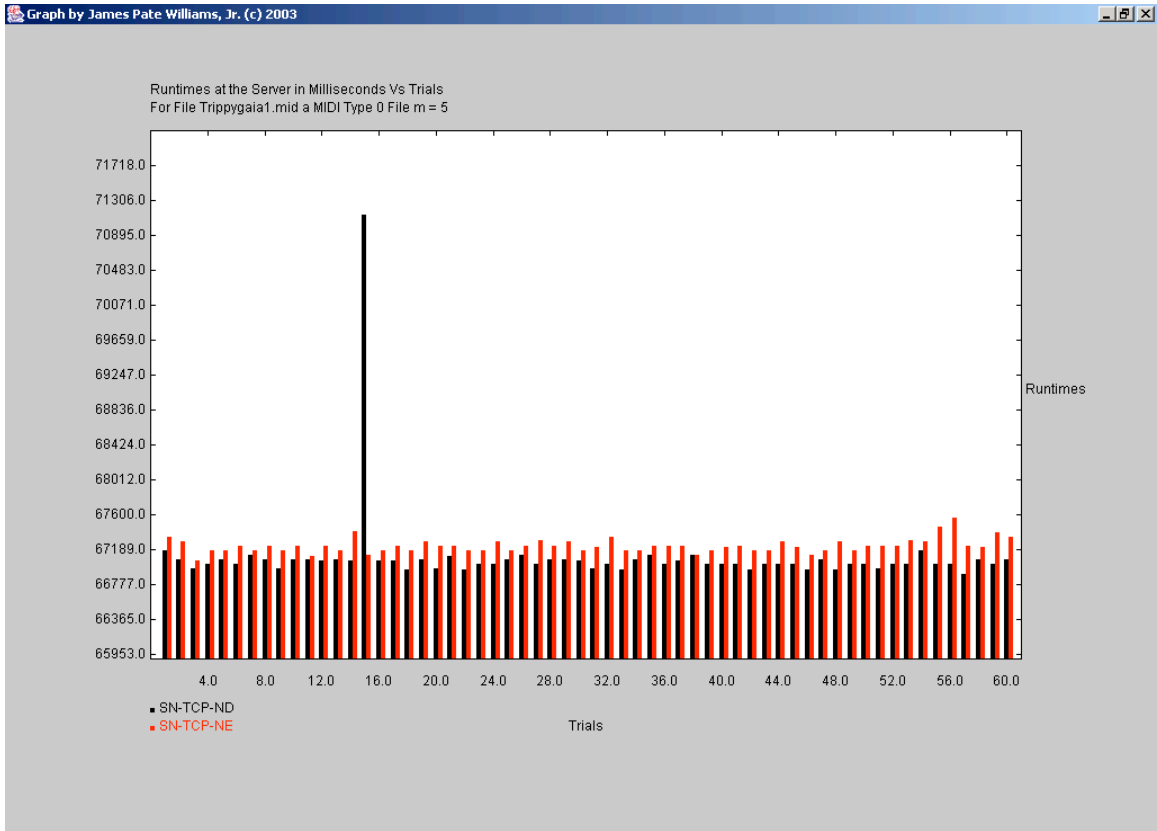


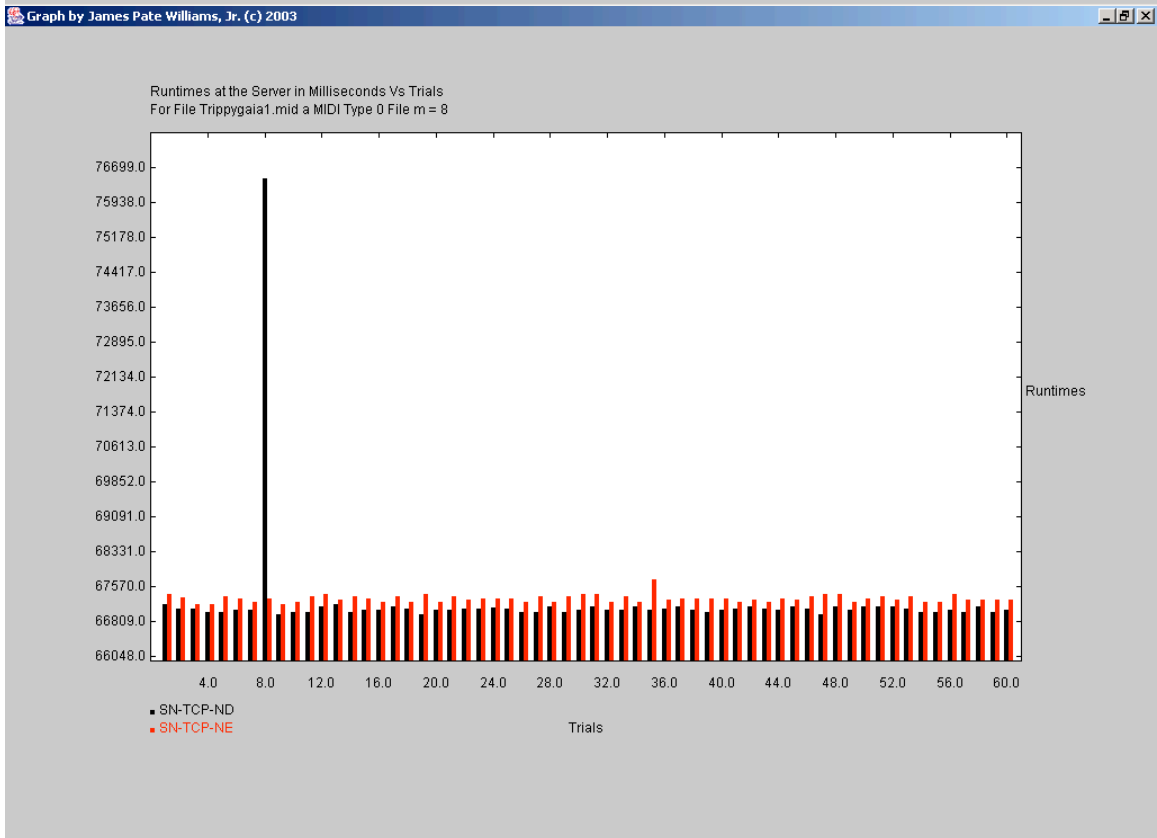
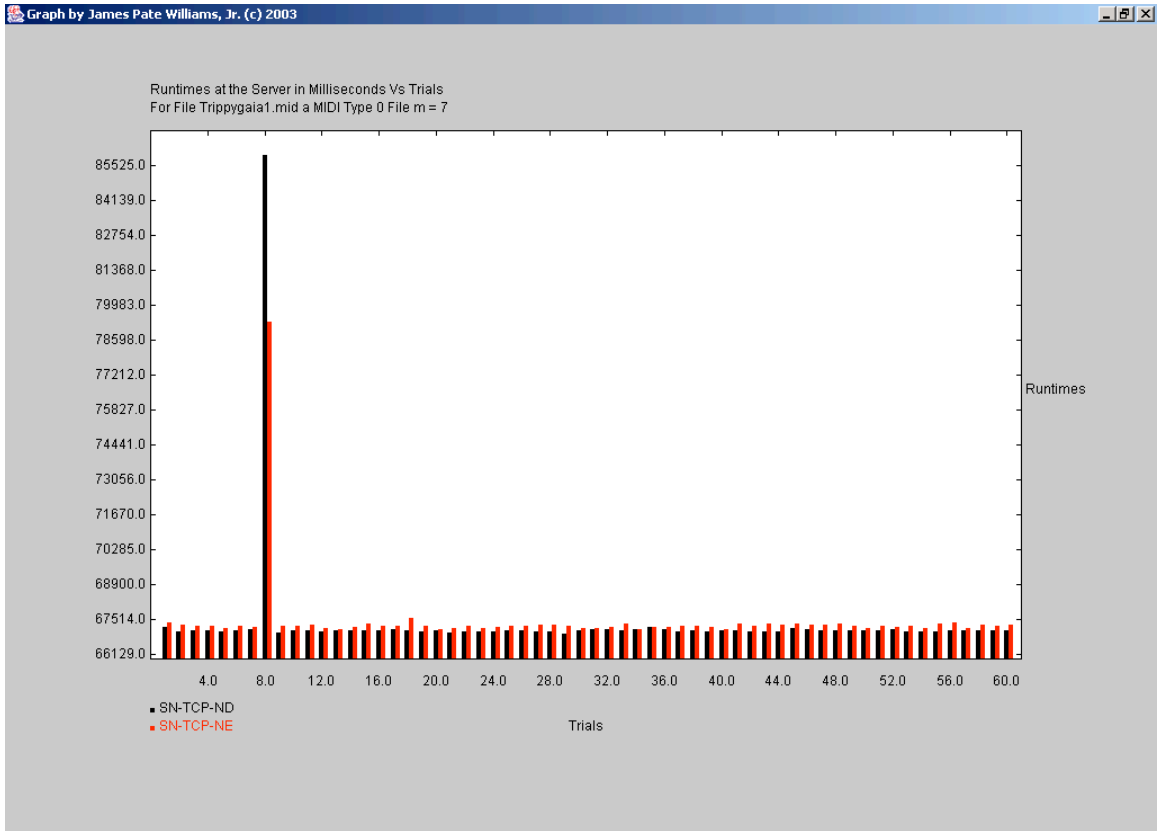












APPENDIX L ATCP-40 VERSUS SN-TCP-ND, SN-TCP-NE, AND ATCP-32

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	67007.83	60	1378.5513	177.9702
SN-TCP-ND	67119.83	60	229.6902	29.6528

Table L-1 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-ND	-112.0000	1397.6515	180.4360	-0.6207	59	0.5371

Table L-2 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	67007.83	60	1378.5513	177.9702
SN-TCP-NE	67490.33	60	1521.7139	196.4524

Table L-3 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-NE	-482.5000	2064.3919	266.5118	-1.8104	59	0.07532

Table L-4 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	67007.83	60	1378.5513	177.9702
ATCP-32	67104.00	60	2151.6688	277.7792

Table L-5 Trippygaia1.mid m = 4

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-ATCP-32	-96.1666	776.0265	100.1846	-0.9598	59	0.3410

Table L-6 Trippygaia1.mid m = 4

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66822.83	60	150.2550	19.3978
SN-TCP-ND	67100.16	60	532.2019	68.7069

Table L-7 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-ND	-277.3333	558.4963	72.1015	-3.8464	59	0.0002

Table L-8 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66822.83	60	150.2550	19.3978
SN-TCP-NE	67230.83	60	82.8985	10.7021

Table L-9 Trippygaia1.mid m = 5



Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-NE	-408.0000	164.1599	21.1929	-19.2516	59	0.0000

Table L-10 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66822.83	60	150.2550	19.3978
ATCP-32	67066.00	60	2039.9494	263.3563

Table L-11 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-ATCP-32	-243.1666	2049.7535	264.6220	-0.9189	59	0.3618

Table L-12 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66992.66	60	1201.5143	155.1148
SN-TCP-ND	67044.83	60	54.3838	7.0209

Table L-13 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-ND	-52.1666	1206.6750	155.7810	-0.3348	59	0.7389

Table L-14 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66992.66	60	1201.5143	155.1148
SN-TCP-NE	67432.83	60	1596.0301	206.0466

Table L-15 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-NE	-440.1666	502.6219	64.8882	-6.7834	59	0.0000

Table L-16 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66992.66	60	1201.5143	155.1148
ATCP-32	66782.83	60	53.5230	6.9097

Table L-17 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-ATCP-32	209.8333	1207.4647	155.8830	1.3460	59	0.1834

Table L-18 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66952.33	60	1286.8044	166.1257
SN-TCP-ND	67371.16	60	2434.3969	314.2792

Table L-19 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-ND	-418.8333	1153.7104	148.9433	-2.8120	59	0.0066

Table L-20 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66952.33	60	1286.8044	166.1257
SN-TCP-NE	67442.50	60	1561.3256	201.5662

Table L-21 Trippygaial .mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-NE	-490.1666	297.0447	38.3483	-12.7819	59	0.0000

Table L-22 Trippygaial .mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66952.33	60	1286.8044	166.1257
ATCP-32	66778.66	60	43.4708	5.6120

Table L-23 Trippygaial .mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-ATCP-32	173.6666	1279.3191	165.1593	1.0515	59	0.2973

Table L-24 Trippygaial .mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66886.16	60	991.5765	128.0119
SN-TCP-ND	67220.00	60	1214.1189	156.7420

Table L-25 Trippygaial .mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-ND	-333.8333	232.7222	30.0443	-11.1113	59	0.0000

Table L-26 Trippygaial .mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66886.16	60	991.5765	128.0119
SN-TCP-NE	67298.83	60	81.4922	10.5206

Table L-27 Trippygaial .mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-SN-TCP-NE	-412.6666	997.0173	128.7143	-3.2060	59	0.0021

