ACCURATE AND PRECISE NETWORK

PERFORMANCE TESTING

IN WINDOWS 2000

By

JOSEPH RYAN HERSHBERGER

Bachelor of Science in Electrical Engineering Technology

Oklahoma State University

Stillwater, Oklahoma

2003

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2005

ACCURATE AND PRECISE NETWORK

PERFORMANCE TESTING

IN WINDOWS 2000

Thesis Approved:

Dr. Keith A. Teague

Thesis Adviser

Dr. George Scheets

Dr. Rao K. Yarlagadda

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

Thanks are due to my fiancée, Rachel, for her patience, understanding, and steadfast love during the crafting of this masterpiece. I also thank my parents, Art and Londa Hershberger, for their continued encouragement and support.

I thank my advisor, Dr. Keith Teague, for his assistance and support. He has been very good to me throughout my years here. I truly look up to him.

I thank Kellen Harwell for assisting with the Visio diagrams. I also thank the many people who helped me make revisions.

I dedicate this in memory of my good friend Austin Beman Barker (1980 – 2005). He loved spending time among God’s beautiful creations. His love for adventure ultimately cost him the highest price. May he rest in peace and may the comfort of the Holy Spirit be with his fiancée and his family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ACRONYMS

ACPI	Advanced Configuration and Power Interface
ADSL	Asynchronous Digital Subscriber Line
API	Application Programming Interface
ARP	Address Resolution Protocol
CDMA	Code Division Multiple Access
DHCP	Dynamic Host Configuration Protocol
FEC	Forward Error Correction
GPS	Global Positioning System
HPET	High Precision Even Timer
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IOCTL	I/O Control
IQR	Inter-Quartile Range
IRP	I/O Request Packet
IRQL	Interrupt Request Level
ISR	Interrupt Service Routine
LAN	Local Area Network

MAC	Medium Access Control
MELP	Mixed Excitation Linear Prediction
NAT	Network Address Translation
NDIS	Network Driver Interface Specification
NIC	Network Interface Card
NTP	Network Time Protocol
OOB	Out Of Band
OSI	Open Systems Interconnection
PIC	Programmable Interrupt Controller
PIT	Programmable Interval Timer
PMT	Power Management Timer
PPM	Part Per Million
PPS	Pulse Per Second
RTP	Real-time Transport Protocol
RTT	Round Trip Time
SMP	Symmetric Multi Processing
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
WAN	Wide Area Network
WDM	Windows Driver Model

CHAPTER 1

INTRODUCTION

Network performance greatly affects most applications that use a network to transport data of some type. How much an application is affected is largely dependent upon the type of data being sent and how closely the end user interacts with the data. Accurately and precisely measuring different metrics for network performance can give the network application developers valuable information to minimize the effects of the network.

Measuring network performance, while valuable, can be costly. The most accurate and precise methods to measure network performance require the use of specialized, dedicated hardware. A hardware solution is not only expensive, it is also less versatile than a software solution. With this in mind, it is both more cost-effective and practical to design a software network performance analyzer with sufficient accuracy and precision that will run on a standard Windows 2000 or Windows XP based personal computer with only moderate hardware support. Because the application runs in this operating system environment, testing a particular configuration will not require the installation of a new operating system and subsequently bypasses the necessity to install any software support that hardware in the new operating system might require.

The application of interest in this case is low bit-rate Voice-over-IP communication, specifically Mixed Excitation Linear Prediction (MELP), which is a 2400 bit-per-second voice coder. The goal of this study is to create a system that will emulate the traffic of the application and accurately measure how that traffic is affected by the network. Some of the typical metrics such as throughput and round-trip time are either not important or do not apply and will therefore not be discussed. There are two metrics that are specifically of interest: jitter and latency. These are discussed further in Chapter 2.

Chapter 2 will discuss what measurements are important in this study as well as some of the fundamental options available when measuring network performance such as where the timestamp code can be implemented, what clock sources are available, and what synchronization options are available. Chapter 3 will review some of the related work discussed in literature and discuss some of the approaches taken in the past. The next five chapters will detail the development of the system used to perform the network tests. Chapter 9 discusses the resulting system and validates its design. In Chapter 10, the results of some network performance tests are analyzed and finally, in Chapter 11, some conclusions are made and future work is addressed.

CHAPTER 2

FUNDAMENTALS OF NETWORK PERFORMANCE ANALYSIS

Network performance analysis can take many forms. This is because there are several different factors of network performance that affect each different type of network application. Some of the performance metrics that can be measured include bandwidth, end-to-end latency, jitter, packet loss, and out-of-order packets (often caused by multi-path routing). A framework for IP Performance Metrics is given in RFC 2330 [34].

2.1. Measurements

In the case of streaming low bit-rate audio, the two primary performance measures of interest are jitter and end-to-end latency. Jitter is defined by the RTP standard [37] and by RFC 3393 [14], which also refers to jitter as delay variation, to be the difference between the end-to-end latencies of two consecutive packets. End-to-end latency, referred to as one-way delay in RFC 2679, is defined to be the time difference between the moment the first bit of a packet is transmitted to the moment the last bit is received [7]. The reason end-to-end latency is considered only secondary is that it does not have an adverse effect on the application’s ability to effectively deal with the network conditions.

The minimum end-to-end latency for a network path caused by the propagation through each link and processing by each router sets a baseline for the perceived round trip time for a conversation that the user experiences. Jitter, on the other hand, refers to the random increases in the end-to-end latency on top of the baseline. It is typically caused by queuing delays in routers along the path due to competing traffic.

A typical Voice-over-IP application implements a jitter buffer on its receiving side to smooth the stream of audio frames before they are played. Understanding the jitter characteristics of different networks is important in order to optimize the design of the jitter buffer in a Voice-over-IP application. If the buffer is too large, the round-trip time that the user perceives is higher, causing the user experience to seem more like a 2-way radio conversation than a face-to-face conversation. If too small, buffer starvation occurs and gaps in audio playback will severely reduce audio quality. These parameters can vary with network conditions during a conversation. Ideally, the jitter buffer should be designed to adapt to network conditions as they change [35].

2.2. Requirements

Useful measurement of jitter requires the measurements to be made as close to the wire as possible for reasons described in Section 2.3 in addition to a precise clock. Only the clock's precision is important for this measurement, and not its accuracy, because it is based on a difference of times, and any offset will cancel. This is required on both ends of the test because the two differences are compared by the jitter calculation. The end-to-end measurements have the added requirement that both clocks must be accurate (synchronized). In addition to measurements, it is also necessary to generate traffic that closely resembles a system of interest. In this case, we are interested in the network's ability to support low bit-rate speech traffic. This means that the traffic generated should closely resemble a Mixed Excitation Linear Prediction (MELP) audio stream. The MELP audio codec generates a 7-byte frame of data for every 22.5ms of speech data. These frames are typically sent between two and five at a time in a UDP packet. This number of frames per packet is referred to as the bundling factor. For all of the tests conducted in this study, a bundling factor of two will be used. This means that the data source should generate a 56-byte UDP packet comprised of 42 bytes of Ethernet/IP/UDP headers and 14 bytes of MELP data every 45ms as shown in Figure 2.1. The minimum frame size for Ethernet is 64 bytes, which includes a frame-check sequence. This means that the frames will also include 4 bytes of padding, which is completely wasted bandwidth. It is clear that this is not very efficient, but sending more frames per packet would increase the perceived delay.

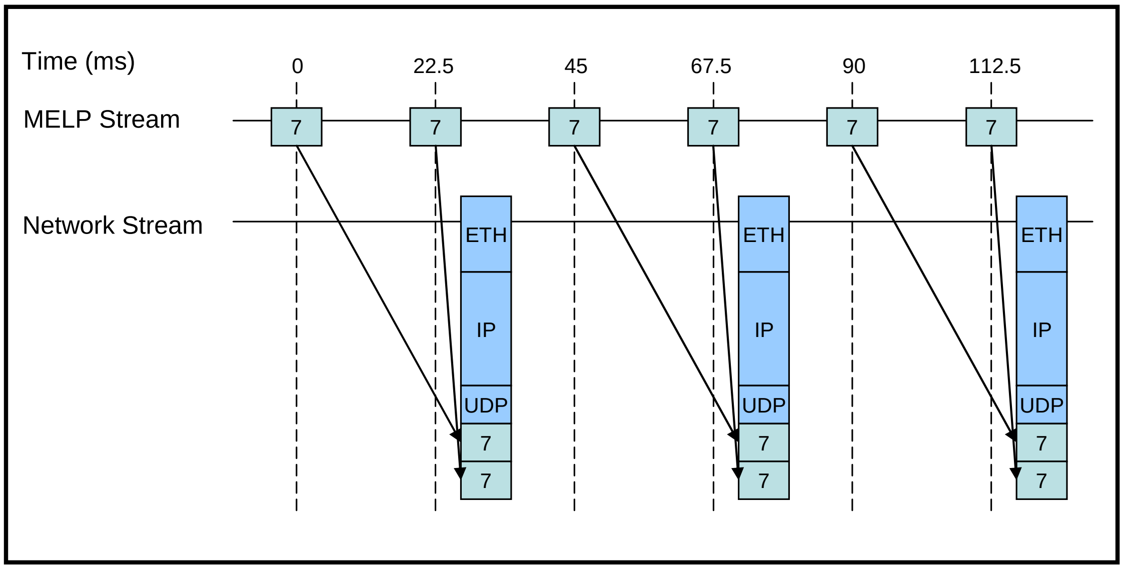


Figure 2.1 – MELP Stream Packetization

2.3. Timing Location

The location at which the measurements are taken is very important. The notion of wire-time is defined in RFC 2330 [34] as the time the packet is transmitted or received on the network in contrast with the host-time which is the time that the timestamp is generated. In RFC 2679 [7] the difference between host-time and wire-time is defined to be comprised of a systematic error and a random error. The farther away from the Ethernet hardware the measurements are taken, the more the systematic and random error will be added. High system utilization can greatly amplify the random error in Windows 2000/XP because they are not real-time operating systems. A real-time operating system gives the guarantee that the program flow is deterministic (to some degree) thereby eliminating a majority of the random timing errors that are location-related. Not only do the number and priorities of other running processes affect the random error, but also the frequency and processing time of hardware and software interrupts. There are at least four locations where packets can be timestamped: a user-mode application, a protocol driver, a NIC driver, or in the NIC hardware as shown in Figure 2.2.

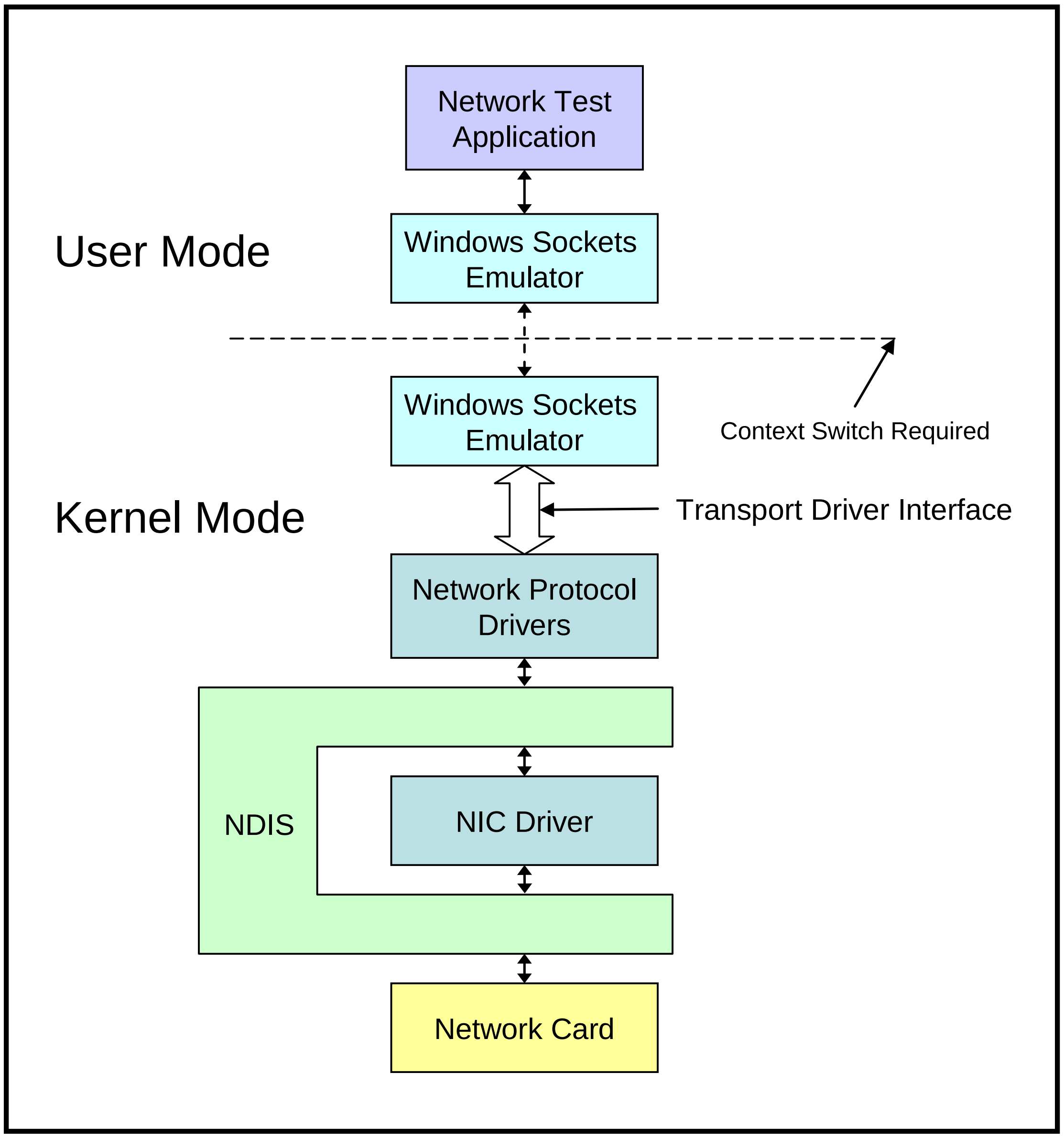


Figure 2.2 – Available Timing Locations

2.3.2. User Mode Application

The least desirable place to take measurements is in the user mode application after waiting to send or receive a packet through the Windows socket interface. The packet is queued and processed for an unknown length of time by Winsock, the protocol drivers, the NIC driver, and the NIC itself in addition to the context switch required to change to kernel mode. When sending, these delays not only affect the measurement, but also the traffic generation.

2.3.3. Protocol Driver

A protocol driver sits just above the NIC driver and implements a network protocol such as TCP or UDP. Timestamping packets in a protocol driver eliminates the context switching and queuing associated with transferring data and control from kernel-mode to user-mode. This reduces both the systematic error and the random error.

Unfortunately, when working with the kernel, there is typically not source code available for a particular driver, so customizing it is not feasible. The only way to get packet timestamps for IP packets from an unmodified protocol driver is through the IP header option for timestamps. This timing is done by the IP protocol driver using an unknown, implementation-specific clock. As such, it is not possible to synchronize it with a reference clock except to assume it is based on the system clock and synchronize that. The time format used is a 32-bit field that represents the number of microseconds since midnight UTC.

There is an open source project known as WinPcap [6, 13, 36] which is implemented partly as a protocol driver. WinPcap provides the framework for accessing the packets while they are still in the kernel. It also provides a mechanism to return timestamps with the packets as they are received. Even if the timestamps were not applied to the packets in WinPcap directly, its interface would be required to return the timestamps since that is not possible through the Winsock interface.

2.3.4. NIC Driver

The NIC driver directly controls the NIC hardware and is notified with an interrupt when a packet is sent or received by the NIC. The effects of the operating system on the timing measurements are quite low in the NIC driver because context switching and Network Driver Interface Specification (NDIS) queuing are eliminated. Some NICs, in an effort to more efficiently communicate with the system by reducing the interrupt frequency, will buffer multiple packets in hardware before interrupting and transferring them to the host. This is another source of random error that can be introduced into the measurements.

The NDIS includes a mechanism for passing additional information, referred to as out-of-band (OOB) data, to higher level drivers. This OOB data includes the time a packet was sent or received, formatted as the number of hundreds of nanoseconds since midnight January 1st 1601. The existence of this mechanism means that the information could be retrieved in the WinPcap protocol driver and passed on to the user mode, allowing measurements that are closer to the wire. Unfortunately, the generation of this information is optional and we have been unable to find any reference to any driver for any NIC which does generate it. Without detailed hardware specifications of any particular NIC, writing a replacement driver that includes this functionality is infeasible. Even with that detailed information it would be impractical.

The Microsoft Windows 2000 DDK [4] ships with a sample NDIS driver for the Intel EtherExpress Pro/100+. Having device driver source code for a NIC that is fully functional makes it possible to add small pieces of code without having to either know or have access to the detailed hardware specifications for the NIC. The driver was modified in two ways to provide timestamps that are based on the Pentium Time Stamp Counter (chosen because if its extremely low call overhead). Within the interrupt service routine (ISR) for the NIC, a timestamp value is read from the Pentium Time Stamp Counter. This timestamp is stored until the type of interrupt is determined. If determined to be a receive interrupt, then the raw timer value is stored in the OOB data. The protocol driver can then check for the timestamp in the OOB data and, if present, read the Pentium Time Stamp Counter again to compute the time that passed. If determined to be a transmission-completion interrupt, the protocol driver will have generated a raw timestamp made at the time that it generated its primary timestamp for the packet and stored the timestamp in the OOB data. The timestamp generated in the ISR is subtracted from the timestamp already in the OOB data and the result stored in its place. These timestamps can then be used to adjust the primary timestamps to reflect a time that is closer to the wire-time. This is discussed further in Chapter 5.

2.3.5. NIC Hardware

The ideal method of timestamping packets is using a hardware clock on the NIC that timestamps the packets as they arrive. This eliminates all operating system effects and measures the wire-time exactly. Unfortunately, most NICs do not have this capability and most often, one of the previous methods will need to be employed.

Another possibility is to use devices such as the cards produced by Endace called DAG cards [10, 22]. These cards provide a hardware clock and timestamp the packet at the very beginning of its reception. Donnelly et al. [15] use this card to measure the systematic and random delay errors of the RIPE NCC software-based measurement system [40]. The cards are capable of being synchronized to several different time sources including GPS or CDMA.

The CDMA signal provides a time signal nearly as accurate (within approximately 5µs) as GPS (within approximately 100ns) and does not require the clear view of the sky that GPS does [1]. One limitation of these cards is that they only provide the ability to receive packets, not transmit them. Another solution would still be needed to transmit the packets, though a host using this card could passively monitor the traffic as it is transmitted. We have been unable to acquire one these cards to compare its performance with other methods and it is left as future work, though based on the design, it should be superior to all other methods described here.

2.4. Timing Methods

In a Windows environment it is difficult to get accurate and precise time sources. There are several time sources natively available in Windows, but because Windows is not a real-time operating system, no one of them performs well enough to be useful in the tests. To be at all useful in this study, a clock that is precise to approximately 100μs is needed. To truly be able to see the fine-grained jitter, a timer with a 1μs resolution is required. In addition, the clock needs to be synchronized to UTC for end-to-end latency measurements.

2.4.1. Real-Time Clock

The real-time clock, first added to the IBM-AT architecture in 1984, is relatively accurate, however long-term drift can occur due to an uncontrolled oscillator. It is not a good tool for precise, short-term measurements because it is only updated once every 10 to 15ms. It is accessed through calls to `GetSystemTime()` or `GetTickCount()`. Another interface available in the kernel is `KeQueryInterruptTime()`. In Windows 2000, `KeQueryInterruptTime()` has a resolution of 15.625ms (the same resolution as `GetTicks()`), but in Windows XP the resolution is 976.6 μs. It is implemented as simply a memory read from a periodically updated location, so call overhead is negligible (~8 cycles). This resolution is still not high enough to be useful.

2.4.2. Multimedia Timer

The multimedia timer that is available in Windows is typically capable of resolutions between 1 millisecond and 1,000 seconds. This timer is typically used for timing in MIDI sequencers. This timer does not provide measurements with high enough resolution to be useful. In addition, setting the resolution to 1ms will degrade system performance. It is accessed through calls to `timeGetTime()`. This function is part of the Windows Multimedia API which is not available in the kernel, making it of limited usefulness anyway.

2.4.3. Pentium Time Stamp Counter

The Pentium Time Stamp Counter is a 64-bit internal counter register in the processor. It counts at the speed of the processor. This means that on modern machines the resolution of these timers is very good (less than 1ns). Unfortunately, modern processors also have features such as throttling of the processor speed to reduce heat and power consumption. These large, numerous changes in frequency make the time stamp counter unreliable on some machines and therefore useless for timing. With this feature disabled, the speed of the processor is observed to provide long-term accuracy (less than 0.1 PPM) [29]. It is very efficient to call; only approximately 15 cycles are required. The time stamp counter is accessed through the `rdtsc` assembly instruction. In some cases is it accessible through the performance counter API as discussed in Section 2.4.4.

Attila Pàsztor and Darryl Veitch [29] make use of the Pentium Time Stamp Counter to make a more accurate system clock in Linux, BSD, and RT-Linux. They synchronize the clock using NTP, but focus on making a rate-stable clock as opposed to the "`ntpd`" which attempts to stabilize the offset. This could be a good approach, but their system was designed to run in a UNIX environment, so testing this method in Windows is left as future work.

2.4.4. Performance Counter

The Performance Counter is typically a hardware counter in the chipset of the computer. It is accessed through calls to `QueryPerformanceCounter()`, though it can be implemented in different ways[2]. Two common counters that are accessed through this API are the Programmable Interval Timer (PIT) provided by the 8254 Programmable Interrupt Controller (PIC), and the Power Management Timer (PMT) that is part of the Advanced Configuration and Power Interface (ACPI). The PIT was introduced in the IBM PC in 1981 and uses a 1,193,182Hz source, but the measurement of the call overhead is not available because the PIT is only used on Pentium class and older machines. The PMT uses a 3,579,545Hz source and has an approximate call overhead of 600ns.

Another timer included as part of some new PC chipsets is called the High Precision Even Timer (HPET). This timer specification was jointly developed by Microsoft and Intel. It is specified to have source clock of at least 10MHz[2] but in practice, it is using a 14,318,180Hz clock [3]. We currently have no motherboard that includes the HPET, but Microsoft Test Engineers found a 61% performance increase in calls to `KeQueryPerformanceCounter()` [2].

In a Symmetric Multiprocessing Machine (SMP) the Performance Counter is implemented with the Pentium Time Stamp Counter, so it will have similar performance and slightly more overhead because it is accessed through the API.

2.4.5. Dedicated Time and Frequency Processor

A dedicated time and frequency processor, in this case the Symmetricom BC637PCI card, provides both accurate and precise timing over long time periods. The software provided with the card is written to allow access to the card only from User-Mode, so eventually it was necessary to write a device driver for it that allowed access to other drivers in the kernel, as discussed in the next chapter. It uses a 10 MHz master clock and therefore has a resolution of 100ns. It is accurate to less than 2 µs when synchronized to a GPS receiver and accurate to less than 5 µs when synchronized to other equipment via the IRIG-B time-signaling protocol. The overhead required to access the card is relatively high, about 1.3 µs, because it requires a bus access.

2.5. Clock Synchronization

Clock synchronization is required to compute latency. Because latency is the time it takes a packet to travel from one computer to another, it must be possible to read from synchronized clocks on the two computers.

2.5.1. NTP Synchronization

The Network Time Protocol (NTP) is widely used on the Internet to synchronize clocks to the atomic time standard [24]. The protocol utilizes a tiered approach to synchronize clocks. The servers that are synchronized either directly to a Cesium clock or to a GPS receiver are referred to at Stratum 1 servers. Stratum 2 servers are synchronized to Stratum 1 servers and so on. The typical end device will connect to a Stratum 2 server or higher, so as not to overload the primary servers.

NTP is used for synchronization of clocks on a large time scale such as minutes or even days [31]. It makes use of changes at small time scales to provide the synchronization at larger time scales. This means that NTP will actually make the clock less reliable on a small time scale due to its adjustments and is clearly not an appropriate technology for synchronization in this application.

The NTP implementation that is built into the Windows "`net time`" application is only intended to keep the time synchronized to within a few minutes. This level of synchronization is needed by the default authentication protocol (MIT Kerberos version

5) for Windows 2000. Another option that comes with Windows 2000 is the "`W32Time`" service [23] which is a fully compliant SNTP client [25] with about the same synchronization target.

The "`ntpd`" operating system daemon can potentially achieve synchronization to within a few milliseconds in ideal conditions, but in the networks of interest in this study its actual performance is rarely better than hundreds of milliseconds.

Another implementation of NTP by Darryl Veitch et al. [41] makes use of the Pentium Time Stamp Counter and the timestamping of NTP server packets and focuses on rate synchronization and offset synchronization as separate problems. Assuming a symmetric low-latency connection to a nearby NTP server, they were able to get synchronization results of within 30μs and a rate of within 0.02 PPM. Even better offset results (within 1μs with rate stability of 0.1 PPM) were achieved when using a Real-Time operating system (RT-Linux) [29].

2.5.2. Time Processor Calls in User-Mode

The time synchronization issue is can be handled in hardware, instead of software, if a dedicated time processing card, such as the Symmetricom BC637PCI is used. The card can be synchronized to sources such as GPS, IRIG-B, IEEE-1344, or Pulse per Second (PPS) signals independently of the software. Timestamps can then be directly read from registers on the card. Due to the fact that the factory-supplied software does not provide access to the time processor card from within the kernel, an attempt was made to relate system times that were measured in the kernel with those measured in user-mode. The performance counter was synchronized to the system clock in user-mode [27] to give a high resolution version of the system clock in user-mode. Plots of both the kernel-mode system time and the user mode system time proved to be for the most part linear. The user-mode clock was compared with the time processor's clock and found to be nearly linear, so it was assumed that the system clock was simply not running at quite the correct frequency. To correct this using the time processor, a process was designed in which two readings were taken in user mode of both the system clock and the time processor's clock: once at the beginning of the test and once at the end shown as events "x" and "y" in Figure 2.3. This approach is similar to that taken by J. Curtis et al. [12] with the exception that they made a comparison reading every second. These two points were used to calculate a frequency error and an offset error for the system clock with respect to the time processor's clock by solving a linear system of equations. After the test was over, all of the packet times were converted to times corresponding to the time processor's clock.

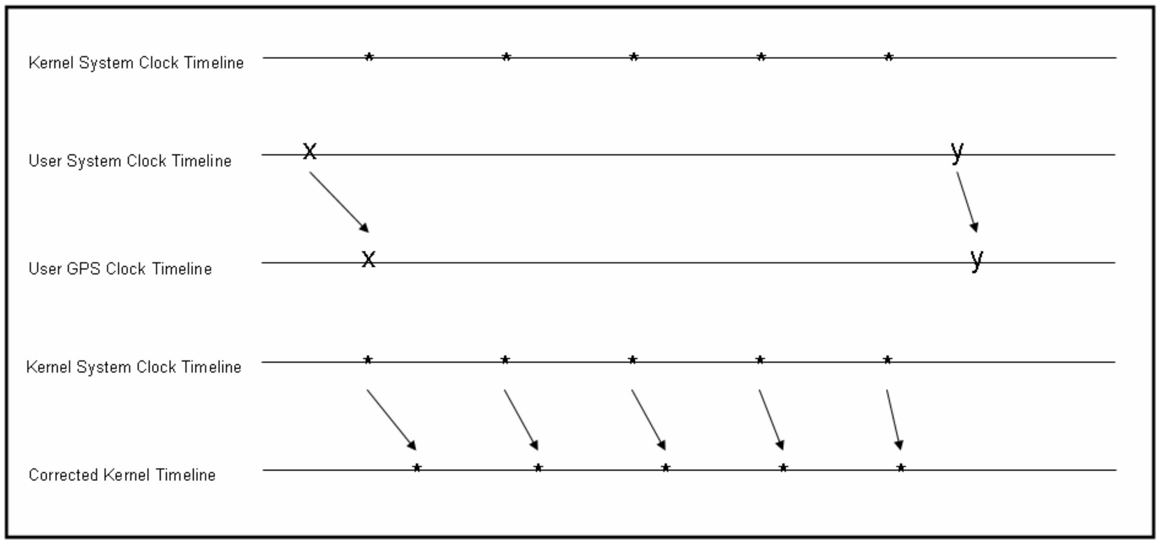

Kernel System Clock Timeline
User System Clock Timeline
x
y
User GPS Clock Timeline
x
y
Kernel System Clock Timeline
Corrected Kernel Timeline

Figure 2.3 – System Clock Correction using GPS

The results of these tests were poor. The latency measurements drifted and sometimes became negative. It was clear that there was a serious problem with one of the clocks which caused the drift in the measurements. Because of the drift, this method was not acceptable and it became necessary to write a replacement driver for the time processor card to allow access to the time processor's clock from within the kernel as well as in user-mode.

Using this new driver, it was possible to fully investigate the problem with the original algorithm. In one case, shown in Figure 2.4, the user-mode system clock is seen to be drifting away from the kernel-mode system clock at approximately 1 µs/s (an error of 1 PPM). Given the nature of the measurements being taken, this is significant. Figure 2.5 shows the latency computed by this method for a test run on a LAN. It is clear that the latency is not reasonable due to its steady increase. The rate of increase is based upon the clock synchronization errors on both machines involved in the test. It was later discovered in the WinPcap protocol driver source code that the kernel-mode system clock is not actually synchronized to the system clock, but is only initialized by the system clock and is free-running from then on.

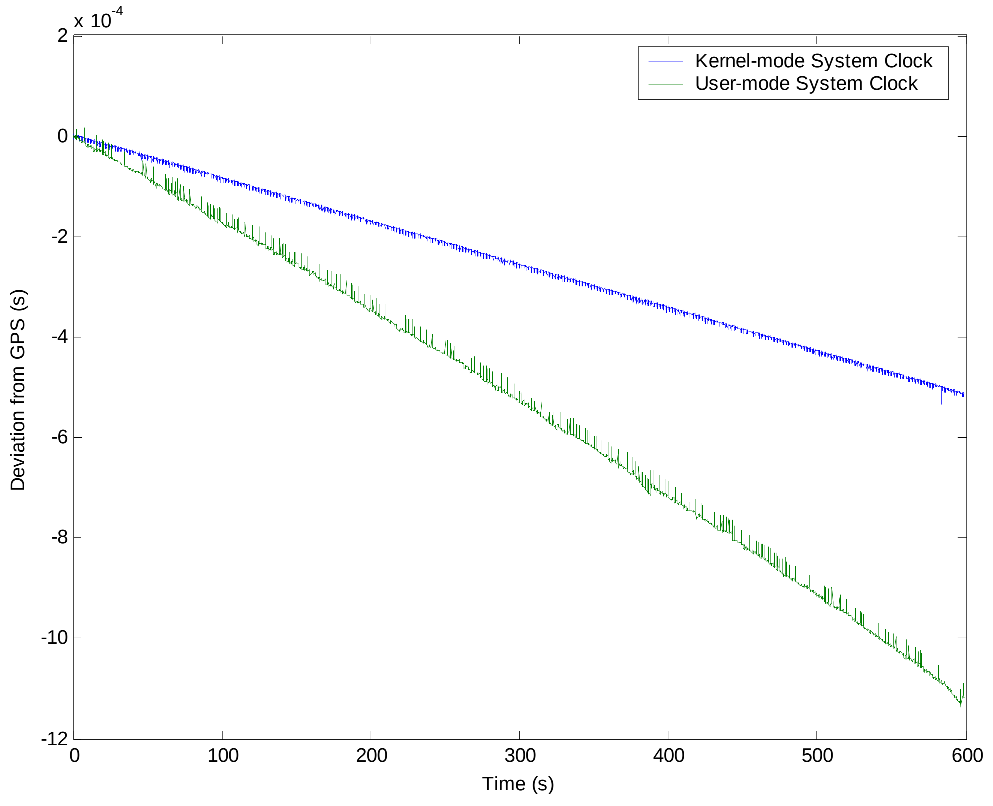


Figure 2.4 – System Clock Comparison

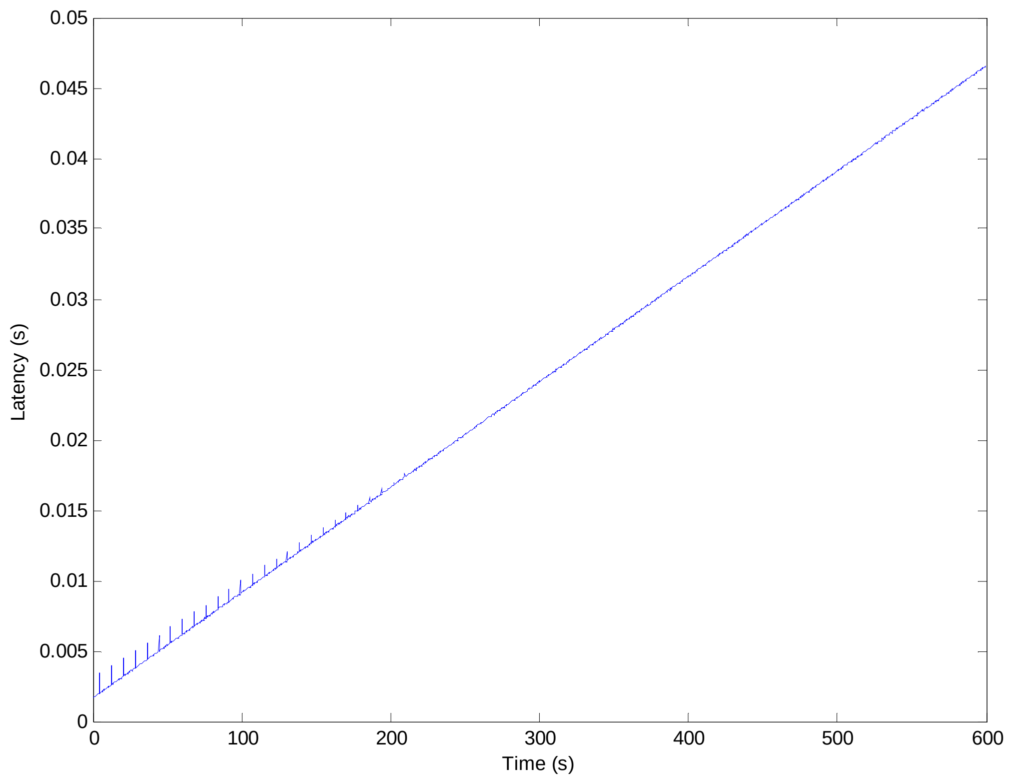


Figure 2.5 – Latency Computed from System Clock Comparison Test

2.5.3. Time Processor Calls in Kernel-Mode

After a new driver for the time processor card was developed, it was used in the kernel to directly measure the timing of the packet transmission and reception. Because the time stamps are taken directly from the time processor cards and the time processors are synchronized with one another, the time stamps are absolute (to within 2 μs) and can be directly compared.