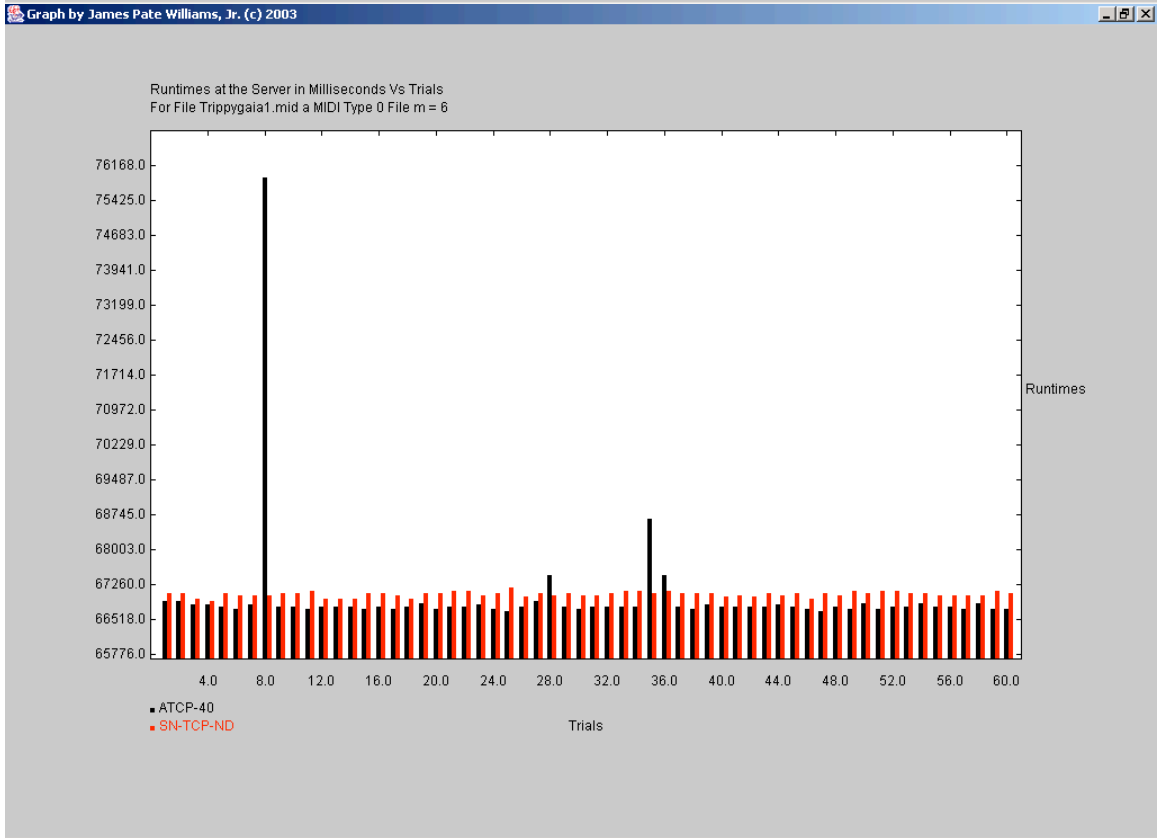
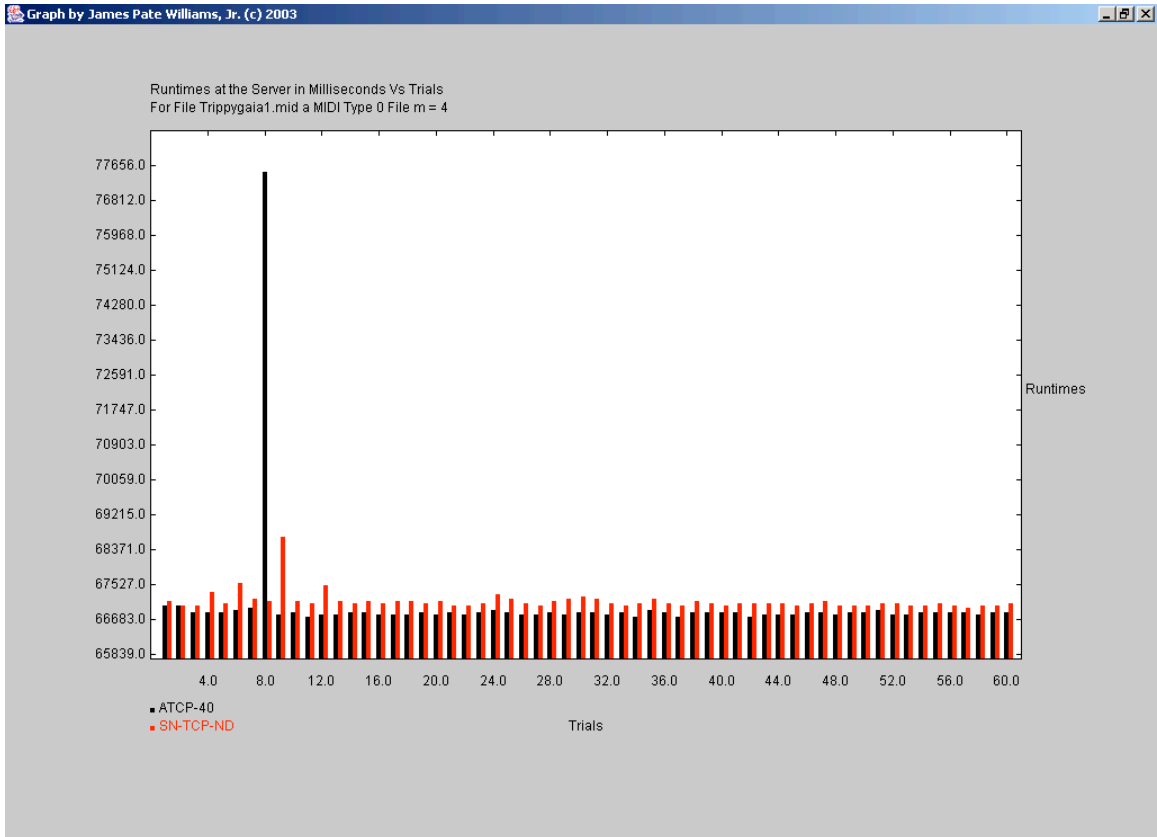
Table L-28 Trippygaial .mid m = 8

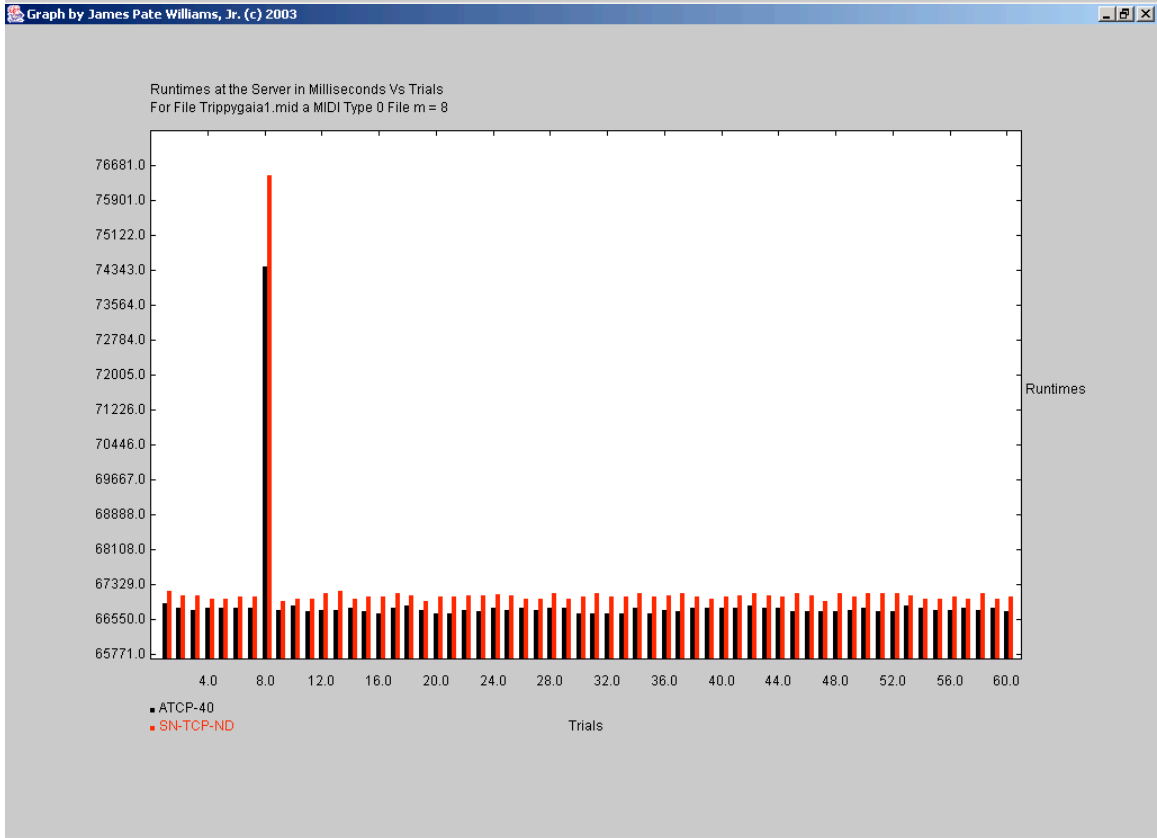
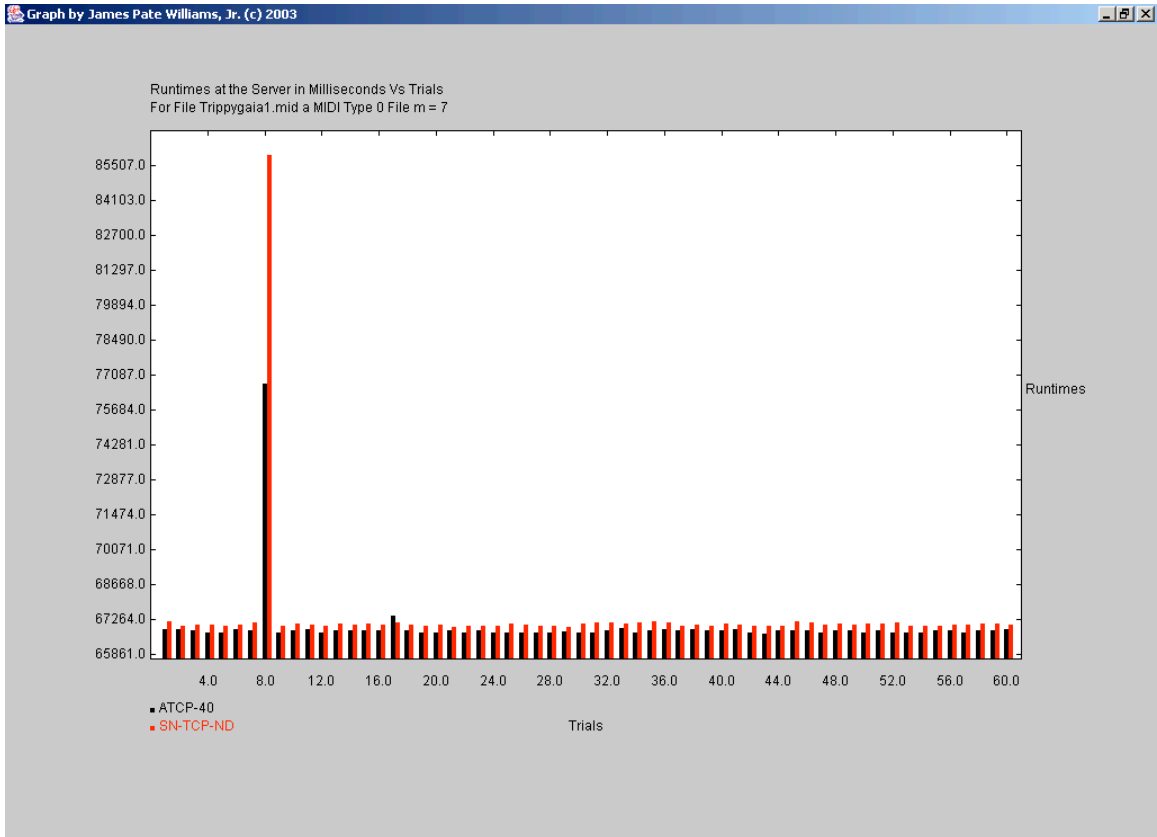
Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-40	66886.16	60	991.5765	128.0119
ATCP-32	66758.83	60	48.9583	6.3204