2.5.4. Sub-Microsecond Synchronization

If the time processors are in close proximity, then it is possible to synchronize the cards to within 100 nanoseconds. The BC637PCI card is equipped with an external event pin and a register for adjusting for propagation delay. The register allows for correction when using the IRIG-B time code bus between two Time Processors. To compute the offset required for use in the propagation delay register, a pulse is generated external to the two cards and routed through two equal length wires to the external event pins on the cards. The event time registers are then read and compared. Because the event registers should represent the same event and refer to the same moment in time, the difference between the values should be used as the correction factor. In testing this method, it was found that after setting the propagation delay register, the event times matched to within 100 nanoseconds for all subsequent events.

CHAPTER 3

RELATED WORK

Researchers have focused on measuring many different aspects of Internet dynamics and performance and have taken different approaches at measuring them. Thomas Chen et al. review a majority of the different approaches for both active and passive performance measurement [8]. In all of the studies of network performance, some form of timestamping was required, though the choices varied greatly based on required accuracy, deployment capability, and cost.

3.1. Original Network Performance Test Application

It was initially developed by a senior design II team in the school of Electrical and Computer Engineering department at Oklahoma State University and was called Network Performance Application (NetPerf). It has a Winsock network interface with user-mode timing. The timing relied on NTP to synchronize the system clock which was then queried via the `GetSystemTimeAsFileTime()` API. An attempt was made to improve the resolution by making a call to `QueryPerformanceCounter()` when a timestamp was needed but nothing was done to synchronize those calls with the system clock. This means that the original system had large errors due to time measurements being so far from the wire in addition to the poor clock being used to generate the timestamps. What little statistical analysis it had was poorly documented. It was frequently hacked by others in an attempt to correct errors, even though they did not understand how it was initially intended to work. The user interface was very simplistic and utilitarian. Many variables were uninitialized leaving unfriendly default values when requesting user input. The "Connection Parameters" dialog would discard values after they were stored in the primary application, which meant that if a user needed to return to the dialog to change one setting, it would be necessary to reenter all of the parameters for the test. The tests were limited to only being able to send data from the machine that originated the connection. When a test actually begins, the parameters of the test are sent to the server, but in the original application, only integers were used to send data that included the packet send rate (a floating point variable) so the value was truncated to its integer part. This was not significant, however, because as stated earlier the server was incapable of transmitting. This test system demonstrated a naïve approach to network performance testing that was prone to significant errors as well as being unpredictable. This system aims to greatly improve the accuracy of the measurements made such that detailed information about the networks being tested is visible.

3.2. Measurements Using Clocks Not Synchronized with GPS

Clocks not synchronized with GPS are easy to deploy and are typically relatively low cost. Unfortunately, these clocks suffer from poor accuracy and cause lots of extra post-processing work to compensate for that poor accuracy.

3.2.1. Vern Paxson's Work

Vern Paxson performed two experiments: one at the end of 1994 and one at the end of 1995. The two experiments involved 35 different sites running his measurement daemon. His PhD dissertation [31] discusses, in great detail, all aspects of his Internet measurement experiments. He used TCP bulk transfers of 100 kilobyte files for traffic. At the end of these 20,800 tests, he analyzed properties such as the route asymmetries, bottleneck bandwidths, TCP implementation problems, packet loss patterns, and much more. His measurement daemon uses `libpcap` [18] as the interface to the packet filter on many different architectures running UNIX or BSD operating systems. This means that the clock available for each measurement is of questionable quality. He later attempted to improve the quality of the measurements by removing timing artifacts from the data [32]. His work was followed by Sue Moon, et al. who approached the problem as a linear program [26] and subsequently by Li Zhang, et al. who took a convex hull approach [42].

The design of Paxson's system is focused on deploying software test daemons to many volunteer sites around the Internet. All of the supporting test sites provide a machine on which to run the test software. These machines consist of a wide variety of hardware architectures and operating systems, and as such, the software must be flexible and able to function well on a large variety of configurations. Because of this, no specialized hardware can be used to provide superior timestamping capability.

3.2.2. ICMP Based Profiling

The most common form of simple network test is to measure the Round Trip Time (RTT) using an ICMP ping packet. This approach was modified by Kimberly Claffy et al. [9] to make use of ICMP Timestamp Request packets. This allowed them to measure the end-to-end latency for both the outgoing and incoming path without requiring the deployment of dedicated hosts or daemons. Naturally, the clocks used to make the remote timestamps were completely unknown and most likely unsynchronized and, as such, suffered from the same accuracy issues addressed by Paxson et al.

3.3. Measurements Using Clocks Synchronized with GPS

A clock synchronized with GPS has the obvious advantage of being accurate, but most GPS solutions are expensive, not readily available, and require rooftop antenna installation. For widely deployed test systems, antenna installation often eliminates GPS as an option.

3.3.1. NTP Software Clock Synchronized with GPS

The RIPE Internet delay measurement project [40] was conducted at many ISPs in Europe, the Middle East and parts of Central Asia. Each test point was an identical machine that was provided by RIPE and placed at the border router of the ISP. Each test point is a PC running BSD and using “`ntpd`” as the software synchronization protocol. It is synchronized with an external GPS receiver through a “Totally Accurate Clock 2”

[5] interface board using a Pulse Per Second (PPS) signal read by the PC's parallel or serial port. This software clock solution is the least precise and efficient of the GPS solutions. It requires the constant adjustment of a relatively unstable software clock by "ntpd" to maintain synchronization.

J. Jeong, in his M.S. thesis [19], argued for the necessity of one-way delay measurements due to the asymmetric routes found commonly in the Internet. He used a nearly identical configuration, with the exclusion of the "Totally Accurate Clock 2," to measure one-way packet delay and loss. This system was expected to be deployed within the Korean Commercial Network and the Asia Pacific Advanced Network.

3.3.2. Time Processor Card

The Surveyor project [20] aims to provide delay and loss information as well as routing information continuously and in near real-time (within 5 minutes of the current time), thus providing researchers or network engineers valuable information about current and past states of the Internet. Each measurement PC runs BSDI and contains a TrueTime bus-level timing card that is similar to the BC637PCI to provide timestamps for network events. Timestamps are recorded from within a modified BSDI network driver to get as close to the wire-time as possible. This project by far most closely resembles the study discussed in this document.

3.3.3. DAG Based Measurements

The most accurate and precise Internet performance measurements make use of the synchronized, hardware based timing solution provided by the DAG [10, 22] series cards. These cards have an onboard clock that can be synchronized to GPS or CDMA [1] and then used to timestamp each packet as it arrives in hardware. This not only takes the load of generating timestamps and synchronizing the clock off of the CPU of the measurement machine, but also provides an accurate and precise timestamp that represents the wire-time of the packet.

Attila Pasztor et al. [30] use them in a receiver in conjunction with an RT-Linux based traffic generator to implement an active probing infrastructure. They investigate several options including Linux and FreeBSD, but ultimately choose this to be the superior design. With this configuration, they achieve impressive timing results which are necessary for determining link rates using low bit-rate probe streams that show spaced out inter-arrival times on the receiver. This is the ideal solution for active probing short of a fully dedicated hardware solution. The final configuration covered by this document would closely resemble that of this active probing infrastructure if it were not important to use Windows XP/2000 as described in Chapter 1.

Doru Constantinescu et al. [11] use the DAG cards in independent measurement points to monitor traffic as well as actively probe the network. Their tests measure router performance in loaded and unloaded conditions using UDP traffic that resembles TCP traffic patterns. The primary focus is on one-way transit time (end-to-end latency) and how it is affected by routers.

Stephen Donnelly et al. [15] cover the use of DAG hardware to accurately measure the error of the RIPE NCC software based active probing system [40]. They determine the systematic error and the random error as described in RFC 2679 [7]. With this error information, the systematic error can be removed and the measurements improved. The random error distribution gives a range for the confidence in the results obtained from the software system.

3.3.4. Other GPS Solutions

Ian Graham et al. [16] developed a system for passive measurement of network traffic which made use of a GPS receiver's PPS signal. This signal was read by the PC's ring indicate (data carrier detect is sometimes used instead) input of its serial port to correct timestamps that were generated by `libpcap` in Linux. The packets were stored once a second along with the GPS corrections. Any packets that could not be stored in that one second were discarded.

Based upon the work of Ian Graham et al. [16], J. P. Curtis et al. [12] performed a passive measurement study of Voice-Over-IP network traffic using standard `libpcap` capturing and timestamping. To correct the timestamps generated by `libpcap`, they inserted zero-length packets every second with the GPS interrupt. These corrections take care of inaccuracies in the system clock, but do nothing to address error introduced by the difference between wire-time and the time the measurements are made in `libpcap`. They took another approach with ATM traffic which allowed them to make use of some limited hardware timestamping capabilities in the ATM NIC and get timestamps that represent wire-time.

CHAPTER 4

BC637PCI DRIVER

The BC637PCI Time and Frequency Processor card from Symmetricom (formerly Datum (formerly Bancomm)) provides an independent hardware clock available through the PCI bus in a PC. The original software provided with the card only included a simple driver that mapped the memory resources of the card to user mode and then relied on a user-mode DLL to implement all of the logic required to control the card. This meant that, within the kernel, there was no way to access the driver.

A lot of effort went into avoiding the development of a replacement driver for the BC637PCI, but after it was decided that the development was necessary, quick progress was made in a relatively short amount of time with the guidance of Programming the Microsoft Windows Driver Model by Walter Oney [28]. Some of the original reasons for avoiding the development were the anticipated problems with making a reliable driver in a reasonable amount of time, the added work of implementing the card configuration portions of the driver to replace the configuration capability of the provided software, and concerns about the required boilerplate code required for handling complex issues such as power management and Plug ’N Play. The card configuration functions include setting various modes and communicating with the onboard GPS module for determining satellite coverage, which would be a majority of the work even though it is only needed initially. It is possible to avoid this development by installing the old driver to configure the card and then installing the new driver for use during testing, however it is also inconvenient.

4.1. Getting Started

To begin the development of the driver, the WDM Driver wizard supplied with Oney's book [28] was used to generate as much of the relevant driver boilerplate as possible. This was a huge timesaver. Many of the functions that don't need unique handling on this card were written automatically and did not need to be modified. From this initial code base, the first step was to acquire all of the card's resources and keep track of them. This card has two memory resources (Dual-Port RAM and Device Registers) and an interrupt request line. The card's interrupt is not currently handled or enabled by the new driver because it is not needed for the testing system, but it is initialized and stored so that it is easily available for use in the future.

It can be difficult to identify the two memory resources due to misleading documentation. There are two other resources that get enumerated called device private resources, but they are undocumented. They are interleaved with the memory resources during the enumeration process. Each one only contains a single numeric parameter (one contains "0" and the other contains "1") that is presumably intended to be used to identify which memory space is which, but it could just be circumstantial. The documentation for the BC637PCI card [39] states that the Dual-Port RAM should be detected with a 0x1000 byte size and the Device Registers should be detected with a 0x40 bytes size, but both are detected as 0x1000 bytes. It also states in a different section that the current size of the

Dual-Port RAM is really only 0x800 bytes. This discrepancy was a bit misleading and initially led to some implementation problems. After discovering the true identities of the memory resources, they were mapped to kernel-mode memory space and stored in two pointers for future use.

4.2. Read Time

The primary purpose of the driver is to be able to read the current time from the card in both kernel mode and user mode. To initiate a time capture event on the card from the driver, simply access the `TIMEREQ` register. The time is then immediately latched into the `TIME0` and `TIME1` registers and can be read back. The format of the `TIME` registers is selectable between a binary format and a decimal format. The binary format provides the microseconds and the nanoseconds in `TIME0` and a 32-bit UNIX time in `TIME1`. The decimal format provides separate bits for the days of the year, hours, minutes, seconds, microseconds, and nanoseconds. For the purposes of this system and therefore this driver, the binary format will always be used for convenience.

The user-mode implementation of `ReadTime` is a synchronous I/O Control (IOCTL) command. This will allow the function to be available to any process in user mode that has access to open a handle to the kernel-mode driver. It would also be possible to simply map the PCI memory to user space as the provided software did, but this is not as easy to control because it doesn't use the constructs provided by the kernel interface. It is also not as flexible to use for the same reason, but it has the advantage that access to the card requires less overhead from user mode. This could be implemented in parallel in the future if deemed necessary or useful.

The kernel-mode implementation was originally a synchronous INTERNAL_IOCTL command to provide time services to other drivers in the system. This implementation suffers from several inherent problems based on the interface. First of all, the overhead of the interface is non-negligible. Also, because the interface is synchronous, it could possibly cause the calling driver to be required to wait for a result. This blocking is acceptable when running at `PASSIVE_LEVEL`, such as when timing something that is happening based on a call from user-mode, but if timing something generated by hardware, then it is not acceptable since the code is then run at `DISPATCH_LEVEL`. Code running at `DISPATCH_LEVEL` cannot block!

The typical solution to this problem is to simply use an asynchronous INTERNAL_IOCTL which will request the time and then call a completion routine when the measurement is acquired. Because of the typical use of the time provider, this is unacceptable. It would not be reasonable to try to handle the receipt of time measurements asynchronously because of the large overhead of keeping track of requests in the calling driver and trying to associate those with the events they are timing, which would require extra buffering to keep those events in memory until their timestamp is returned. The asynchronous interface also has a higher overhead than the synchronous one because of its callback function.

These difficulties are a result of the interface, not the underlying time request function. To avoid them, `ReadTime` was implemented last as a direct-call interface. This eliminated the limitations on IRQL and is more efficient because no IRP needs to be allocated. This interface has similar overhead to that of the provided software in user mode.

4.3. Configuration

The interface to the configuration of the card is through a Dual-Port memory command protocol used in association with the ACK register. There are a few things about accessing the Dual-Port RAM that are unlike accessing the PCI registers on the card. For one thing, the Dual-Port RAM can't be accessed more than one byte at a time from within the mapped memory space. If larger accesses are attempted, then all bytes of that variable will be equal to the most significant byte of the memory. Also, the memory is implemented as Big-Endian whereas the PCI registers are Little-Endian. The top-most 8 bytes of the Dual-Port RAM contain pointers into the memory-space in which to find the four main memory sections: the Input area, the Output area, the GPS Packet area, and the Year area.

The main command protocol is implemented through the Input and Output areas. A command and its operands are written to the Input area and then bit 7 of the `ACK` register is set. This tells the Time Processor to read the command. When the command is completed, the Time Processor sets bit 0 of the `ACK` register, after which the Output area will contain any results from the command. Packets can also be sent to or requested from the GPS module on the card. This is done by putting the GPS packet in the GPS Packet area of the Dual-Port RAM, writing the appropriate command to the Input area, and then setting bit 7 of the `ACK` register. Any incoming GPS packets are signaled by the Time Processor in two ways: bit 2 of the `ACK` register is set and the GPS Packet interrupt is signaled.

The configuration operations are written as IOCTL commands to provide access to user or kernel mode processes running at `PASSIVE_LEVEL`, though it is expected that calls will only occur from user mode. The general command interface is implemented in the driver, but at this time, only the commands for setting the timing mode and setting the propagation delay are exposed. The rest of the configuration command set can be implemented very easily, but are not needed at this time.

A small configuration utility has been written to access the IOCTLs from user-mode to facilitate testing and card configuration. It is far from a complete implementation, but it does enough to allow a user to get the card ready to use with the current system. A replacement DLL for that supplied with the card was also written to allow applications that use that old DLL interface to access the card through the new driver. The functions exported by the DLL that are not implemented in the driver simply do nothing. This allows the device capabilities to be easily extended and old software to function as well as new software written to directly access the driver.

CHAPTER 5

WINPCAP

WinPcap [6] is an open source library for packet capture and protocol analysis on Win32 platforms. It was originally adapted from the `libpcap` BSD library [18], but now provides extended functionality on the Win32 platforms. This is the protocol driver that is used by the test system to access packets from any NIC in the computer.

5.1. Structure

WinPcap includes a kernel-level packet filter named `npf.sys`, a low-level dynamic link library named `packet.dll`, and a high-level and system-independent library named `wpcap.dll`, which is based on `libpcap` [18].

The packet filter is a device driver that enables Windows 95, 98, ME, NT, 2000, XP and 2003 to capture and send raw data from a network card. It is implemented as an NDIS protocol driver.

`Packet.dll` is an API that can be used to directly access the functions of the packet driver, offering a programming interface independent of the Microsoft OS.

`Wpcap.dll` exports a set of high level capture primitives that are compatible with `libpcap`, the well known UNIX capture library. These functions allow a developer to capture packets independent of the operating system and the underlying network hardware.

5.2. Capabilities

The WinPcap library provides an interface to the network subsystem in Windows that is typically hidden from user mode applications. It allows all packets that arrive at the NIC to be captured using the NDIS before any processing is done to it by the typical protocol drivers such as TCP or UDP. This allows even packets that are corrupted to be captured before they would be thrown away when the checksum is discovered to be bad. It is possible to configure the NIC in promiscuous mode so that even packets that are not destined for the NIC can be captured. The driver in WinPcap is capable of efficiently filtering incoming packets as they are received. The filter is implemented as a compiled string that is processed by a virtual machine in the driver. The virtual machine code is then compiled to the host machine language just before execution. It is an implementation of the Berkley Network Packet Filter [21].

WinPcap extends the functionality of libpcap specifically on Win32 platforms in several ways. One, when incoming packets are captured, they are time-stamped by a precise clock that is synchronized to the RTC in the kernel as soon as it passes the filter. Another feature provided is the ability to inject packets directly into the network without going through the WinSock interface. A subset of this feature allows a program to queue packets with time-stamps and send them synchronously. All of the packet send times are relative to their time-stamps based on the time-stamp of the first packet in the queue.

5.3. Modifications

The interface provided by WinPcap is great for packet capture and protocol analysis applications, but to analyze network performance there are a few more things that are needed. Three changes were made to the transmission functionality of WinPcap to make it suitable.

The first modification required changes to the driver and the packet DLL. This modification causes the driver to time-stamp each packet as it is injected into the network. The time-stamp is stored in the packet header and is then copied back to the packet DLL where it overwrites the buffer that was passed into the send function. This allows an application to know when the packet was injected into the network. The limitation here is that the timestamp is made when the NIC driver is asked to inject the packet, and is not directly related to when it is actually injected. An improvement would be to use the OOB data in NDIS and a modified NIC driver to timestamp and return the actual time that the packet was injected. Because packet injection is an asynchronous process, it required a structural change to the protocol driver. It was necessary to cause the IOCTL that initiates the packet-queue transmission return a pending status to the protocol driver until the transmission-completion interrupt had triggered for every packet sent from the queue. It was also necessary to keep track of the packets after they were transmitted so that the timestamp returned from the NIC driver could modify the timestamp in the buffer and be returned to user mode. It was then necessary to monitor when a queue had been fully transmitted so that the IRP could be completed and control returned to user-mode.

When using the timestamp correction on an SMP computer, it is necessary to guarantee that both timestamps for the correction, meaning the timestamp in the protocol driver and the timestamp in the NIC driver, be processed by the same CPU. This is because the timestamps are acquired from the Pentium Time Stamp Counter which is unique to each processor and so they are not guaranteed to be synchronized. On the receive side, this is not a problem because the reception handler is initiated by an interrupt, which means that the computer is operating in an arbitrary thread context. The received packet it then notified up to the protocol driver in the same context. This means that the same processor must be recording the timestamps. On the sending side, the application requests a transmission so the protocol driver is executed in the context of the application. The completion interrupt, on the other hand, is executed in an arbitrary thread context, which means it could be executing on a different processor and the difference in the values read would have no meaning. This can be avoided by assigning a processor affinity to the application, which will restrict it's execution to a single processor. This con be done directly using the task manager. An affinity must also be set for the interrupts in the NIC driver so that they can be guaranteed to execute on the same processor. To accomplish this, a filter driver supplied with the Windows 2003 Resource Kit called "IntFiltr" is used. This filter driver is installed in the driver stack above the

NIC driver and provides a graphical interface to set the processor affinity for the interrupts in that driver.

The second modification was to replicate the PacketSendPackets function in the packet DLL as PacketSendPacketsRef and change the new function to allow the reference times for synchronous transmission to be specified as parameters. The reason this is important is that when sending a queue of packets synchronously with PacketSendPackets, the reference times are always the time-stamp of the first packet in the queue and the current time. If all of the packets to be sent throughout a long transmission sequence are not available at the beginning of a transmission or if the number of packets to be sent is prohibitively large, there is no way to accurately set the time that should pass between the last packet in one queue and the first packet in the next queue. By providing the reference times explicitly, that delay can be accurately implemented. This change was later extended to propagate the timing reference all the way into the driver. This way the transmission restart does not have the added delay of switching to kernel-mode after the appropriate send time is reached.

The third modification was to replicate the pcap_sendqueue_transmit function in the wpcap DLL as pcap_sendqueue_transmit_persist. The new function was then modified to retain the time-stamp of the first packet in the first queue passed to it as a reference. In subsequent calls, that same reference is used instead of the time-stamp of the first packet in the current queue. This reference in then passed to the PacketSendPacketsRef function in the packet DLL.

These changes extend WinPcap's functionality to transmit packets close to the requested time while knowing what the error in transmission time is. The reception capabilities provided by WinPcap are already sufficient and do not require modification with the exception of timestamping. It was necessary to replace the time stamping call with a call to the time processor, if available, and to modify the primary timestamp with a timestamp from the NIC driver, if present.

CHAPTER 6

GENERIC SOCKET EMULATION CLASS

Simply having access to the network interface is not sufficient for sending packets across wide area networks. Many other capabilities are required to traverse the varying topologies that are of interest. The CPCapSocket class was developed to handle these issues and provide a simple interface to send and receive packets and collect the timestamps associated with those events. It attempts to closely emulate the interface provided by the Winsock.

6.1. Supported Protocols

For the purposes of VoIP testing there are two protocols of interest that are supported by the CPCapSocket: UDP and a UDP variant that does not include the data payload in the checksum. UDP is of interest because it is the most often used transport protocol for VoIP systems. The variant is an ideal protocol because many voice codecs have the ability to handle bit errors and make use of the data that is preserved. Some even have FEC capability and can actually fix errors. If the checksum protects this data, then a bit-error in the data will cause the entire packet to be discarded. In a real-time, low-latency system like VoIP there is no time to request a retransmission without creating an unacceptable delay for the user. Therefore, retaining all possible data is far more desirable than discarding it all. It is still necessary to have a checksum protect the header, because if a bit error occurs in the header, it is possible that the packet is not even part of the stream and will corrupt the system.

6.2. Establishing a Connection

The communication channels used by this system are not technically connections, because they simply transfer datagrams between hosts. However, if there is a NAT between the two machines that must communicate, a route must be established. An association packet is used to establish that route and a return packet is sent to verify that the route is operational.

6.2.1. Selecting a Device

The first step in establishing a connection is deciding which NIC to use for the connection. As the system exists now, any device that is registered as a NIC in Windows 2000 can be used as a communication device. Unfortunately, this excludes dial-up networking connections, which are used by modems, VPN connections, and some cellular devices. This is an unfortunate limitation of the WinPcap, but may be available in the future. Choosing a NIC is done by looking in the routing tables maintained by Windows to find the default gateway device. This device is then opened with WinPcap and its MAC address and default IP address are stored as the local addresses for the device.

6.2.2. Bind a Socket in Winsock

WinPcap is not a true reception interface; it is simply a sniffing interface. Because of this, a socket on the desired local port must be allocated and bound in Winsock to prevent Winsock from sending “Destination Unreachable” ICMP packets back for every packet received. A random local port number is selected between ports 2000 and 4000 until a socket is successfully bound.

6.2.3. Create Packet Template

At this point, all of the initialization needed for the local side of the communication is complete. Because the local host is establishing the connection, the remote IP address and port are provided by the user. The only other information needed is the MAC address to send the packet to, such that the packet will reach the desired remote host. ARP is used to attempt to resolve the MAC address of the remote host from its IP address. This will only be successful if the remote host is on the LAN and is reachable by the broadcast ARP packets. In most non-trivial cases, such as those of interest in this study, this is not true. If the ARP for the remote host fails, the routing tables are used to determine the best route to the remote host. This route contains the IP address of the next hop of the route. ARP is then used again to resolve the MAC address of the next hop in the route, based on its IP address. If this does not succeed, then there is no route to the remote host and the connection will fail.

Now that a remote MAC address for the communication with the remote host has been determined, a packet template is generated that will be used for the creation of all packets. It contains such things as the MAC address, IP address, and port for the local host and remote host or next hop in the route to the remote host. The remote host IP address and the local host port number are used to create a receive filter for WinPcap so that only packets sent by the remote host and associated with this communication channel are sent to the opened interface on the local host. Finally, the association packets are exchanged to finalize the connection process.

6.3. Listening for a Connection

For a client to successfully listen for an incoming connection, it must either have an Internet routable address or some other pre-established route such that the initial association packet will reach it.

6.3.1. Selecting a Device

When listening for a connection, it can be difficult to know what interface to listen on. As the system is set up now, if the remote host IP address (or some IP address on the same network segment) is known, the device associated with the best route to that address is used. If no address is provided, a device that is associated with a remote route is selected. In the future it may be useful to allow a user to directly select which interface to listen on, but at this point that is unnecessary.

6.3.2. Bind a Socket in Winsock

Just as in the connection process, it is necessary to allocate and bind a Winsock socket to the local port to prevent Winsock from sending “Destination Unreachable” ICMP packets in response to all received packets. In this case, the local port that is bound is a fixed port number that is not random, but one requested by the user.

6.3.3. Receive an Association Packet

Before any further information about the connection can be determined, a packet from the connection host must be received. A temporary receive filter is set in WinPcap so that only packets destined for the local port that is being listened on are sent to the opened interface. When an association packet is received, the remote host’s IP address and port are garnered and stored. The packet filter in WinPcap is then updated to include the remote host’s IP address in addition to the local port. This has the effect of no longer listening for packets from any host but from then on only accepting packets from the associated host.

6.3.4. Create Packet Template

The packet template is created in the same way that it was on the connection side. It will be used to transmit all packets to the remote host.

6.3.5. Reply with Verification Packet

At this point, the communication channel should be fully functional. This is verified by sending the association packet back to the connecting host. If this packet is successfully received by the connecting host, then the bidirectional channel is established and ready to use.

6.4. Sending Packets

The CPCapSocket class provides several interfaces for sending packets. They each have their own advantages and limitations. Individual circumstances dictate the best choice of interface.

6.4.1. Creating the Packet

Before data can be sent to a remote host, it must be wrapped up in a packet. If the data is larger than the maximum allowed datagram size of 1500 bytes then it must be broken into separate packets before it is transmitted. The packet template created during the connection process is used to initialize a majority of the packet header fields. The only fields that must be generated for each packet are the various size and checksum fields.

6.4.2. Send

The most basic interface allows the immediate transmission of one packet over the open socket. Internally, this is implemented by creating a send queue, adding the packet, and sending the queue. This interface is only used to send the association packet during the connection process.

6.4.3. SendDelayed

The `sendDelayed()` interface allows for the transmission of a packet a fixed amount of time after the transmission of the previous packet. This is useful when it is desirable to specify the delay from one packet to the next without having to keep track of when the previous packet was transmitted and at what time the new packet should be transmitted. This is also implemented as a queue with a single packet added before transmission, but it makes use of the new `pcap_sendqueue_transmit_persist()` function that was added to WinPcap to allow for the transmission time to be referenced from the previous transmission time. This interface is one possibility for the transmission of test packets. It has the advantage of being able to update the status after each packet; however, it is far less efficient for sending a long sequence of packets and is far more likely to allow operating system delays to change the transmission time.

6.4.4. Queue / SendQueue / GetQueueSentTimes

This interface allows a user to queue packets for transmission at set offsets using the `queue()` function. That queue of packets can then be sent with the `sendQueue()` function. It makes use of the same `pcap_sendqueue_transmit_persist()` function that allows the next queue that is sent to still be relative to the previous queue's initial transmission time. After a queue has been successfully transmitted, the exact times at which the packets were actually transmitted can be retrieved with the `getQueueSentTimes()` function. Although the packets should be sent at the exact time specified, this is typically not the case due to the fact that the system does not run on a real-time operating system. The actual times of transmission are used to compute the jitter and latency instead of the desired transmission time so that only the effects of the network are measured.

6.5. Receiving Packets

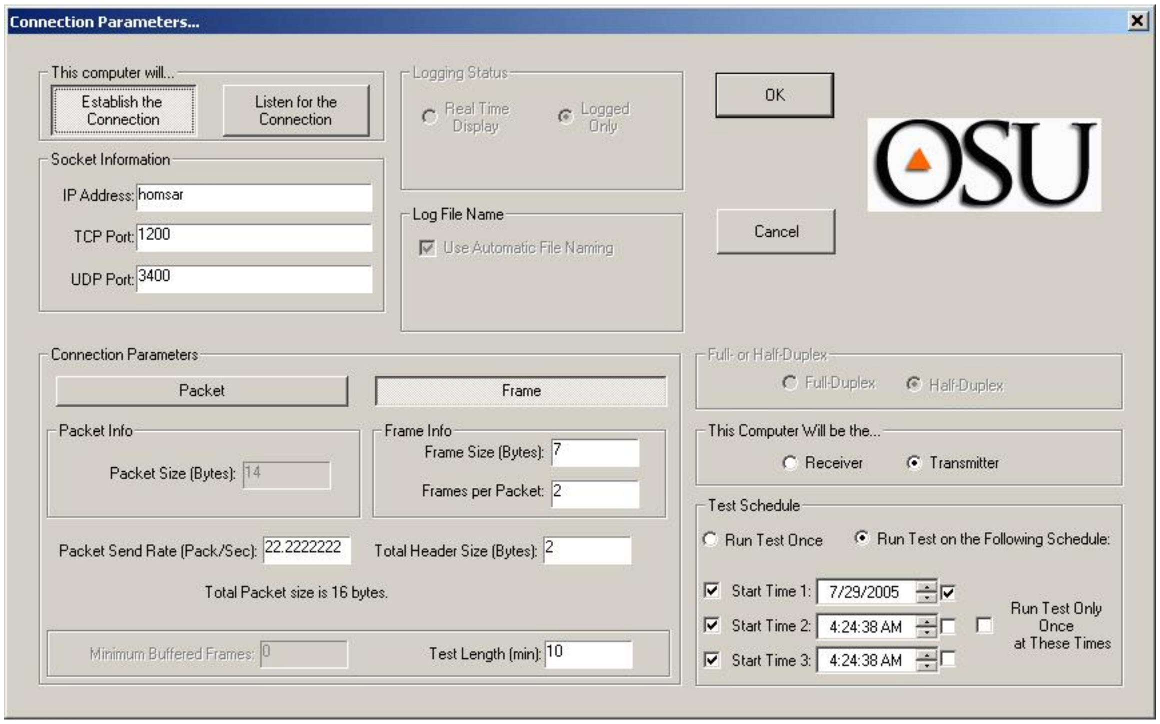
Packet reception makes use of a somewhat simpler interface. The `recv()` function simply blocks, waiting for the receipt of a packet that passes the filter that was set in the connection process. When a packet is received, the rest of the packet is validated. The checksums are verified to match before the payload is copied into the output buffer. The timestamp associated with the receipt of the packet is also passed back to the user.

CHAPTER 7

LOW BIT-RATE NETWORK JITTER AND LATENCY TESTER

The Low Bit-rate Network Jitter and Latency Tester (LoBiNeJiLaTe) is the software that makes use of all of the software described in previous chapters. Then establishing a connection, all of the desired test settings are entered in the dialog pictured in Figure 7.1. These settings are stored in the registry to that it is not necessary to reenter them if they are not to be changed. It is not necessary to open the connection parameters dialog at all. If the settings that were used the last time the application was used are still appropriate, then all that is necessary is opening the application and clicking the "Start Connection" button.