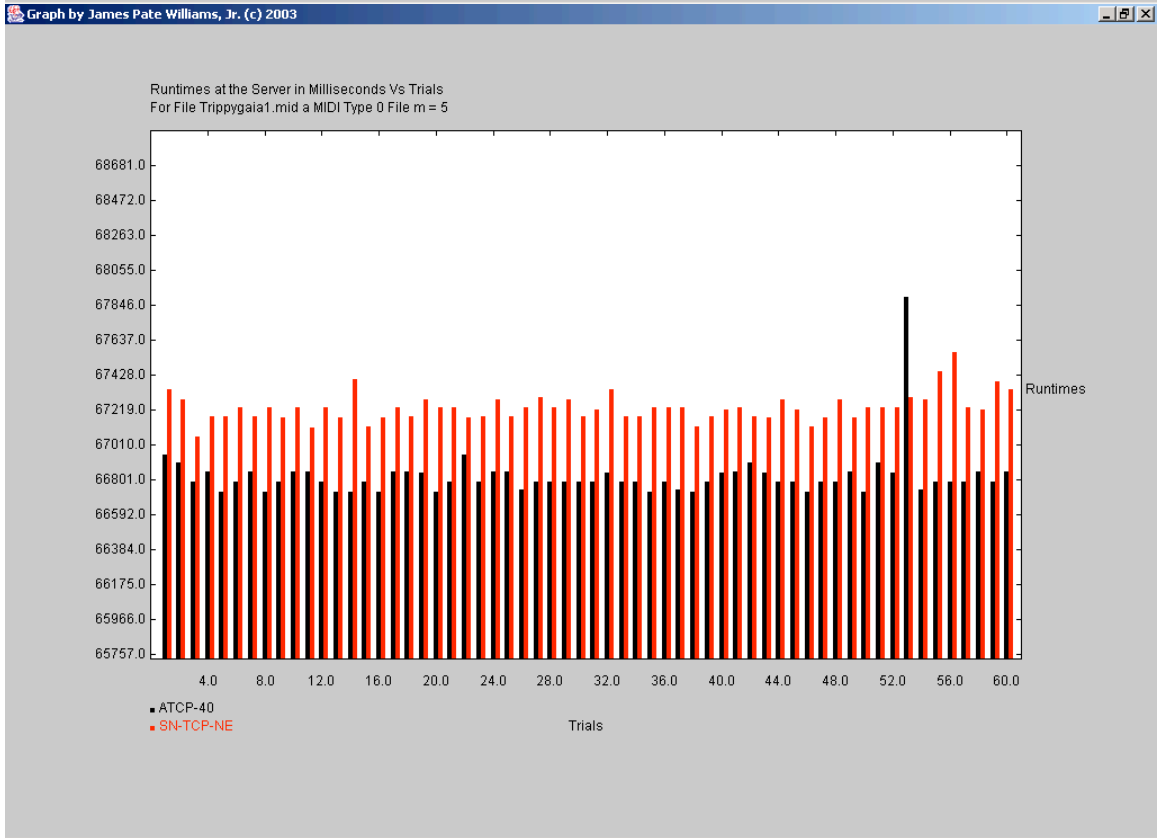
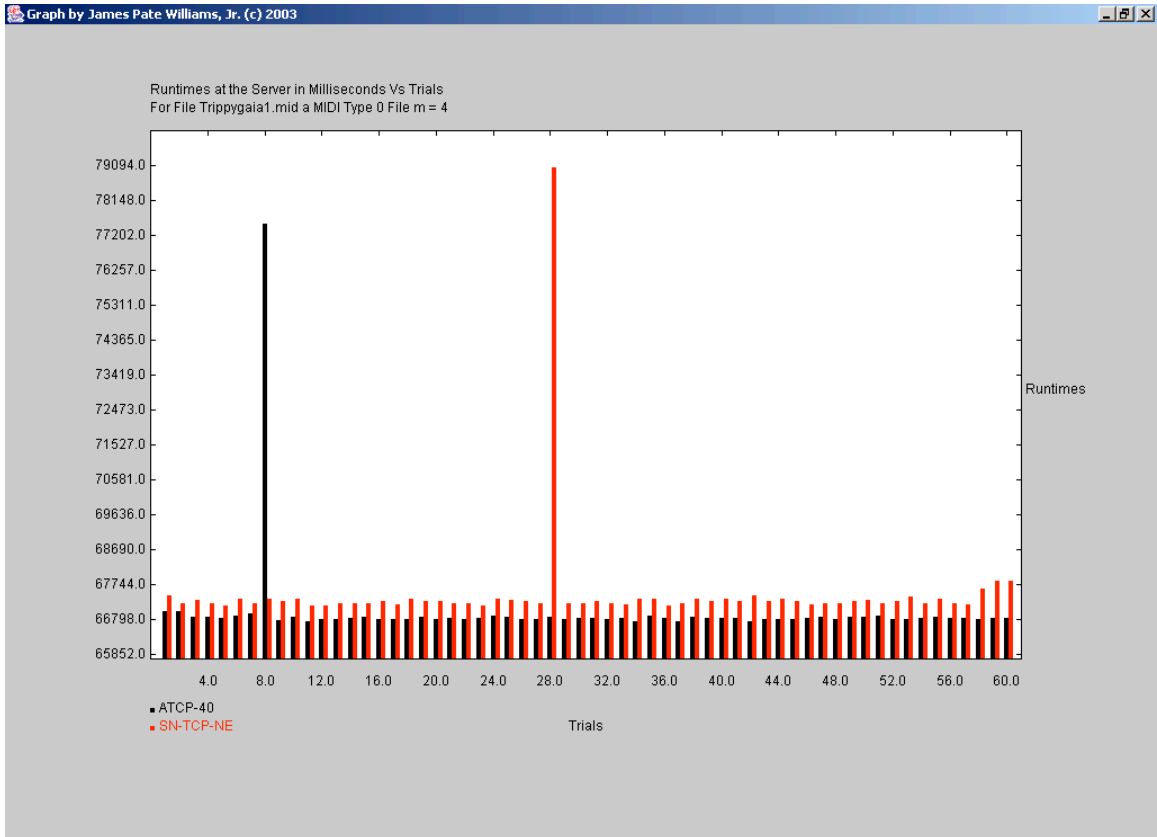
Table L-29 Trippygaial .mid m = 8

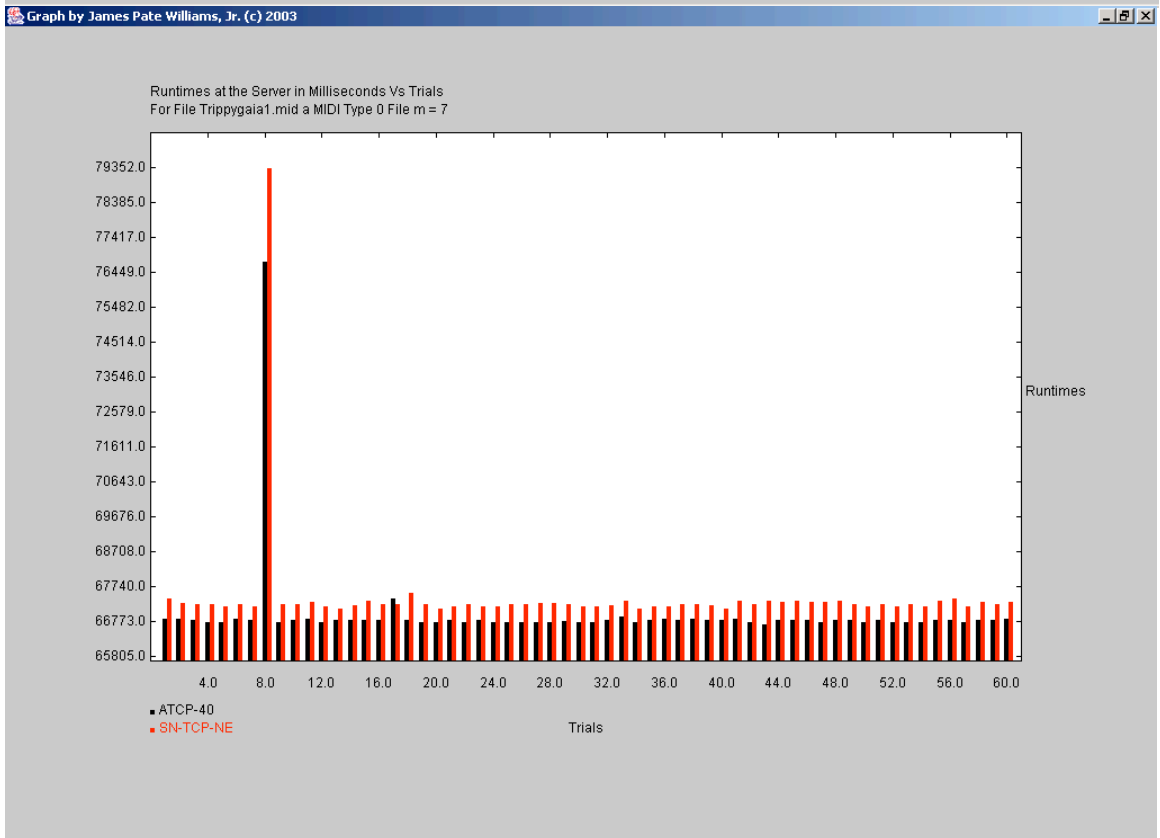
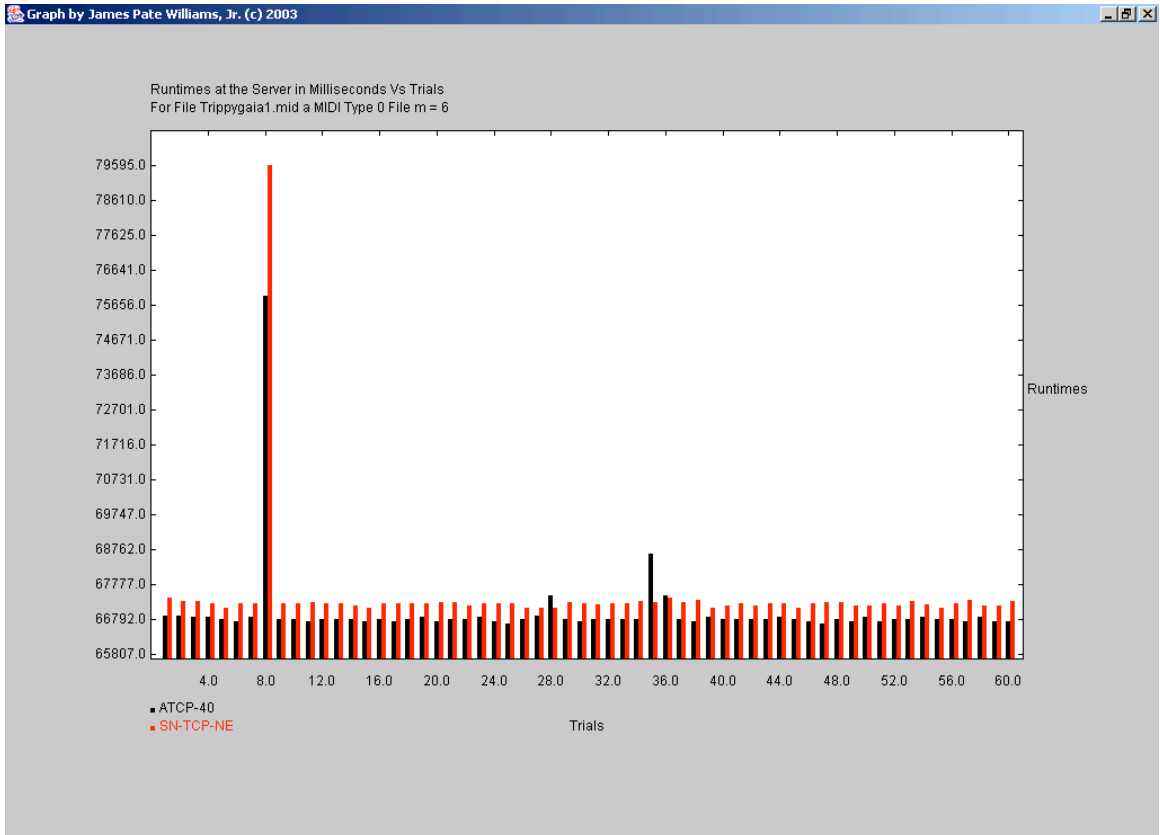
Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP40-ATCP-32	127.3333	1002.7840	129.4588	0.9835	59	0.3293

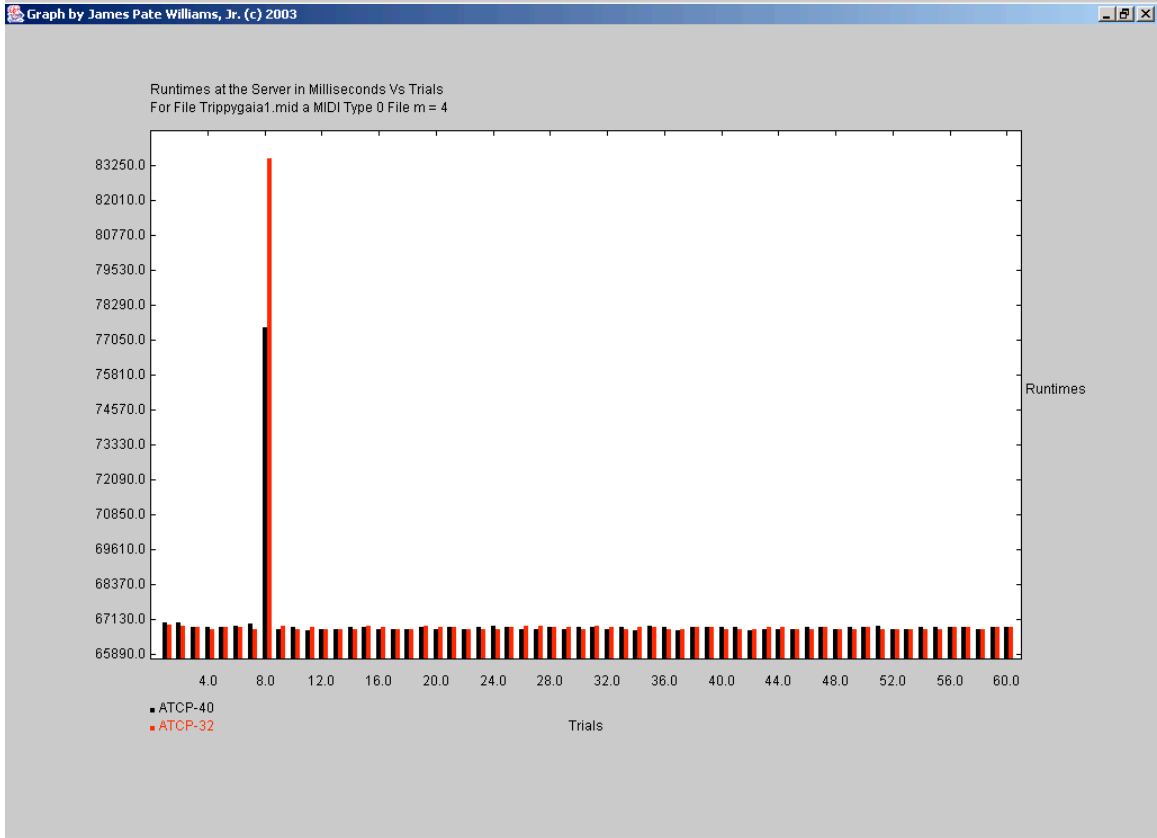
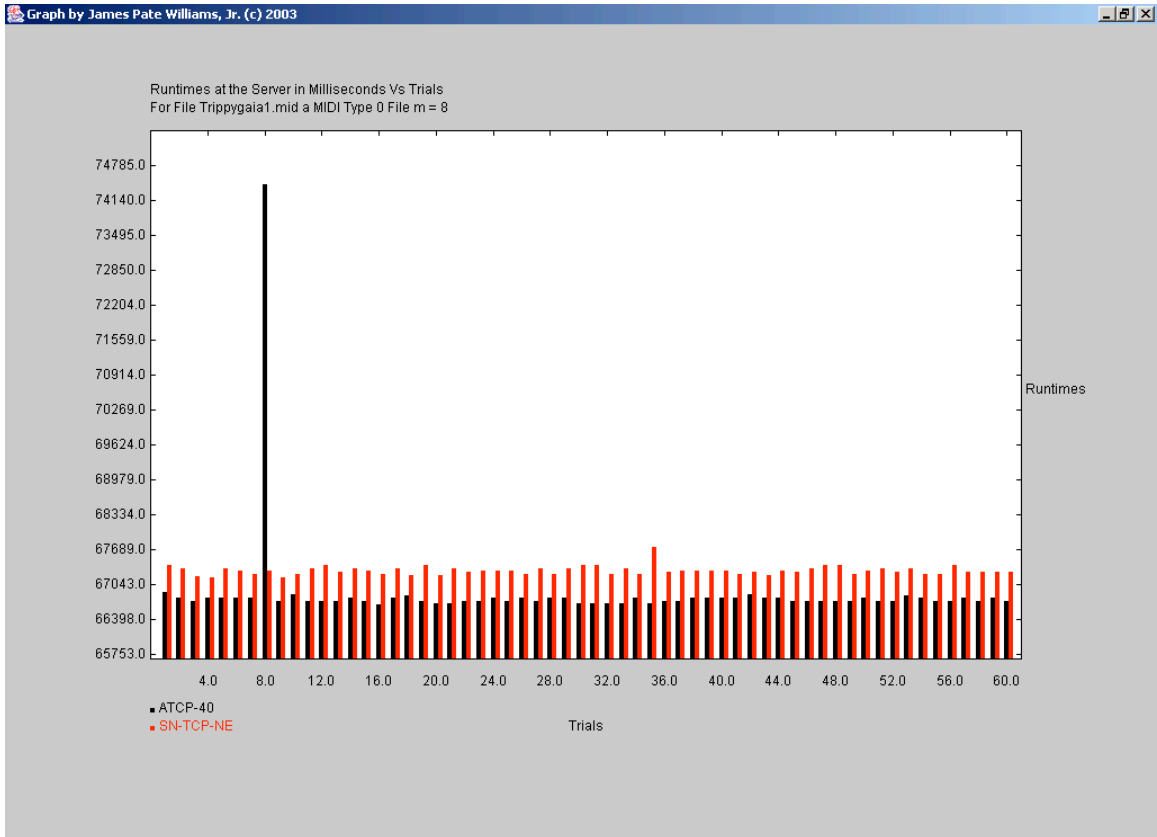
Table L-30 Trippygaial .mid m = 8