It is a multithreaded application with the basic flow shown in Figure 7.2. The control thread is responsible for managing the tests. The scheduler will delay the execution of the control thread if desired. The TCP thread is responsible for transferring the test parameters from the client to the server before a test begins and then transferring the server's packet log file back to the client after the test is completed. The UDP thread actually conducts the test by sending packets or receiving them. The file thread is used to write the packet logs asynchronously during a test. The main application thread is left to run the user interface.


Connection Parameters...

This computer will... [✓ Establish the Connection] Listen for the Connection

Socket Information
IP Address: homsar
TCP Port: 1200
UDP Port: 3400

Logging Status: Real Time Display [✓ Logged Only]

Log File Name: [✓ Use Automatic File Naming]

OK
Cancel

Connection Parameters
Packet [✓ Frame]

Packet Info
Packet Size (Bytes): 14

Frame Info
Frame Size (Bytes): 7
Frames per Packet: 2

Packet Send Rate (Pack/Sec): 22.2222222
Total Header Size (Bytes): 2

Total Packet size is 16 bytes.

Minimum Buffered Frames: 0
Test Length (min): 10

Full- or Half-Duplex: Full-Duplex [✓ Half-Duplex]

This Computer Will be the... Receiver [✓ Transmitter]

Test Schedule: Run Test Once [✓ Run Test on the Following Schedule:]

✓ Start Time 1:	7/29/2005	✓	
✓ Start Time 2:	4:24:38 AM	☐	☐ Run Test Only Once at These Times
✓ Start Time 3:	4:24:38 AM	☐	

Figure 7.1 – Test Parameters Dialog

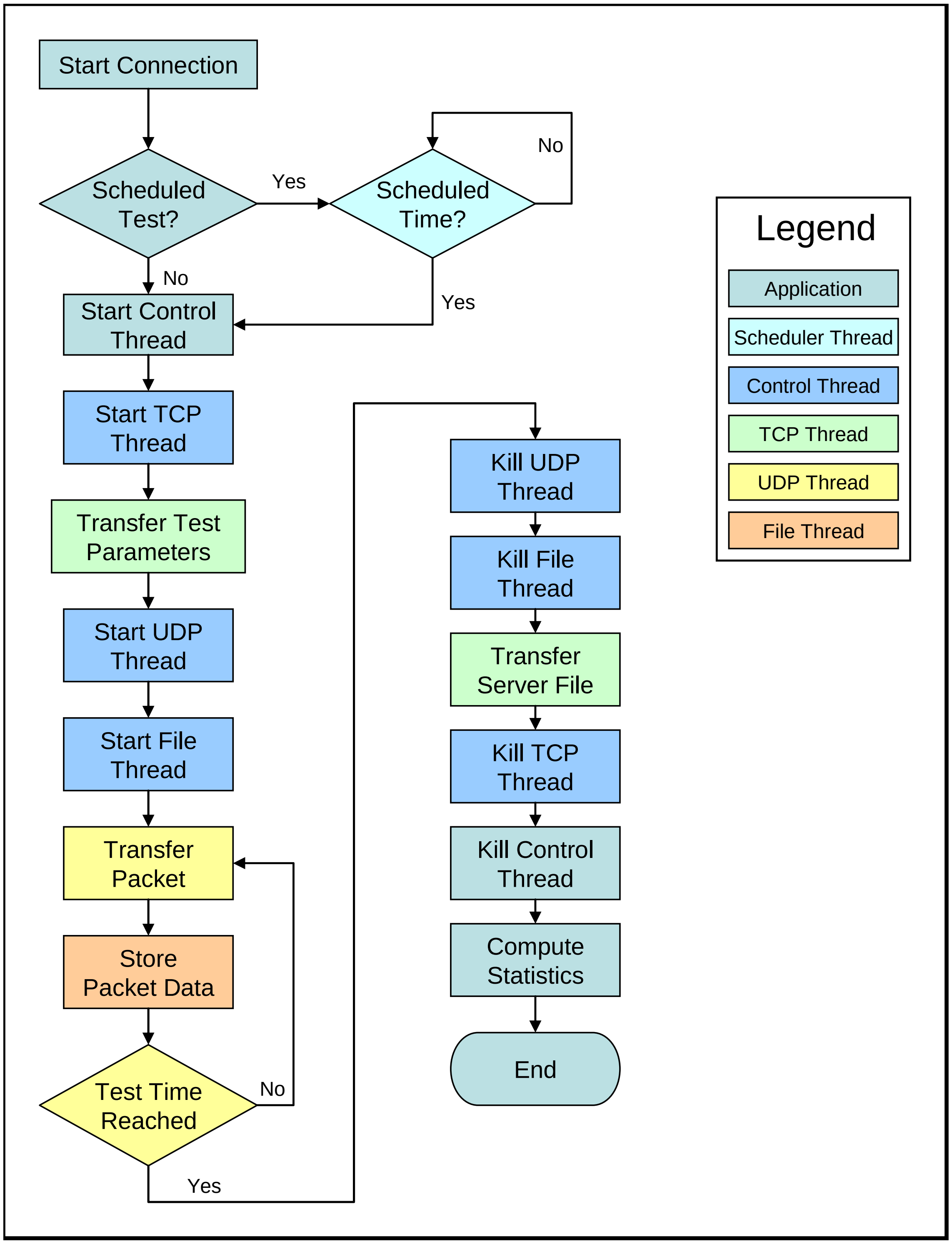


Figure 7.2 – Application Flow Chart

The networking code for the test packets is implemented with the CConnection class. This class implements Winsock code to create and use TCP and UDP connections in addition to containing the CPCapSocket class discussed in the last chapter. This greatly simplifies the code required to run a test. This means that is it possible to use the same object to communicate via Winsock or the WinPcap interface.

The statistics are implemented in a simple, direct way such that modifying them and adding to them should be straightforward. Four measurements are computed for each test: Jitter, Latency, Inter-Transmission Time, and Inter-Arrival Time. The minimum, maximum, average, variance, median, and inter-quartile range are computed for each measurement as discussed in Chapter 8.

CHAPTER 8

STATISTICAL ANALYSIS TOOLS

The results of the network tests are analyzed using three different classes of statistical tools. The first is classical statistics: the minimum, maximum, mean, and standard deviation. The second is robust statistics: the median and the inter-quartile range. The final is the Probability Density Function (PDF) in the form of a histogram. Each of these measures is applied to the four metrics at the end of the test: the inter-transmission time, inter-reception time, end-to-end latency, and jitter.

8.1. Classical Statistics

The minimum, maximum, mean, and standard deviation paint a good picture for the overall properties of a dataset. This can be very informative if the data is consistent, however, if the data contains outliers, these functions can give misleading results as they are dominated by the outliers.

8.2. Robust Statistics

Robust statistics are highly immune to outliers. They are often used when data is polluted with impulsive noise or otherwise abnormal points.

8.2.1. Median

The median is a robust substitute for the mean. It measures the 50th percentile of a series. When the number of points in the series is odd, the median is the center point after the data is sorted. When even, it is the mean of the two center points.

8.2.2. Inter-Quartile Range

The inter-quartile range, also known as the central variation, is a robust substitute for the standard deviation. It gives a sense of the variation in the main body of the data, excluding outliers. It is the difference between the 25th and the 75th percentile. The IQR of a series can be directly compared to the variance of that series if the IQR is multiplied by 0.7413.

8.3. Probability Density Function

The probability density function, in the form of a histogram, is useful for determining how the data is distributed which is not obvious from the times series. The x-axis is the sequence of bins for values found in the time series and the y-axis is the number of times a measurement was within one of those bins. Figure 8.1 shows an example of a histogram and its associated time series. The large spike at just over 25ms shows up as a small mark at 25ms on the histogram. The majority of the data is around 11.5ms and as such, the largest spike in the histogram is at about 11.5ms.

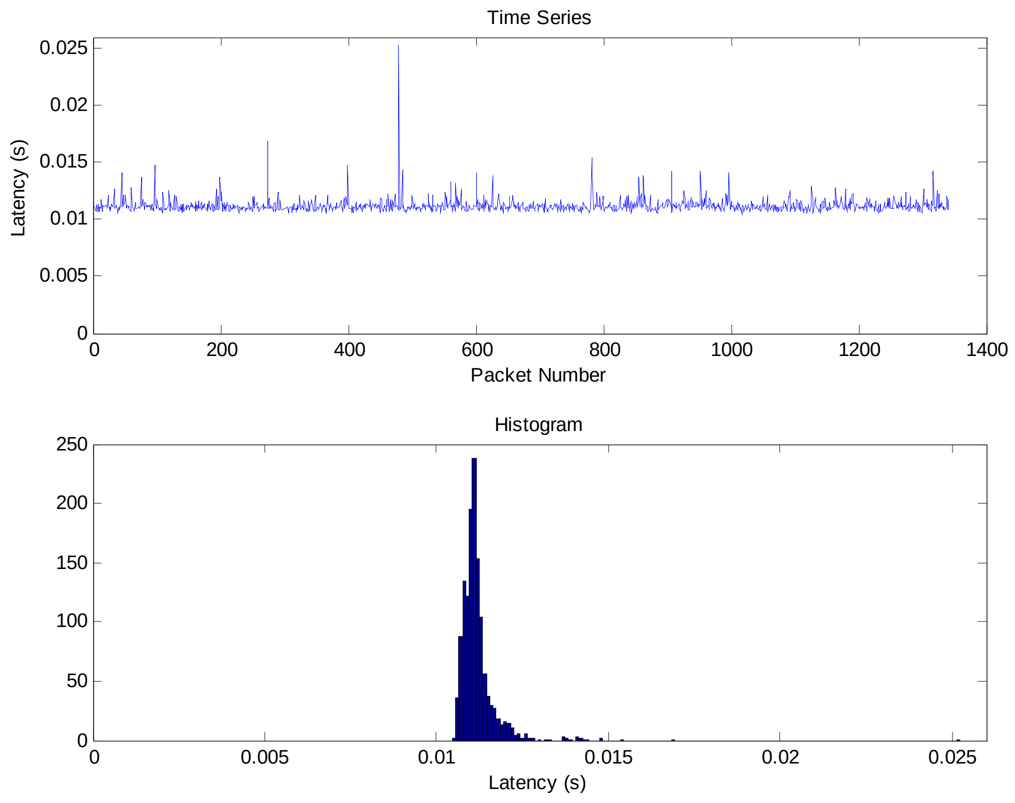


Figure 8.1 – A sample Histogram and its Associated Time Series

CHAPTER 9

RESULTS

All of the components described in the previous chapters combine to create a test system capable of accurately and precisely measuring the jitter and latency of a network connection in the context of low-bit-rate VoIP streams. The resulting system is detailed in Figure 9.1. The system, having precise and accurate timing, allows versatile network testing that utilizes a wide variety of NICs. If the Intel EtherExpress Pro/100+ NIC is used when applicable, it is possible to achieve more accurate results. The NIC can also be used to get a general idea of the magnitude of the error incurred by the NDIS interface and the interrupt processing.

9.1. Final System

The final system provides several options for testing networks depending on the required flexibility versus required accuracy. For maximum flexibility, if the modified version of WinPcap does not detect a time processing card, it will automatically use the Pentium Time Stamp Counter instead. This will allow measurements of jitter with reasonable precision. However, without a synchronized clock the latency calculations will be meaningless. Advanced software clock synchronization could possibly be used to get somewhat accurate timing in the future to produce meaningful latency data for low bandwidth links such as cellular data when the time processor is not available.

When the time processor is available, all packet timestamps are applied in the protocol driver, `npf.sys`, at the earliest and latest times possible for reception and transmission, respectively. This produces results sufficient to measure all but the fastest of measurements without the error becoming noticeable. This mode is used most frequently because the protocol driver is abstracted away from the hardware by the operating system and any device that appears as a NIC in the PC can be used in measurements.

If further accuracy and precision are required and it is possible to use 10 or 100Mbit Ethernet, then the Intel EtherExpress Pro/100+ NIC can be used. The driver for this card is modified to record a timestamp as soon as the hardware interrupts the PC to notify it of a newly received packet or a transmission that has successfully completed. Because the Pentium Time Stamp Counter has low call overhead and high precision, it is used to generate a differential timestamp to correct the GPS timestamp for call placement. This has the effect of recording the time in the interrupt of the NIC without slowing down the interrupt with a call to the timer processor.

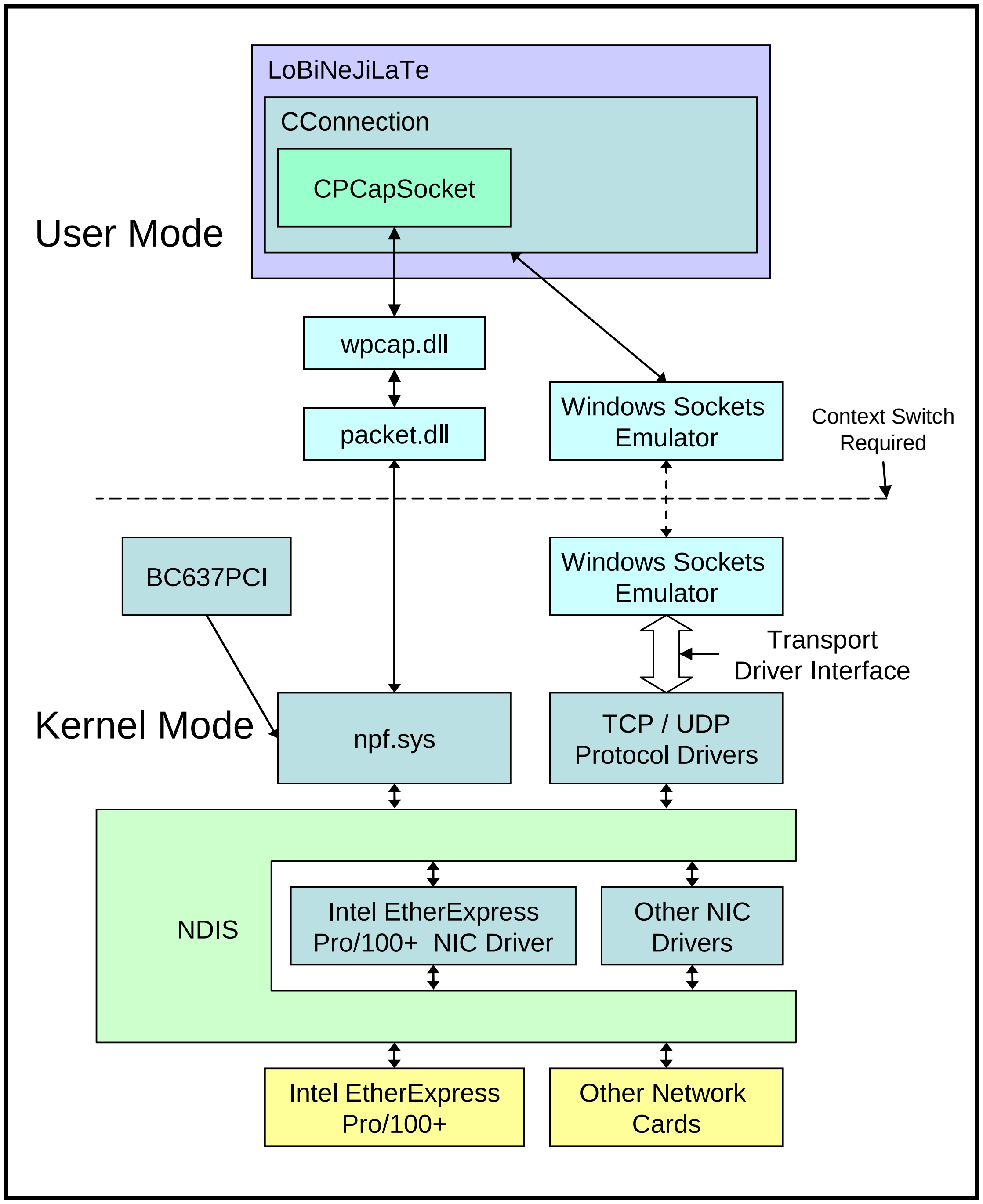


Figure 9.1 – Layout of the Network Testing Application

9.2. System Validation

Though no hardware timing solution, such as a DAG card, is available, we will attempt to show that the results of the network performance tests are sensible, or at least not obviously grossly incorrect. Two issues will be addressed: the host-time to wire-time discrepancies and network latency scaling. In all of the tests, the transmitter is running at real-time priority in an attempt to make the transmissions as accurate as possible. All of the packets that are transmitted are queued up 60 seconds at a time. This allows the transmitter to remain in kernel space for 60 seconds at a time, which greatly reduces context switches and improves accuracy. This section will attempt to show the approximate error in the system. As Vern Paxson states in Strategies for Sound Internet Measurement [33], it is important to state how accurate a test is when reporting its results.

9.2.1. Timing-Location Error

Though it is not possible to fully measure the wire-time to host-time error without a DAG card, a fairly good approximation can be made using the Intel EtherExpress Pro/100+ NIC (e100b) with the modified driver. It modifies the timestamps generated in the protocol driver such that they represent the time that the NIC notifies the computer that it has received or started transmitting a packet. This means that the only sources of error are the delay in the hardware itself and any interrupt processing delay caused by other frequently interrupting devices.

When transmitting packets, the e100b provides an interrupt when the packet data has been successfully copied out of memory and into the NIC's controller. This eliminates the delays in NDIS queuing the time required to copy the data. The correction of this transmission time was tested on two computers: one having two processors and one having only one.

The delay introduced between a packet notification and its processing in a protocol driver is dependent upon the PC on which the tests are run. An SMP machine can much more efficiently process interrupts and as such, the performance is much better meaning that the packets are processed in the protocol driver relatively close to the wire-time. In a single-processor machine, the CPU must handle all aspects of the system and will not have near the performance of an SMP machine for time critical operations because it will often be busy with system operations while a time critical function is pending.

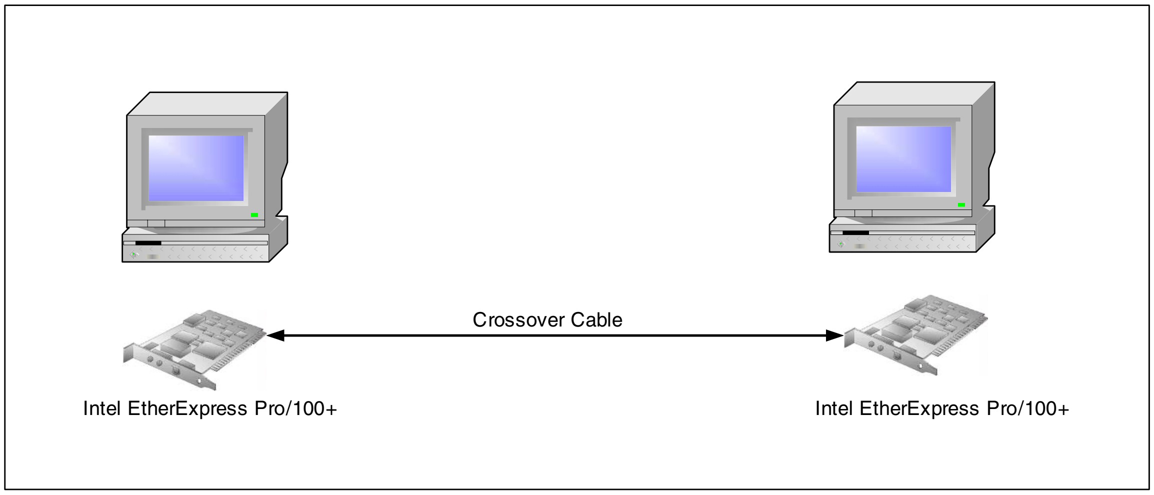


Figure 9.2 – System Validation Test Configuration

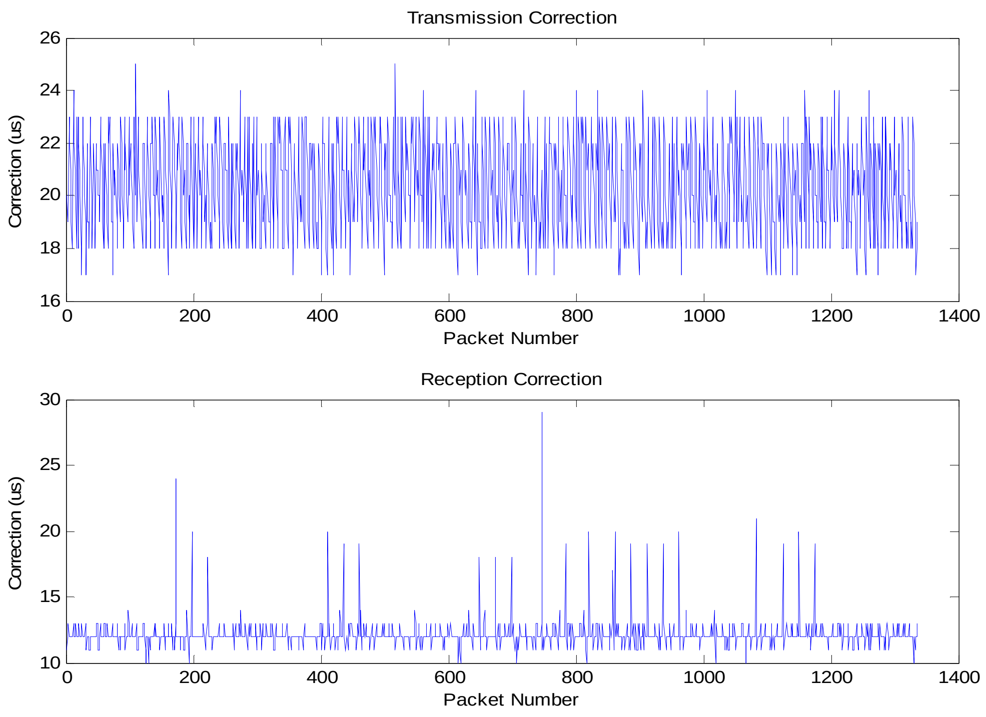


Figure 9.3 – 100Mb/s Dual-Processor Sourced NIC to Protocol Corrections

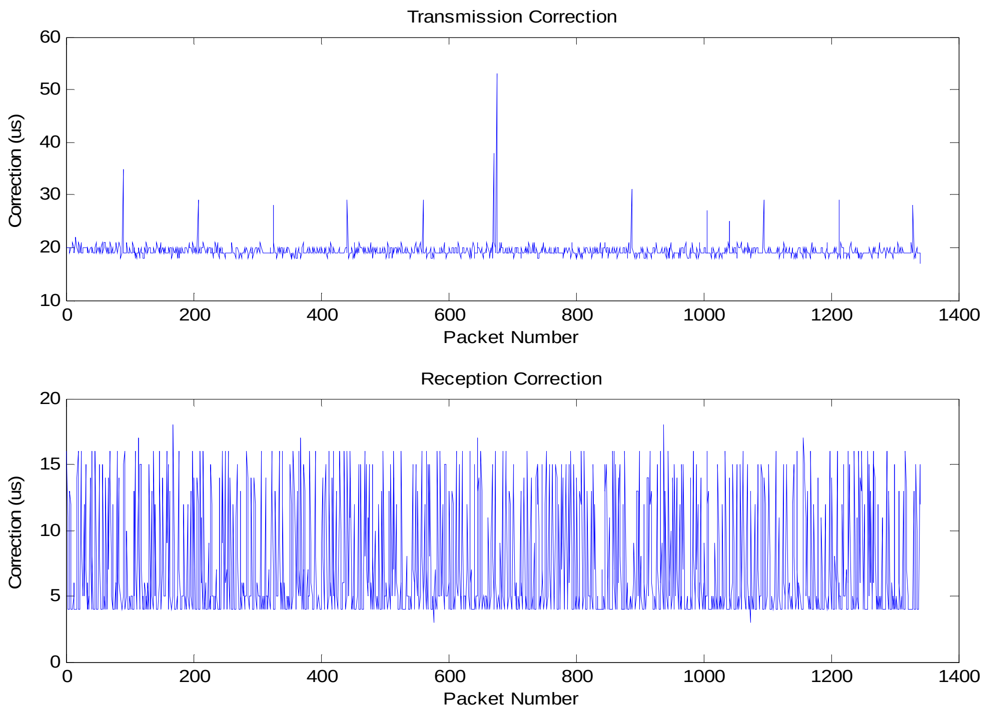


Figure 9.4 – 100Mb/s Single-Processor Sourced NIC to Protocol Corrections

On the dual-processor machine, the transmission error was relatively white with no structure to speak of. The average delay is 20 µs. The single processor machine showed delay that was much more structured. This can be attributed to system interference with the processing of the transmission request. There seem to be many instances in which another regularly timed process on the computer is apparently interfering with the testing. It is more prevalent on single processor machines, presumably because the system shares the processor with the testing software all of the time. This supports the use of a real-time operating system on such hardware, which would greatly reduce these effects. The average delay on the single-processor machine is 21 µs.

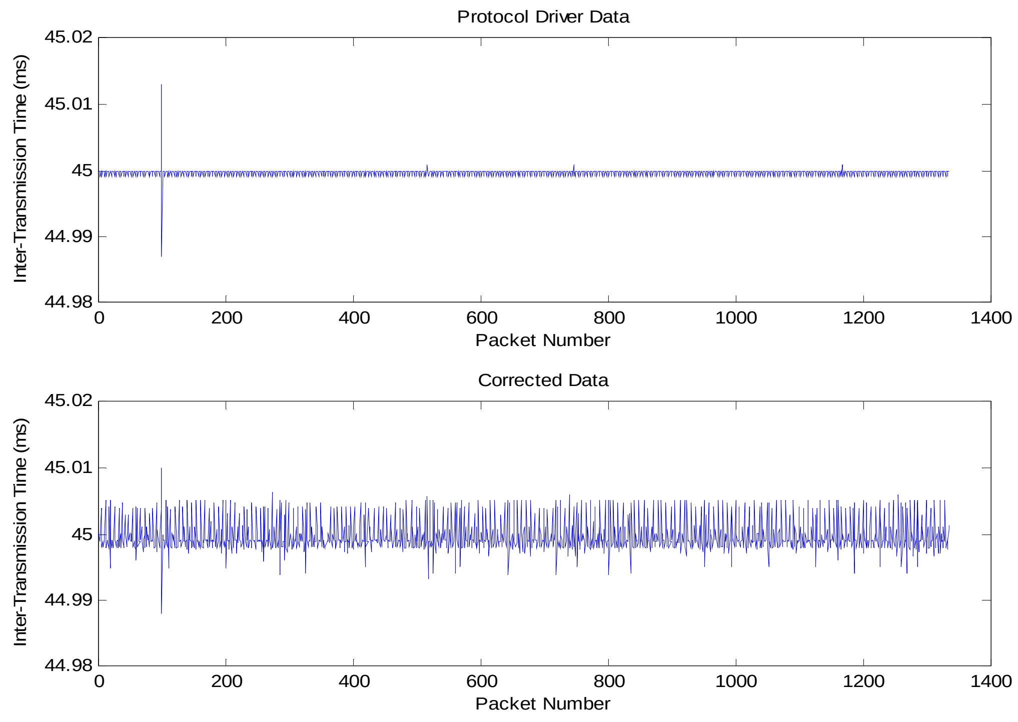


Figure 9.5 – 100Mb/s Dual-Processor Sourced Inter-Transmission Time

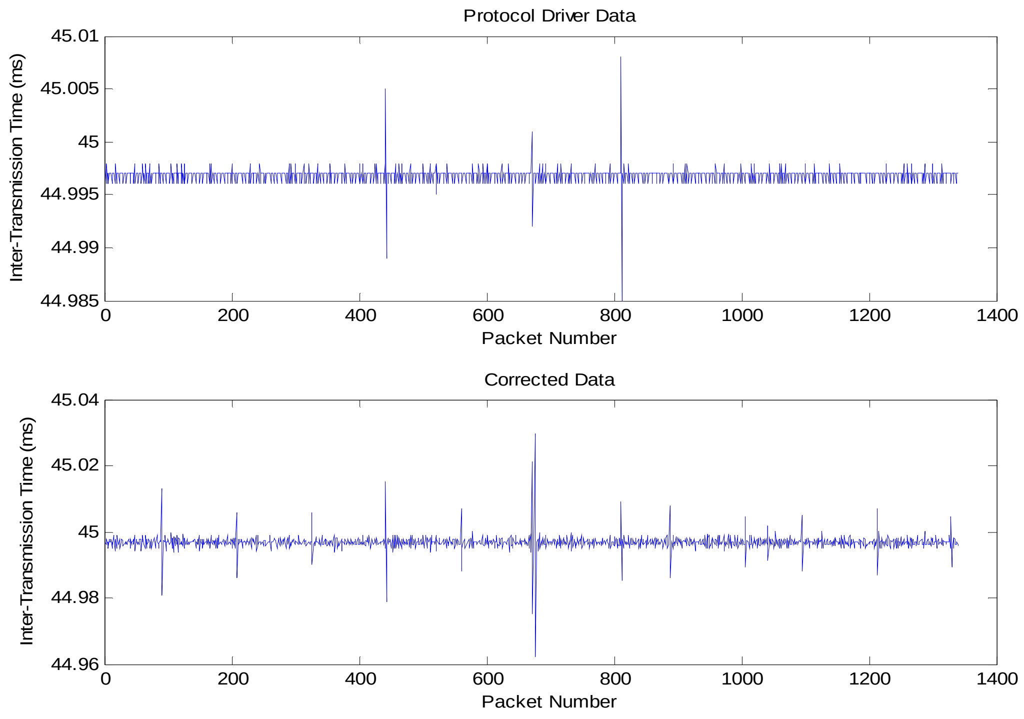


Figure 9.6 – 100Mb/s Single-Processor Sourced Inter-Transmission Time

As a result of these adjustments, the timestamp data is less perfect than the original timestamps from the protocol driver, which should be expected. It is not reasonable to expect that every packet is transmitted within 1 µs of the desired time. This appears to be the case based on the fact that without the network driver correction, there are few options for variation. The timestamping in the protocol driver is a small, fixed number of instructions away from the code which waits for the correct time to schedule the transmissions. This means that the only transmission error the clock will measure is the amount of time that the scheduling clock delayed beyond the scheduled time. The scheduler was not converted to use the time processor due to the fact that scheduling performance was already sufficient. This means the timestamp measurement would measure any error in the scheduler since they are not based on the same clock.

For these tests the inter-reception time should be the same as the inter-transmission time, 45ms, because the network is simply a crossover cable and should not produce any additional delays. Figures 9.7 and 9.8 show that the corrections significantly reduce the deviations from 45ms. This depends on the transmission, though, because if the packets were not sent at exactly 45ms intervals, then the reception of the packets will not approach 45ms intervals.

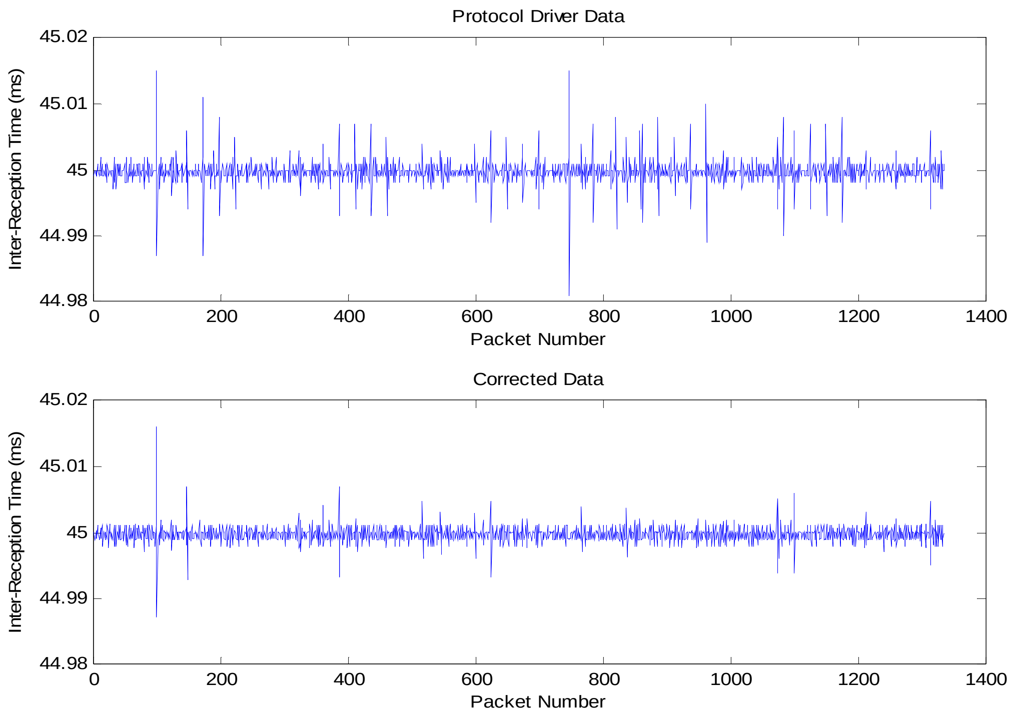


Figure 9.7 – 100Mb/s Dual-Processor Sourced Inter-Reception Time

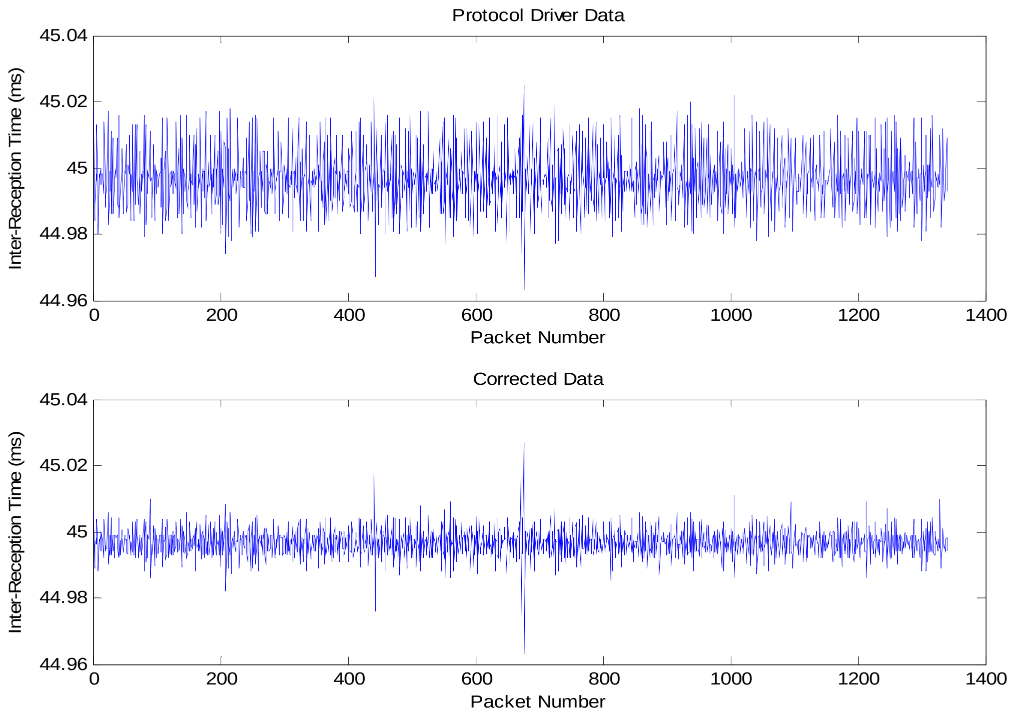


Figure 9.8 – 100Mb/s Single-Processor Sourced Inter-Reception Time

It is not really possible to validate the packet reception or transmission timing capabilities independently. The received packets are sent from an imperfect transmitter so both are needed to compute the effects of the network. Without a good reference for the packets received and it is not possible to distinguish whether the error is in the reception timing or in the transmission timing. If the packets are transmitted at a time other than the time they are believed to be transmitted, then it is correct for the receiver to see deviation from the specified transmission rate. Because the reception test is run between two e100b cards at 100Mbits per second over a cross-over cable, it should not be possible for the network to inject any error other than a small, constant propagation delay. Anything other than this seen in the latency plots must be due to errors in the measurement system.