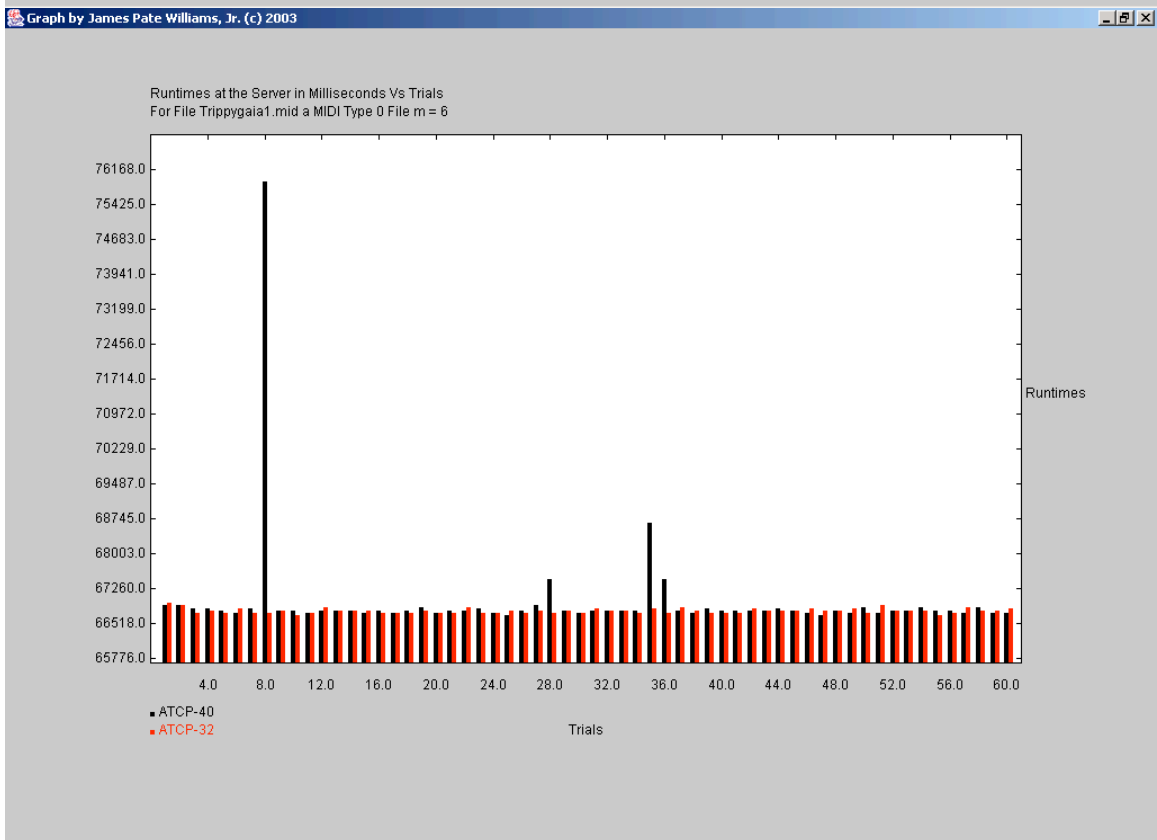
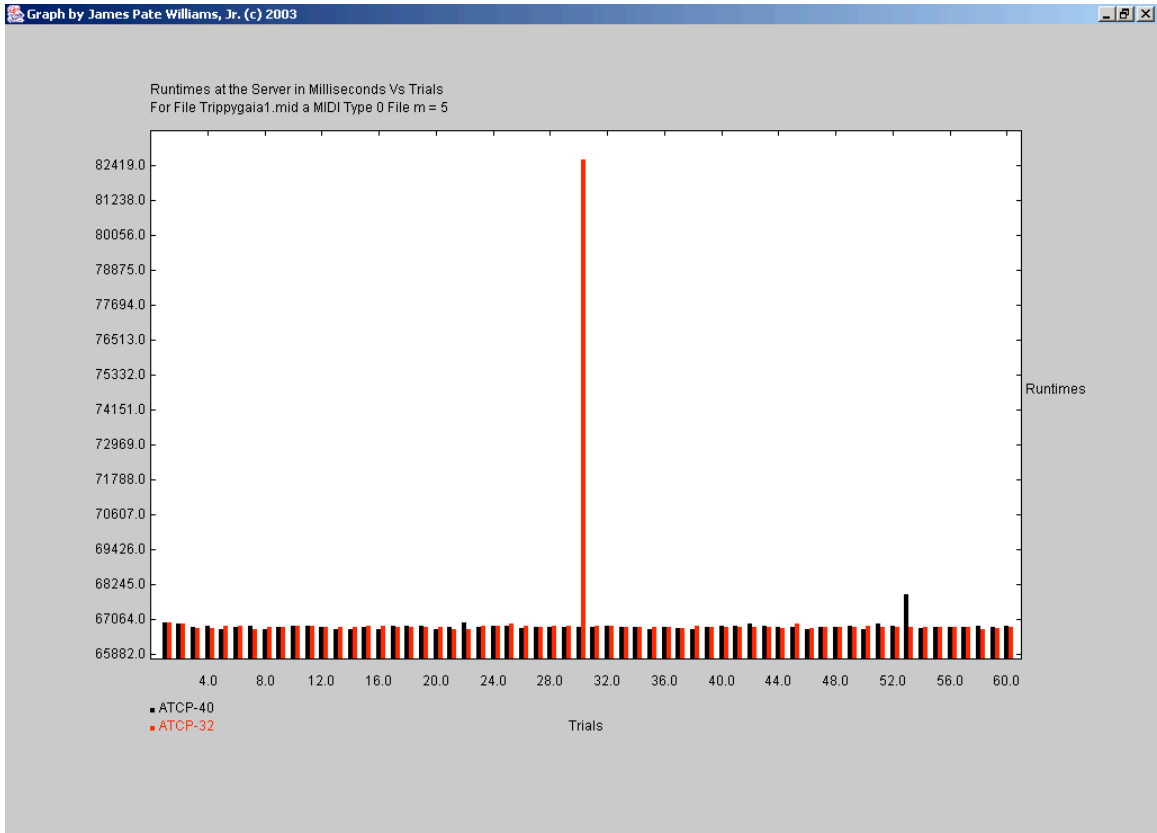




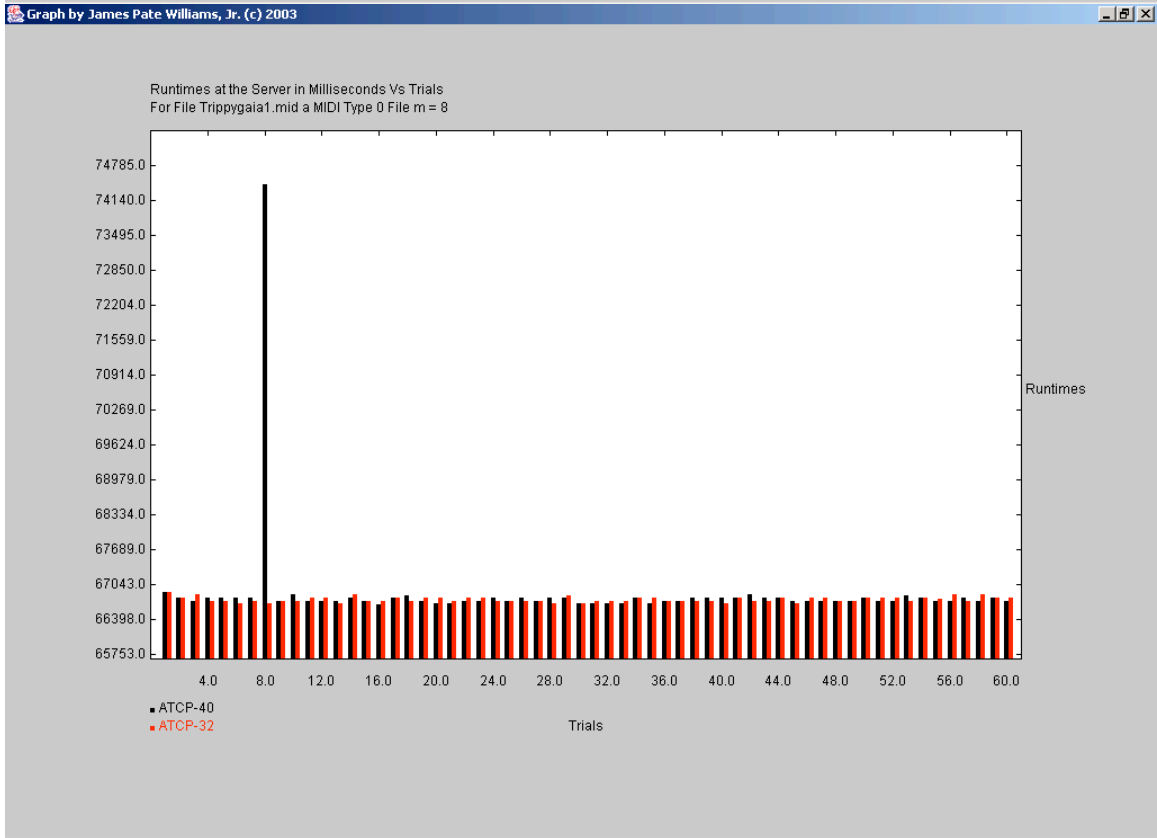
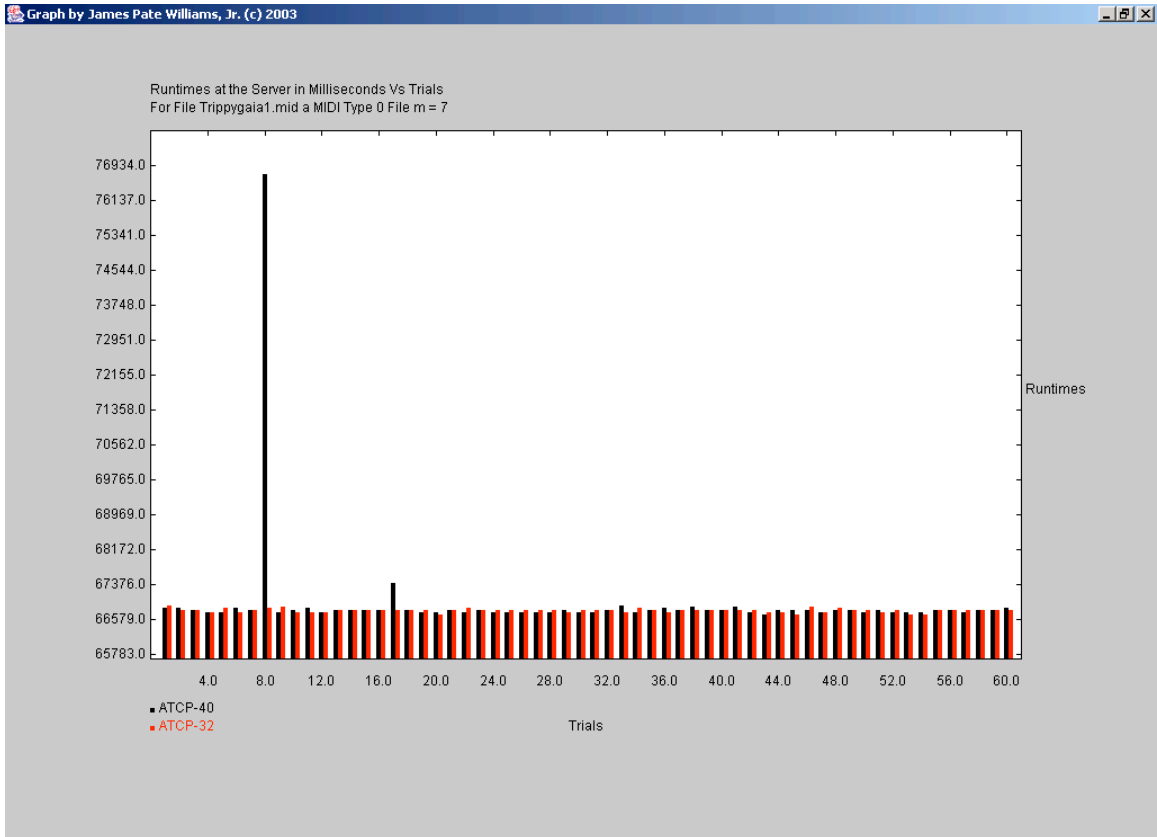