After correcting the jitter calculations, the errors are less than 10μs for both the single-processor and dual-processor tests. Because of the low level of jitter, the latency is consistent. With consistent latency in these validation tests, the system is shown to provide precise measurements of network effects on packet transfer. This means that any jitter revealed in network tests will reveal jitter that is actually caused by the network and not by the test system, above the margin of error in these validation tests.

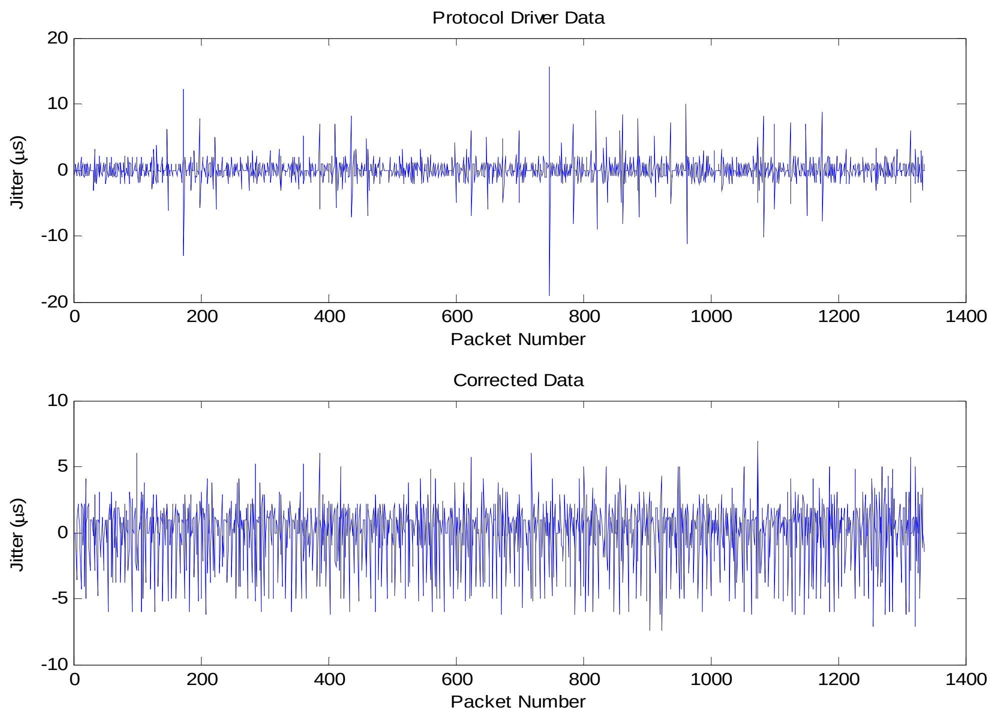


Figure 9.9 – 100Mb/s Dual-Processor Sourced Jitter

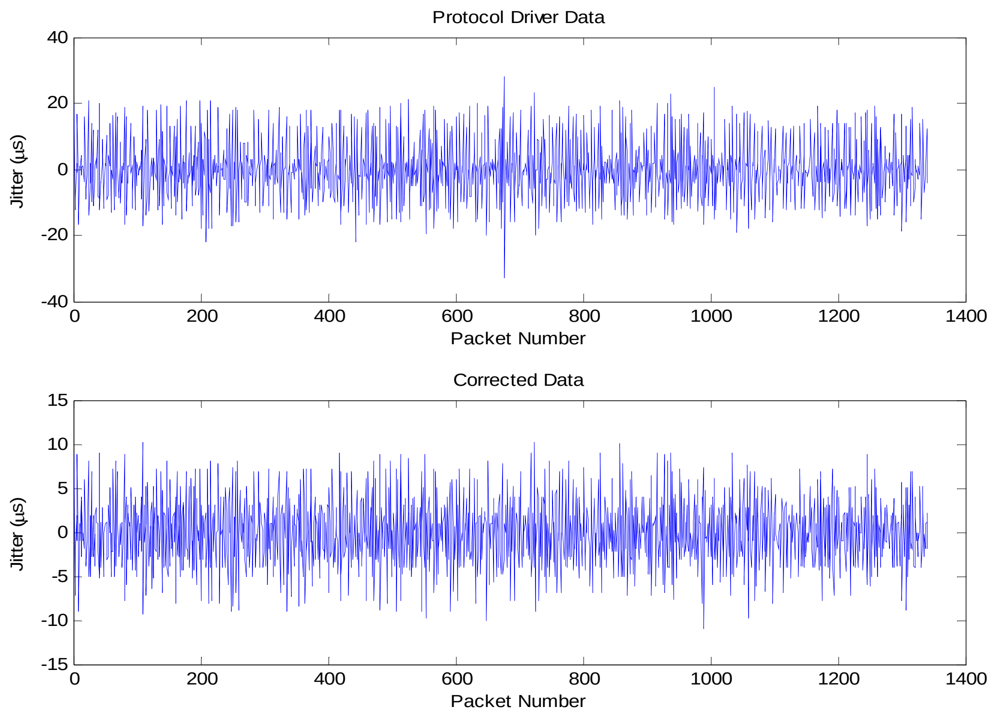


Figure 9.10 – 100Mb/s Single-Processor Sourced Jitter

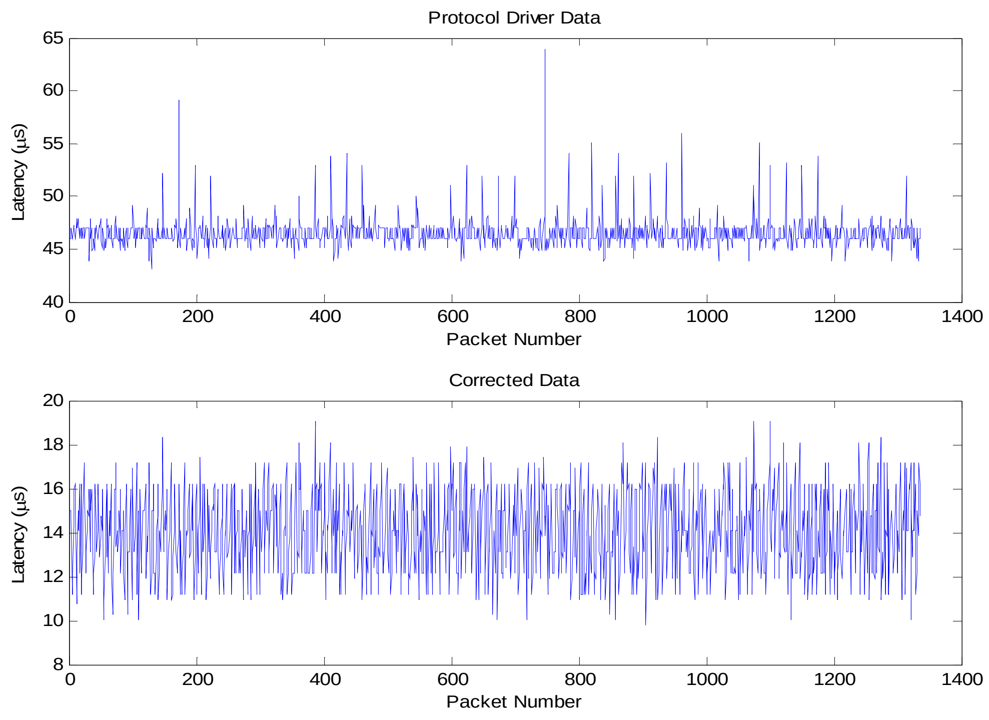


Figure 9.11 – 100Mb/s Dual-Processor Sourced Latency

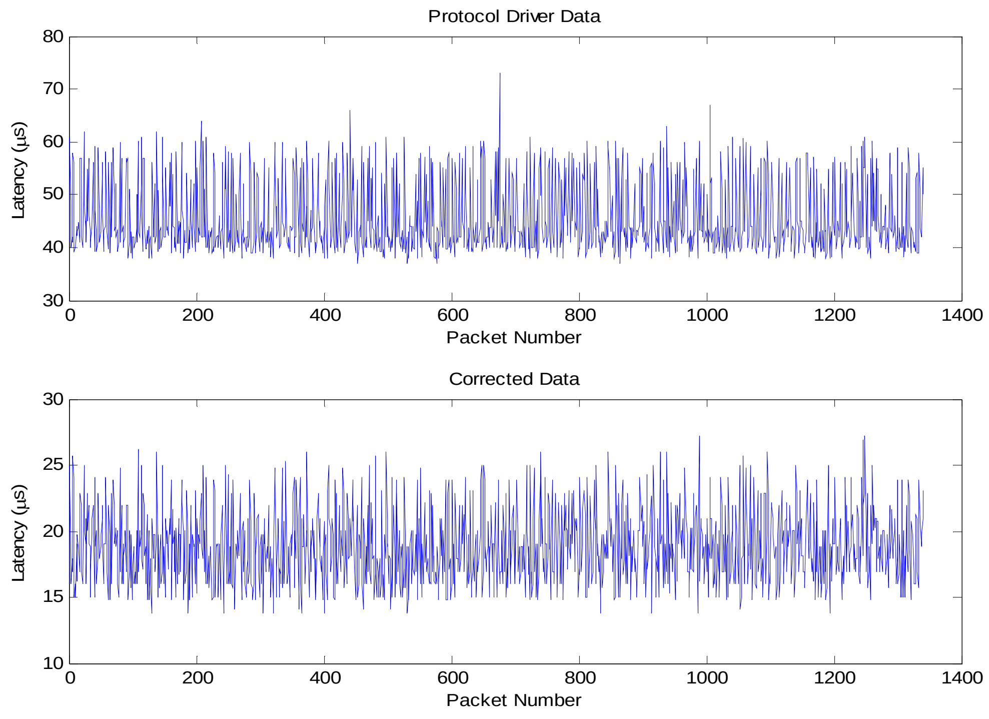


Figure 9.12 – 100Mb/s Single-Processor Sourced Latency

9.2.2. Network-Latency Scaling

The validation of jitter measurements is more straightforward than latency because they are differential. Directly validating latency requires a known reference, such as a DAG card. Without that option being available, the latency measurements can be indirectly validated by comparing the measurements made at different link speeds. The e100b cards were configured to communicate at 10Mbits per second instead of 100Mbits per second. The resulting latencies were then compared with the tests in Section 9.2.1 to evaluate the measurement error associated with the single-processor source and the dual-processor source systems.

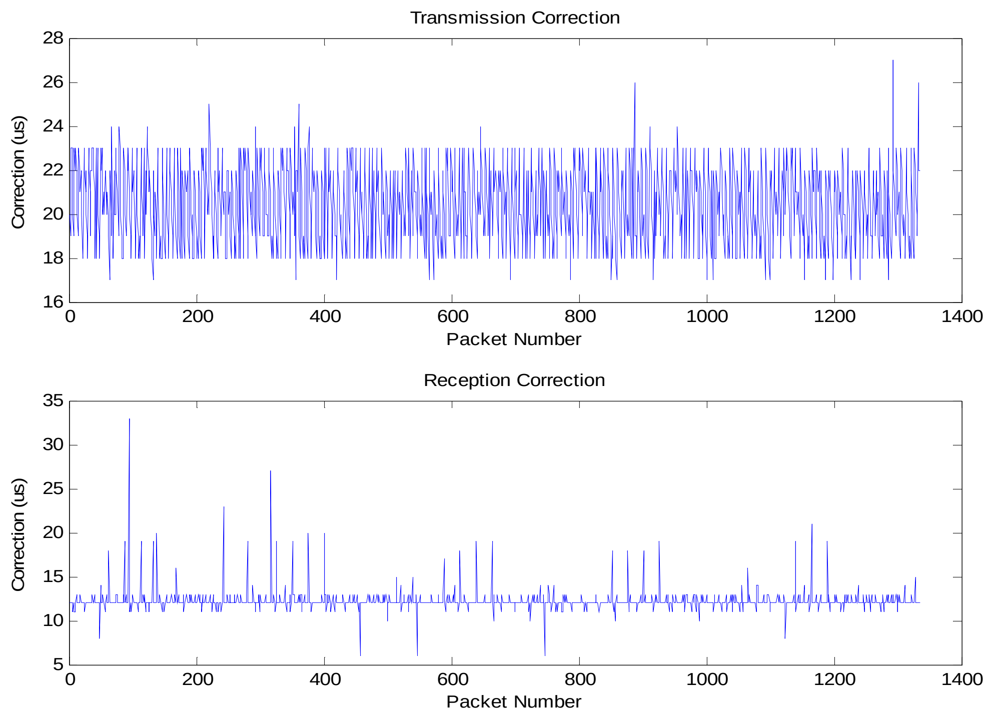


Figure 9.13 – 10Mb/s Dual-Processor Sourced NIC to Protocol Corrections

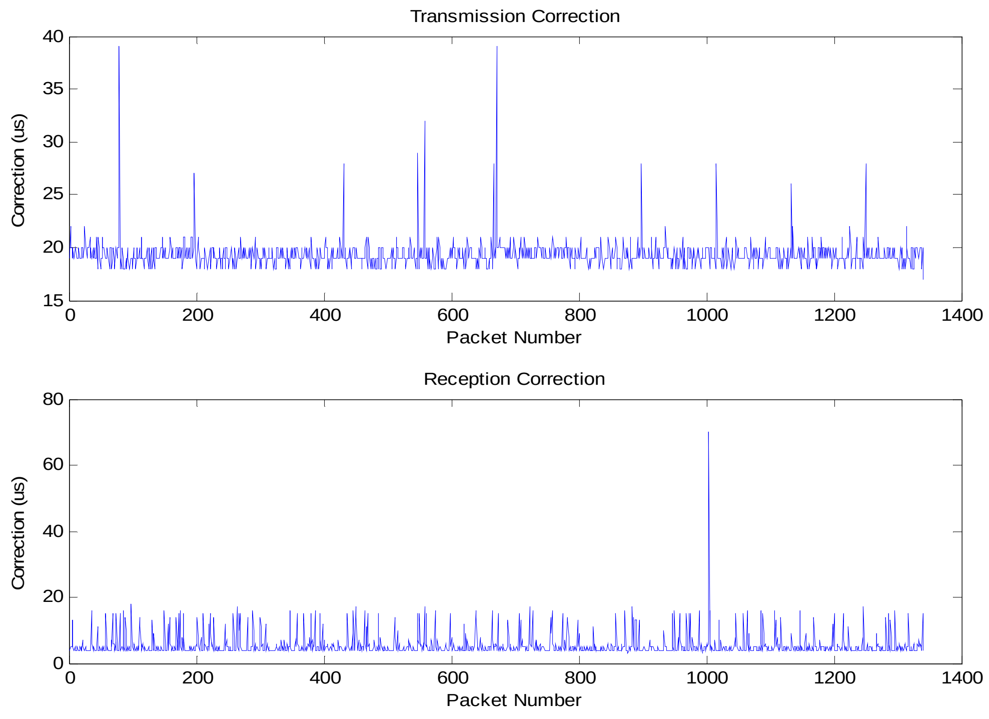


Figure 9.14 – 10Mb/s Single-Processor Sourced NIC to Protocol Corrections

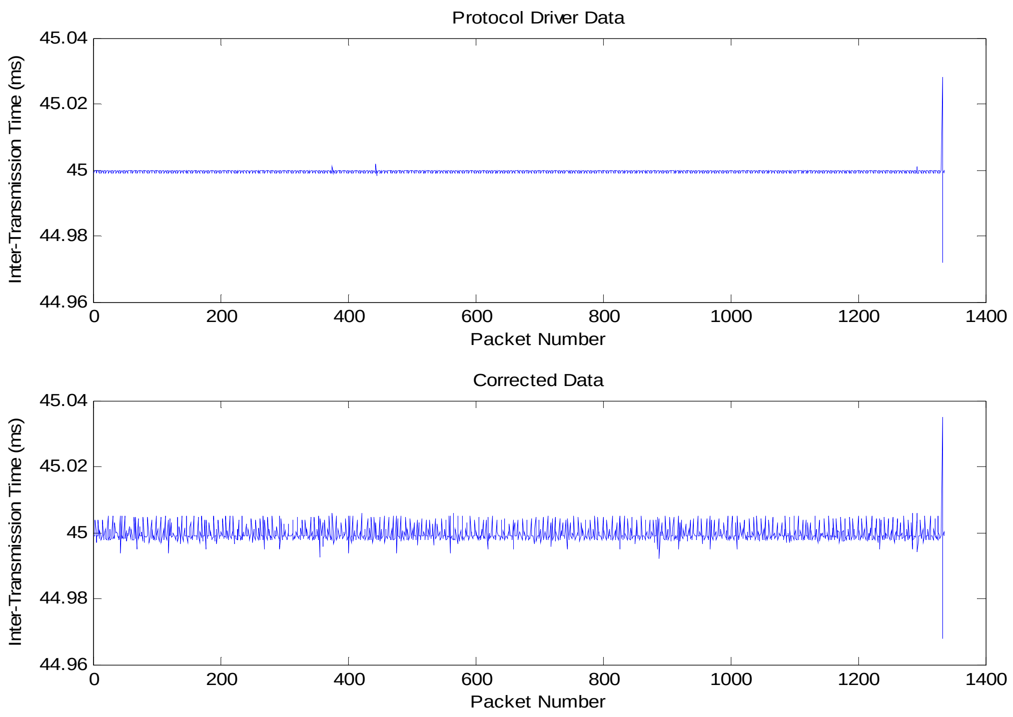


Figure 9.15 – 10Mb/s Dual-Processor Sourced Inter-Transmission Times

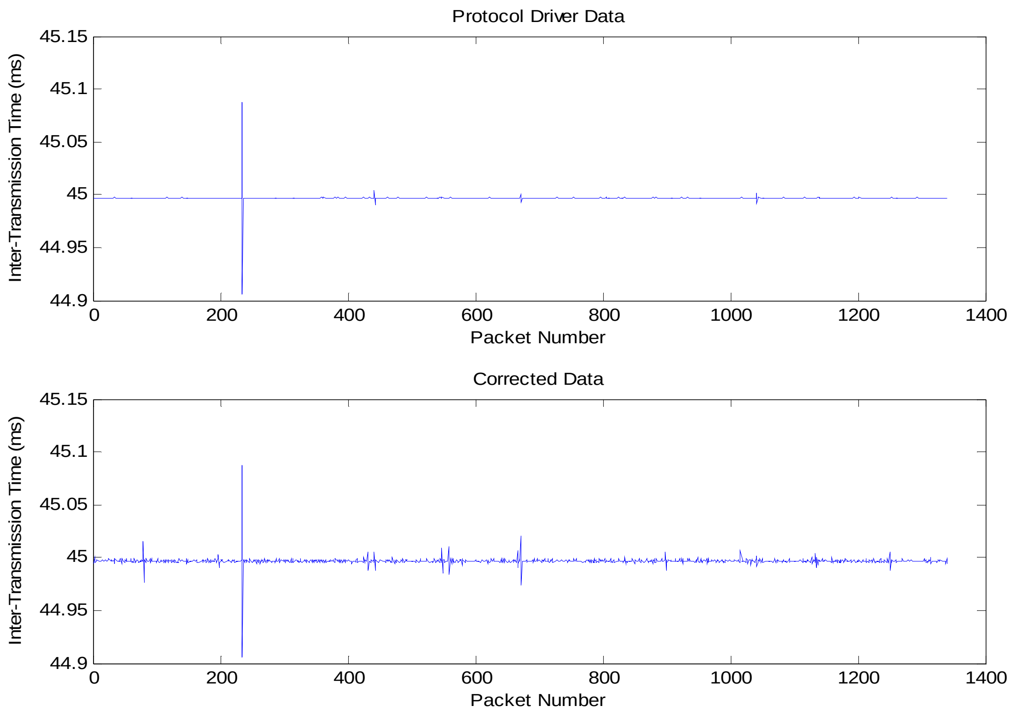


Figure 9.16 – 10Mb/s Single-Processor Sourced Inter-Transmission Times

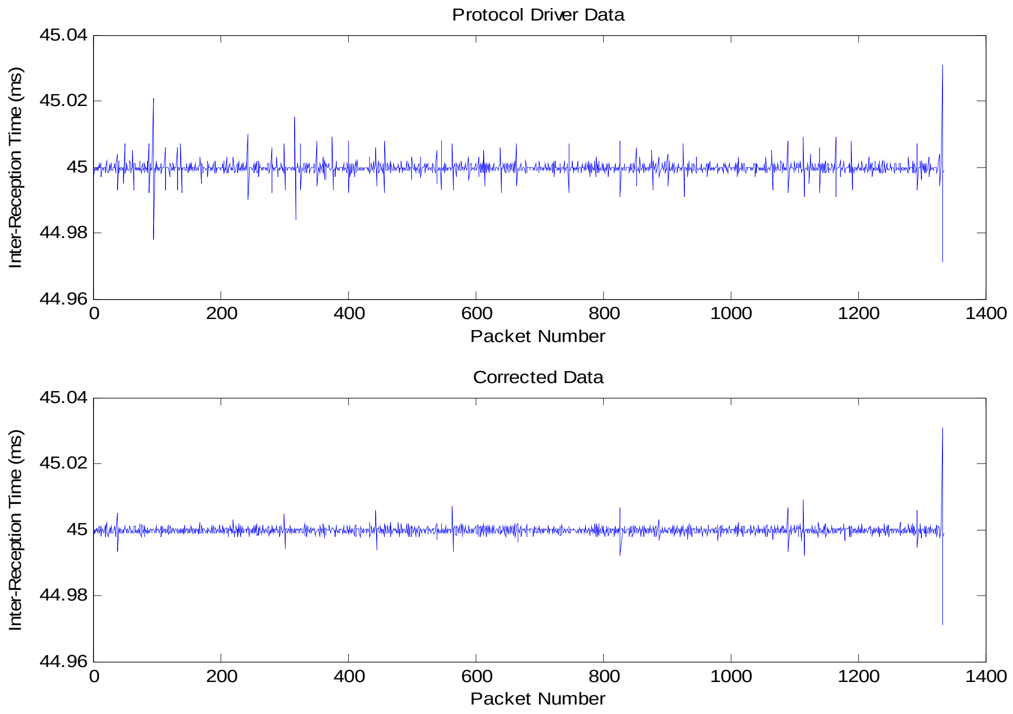


Figure 9.17 – 10Mb/s Dual-Processor Sourced Inter-Reception Times

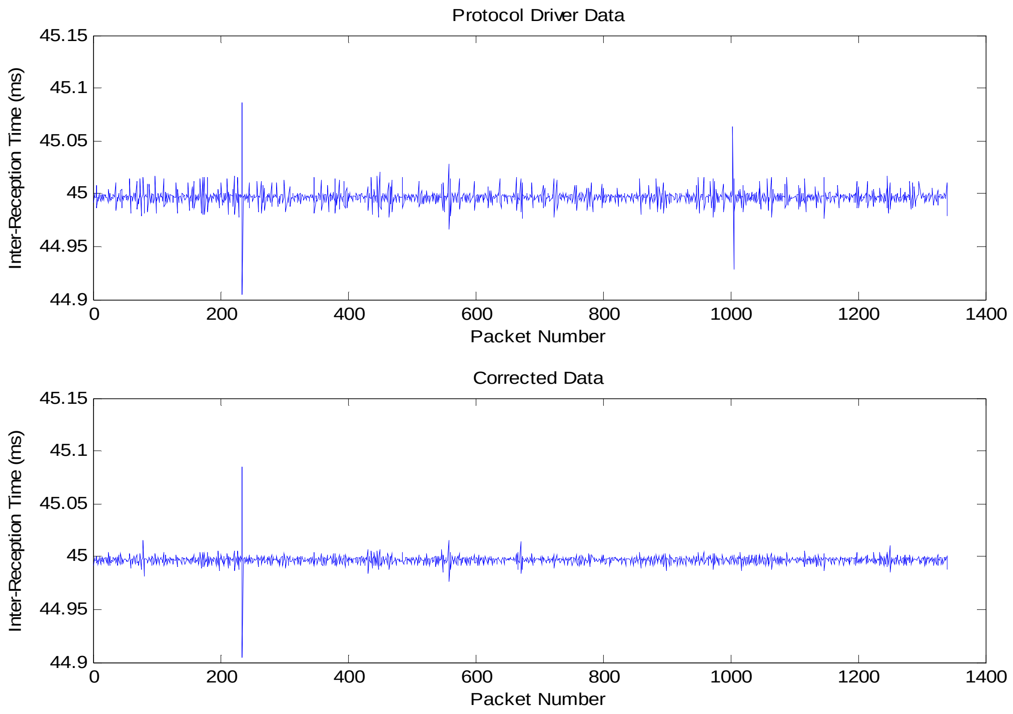


Figure 9.18 – 10Mb/s Single-Processor Sourced Inter-Reception Times

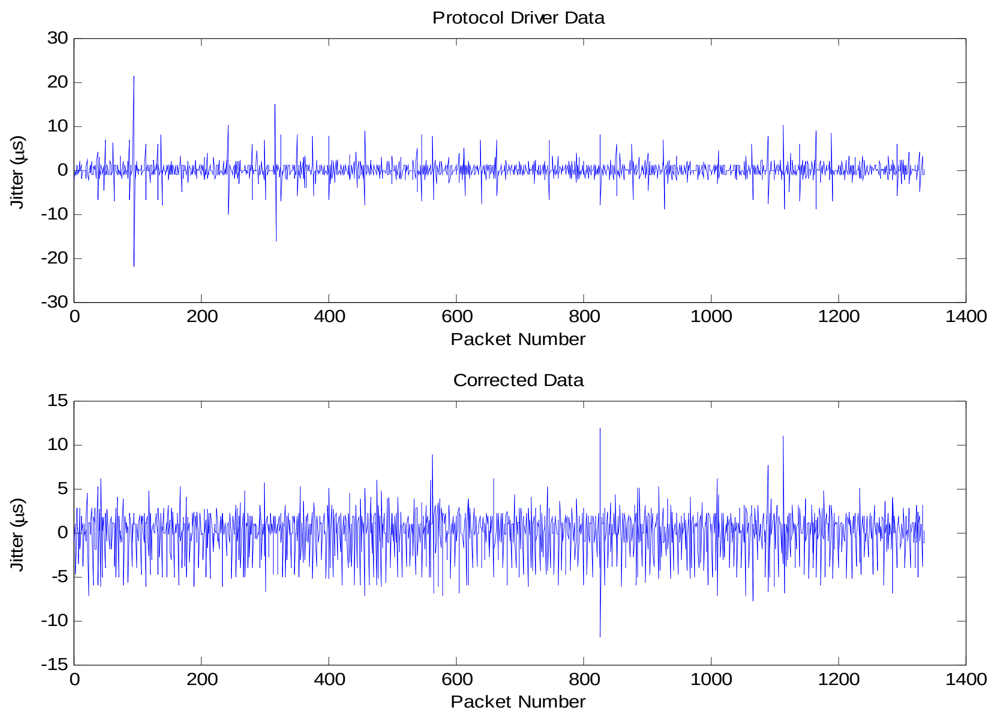


Figure 9.19 – 10Mb/s Dual-Processor Sourced Jitter

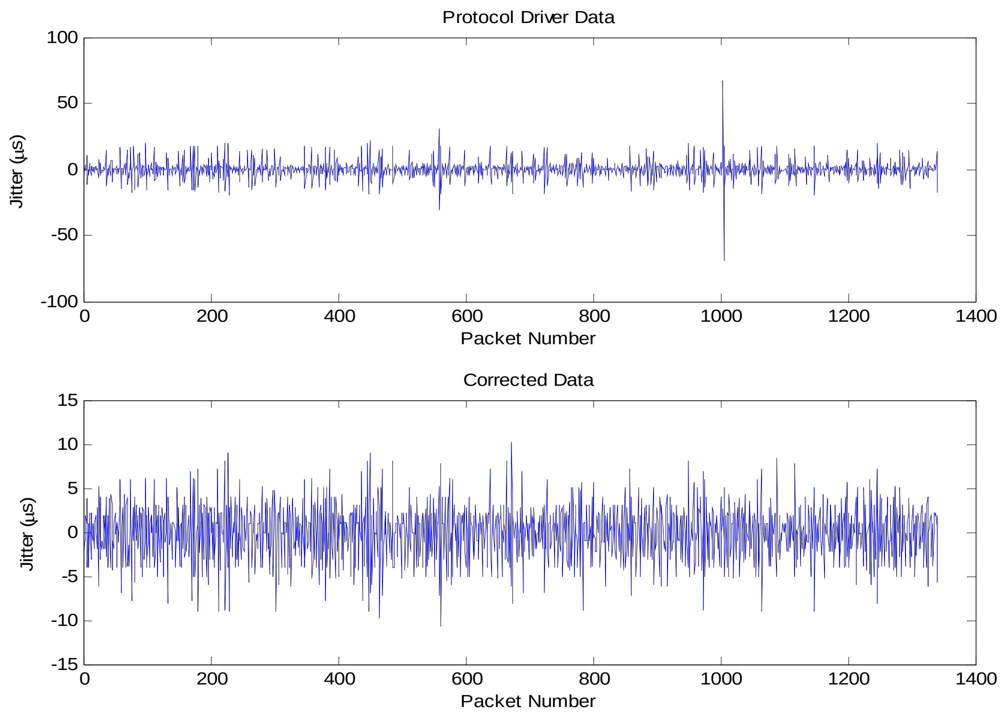


Figure 9.20 – 10Mb/s Single-Processor Sourced Jitter

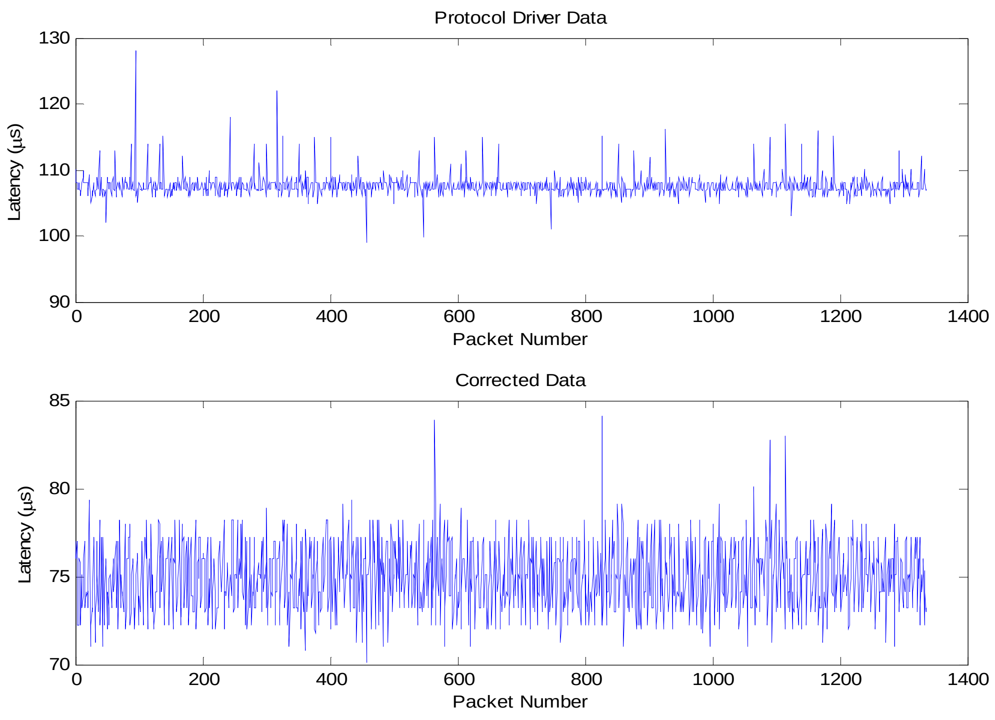


Figure 9.21 – 10Mb/s Dual-Processor Sourced Latency

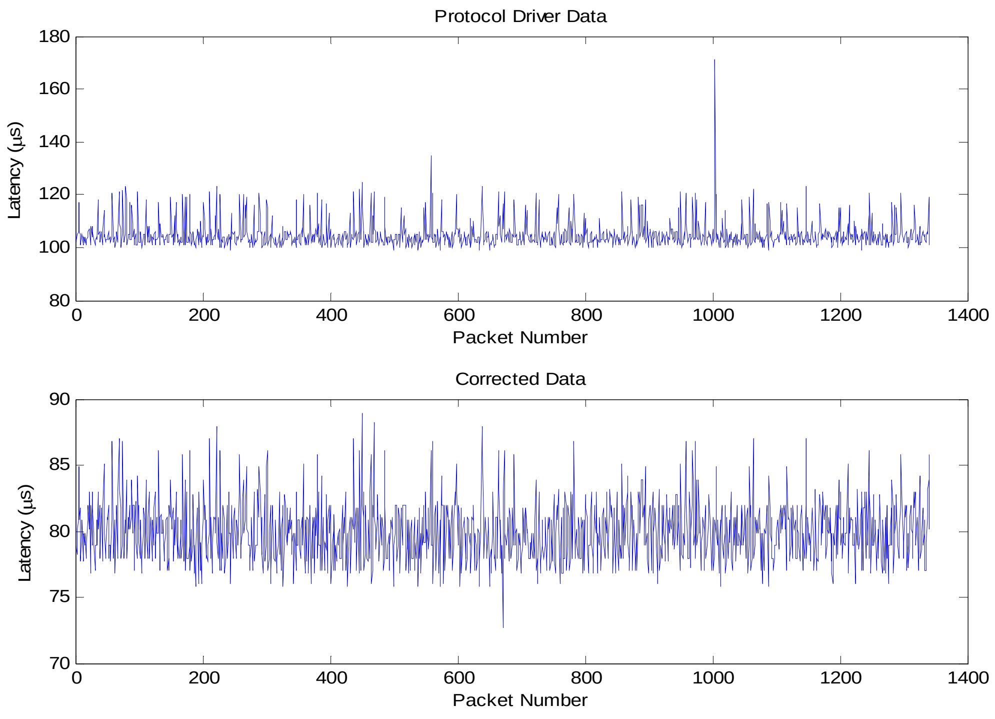


Figure 9.22 – 10Mb/s Single-Processor Sourced Latency

The NIC to protocol driver corrections in Figures 9.13 and 9.14 almost identically match Figures 9.3 and 9.4. This means that regardless of the speed of the network connection, the same delays are incurred by the communication between the protocol driver and the NIC. To compute the remaining error, that introduced by the NIC hardware and interrupt processing latency, it is possible to compare the latencies of two tests whose only variation is the link speed. The error should be independent of the link speed, while the true latency should be directly affected by the link speed. In the case of the dual-processor tests, the latency at 10Mb/s was 75µs and at 100Mb/s was 14µs. Because the link speeds are known, the theoretical latencies can be computed. They are dominated by the transmission rate (800ns/byte at 10Mb/s and 80ns/byte at 100Mb/s), because a short cable was used causing the propagation delay to be approximately 5ns. This can be described as a system of two linear equations with two unknowns. The unknowns are the number of bytes transmitted and the amount of erroneous delay, which is independent of the link speed.

$$0.8 \cdot n + e = 75 \tag{9.1}$$

$$0.08 \cdot n + e = 14 \tag{9.2}$$

The solution to these equations is n = 85 bytes and e = 7µs. The same process can be used for the single-processor tests which yield n = 85 bytes and e = 12µs. These errors in general can be added to the NIC driver to Protocol driver errors found in Section 9.2.1 to find the expected constant error for the measurements. This agreement in n suggests that the link-based delay is constant regardless of machine setup. The actual data transmitted on the wire is 72 bytes, made up of a 7-byte preamble, a 1-byte start of frame delimiter, and the 64-byte payload. The additional 13 bytes of time computed above are assumed to be the time required by the collision detection mechanism before the start of the preamble.

9.2.3. Hardware Based Error

The hardware can cause timing errors if it does not notify the computer after each packet it receives. Some NICs are designed to hold packets in a buffer and notify the PC of multiple packets with a single interrupt to reduce the card's interrupt frequency during heavy-load traffic conditions. In Figure 9.23, two packets near the end of the test were processed by the same interrupt. This is a rare occurrence; it was only encountered once in all of the testing conducted for this study.

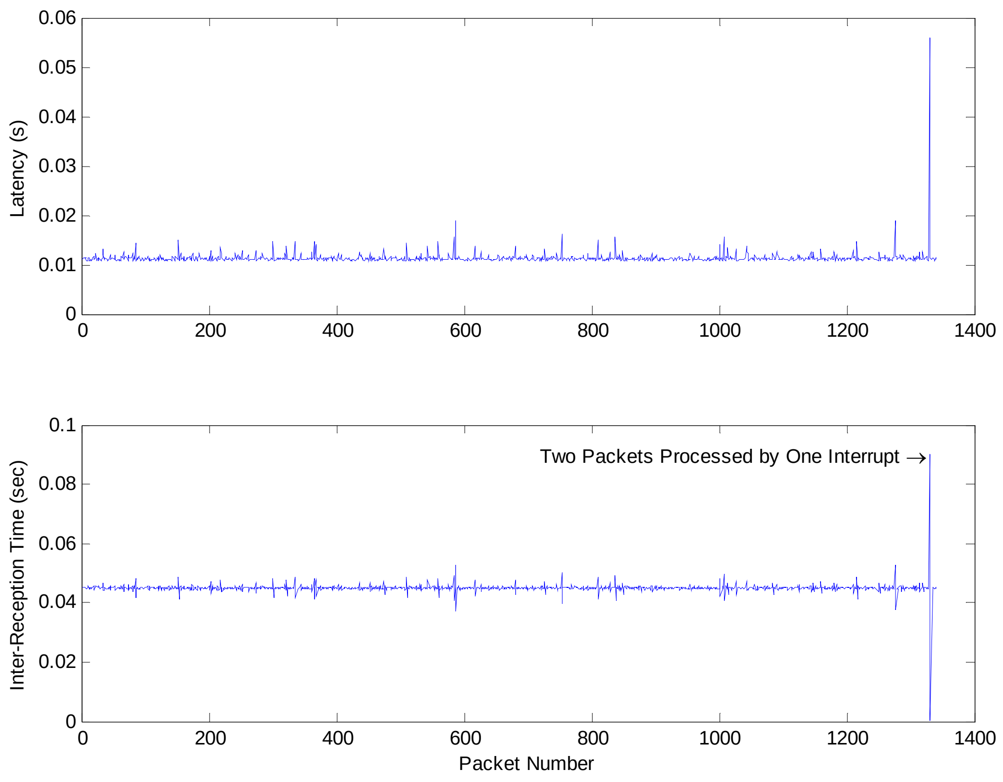


Figure 9.23 – Packet Clustering by the NIC

CHAPTER 10

EXAMPLE NETWORK TESTS

There are many network configurations that a person may need support for when using a VoIP application. In this chapter, we explore the VoIP performance of cellular connections, broadband connections, and wireless LAN connections. In all of the tests, the educational network link at Oklahoma State University is assumed to represent the Internet without any additional network impairments due to its significantly higher link speed than those of the networks discussed here. These tests are in no way meant to be exhaustive or fully representative of a given communication standard. They are simply intended to demonstrate the capabilities of the measurement system and investigate some of the properties of the networks that are revealed by the tests. They are all, however, believed to be accurate measurements of the given network conditions at the time.

10.1. Broadband Networks

Broadband network connections are widely replacing dial-up services providing Internet connectivity in homes and small offices. Many of them are even using that broadband connection to replace their circuit-switched telephone connections with services such as Vonage and Cox digital telephone service. These connections were originally designed to provide Web browsing and digital content download services. The systems provide more downstream bandwidth than upstream bandwidth due to the assumption that the endpoint will be primarily consuming content as opposed to producing it. This asymmetric bandwidth is likely to cause asymmetric delay properties as well. The two broadband connections tested were SBC Global ADSL Internet service and Cox High-Speed Cable Internet service.

10.1.1. SBC Global Asynchronous Digital Subscriber Line Internet Service Tests

The ADSL service does not allow a direct connection to the Internet through the DSL modem. A connection must be made with an SBC server using the Point to Point Protocol over Ethernet (PPPOE). This provides a separate IP address that is routable on the Internet. The EnterNet 300 PPPOE software was used in these tests to establish the DSL connection.

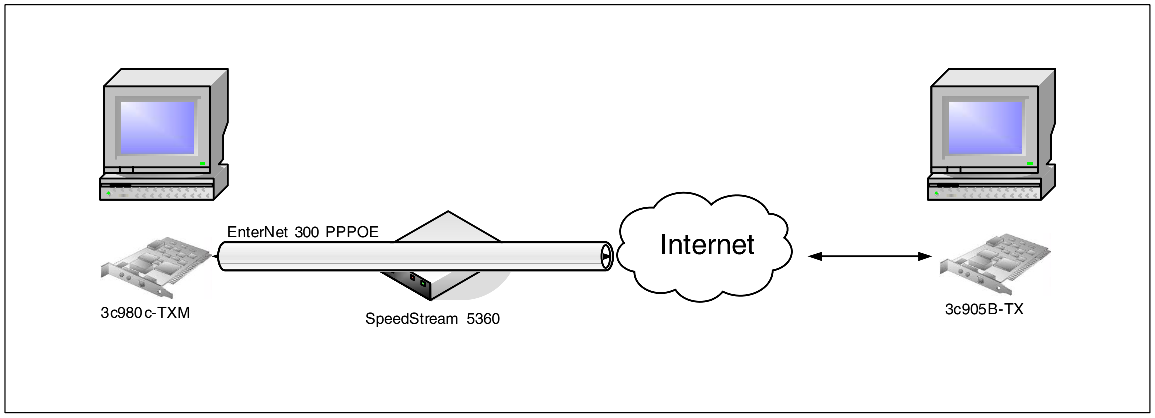


Figure 10.1 – DSL Broadband Test Configuration

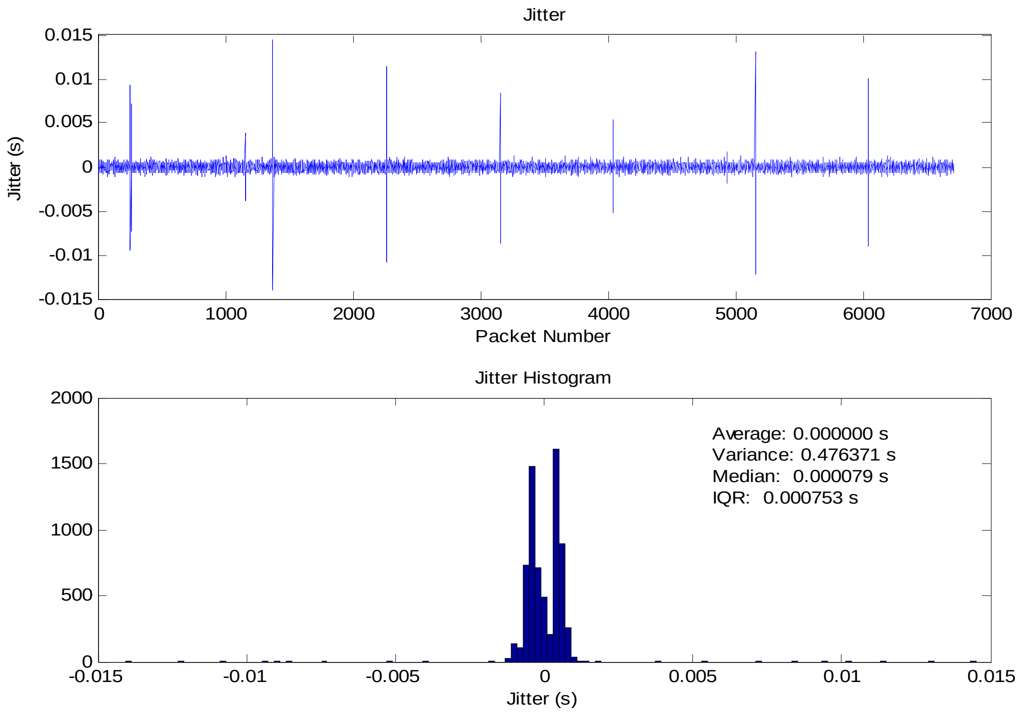


Figure 10.2 – EDU to DSL Jitter (Downstream)

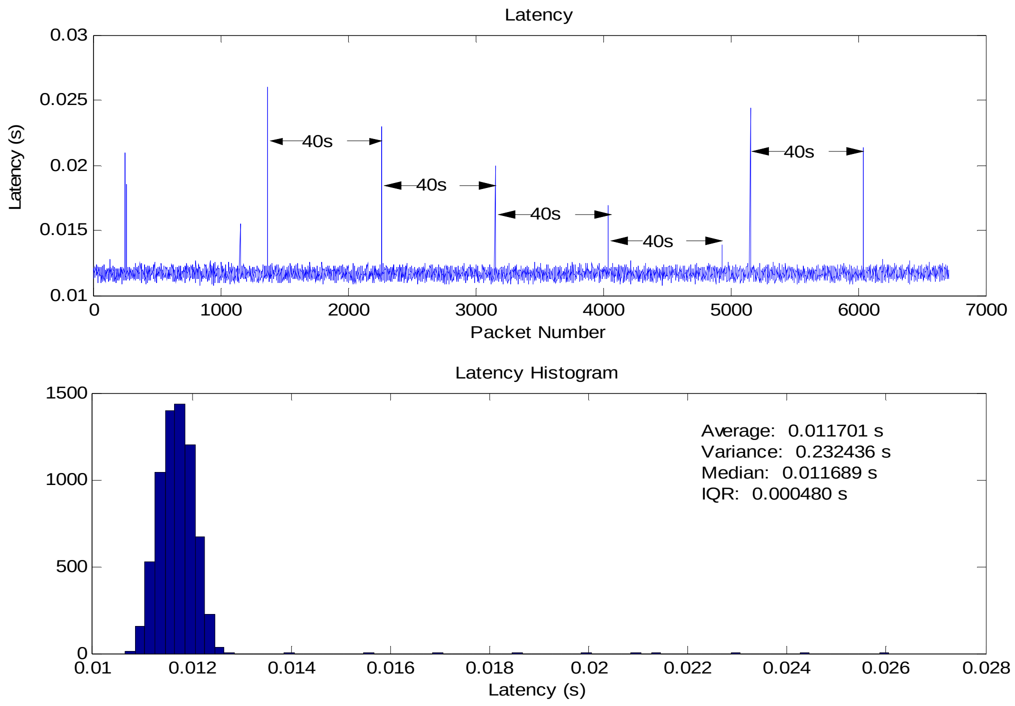


Figure 10.3 – EDU to DSL Latency (Downstream)

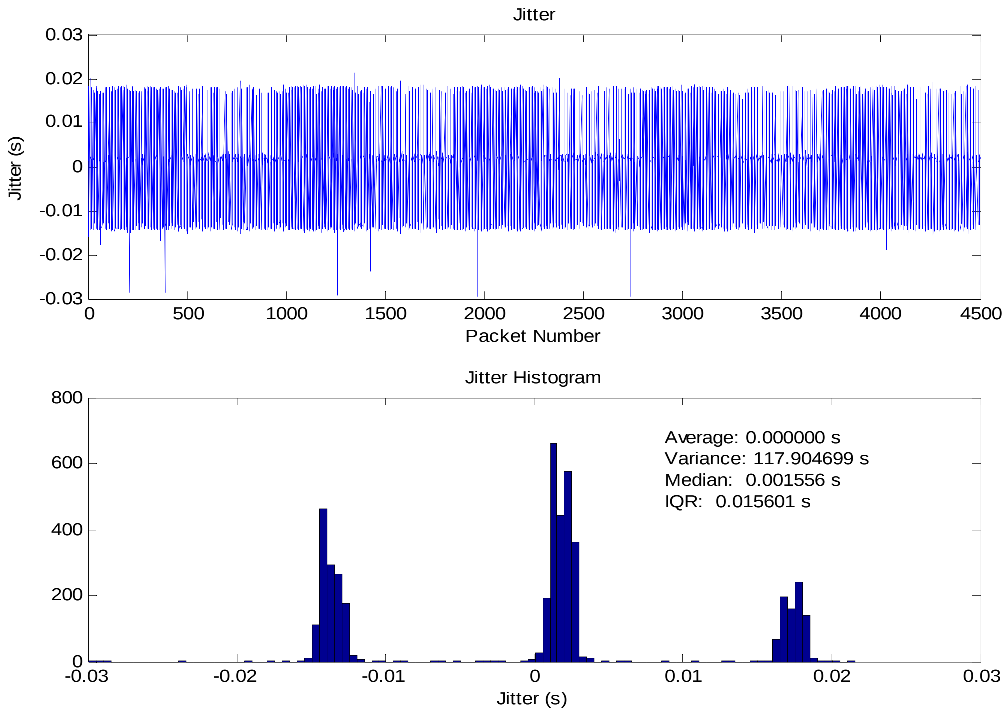


Figure 10.4 – DSL to EDU Jitter (Upstream)

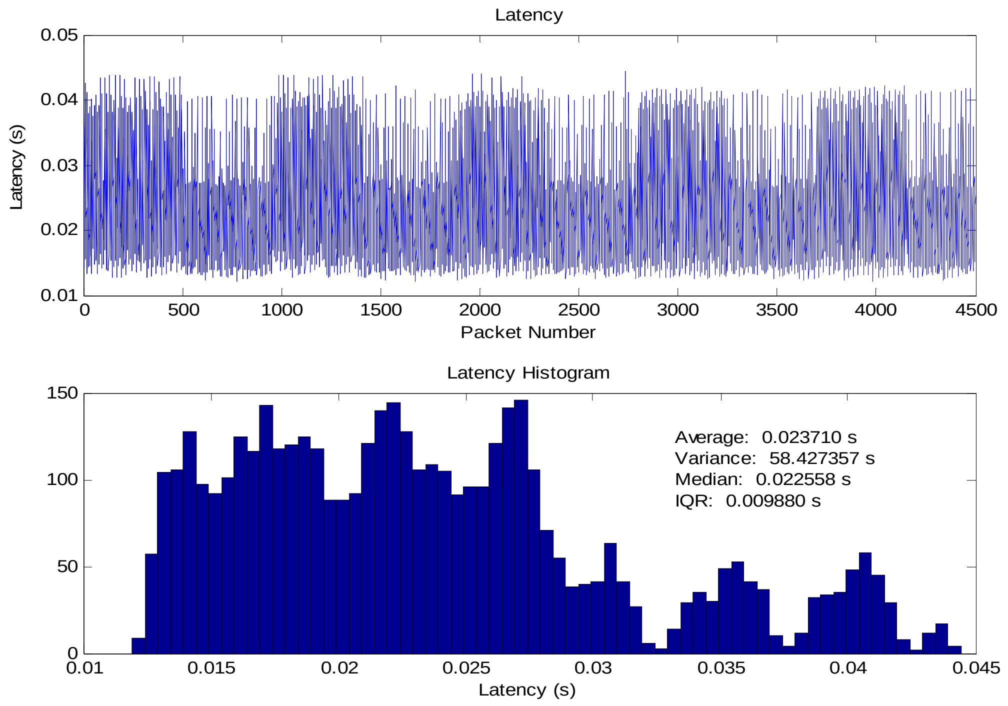


Figure 10.5 – DSL to EDU Latency (Upstream)

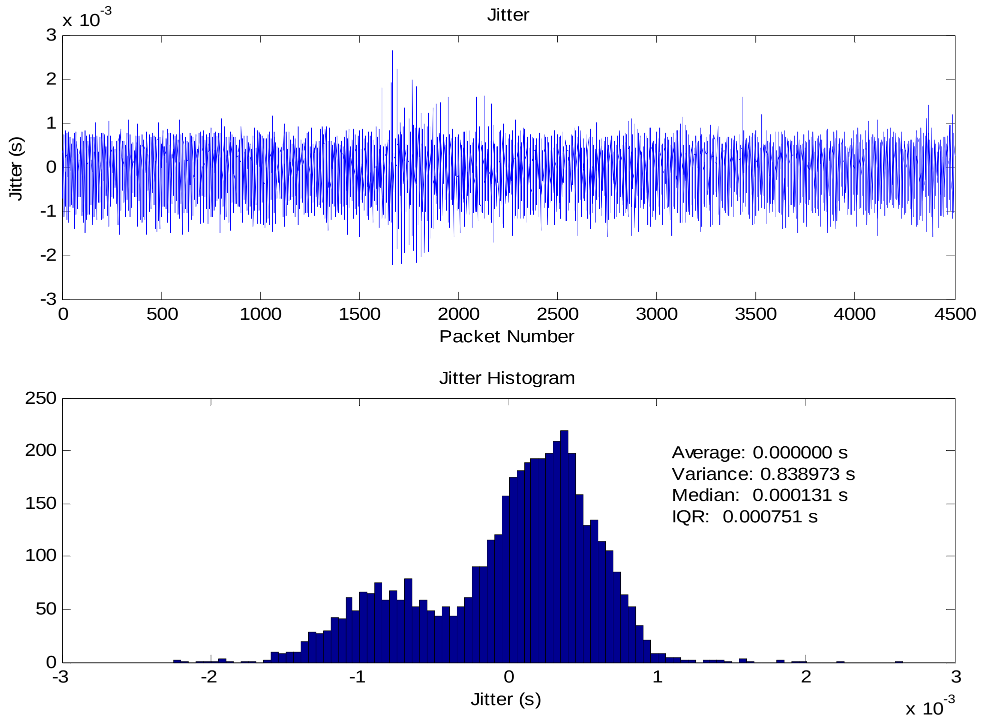


Figure 10.6 – DSL to EDU Jitter (Upstream) – Alternative Test

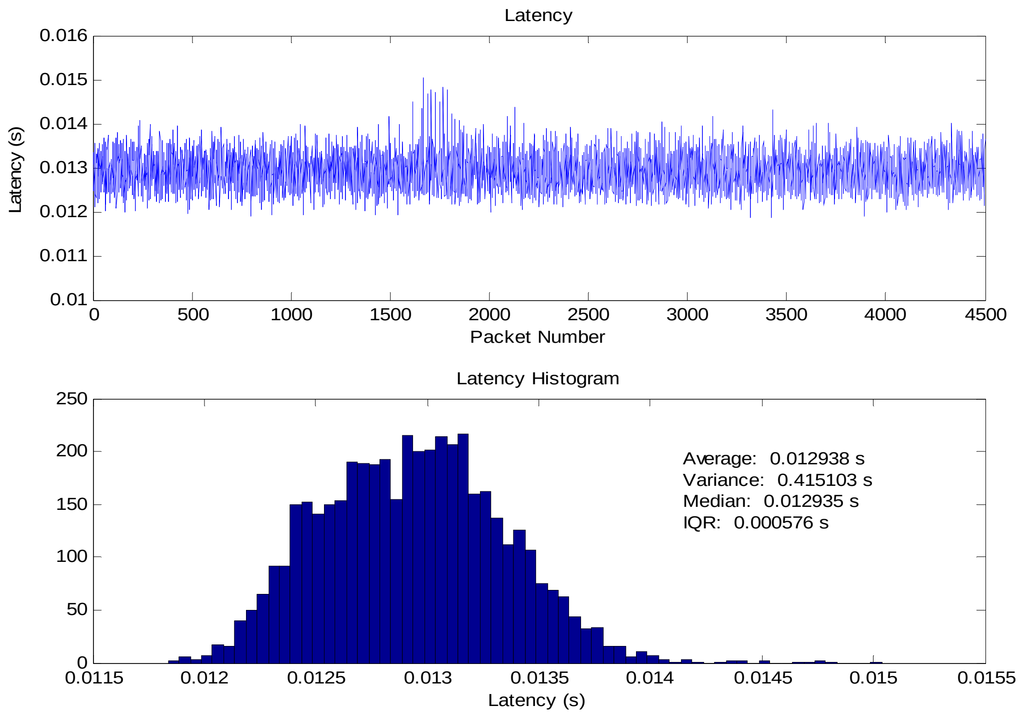


Figure 10.7 – DSL to EDU Latency (Upstream) – Alternative Test

The DSL tests revealed some periodic delay spikes in the downstream channel at 40 second intervals. It is unknown what would cause these spikes, but they will certainly have adverse effects on a VoIP application. There was some inconsistency in the upstream that is possibly due to varying channel allocation or a non-static route. The connection shown in Figures 10.6 and 10.7 ended less than one minute before the connection shown in Figures 10.4 and 10.5 was opened. The test in Figures 10.6 and 10.7 very closely resemble the downstream link with the exception of the impulsive delay. The most jitter seen in any test was less than 20ms, which is less than one frame. The longest delay seen was less than 70ms and typically below 40ms. This suggests that DSL should be able to support VoIP well because of low levels of both jitter and latency.

10.1.2. Cox Cable Internet Service Tests

The cable modem Internet connection is made through a router which implements NAT for the computers on the internal LAN.

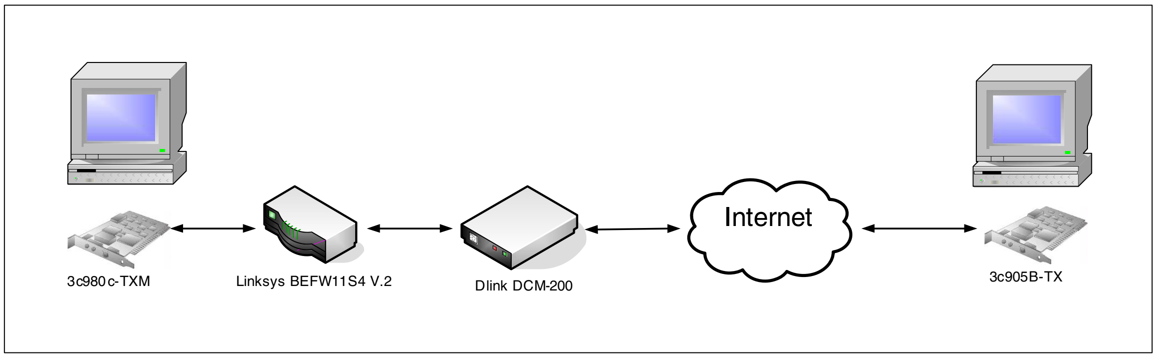


Figure 10.8 – Cable Broadband Test Configuration

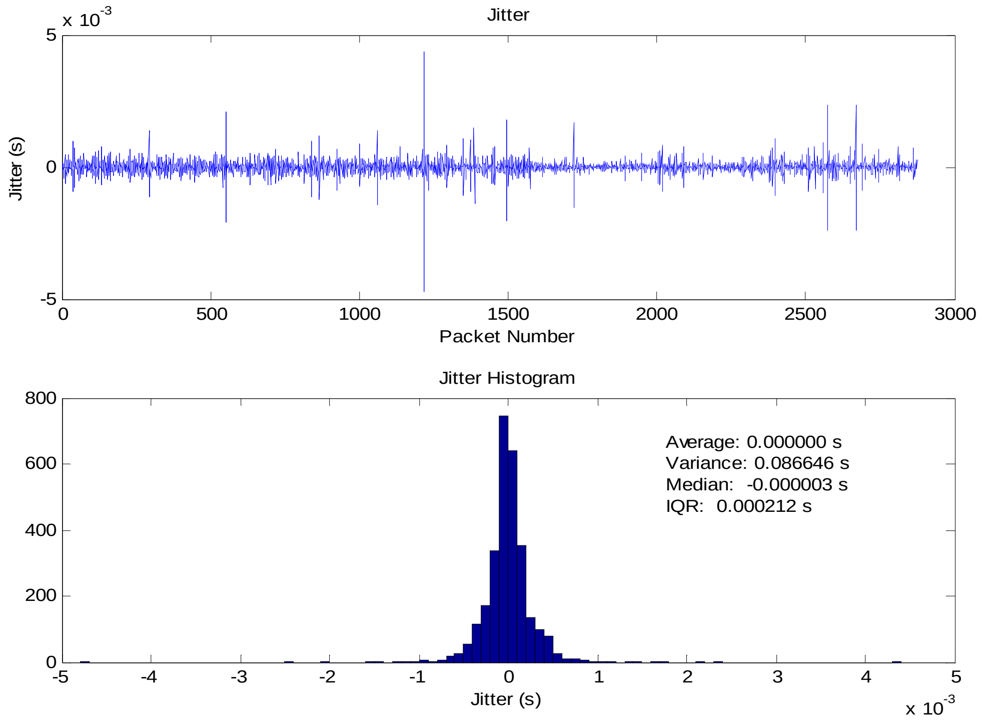


Figure 10.9 – EDU to CABLE Jitter (Downstream)

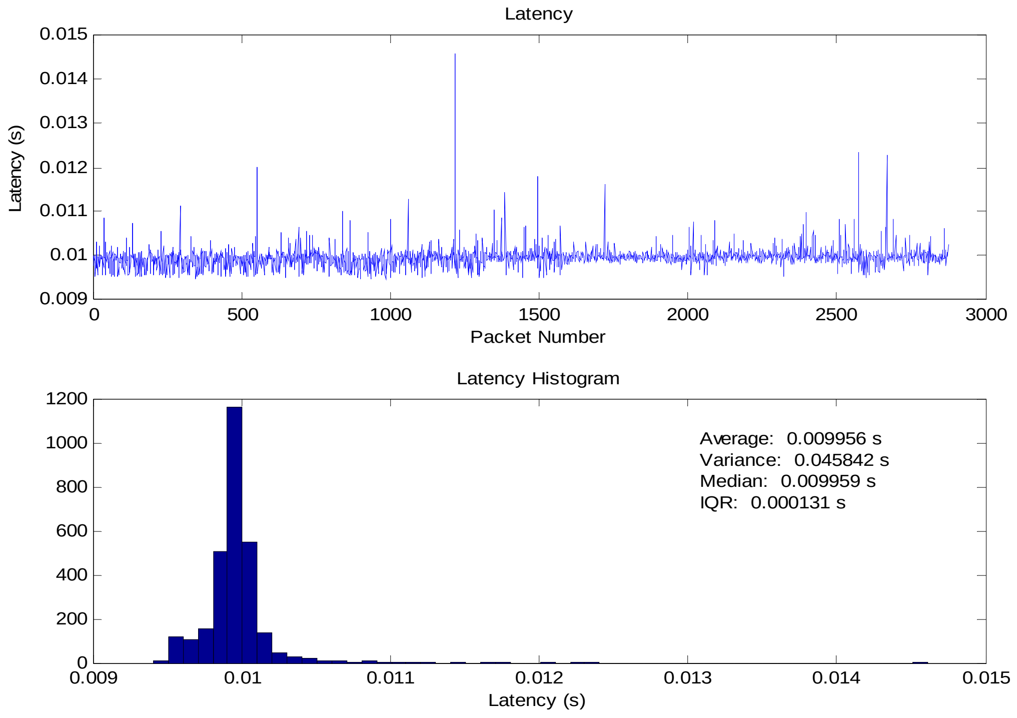


Figure 10.10 – EDU to CABLE Latency (Downstream)

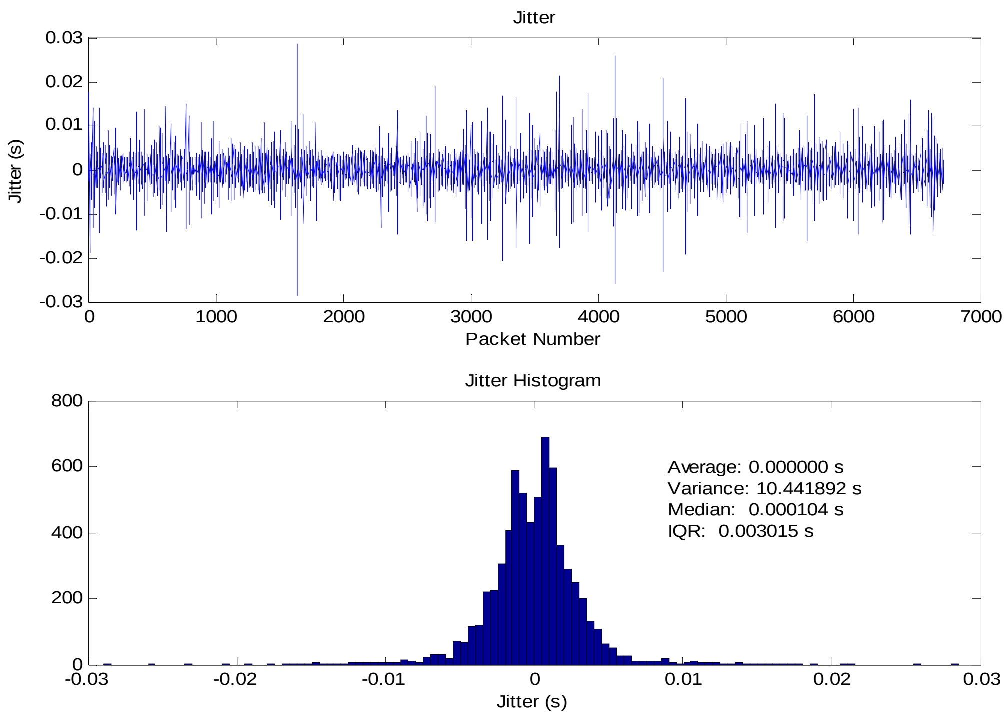


Figure 10.11 – CABLE to EDU Jitter (Upstream)

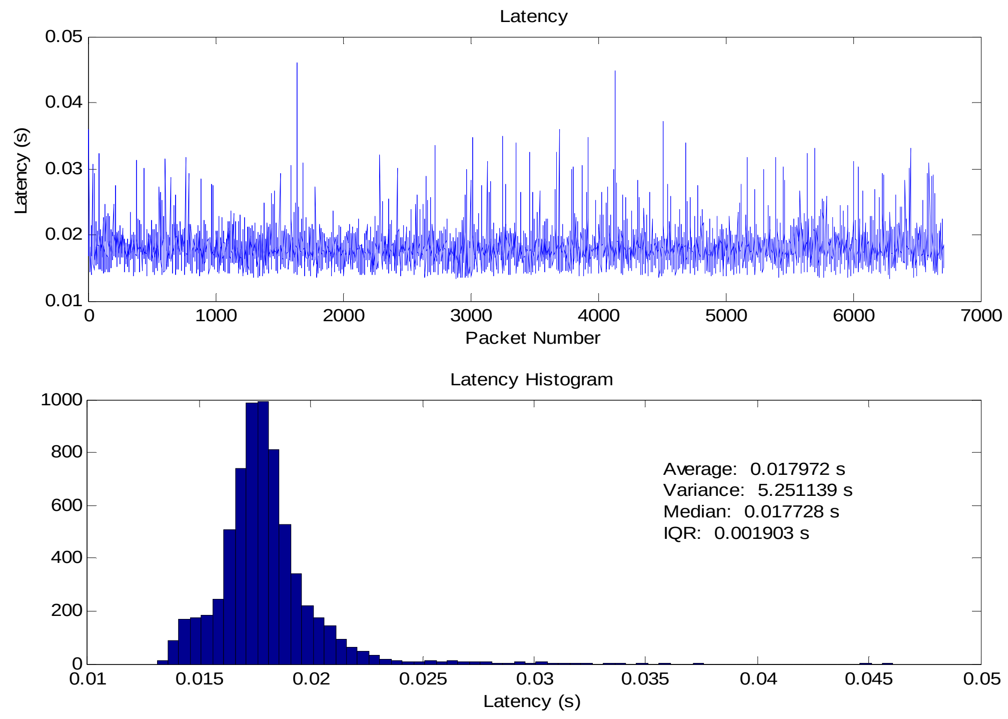


Figure 10.12 – CABLE to EDU Latency (Upstream)

The tests on the Cable connection were very consistent. The downstream had a typical delay of around 10ms and rarely had a spike above 12ms. The largest delay spike seen on the downstream was 60ms. The typical jitter was less than 1ms. On the upstream, delays were higher, as expected. The average delay was around 18ms and rarely went above 35ms. The largest delay seen in the upstream was 50ms. The jitter was nearly always below 10ms. From this data, it appears that a Cable Internet connection will easily support a VoIP system. This is supported by its broad commercial use for that purpose.