APPENDIX M ATCP-TCP-ND VS SN-TCP-NX AND ATCP-X

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66816.00	60	52.5679	6.7864
SN-TCP-ND	67100.16	60	532.2019	68.7069

Table M-1 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-ND	-284.1666	536.3918	69.2478	-4.1036	59	0.0001

Table M-2 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66791.50	60	60.4734	7.8070
SN-TCP-ND	67044.83	60	54.3838	7.0209

Table M-3 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-ND	-253.3333	81.1290	10.4737	-24.1875	59	0.0000

Table M-4 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66280.33	60	1849.4488	238.7628
SN-TCP-ND	67371.16	60	2434.3969	314.2792

Table M-5 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-ND-SN-TCP-ND	-1090.8333	590.2980	76.2071	-14.3140	59	0.0000

Table M-6 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66062.50	60	55.4068	7.1529
SN-TCP-ND	67220.00	60	1214.1189	156.7420

Table M-7 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-NE	-1157.50	1213.3531	156.6432	-7.3894	59	0.0000

Table M-8 Trippygaial.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66816.00	60	52.5679	6.7864
SN-TCP-NE	67230.83	60	82.8985	10.7021

Table M-9 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-NE	-414.8333	102.6181	13.2479	-31.3130	59	0.0000

Table M-10 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66791.50	60	60.4734	7.8070
SN-TCP-NE	67432.83	60	1596.0301	206.0466

Table M-11 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-NE	-641.3333	1597.1589	206.1923	-3.1103	59	0.0028

Table M-12 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66280.33	60	1849.4488	238.7628
SN-TCP-NE	67442.50	60	1561.3256	201.5662

Table M-13 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-ND-SN-TCP-NE	-1162.1666	306.8047	39.6083	-29.3414	59	0.0000

Table M-14 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66062.50	60	55.4068	7.1529
SN-TCP-NE	67298.83	60	81.4922	10.5206

Table M-15 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-SN-TCP-NE	-1236.3333	92.5709	11.9508	-103.4514	59	0.0000

Table M-16 Trippygaial.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66816.00	60	52.5679	6.7864
ATCP-32	67066.00	60	2039.9494	263.3563

Table M-17 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-32	-250.0000	2029.5336	262.0116	-0.9541	59	0.3438

Table M-18 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66791.50	60	60.4734	7.8070
ATCP-32	66782.83	60	53.5230	6.9097

Table M-19 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-32	8.6666	72.5671	9.3683	0.9250	59	0.3586

Table M-20 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66280.33	60	1849.4488	238.7628
ATCP-32	66778.66	60	43.4708	5.6120

Table M-21 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-32	-498.3333	1841.9915	237.8000	-2.0955	59	0.0404

Table M-22 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66062.50	60	55.4068	7.1529
ATCP-32	66758.83	60	48.9583	6.3204

Table M-23 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-32	-696.3333	79.1472	10.2178	-68.1485	59	0.0000

Table M-24 Trippygaial.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66816.00	60	52.5679	6.7864
ATCP-40	66822.83	60	150.2550	19.3978

Table M-25 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-ND-ATCP-40	-6.8333	153.9919	19.8802	-0.3437	59	0.7322

Table M-26 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66791.50	60	60.4734	7.8070
ATCP-40	66992.66	60	1201.5143	155.1148

Table M-27 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-40	-201.1666	1202.3622	155.2242	-1.2959	59	0.2000

Table M-28 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66280.33	60	1849.4488	238.7628
ATCP-40	66952.33	60	1286.8044	166.1257

Table M-29 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCP-TCP-ND-ATCP-40	-672.00	575.4933	74.2958	-9.0449	59	0.0000

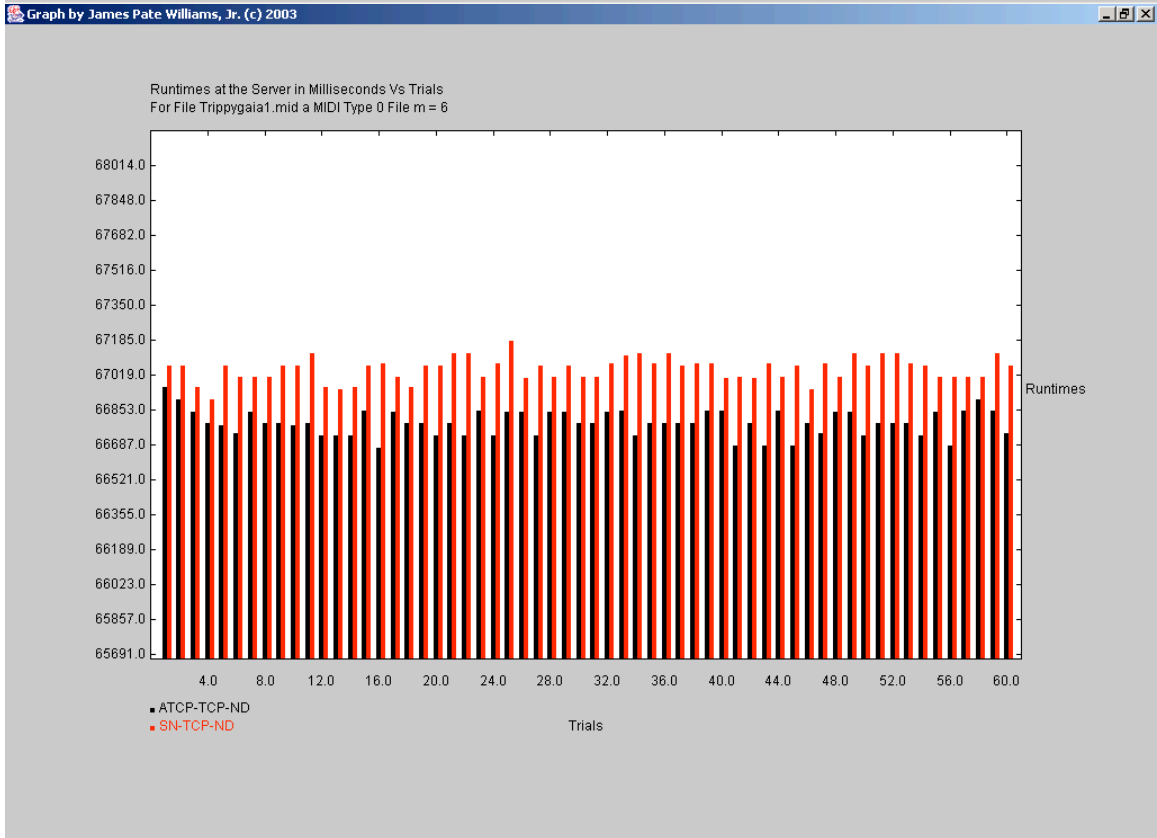
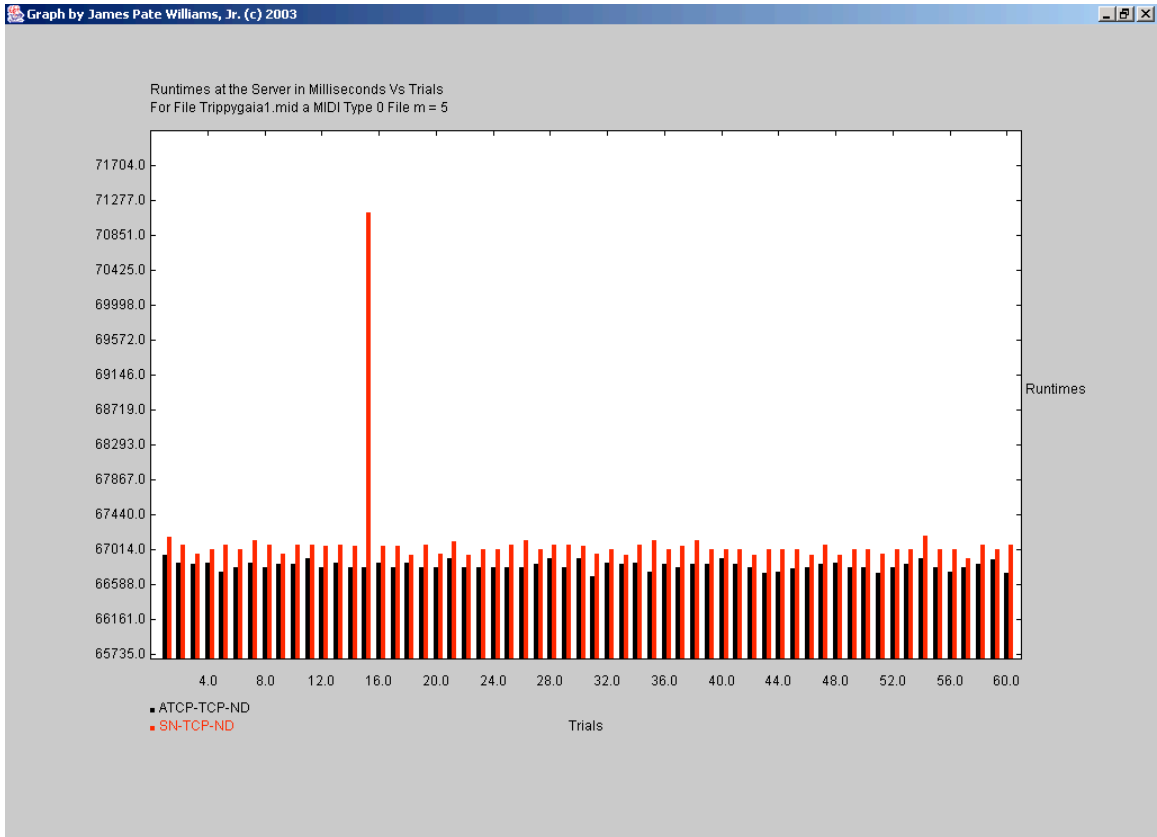
Table M-30 Trippygaial.mid m = 7