The DSL and Cable tests revealed that a Cable connection has better characteristics for VoIP, but only by a narrow margin. The overall characteristics of DSL are actually superior if the upstream connection consistently produced results like those in Figures 10.6 and 10.7. The differences are negligible, though, as both systems are easily capable of providing excellent quality of service for VoIP.

10.2. Wireless LANs

Wireless LANs are fast appearing in many public places such as airports, coffee shops, stadiums, and libraries. Some cell phones now support the ability to use a wireless LAN instead of cellular service for voice and data service in those areas with wireless LAN infrastructure. They are also often used in homes to connect PCs to a broadband connection. The wireless LANs tested were Bluetooth, 802.11b, and 802.11g. In addition, an 802.11b NIC was used to connect to an 802.11g access point and an 802.11g NIC to an 802.11b access point.

10.2.1. Bluetooth Tests

The Bluetooth connections were made using the Personal Area Network (PAN) profile to an Ethernet access point.

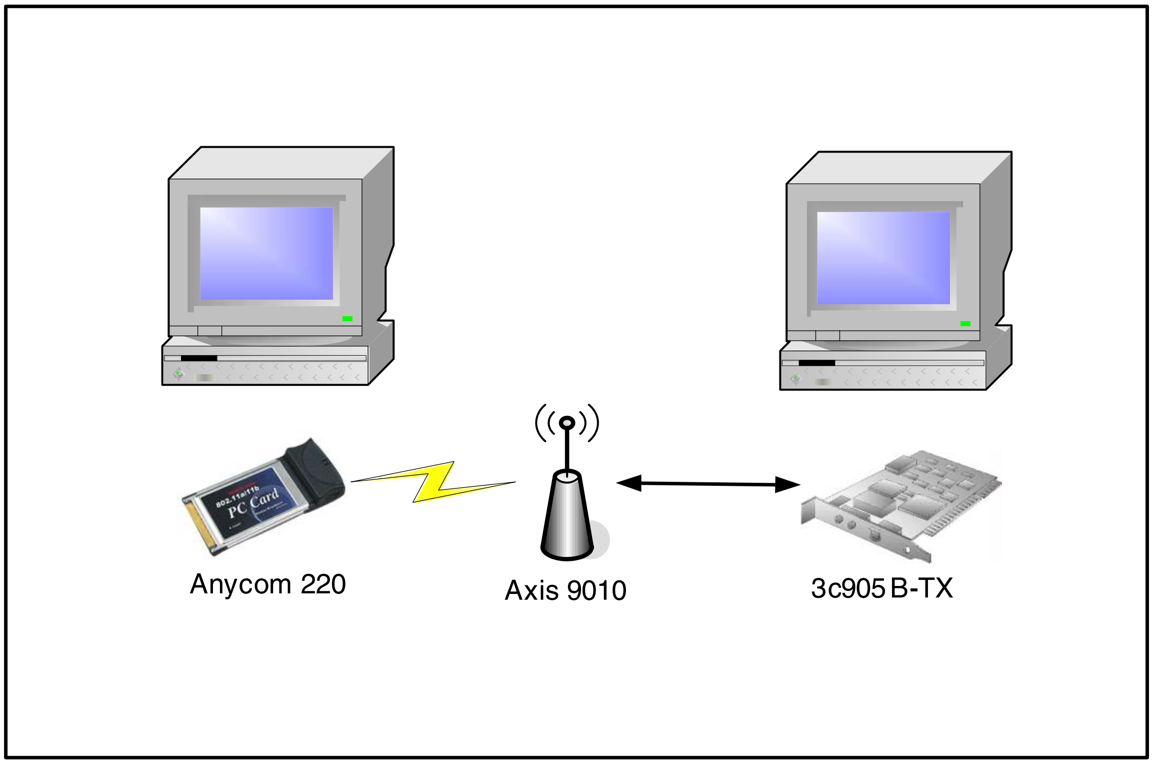


Figure 10.13 – Bluetooth Test Configuration

.

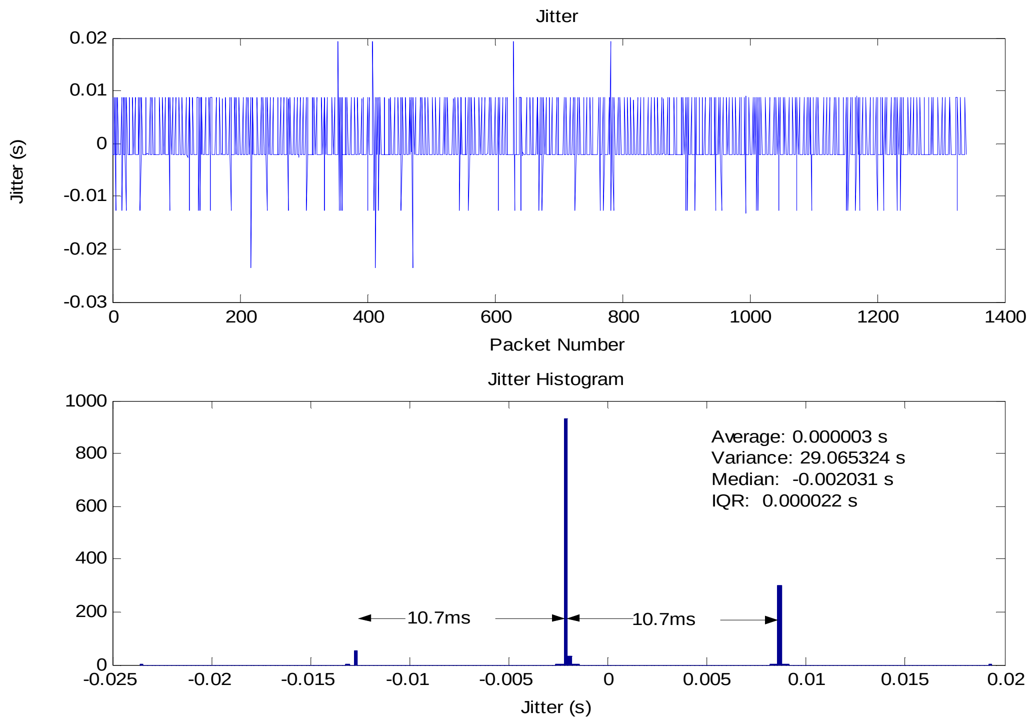


Figure 10.14 – Bluetooth Access Point to NIC Jitter

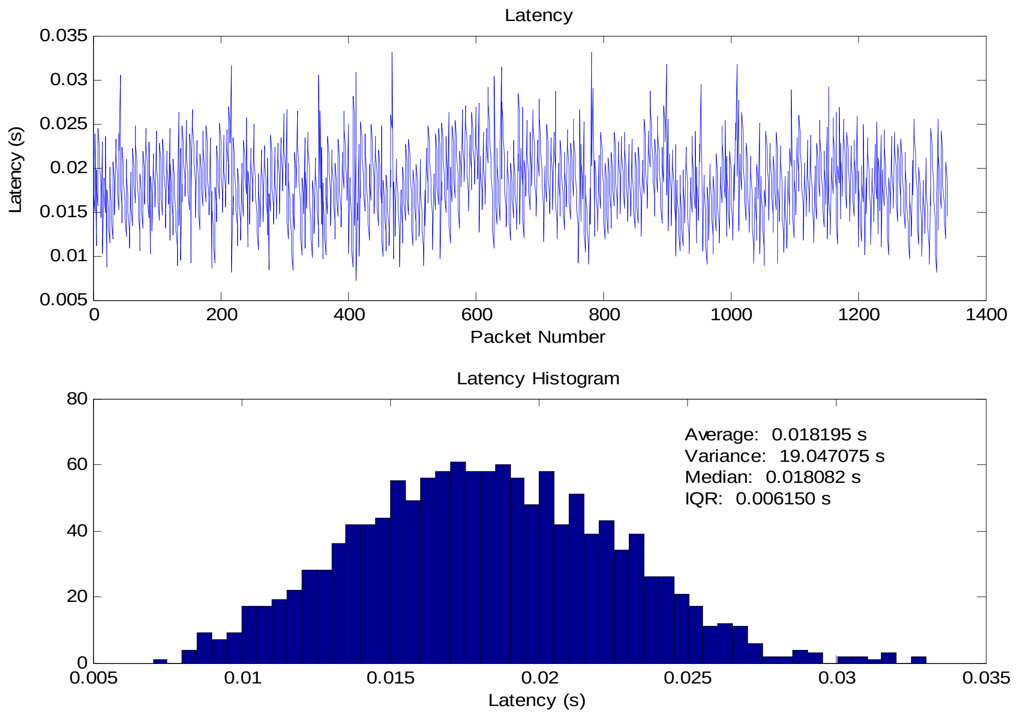


Figure 10.15 – Bluetooth Access Point to NIC Latency

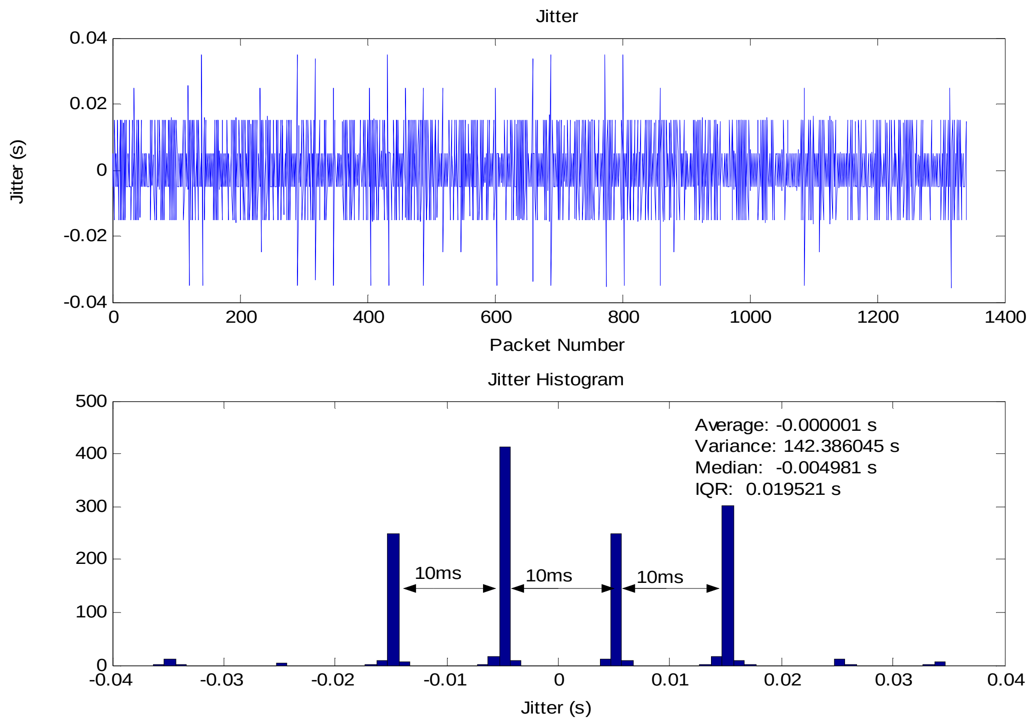


Figure 10.16 – Bluetooth NIC to Access Point Jitter

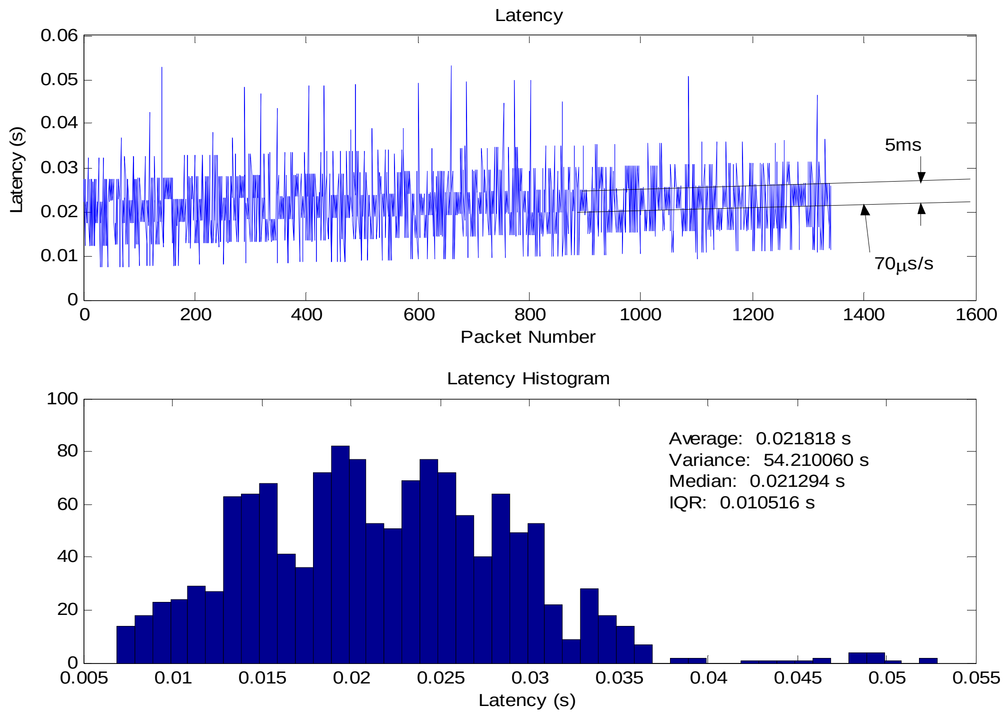


Figure 10.17 – Bluetooth NIC to Access Point Latency

Bluetooth was originally designed to replace cables to devices such as mice, keyboards, and scanners on a desktop and as such, it was not designed with high bandwidth capabilities (only 721 kbps). It is considered here because it is often used to connect handheld computers or cell-phones to a network connection or to share the network connection provided by the phone. The average delay of a Bluetooth connection is about 20ms with peaks that are frequently above 50ms. That is more than 2 frames of speech data so the jitter is non-negligible, but will most likely not severely degrade quality of service if the jitter buffer on the receiver is configured properly. There is a very rigid structure in the Bluetooth data that is presumably due to the synchronization scheme used. The access point was observed to only vary the delay by multiples of 10.7ms offset from zero by 2.1ms as seen in Figure 10.14. This results in the high frequency appearance of the latency in Figure 10.15. The NIC apparently uses a slightly different timing scheme in which the delay is varied only in multiples of 10ms offset from zero by 4.8ms as seen in Figure 10.16. Because the offset is so close to 5ms (half of the interval), the latency appears to have the slowly increasing structure seen in Figure 10.17. Because Bluetooth has a jitter of nearly 40ms, it should probably be avoided if convenient alternatives are available.

10.2.2. 802.11a Tests

The 802.11a connections were established using the Microsoft wireless zero configuration service in Windows XP.

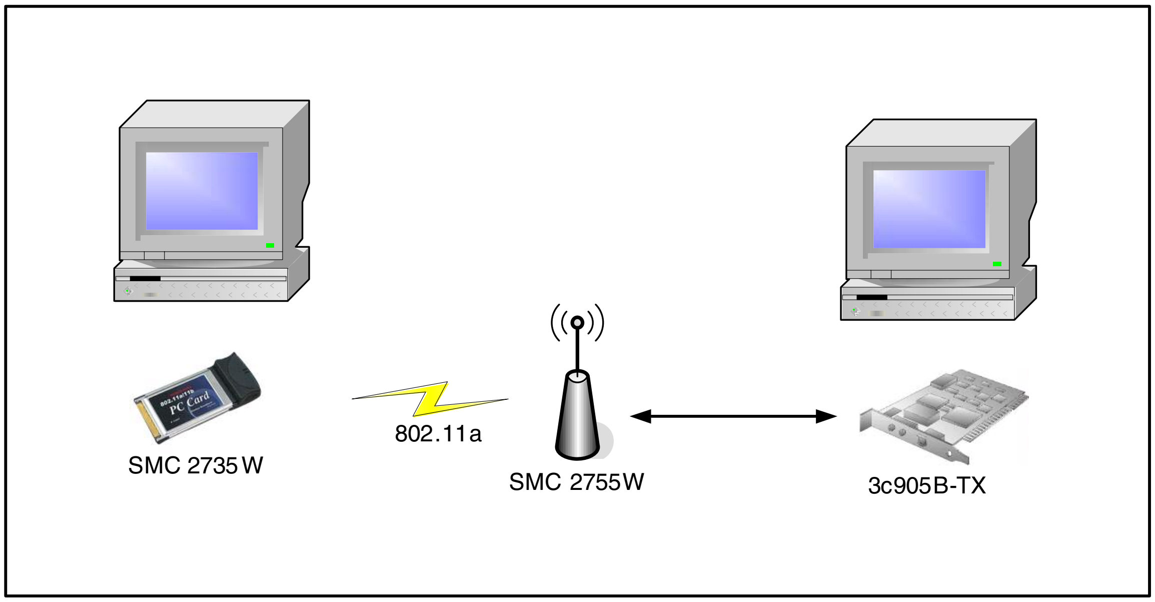


Figure 10.18 – 802.11a Test Configuration

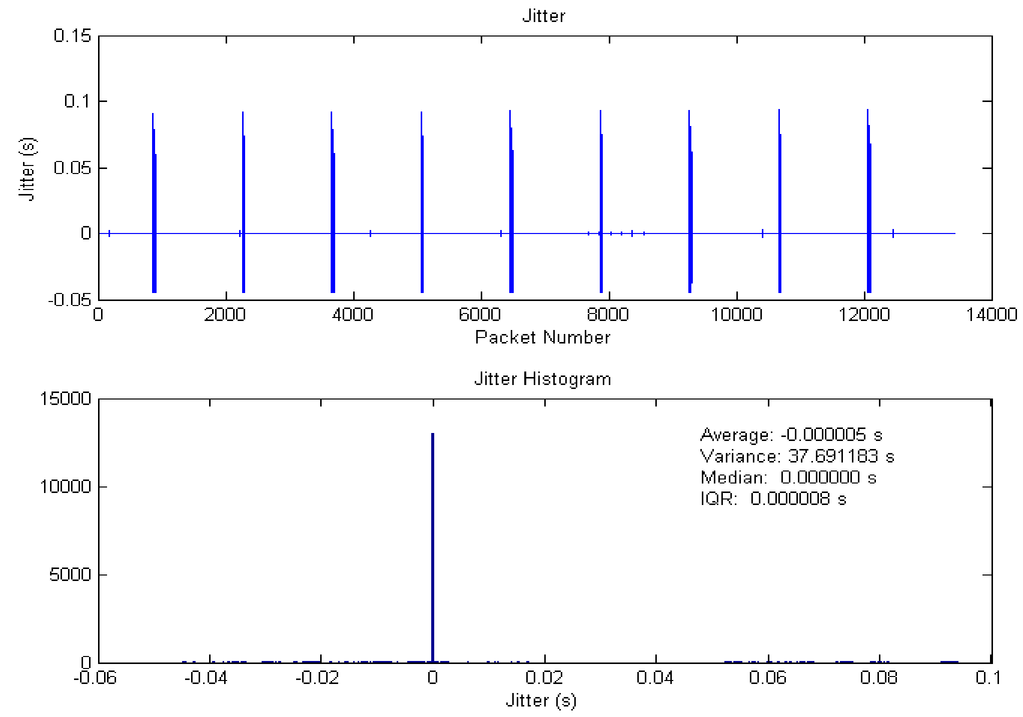


Figure 10.19 – 802.11a Access Point to NIC Jitter

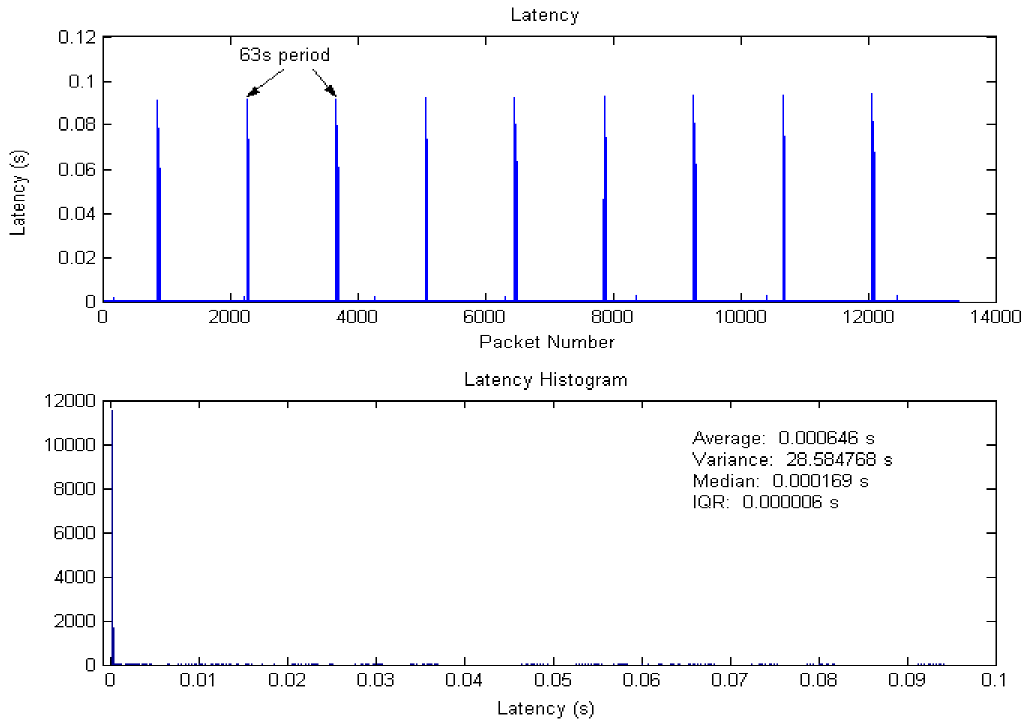


Figure 10.20 – 802.11a Access Point to NIC Latency

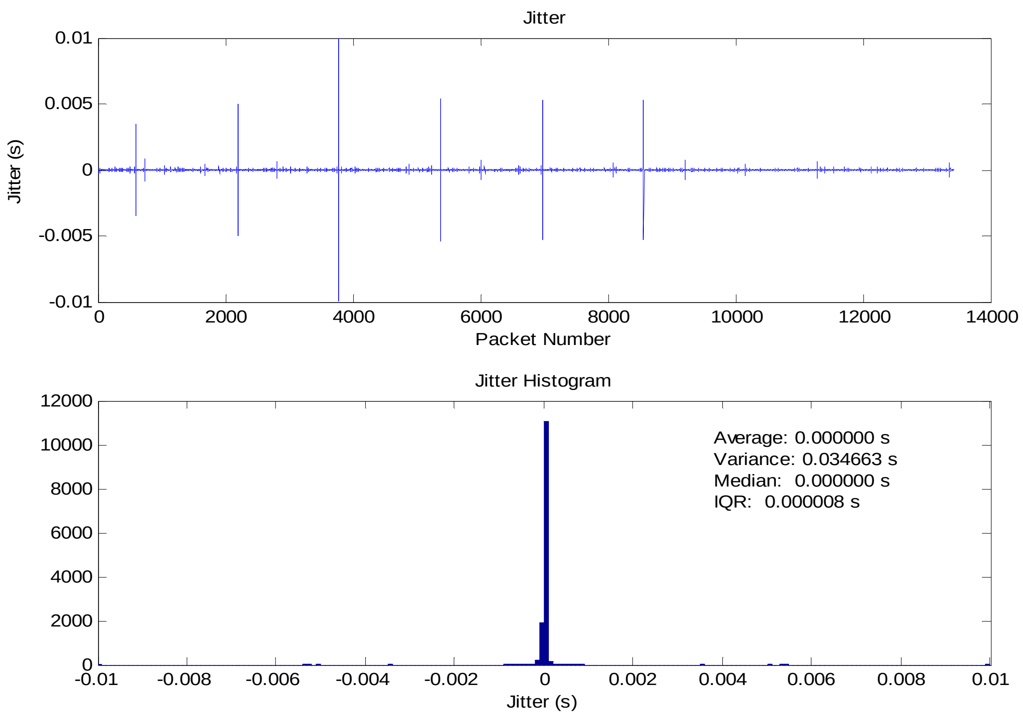


Figure 10.21 – 802.11a NIC to Access Point Jitter

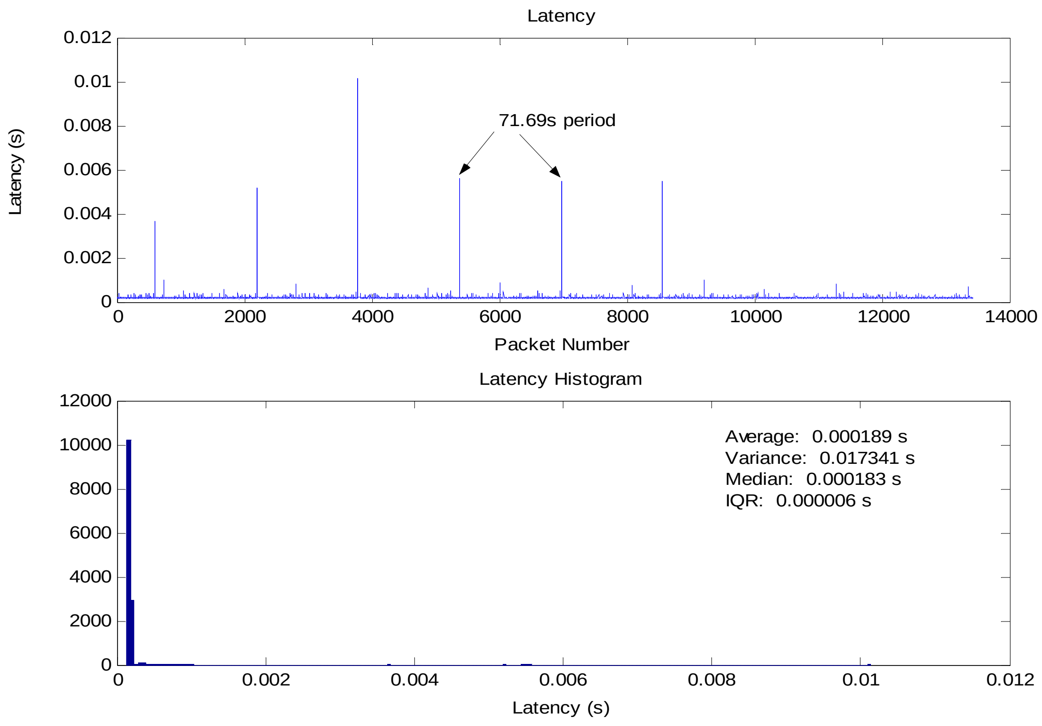


Figure 10.22 – 802.11a NIC to Access Point Latency

The tests of the 802.11a network showed excellent baseline latency (less than 200µs). When the NIC transmitted to the access point, the jitter was always less than 12ms, which should not affect VoIP. When packets were sent the other way, the jitter was nearly an order of magnitude higher. With these delays of over 90ms the VoIP application would need to buffer an additional four frames to avoid buffer starvation. This is certainly doable, but it will add to the total latency perceived by the user. The jitter spikes in both cases are periodic. The small spikes generated by the NIC have a period of 71.69s and the large spikes generated by the access point have a period of 63s. The source of the 71.69s spikes is unknown and assumed to be caused by some feature of the access point. The 63s spikes were determined to be caused by the wireless zero configuration scanning for new access points.

In addition to the large delay, the traffic generated by the NIC included many duplicate receptions. The timestamps on the duplicates were between 70µs and 110µs. Vern Paxson suggests that packet filter software can sometimes duplicate packets [33]. Guy Harris posted on the Ethereal forum that many wireless drivers will errantly provide two copies of each packet to WinPcap [17]. The large difference in timestamps leads me to believe that these two possibilities are probably not true. Srikant Sharma explains that 802.11 will retransmit packets on a wireless link at MAC level if an acknowledgment is not received [38]. This is below the transport layer and is therefore transparent to UDP. However, the 802.11 endpoint is responsible for removing duplicate packets based on sequence numbers in the 802.11 header. It is possible that the access point is failing remove unneeded retransmissions. The duplicate packets come in a single large chunk.

In one test, 5111 packets were duplicated all consecutively, all with similar delays from the original, with eachduplicated exactly once. This doesn't seem likely to be caused by network error.

10.2.3. 802.11b Tests

The 802.11b connections were established using the Microsoft wireless zero configuration service in Windows XP.

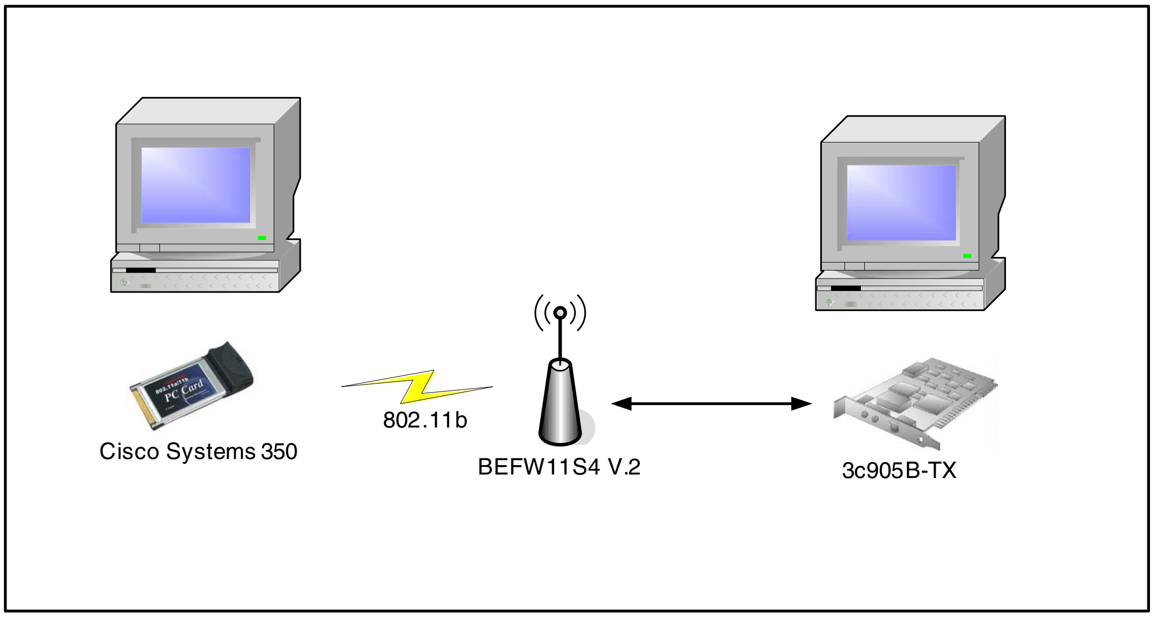


Figure 10.23 – 802.11b Test Configuration

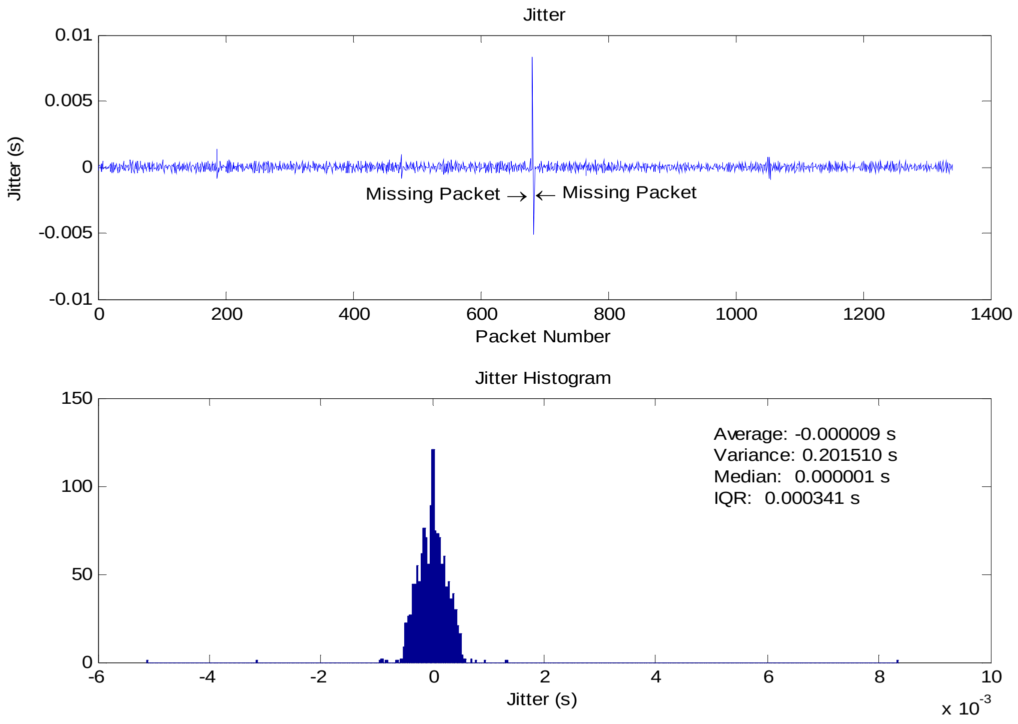


Figure 10.24 – 802.11b Access Point to NIC Jitter

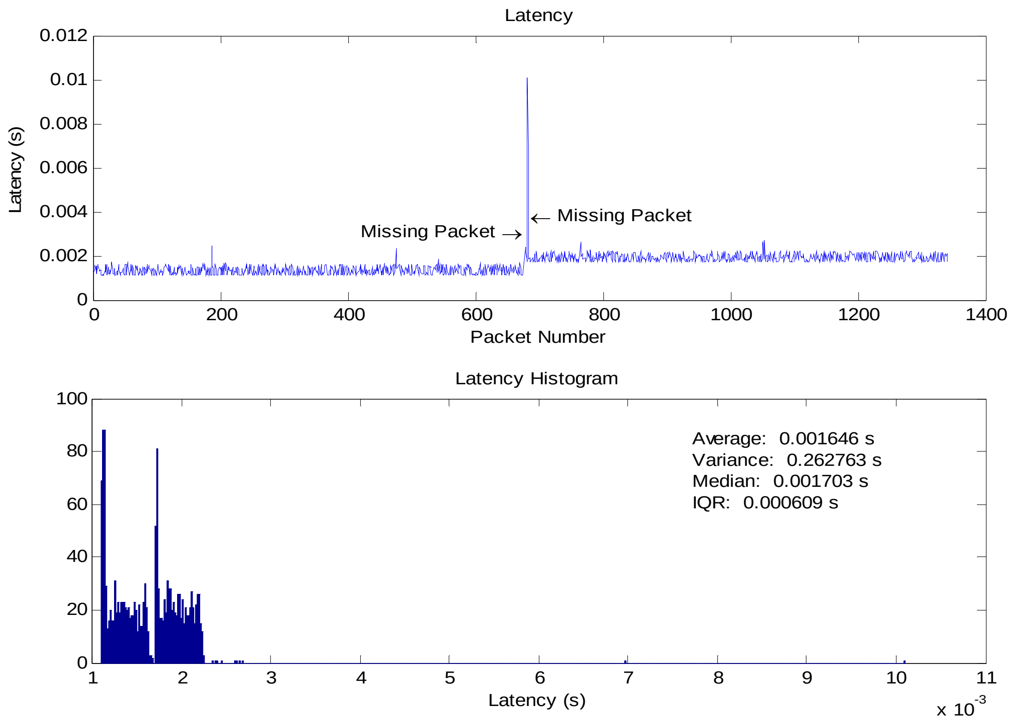


Figure 10.25 – 802.11b Access Point to NIC Latency

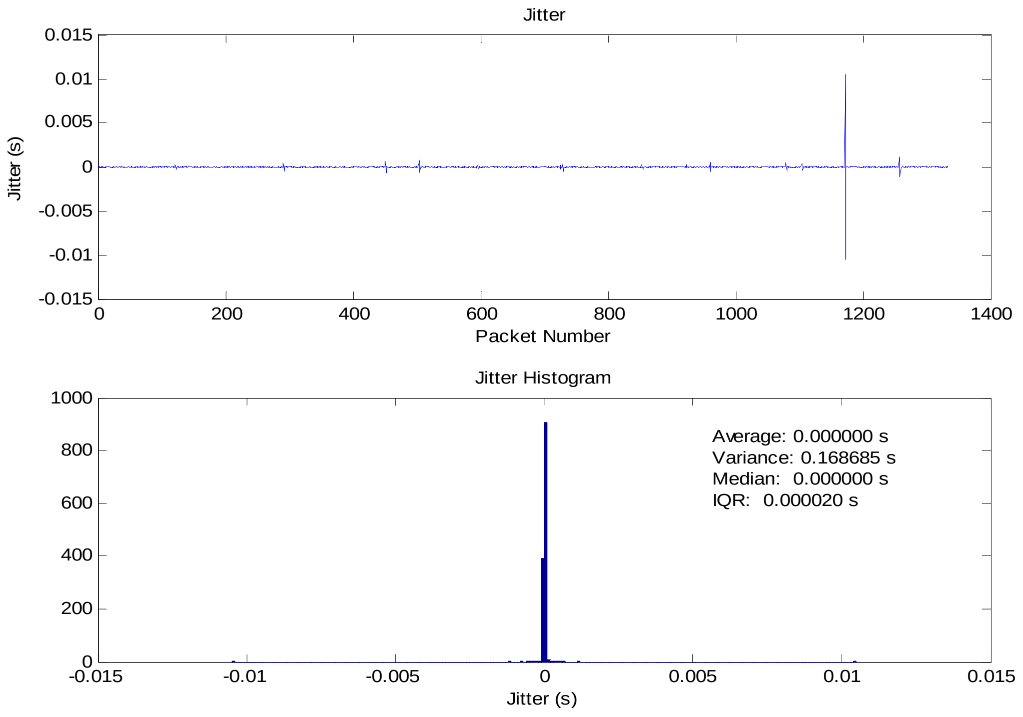


Figure 10.26 – 802.11b NIC to Access Point Jitter

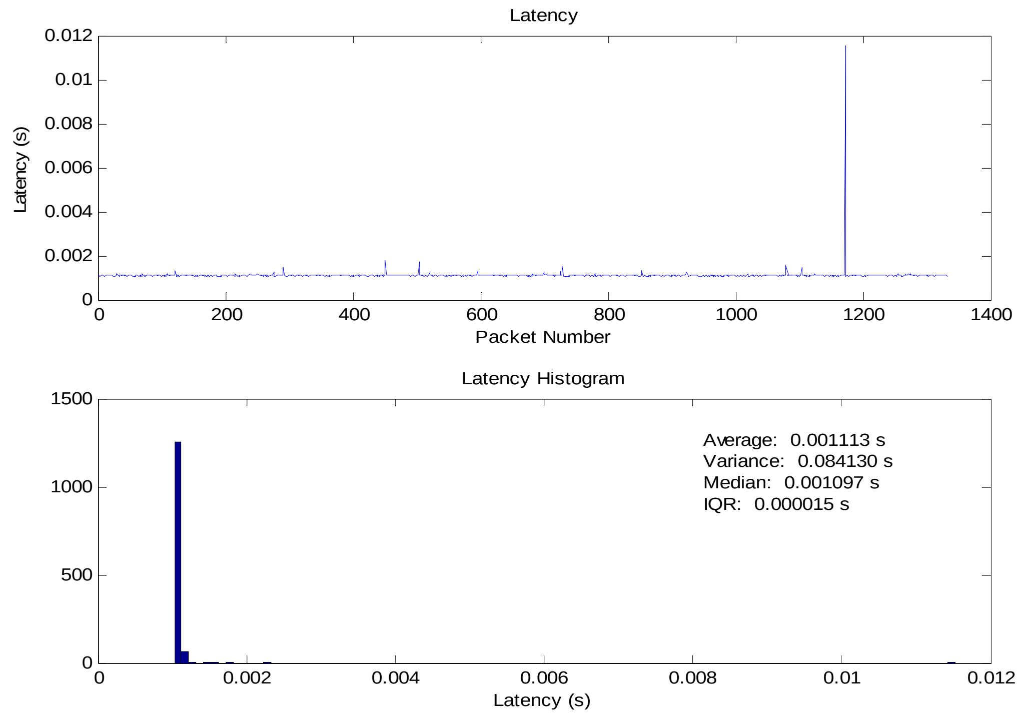


Figure 10.27 – 802.11b NIC to Access Point Latency

The 802.11b wireless LAN standard is probably the most widely deployed wireless LAN. It is capable of speeds up to 11 Mbps, though that says nothing of its delay and loss rate. The data paths to and from the access point appear to be similar in Figures 10.24 through 10.27. The jitter is almost always below 1ms and the latency is typically below 8ms. When the access point was transmitting, there was a steep step observed in the latency from several tests, as visible in Figure 10.25. It is assumed that this is due to internal delay in the access point for radio synchronization. Even with excellent signal strength, a few packets are occasionally lost, also visible in Figure 10.25. The delay and jitter characteristics of 802.11b are small enough to not adversely affect VoIP applications.

10.2.4. 802.11g Tests

The 802.11g connections were established using the Microsoft wireless zero configuration service in Windows XP.

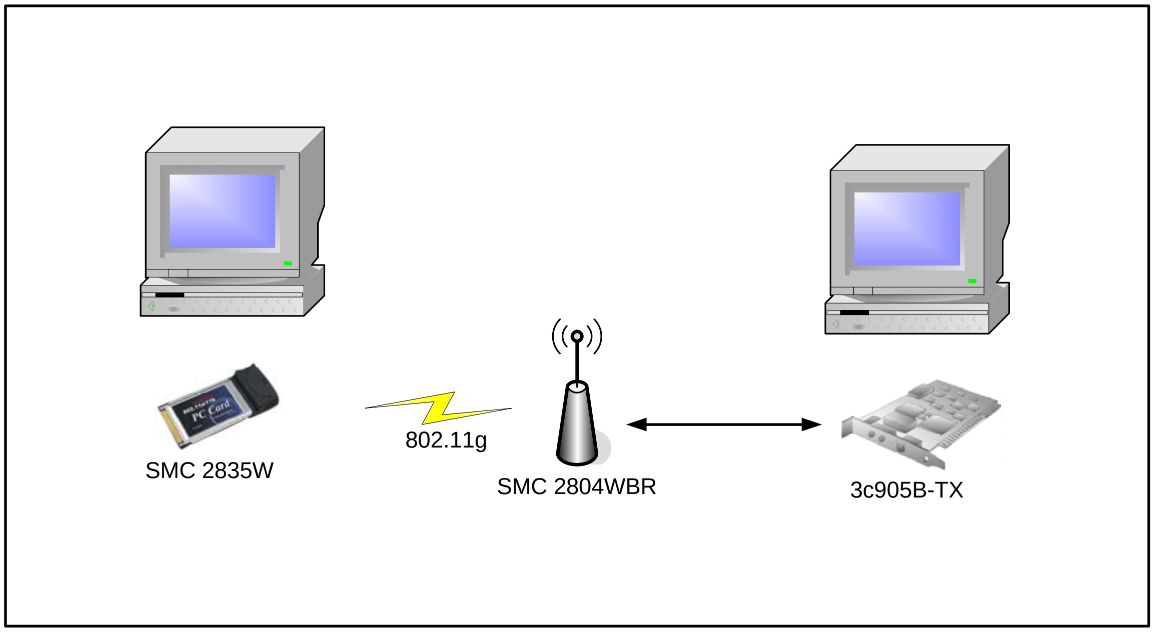


Figure 10.28 – 802.11g Test Configuration

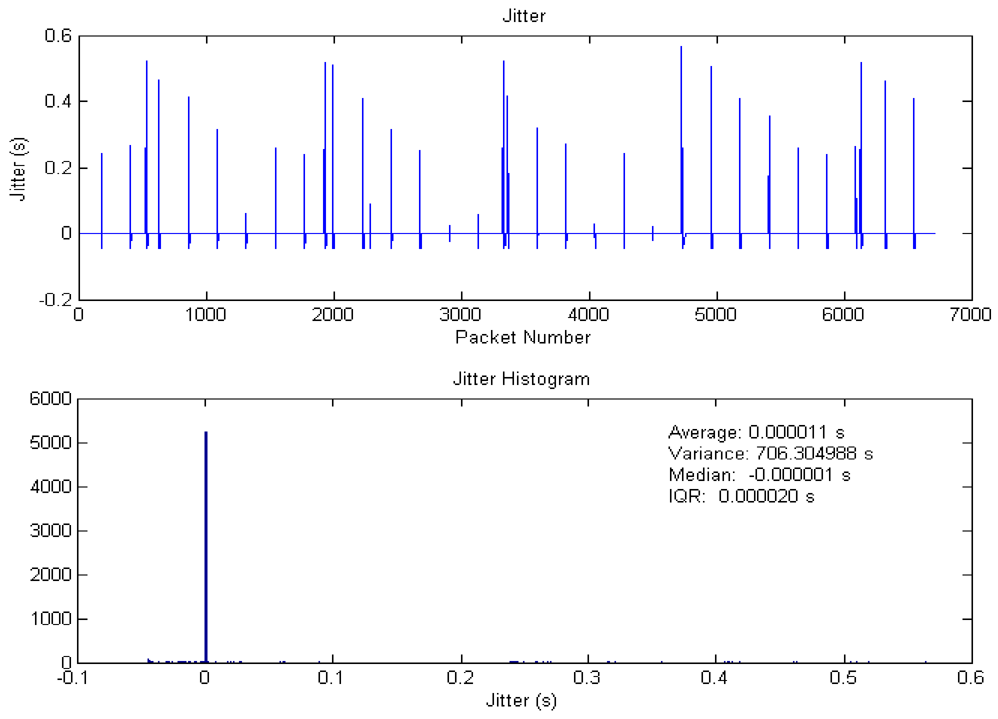


Figure 10.29 – 802.11g Access Point to NIC Jitter

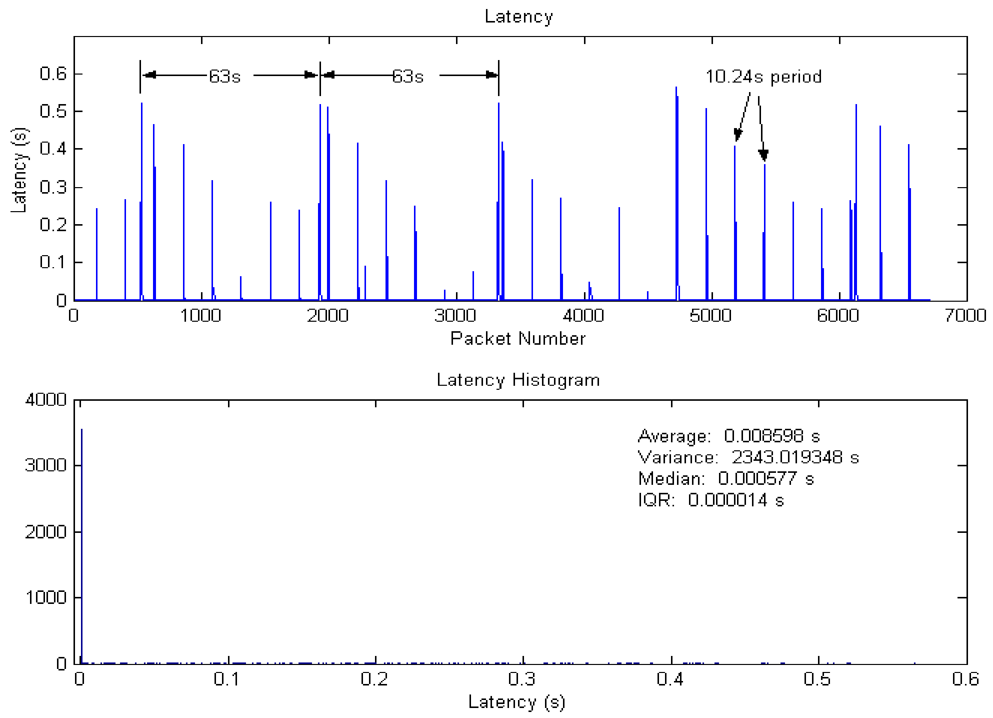


Figure 10.30 – 802.11g Access Point to NIC Latency

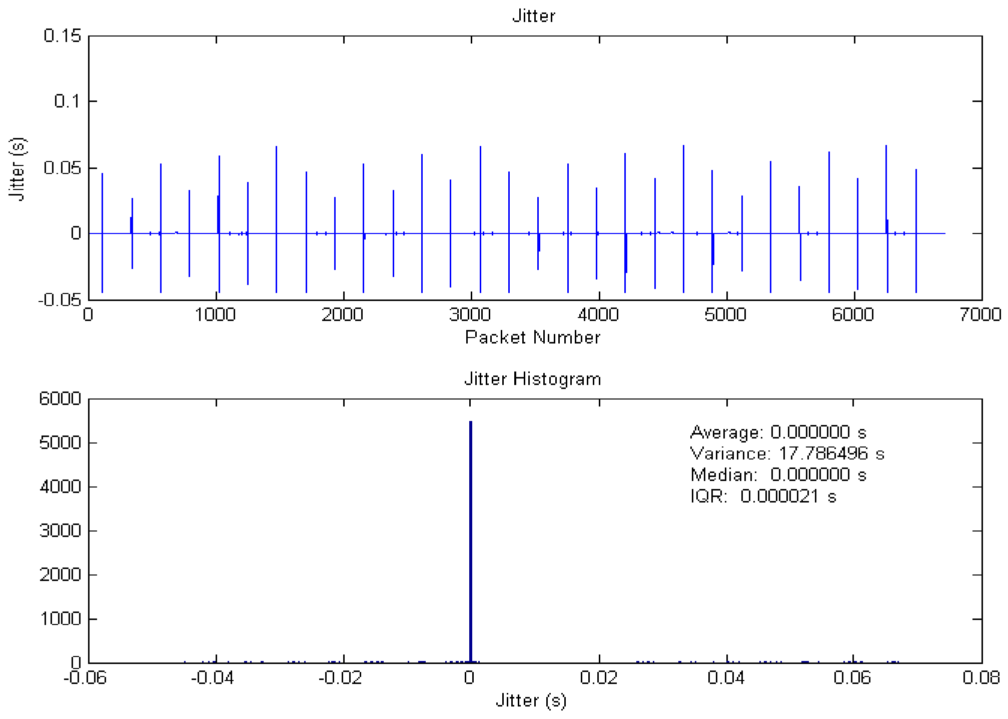


Figure 10.31 – 802.11g NIC to Access Point Jitter

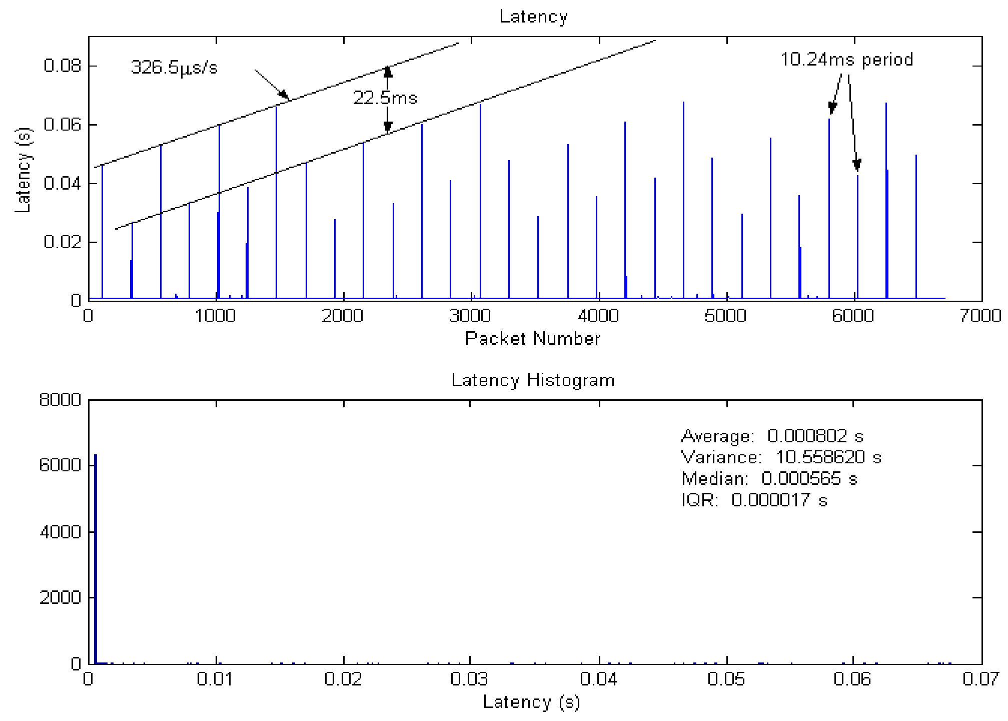


Figure 10.32 – 802.11g NIC to Access Point Latency

The tests of the 802.11g network showed two different periodic aspects. The 63s delay spikes were caused by the wireless zero configuration. The test was later rerun with he wireless zero configuration service disabled, and the 63s periodic spikes no longer existed. The other delays seen had a 10.24s period. This is assumed to derive from some 10ms driven counter that causes an interference event every 1024 counts. The exact cause is unknown. The typical latency is less than 600µs in both directions, but unless the large jitter that was as high as 600ms is eliminated, this network will have poor performance in a VoIP system.

10.2.5. 802.11b NIC with 802.11g Access Point Tests

The Microsoft wireless zero configuration was used in these tests.

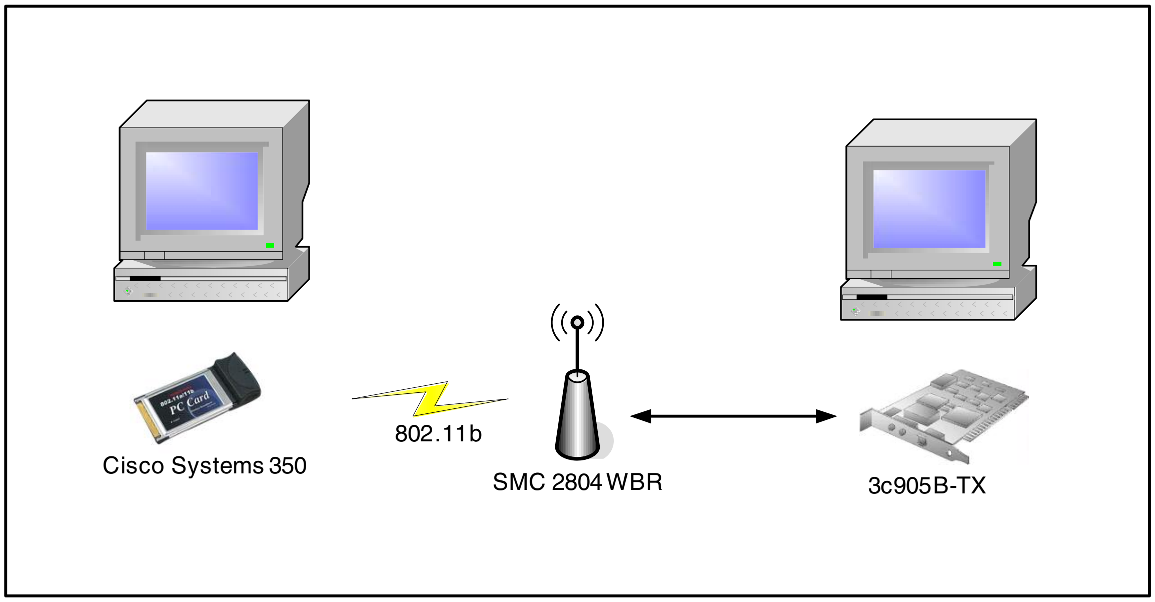


Figure 10.33 – 802.11b NIC and 802.11g Access Point Test Configuration

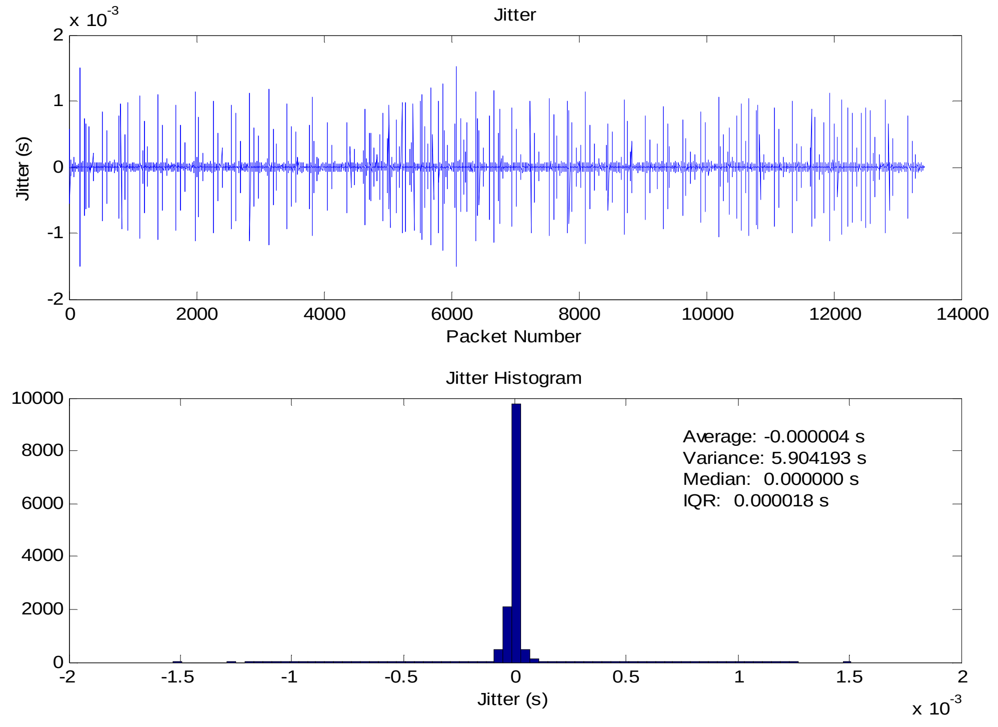


Figure 10.34 – 802.11g Access Point to 802.11b NIC Jitter

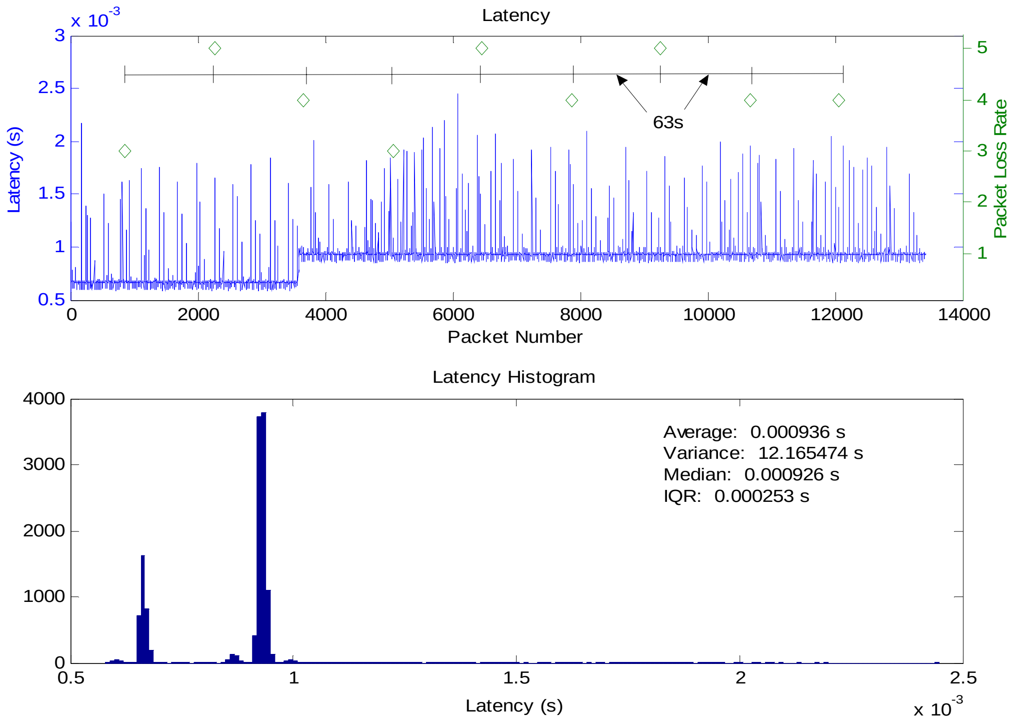


Figure 10.35 – 802.11g Access Point to 802.11b NIC Latency

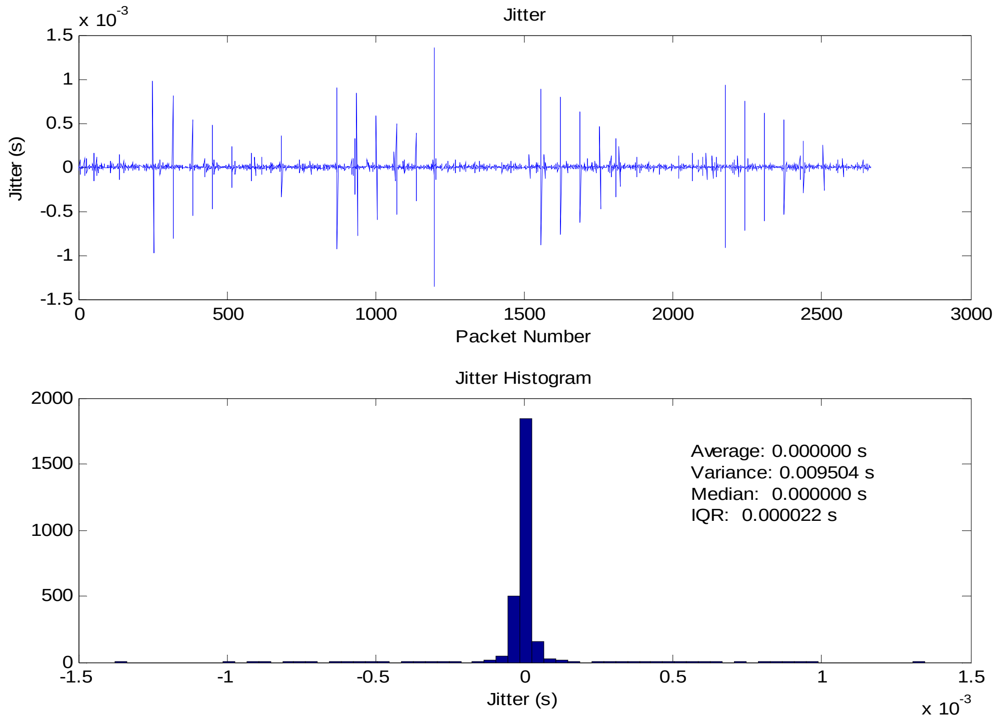


Figure 10.36 – 802.11b NIC to 802.11g Access Point Jitter

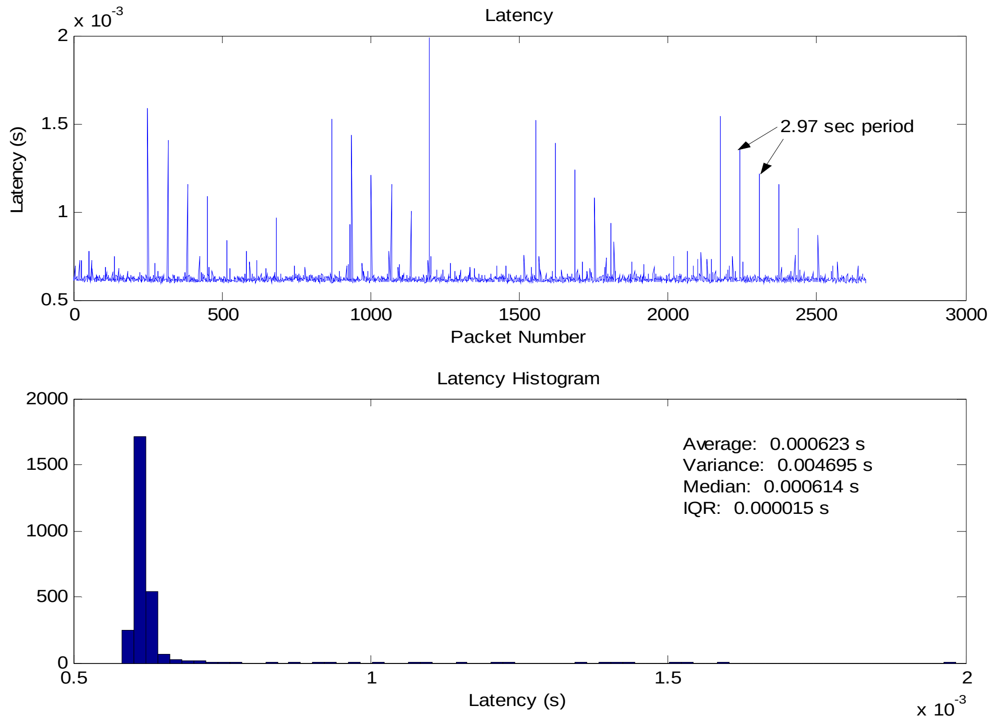


Figure 10.37 – 802.11b NIC to 802.11g Access Point Latency

In the configuration where an 802.11b device is operating in a network using an 802.11g access point, the access point is able to reduce its speeds to support the older device. When the access point is transmitting, the packet loss rate is surprisingly high considering that it is operating in a local network with its endpoints less than 10 feet from one another. As seen in Figure 10.35, between 4 and 5 packets are lost every 63 seconds. This is due to the wireless zero configuration software probing for new access points. On some hardware, this causes the active connection to be interrupted and hence causing the packet loss. Aside from the packet loss, the latency and jitter are exceptionally low. The average latency is below 1ms with jitter below 2ms. The step in the latency that existed in the 802.11b tests appears in this test as well. The source of the 2.97s periodic delays in the tests in which the 802.11b NIC is transmitting is unknown.