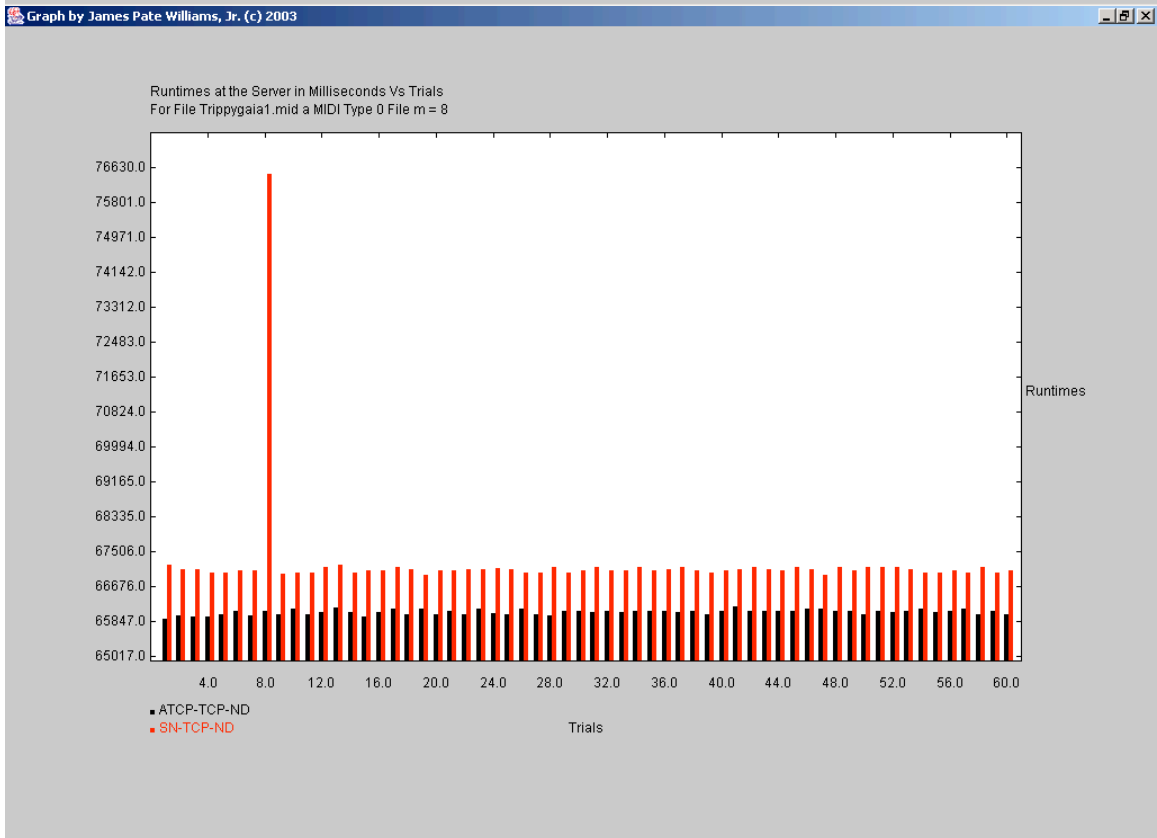
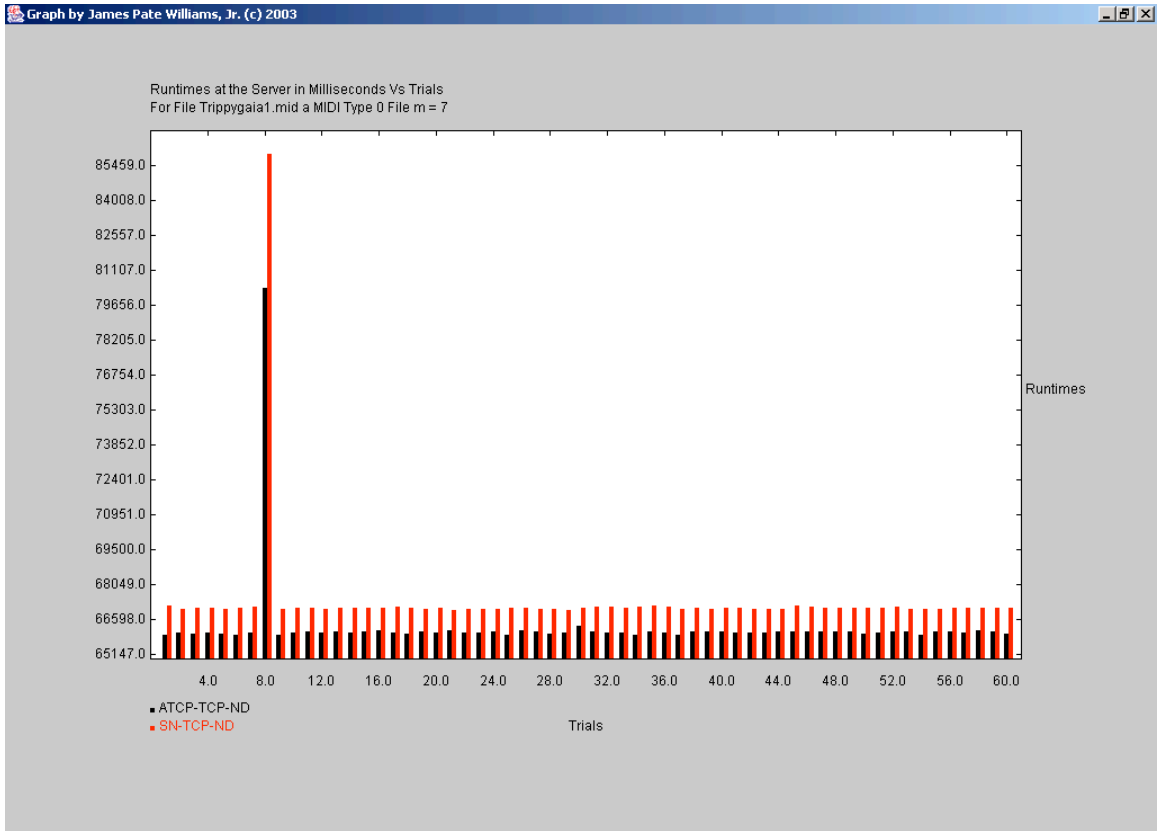
Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-ND	66062.50	60	55.4068	7.1529
ATCP-40	66886.16	60	128.0119	-823.6666

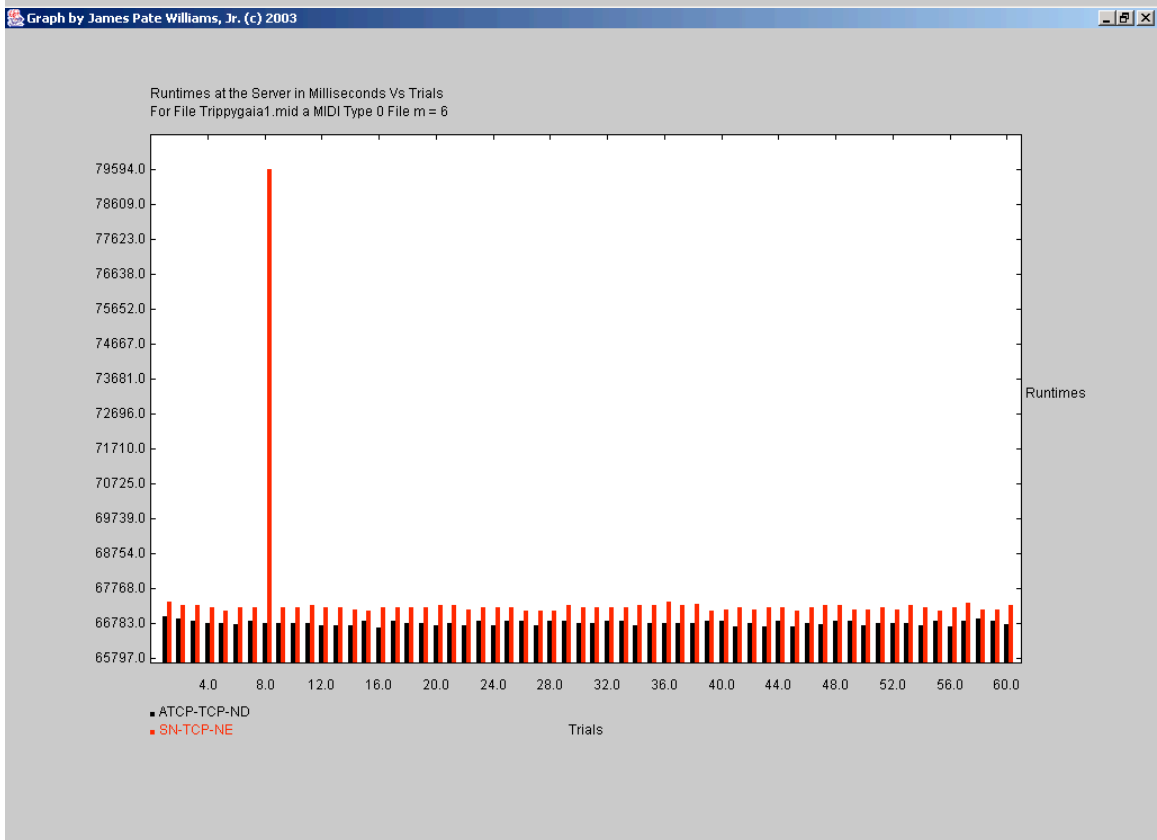
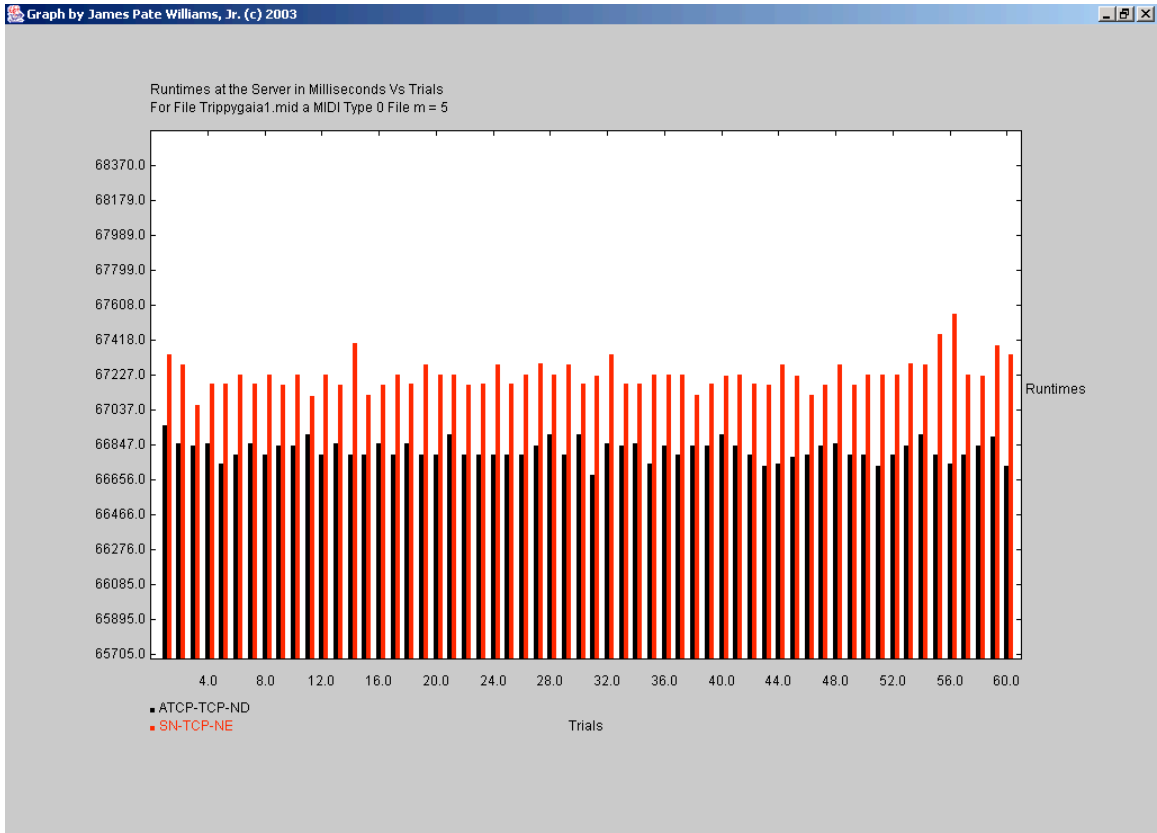
Table M-31 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-ND-ATCP-40	-823.6666	991.1865	127.9616	-6.4368	59	0.0000

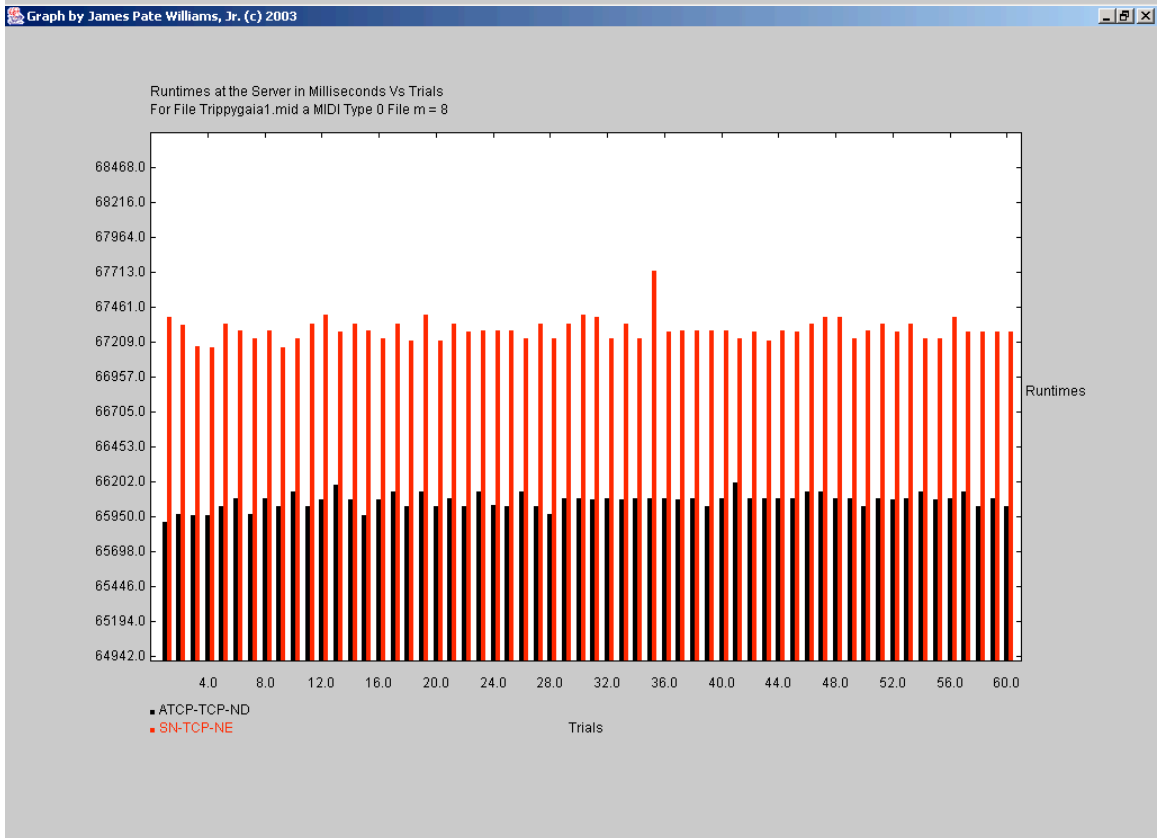
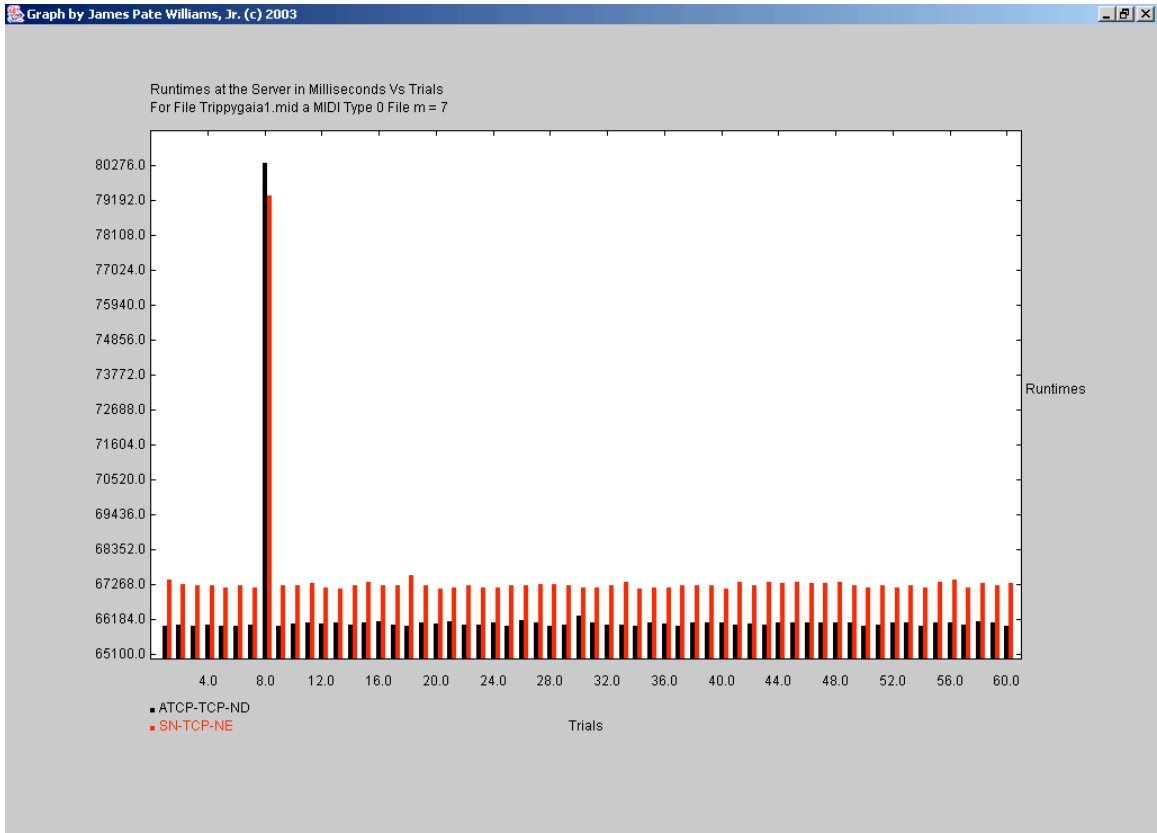
Table M-32 Trippygaial.mid m = 8

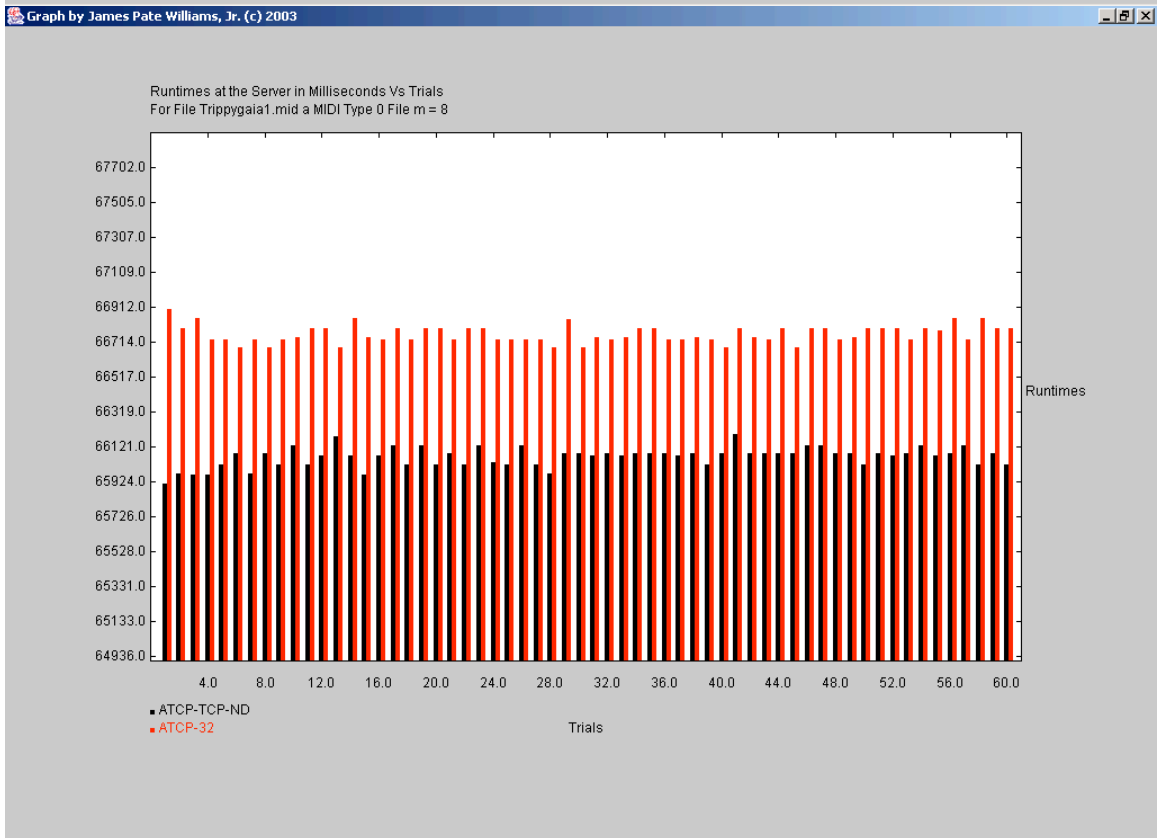
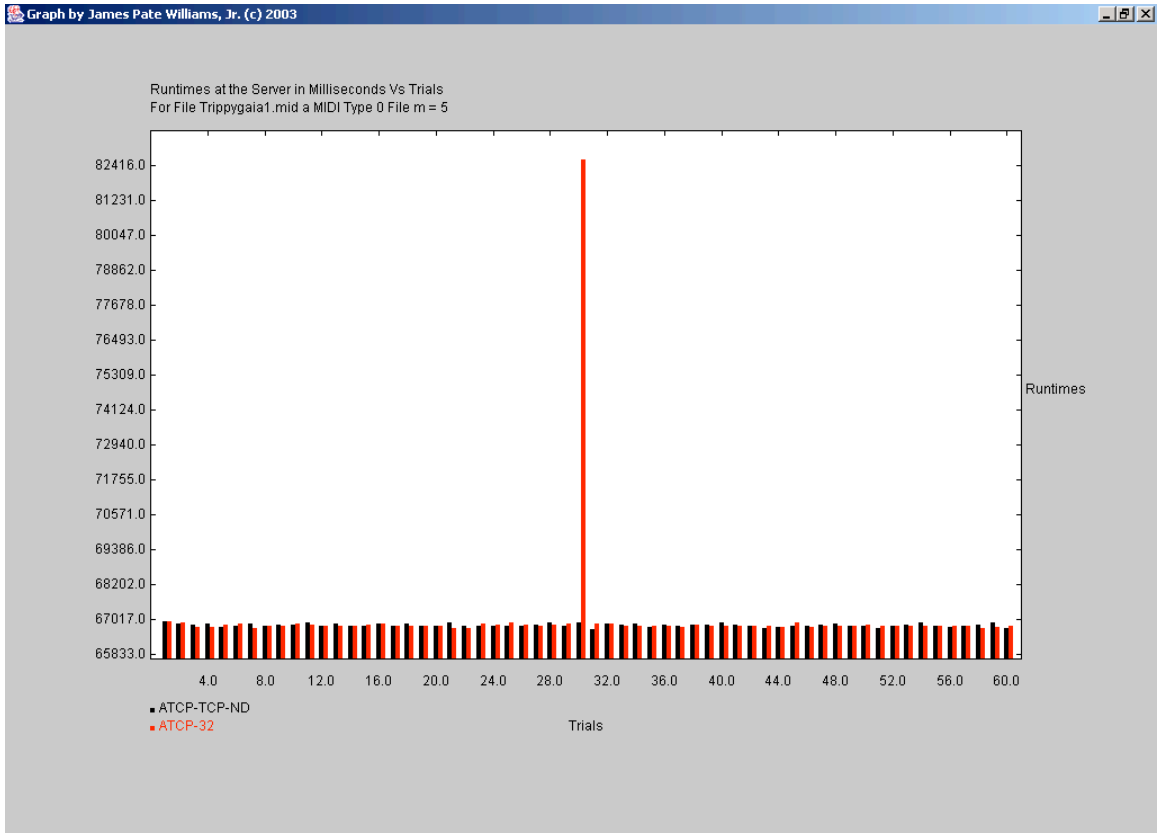


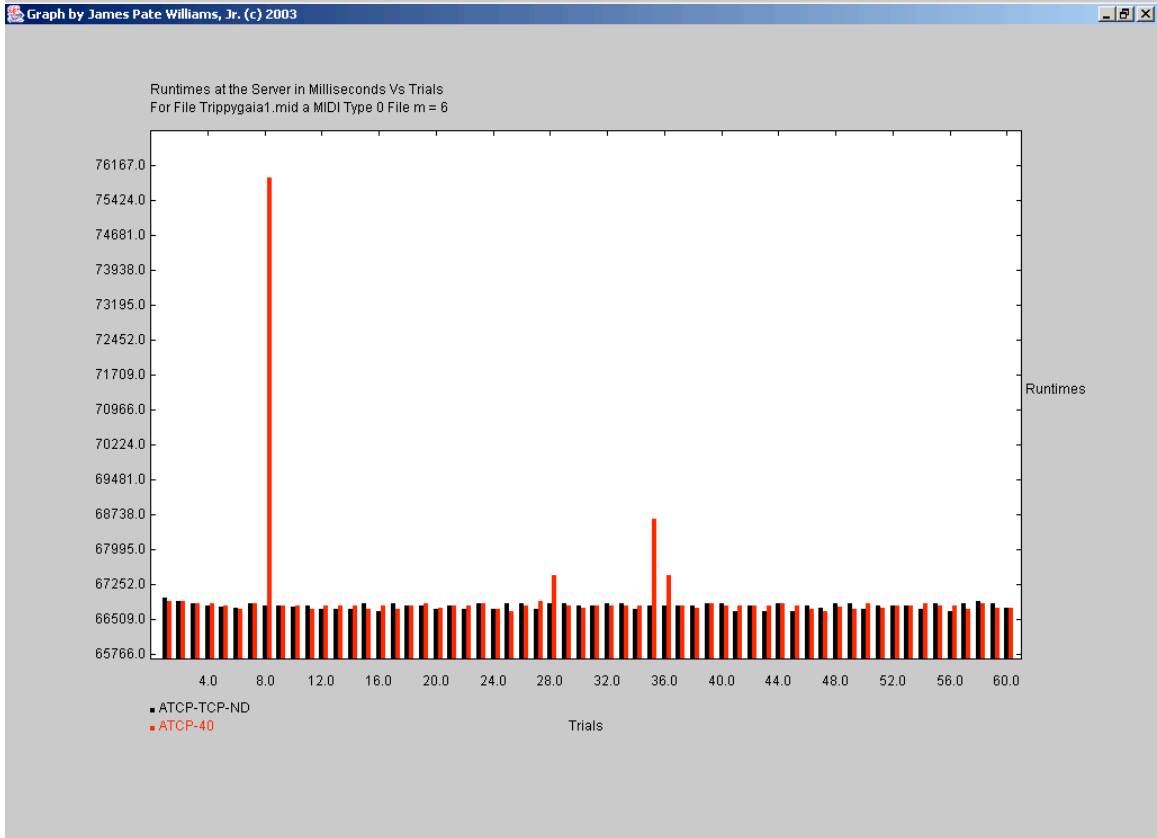
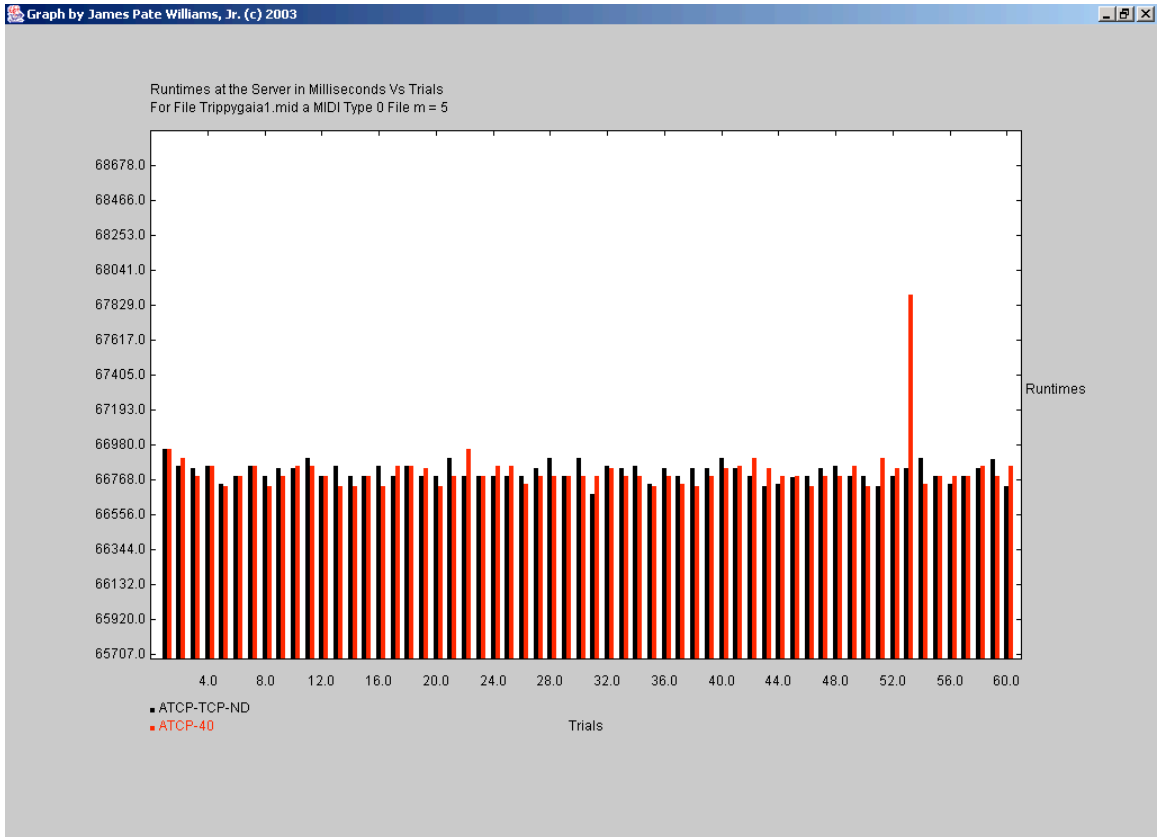


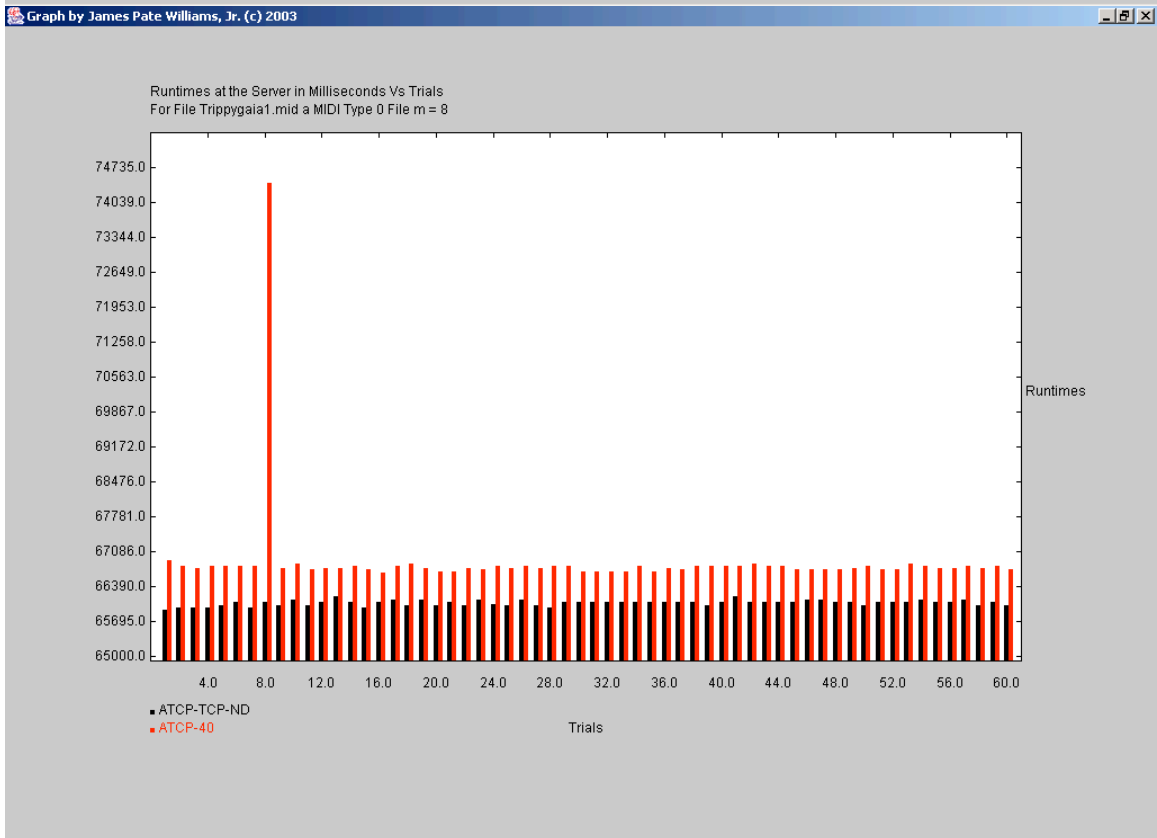
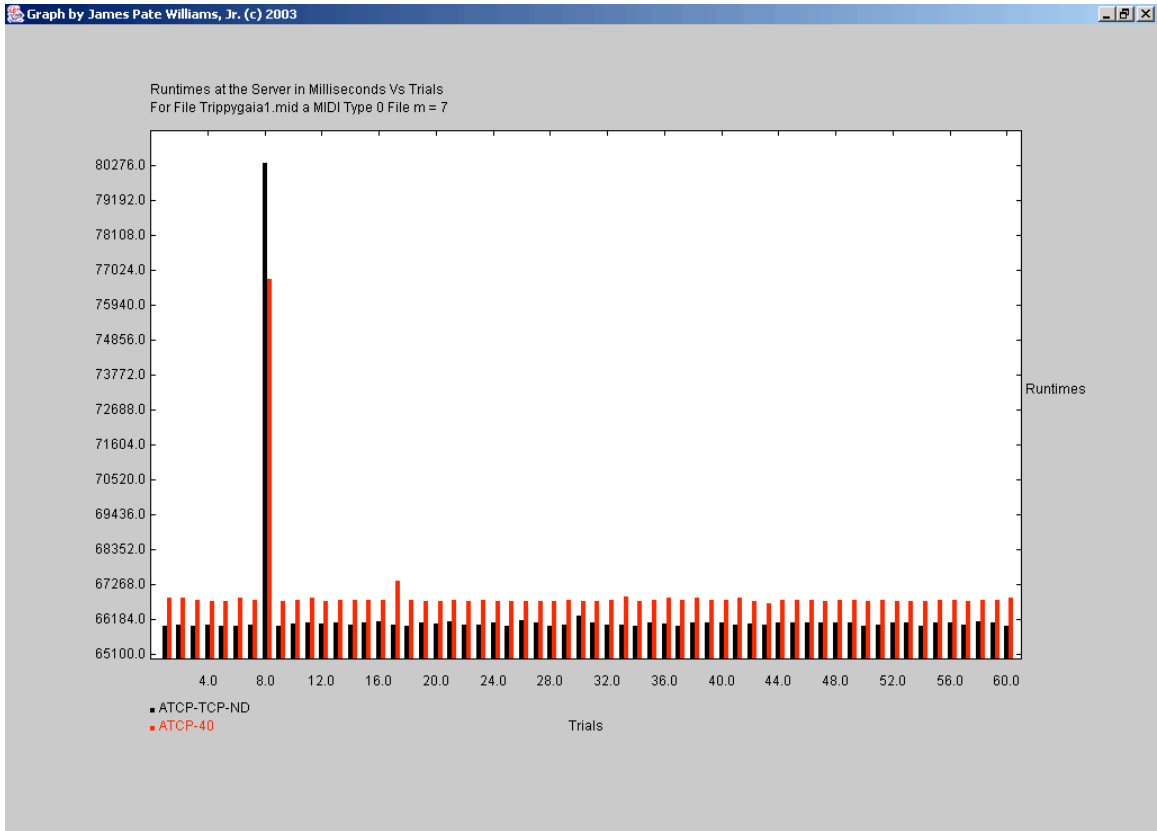












APPENDIX N ATCP-TCP-NE VS SN-TCP-NX, ATCP-X, AND ATCP-TCP-ND

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66525.83	60	2278.37	294.1370
SN-TCP-ND	67100.16	60	532.20	68.7069

Table N-1 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-ND	-574.3333	2348.8759	303.2385	-1.8939	59	0.0631

Table N-2 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66581.50	60	2695.9076	348.0401
SN-TCP-ND	67044.83	60	54.3838	7.0209

Table N-3 Trippygaia1.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-ND	-463.3333	2701.0567	348.7049	-1.3287	59	0.1890

Table N-4 Trippygaia1.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66471.00	60	1645.5452	212.4389
SN-TCP-ND	67371.16	60	2434.3969	314.2792

Table N-5 Trippygaia1.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-ND	-900.1666	2950.9909	380.9712	-2.3628	59	0.0214

Table N-6 Trippygaia1.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66832.83	60	1598.4325	206.3567
SN-TCP-ND	67220.00	60	1214.1189	156.7420

Table N-7 Trippygaia1.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-NE	-387.1666	2014.6456	260.0896	-1.4885	59	0.1419

Table N-8 Trippygaia1.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66525.83	60	2278.3760	294.1370
SN-TCP-NE	67230.83	60	82.8985	10.7021

Table N-9 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-NE	-705.0000	2280.4448	294.4041	-2.3946	59	0.0198

Table N-10 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66581.50	60	2695.9076	348.0401
SN-TCP-NE	67432.83	60	1596.0301	206.0466

Table N-11 Trippygaia1.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-NE	-851.3333	1104.1930	142.5507	-5.9721	59	0.0000

Table N-12 Trippygaia1.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66471.00	60	1645.5452	212.4389
SN-TCP-NE	67442.50	60	1561.3256	201.5662

Table N-13 Trippygaia1.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-NE	-971.50	2270.3026	293.0948	-3.3146	59	0.0015

Table N-14 Trippygaia1.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66832.83	60	1598.4325	206.3567
SN-TCP-NE	67298.83	60	81.4922	10.5206

Table N-15 Trippygaia1.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-SN-TCP-NE	-466.0000	1594.7235	205.8779	-2.2634	59	0.0272

Table N-16 Trippygaia1.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66525.83	60	2278.3760	294.1370
ATCP-32	66937.66	60	1871.3897	241.5953

Table N-17 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-32	-411.8333	2966.7447	383.0051	-1.0752	59	0.2866

Table N-18 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66581.50	60	2695.9076	348.0401
ATCP-32	66487.00	60	1958.3314	252.8195

Table N-19 Trippygaia1.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-32	94.5	741.6093	95.7413	0.9870	59	0.3276

Table N-20 Trippygaia1.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66471.00	60	1645.5452	212.4389
ATCP-32	66778.66	60	43.4708	5.6120

Table N-21 Trippygaia1.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-32	-307.6666	1638.6490	211.5486	-1.4543	59	0.1511

Table N-22 Trippygaia1.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66832.83	60	1598.4325	206.3567
ATCP-32	66758.83	60	48.9583	6.3204

Table N-23 Trippygaia1.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-32	74.0000	1595.2495	205.9458	0.3593	59	0.7206

Table N-24 Trippygaia1.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66525.83	60	2278.3760	294.1370
ATCP-40	66822.83	60	150.2550	19.3978

Table N-25 Trippygaia1.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-40	-297.0000	2295.7730	296.3830	-1.0020	59	0.3203

Table N-26 Trippygaia1.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66581.50	60	2695.9076	348.0401
ATCP-40	66992.66	60	1201.5143	155.1148

Table N-27 Trippygaia1.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-40	-411.1666	1549.6933	200.0645	-2.0551	59	0.0442

Table N-28 Trippygaia1.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66471.00	60	1645.5452	212.4389
ATCP-40	66952.33	60	1286.8044	166.1257

Table N-29 Trippygaia1.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-40	-481.3333	2098.8305	270.9578	-1.7764	59	0.0808

Table N-30 Trippygaia1.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66832.83	60	1598.4325	206.3567
ATCP-40	66886.16	60	991.5765	128.0119

Table N-31 Trippygaia1.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-40	-53.3333	1891.4774	244.1886	-0.2184	59	0.8278

Table N-32 Trippygaial.mid m = 8

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66525.83	60	2278.3760	294.1370
ATCP-TCP-ND	66816.00	60	52.5679	6.7864

Table N-33 Trippygaial.mid m = 5

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-TCP-ND	-290.1666	2282.7299	294.6991	-0.9846	59	0.3288

Table N-34 Trippygaial.mid m = 5

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66581.50	60	2695.9076	348.0401
ATCP-TCP-ND	66791.50	60	60.4734	7.8070

Table N-35 Trippygaial.mid m = 6

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-TCP-ND	-210.0000	2697.2214	348.2097	-0.6030	59	0.5487

Table N-36 Trippygaial.mid m = 6

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66471.00	60	1645.5452	212.4389
ATCP-TCP-ND	66952.33	60	1286.8044	166.1257

Table N-37 Trippygaial.mid m = 7

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-TCP-ND	270.9578	2098.8305	270.9578	-1.7764	59	0.0808

Table N-38 Trippygaial.mid m = 7

Protocol	Mean	N	Std. Dev.	Std. Error Mean
ATCP-TCP-NE	66832.83	60	1598.4325	206.3567
ATCP-TCP-ND	66062.50	60	55.4068	7.1529

Table N-39 Trippygaial.mid m = 8

Difference	Mean	Std. Dev.	Std. Error Mean	T	DF	Sig. 2
ATCPTCP-NE-ATCP-TCP-ND	770.3333	1592.3748	205.5747	3.7472	59	0.0000

Table N-40 Trippygaial.mid m = 8