10.2.6. 802.11g NIC with 802.11b Access Point Tests

The Microsoft wireless zero configuration was used in these tests.

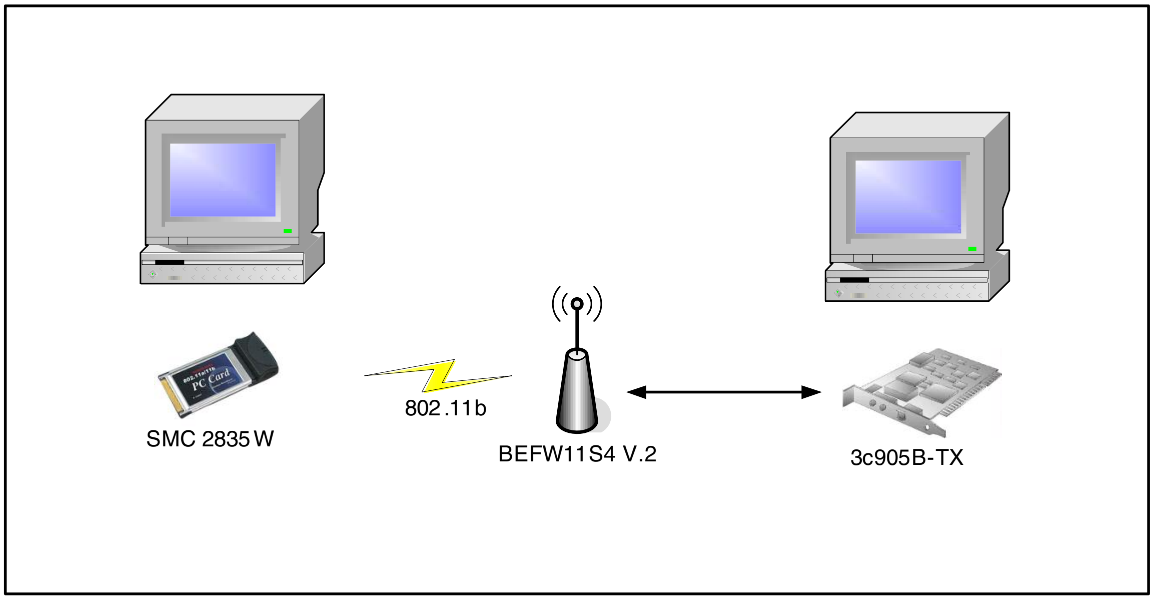


Figure 10.38 – 802.11g NIC and 802.11b Access Point Test Configuration

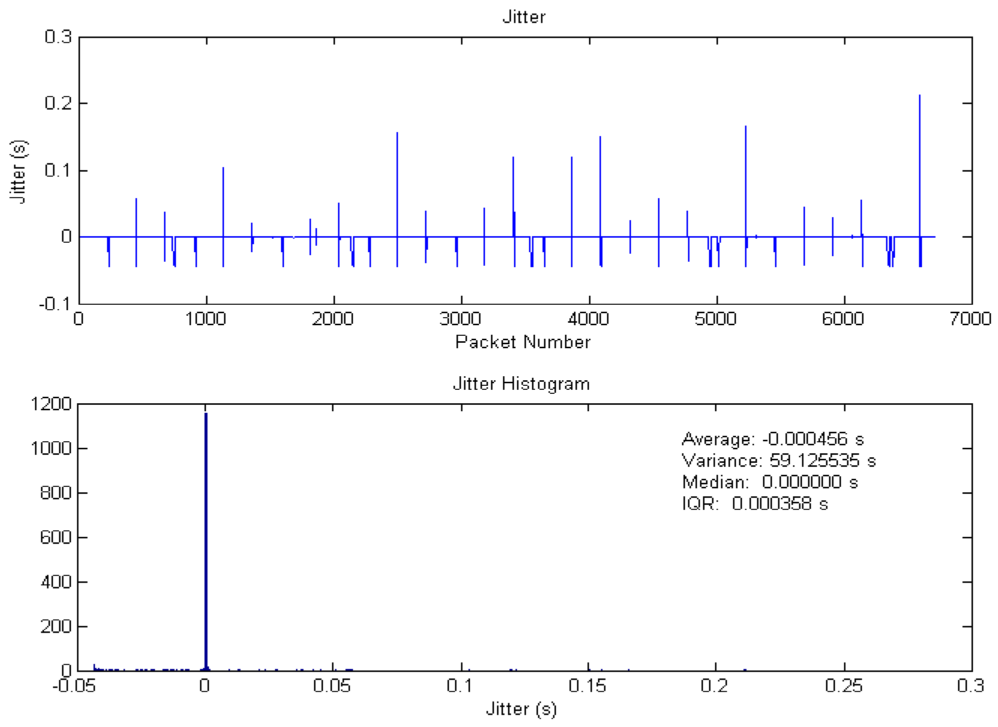


Figure 10.39 – 802.11b Access Point to 802.11g NIC Jitter

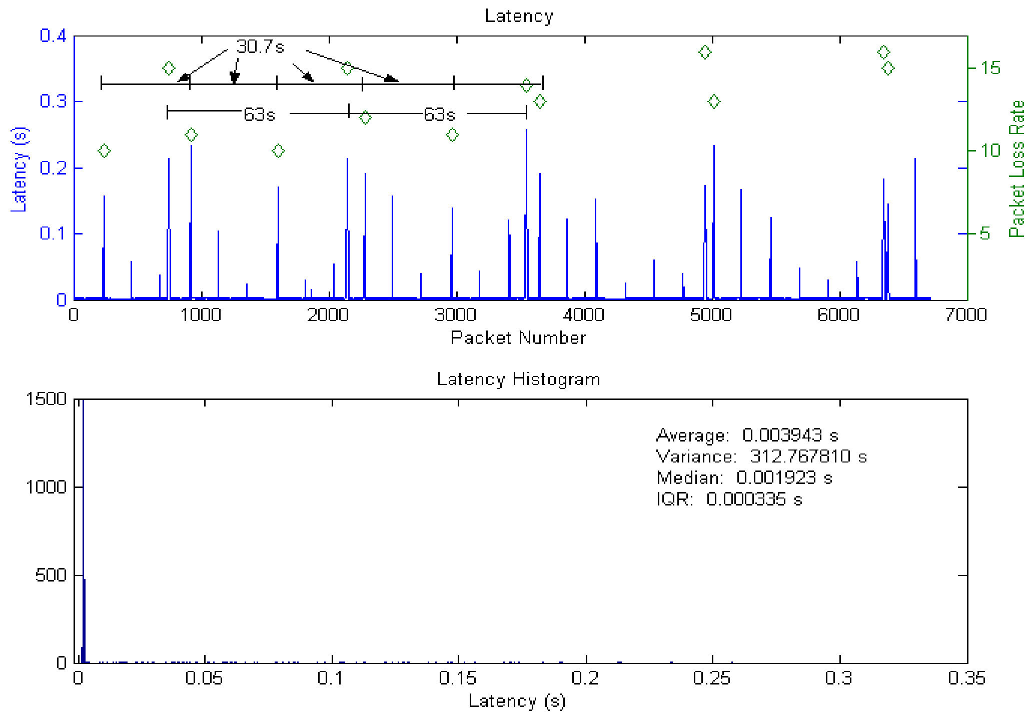


Figure 10.40 – 802.11b Access Point to 802.11g NIC Latency

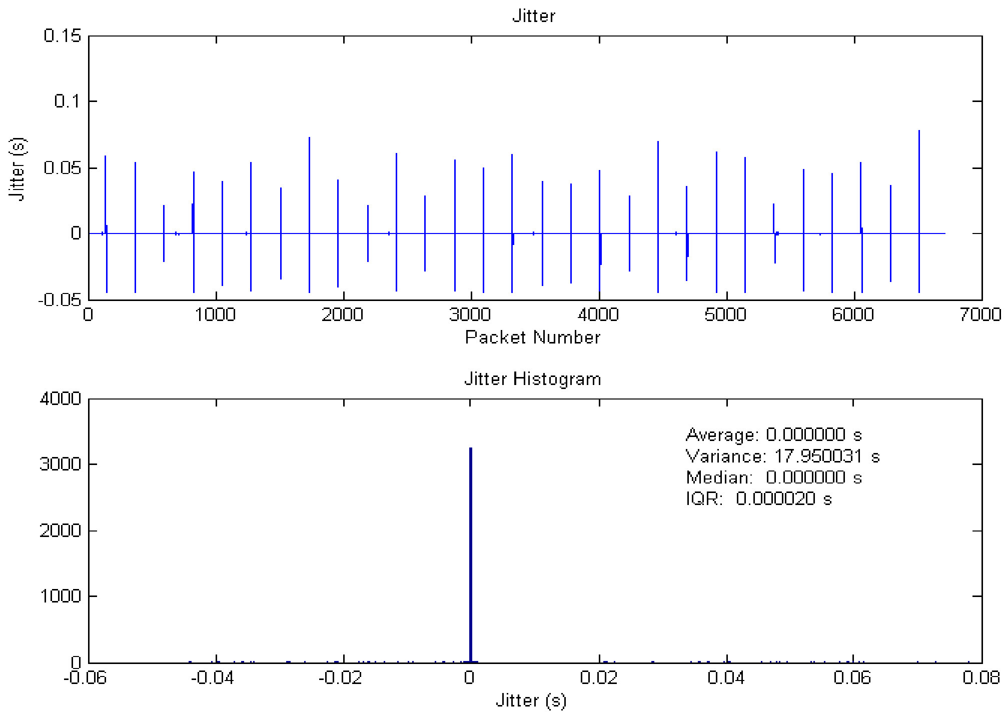


Figure 10.41 – 802.11g NIC to 802.11b Access Point Jitter

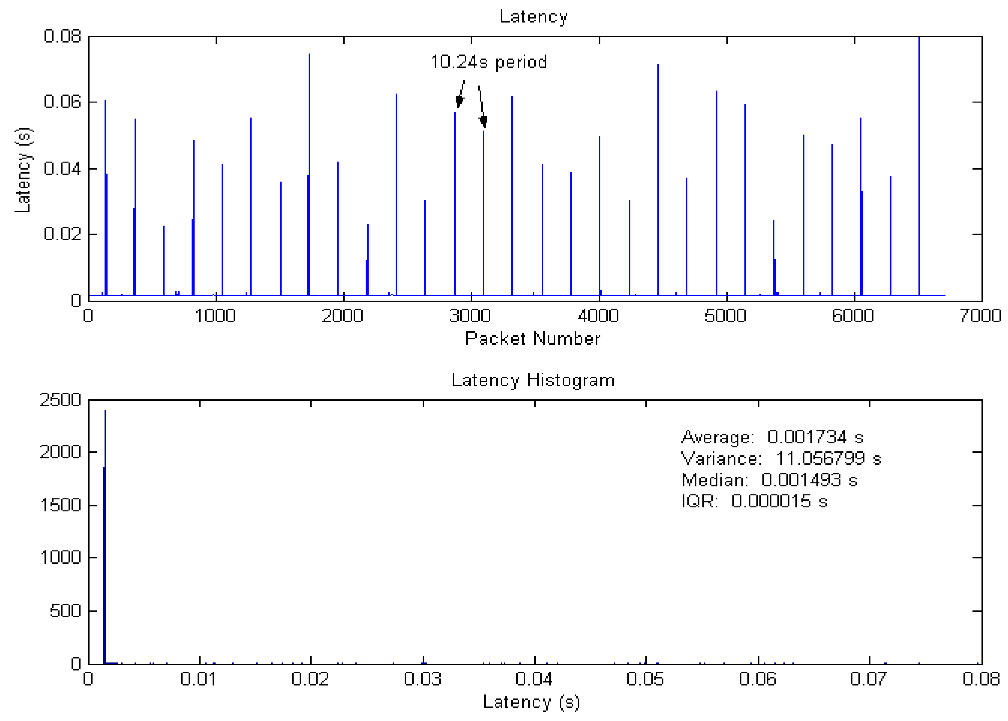


Figure 10.42 – 802.11g NIC to 802.11b Access Point Latency

In the case where an 802.11g NIC is using an 802.11b access point, many of the periodic delays seen in previous tests show up again. When the access point is transmitting, the 63s delays and packet losses appear again. This is again solved by disabling the wireless zero configuration. The 30.7s delays are three times the 10.24s delays. The 10.24s delays are seen when the NIC is transmitting. The source of these 10.24s delays is unknown.

Bluetooth is a reasonable technology for use in a VoIP system based on the characteristics measured with these tests. 802.11 configurations had much lower typical latencies than Bluetooth. Until the source of the large delay spikes can be identified and eliminated, 802.11 hardware that experienced those delays will perform poorly in a VoIP system. It is assumed that the existence of these delays is caused by some aspect of the hardware or software used in the configurations and that the delays are not simply inherent in 802.11g.

10.3. Cellular Data Networks

The primary reason to want VoIP traffic on a cellular data connection as opposed to using the cellular voice service is to encrypt the speech data. Without the support of a service provider, cellular data is the only option. These tests were run on the AT&T and Sprint cellular data networks, which use EDGE and CDMA2000 1xRTT, respectively. Enhanced Data rates for GSM Evolution (EDGE) is a GPRS-compatible data service capable of up to 384 Kbps. Code Division Multiple Access (CDMA) 1x Radio

Transmission Technology (RTT) is the first stage of the CDMA2000 deployment capable of up to 144 kbps.

10.3.1. Windows XP Connection Sharing Tests

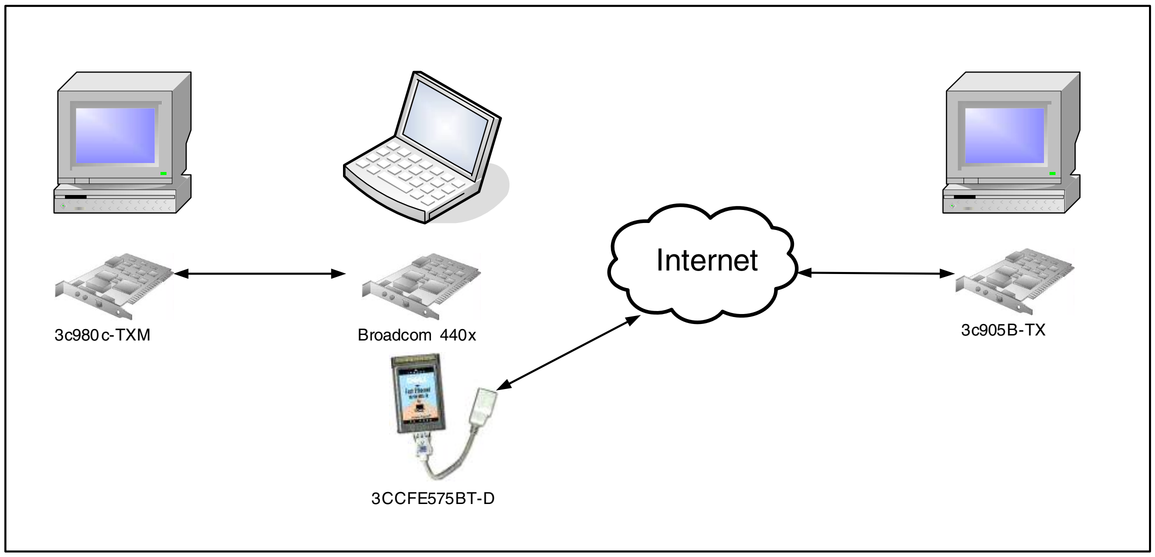


Figure 10.43 – Connection Sharing Test Configuration

Due to limitations in WinPcap, it is not possible to access a dial-up networking connection from within the WinPcap protocol driver. The CDMA driver for the Sprint SPHA620 phone is implemented as a dial-up networking device. This means that it is not possible to directly use the phone with the test software. To work around this, the two wireless data connections were configured on a separate machine and the connections were shared to an Ethernet port using Windows XP connection sharing. A standard Ethernet PCMCIA card was shared to identify the overhead associated with the connection sharing software. The connection sharing is a symmetric process that imposes an additional 150 μs delay on average, but can be impulsive up to 2 ms and in a few cases has had periodic impulsive spikes up to 1.6 ms with a period of 7.875 s. The source of these delays is unknown, but assumed to be caused by other operations on the Windows XP system. This additional noise is negligible because the delays typically seen in the wireless data networks are at least 2 orders of magnitude greater than the worst errors caused by the connection sharing.

10.3.2. Sprint CDMA 1xRTT Tests

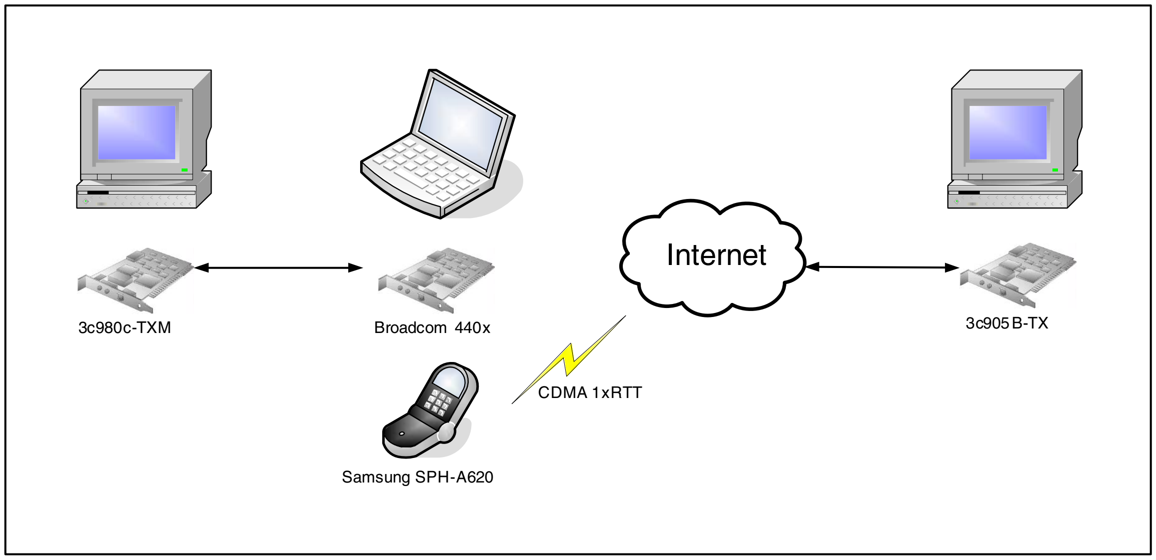


Figure 10.44 – CDMA Test Configuration

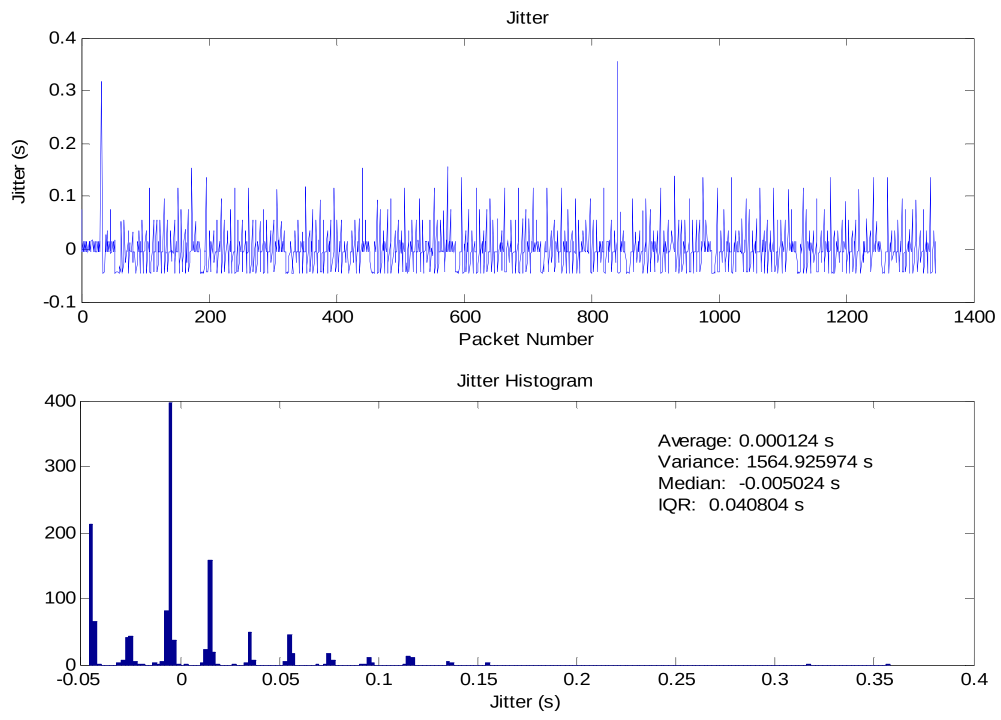


Figure 10.45 – CDMA to EDU Jitter (Upstream)

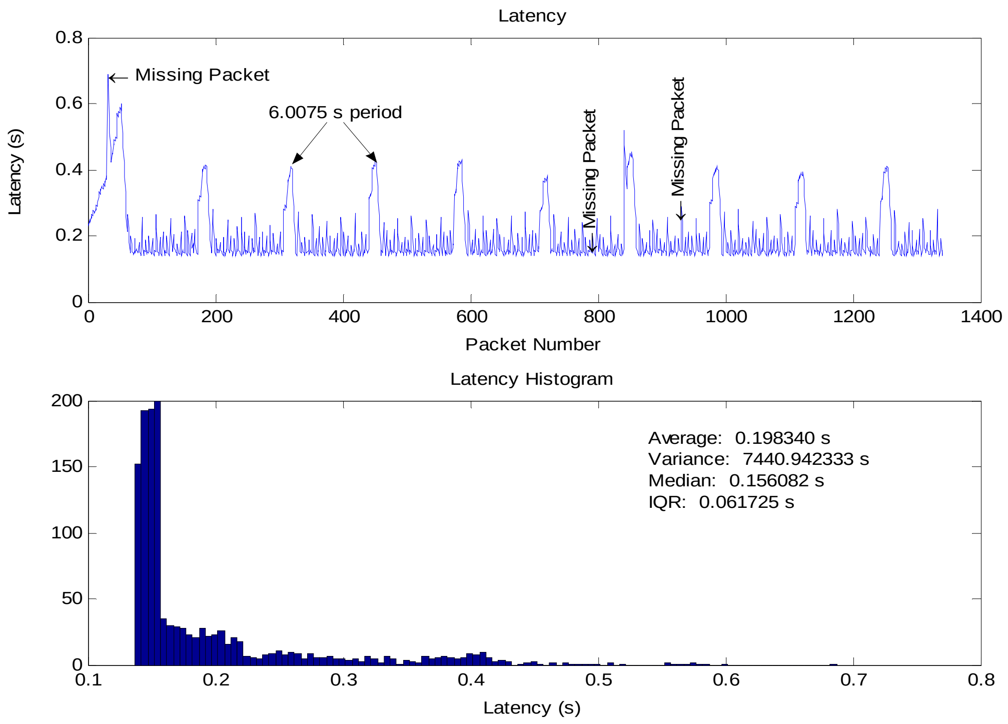


Figure 10.46 – CDMA to EDU Latency (Upstream)

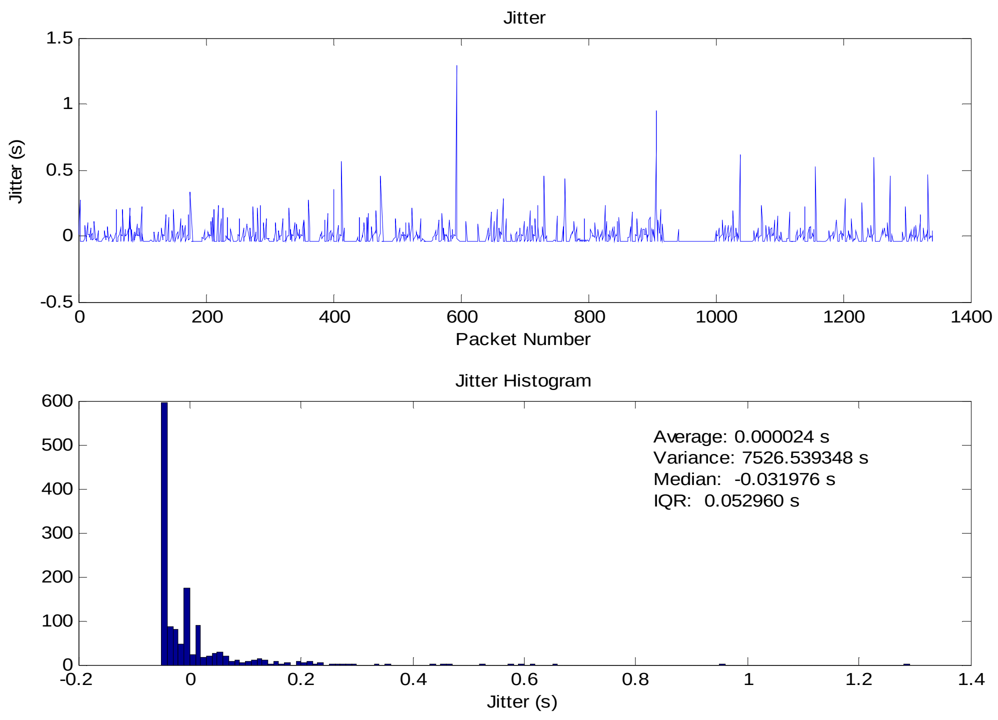


Figure 10.47 – EDU to CDMA Jitter (Downstream)

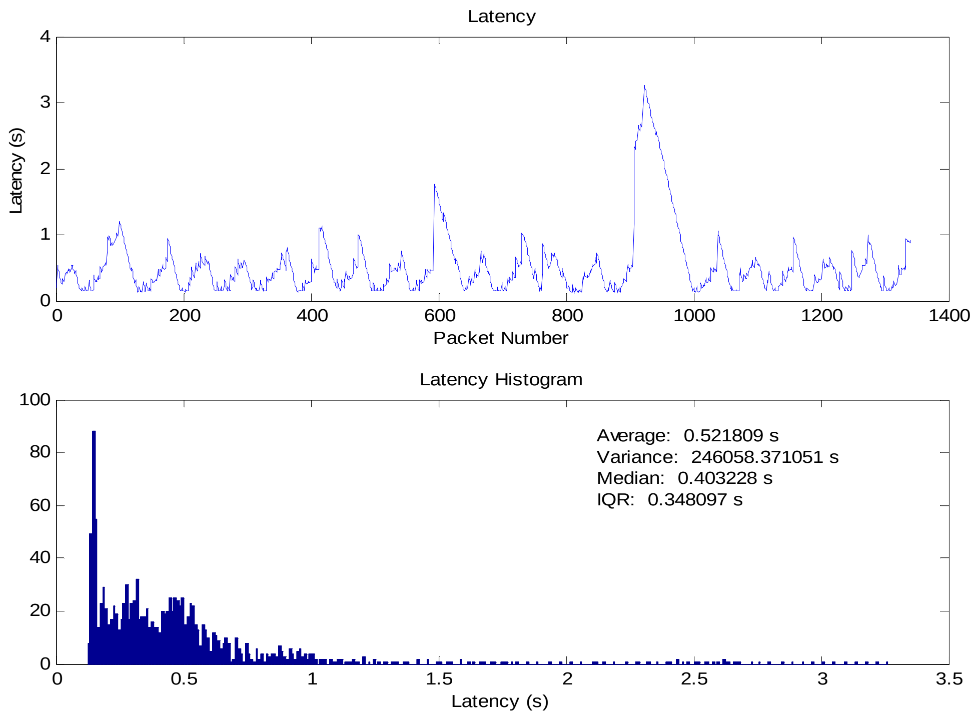


Figure 10.48 – EDU to CDMA Latency (Downstream)

The CDMA tests reveal considerable delays on the upstream network path, with levels reaching as high as 700ms but more often peaking at about 400ms. The jitter is typically below 150ms. The downstream is far worse than the upstream. On several occasions, latency as high as 3.25s was observed, though the average latency was approximately 500ms. The jitter was typically around 200ms. For this connection to be useful for VoIP, a relatively large jitter buffer will be required and the end-to-end delay experienced by the user will probably be unacceptably high.

10.3.3. AT&T Wireless EDGE Tests

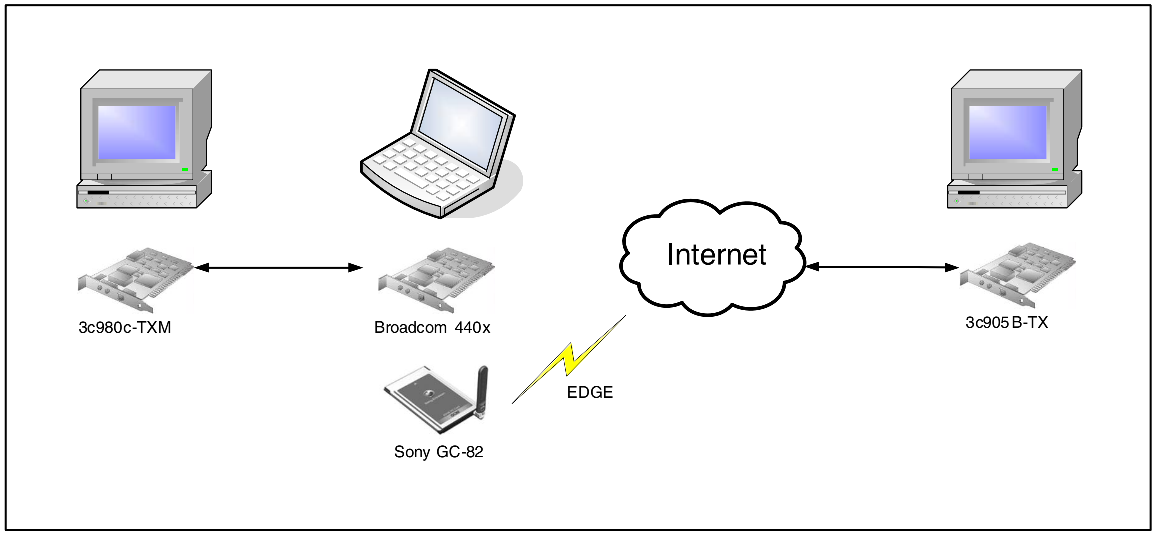


Figure 10.49 – EDGE Test Configuration

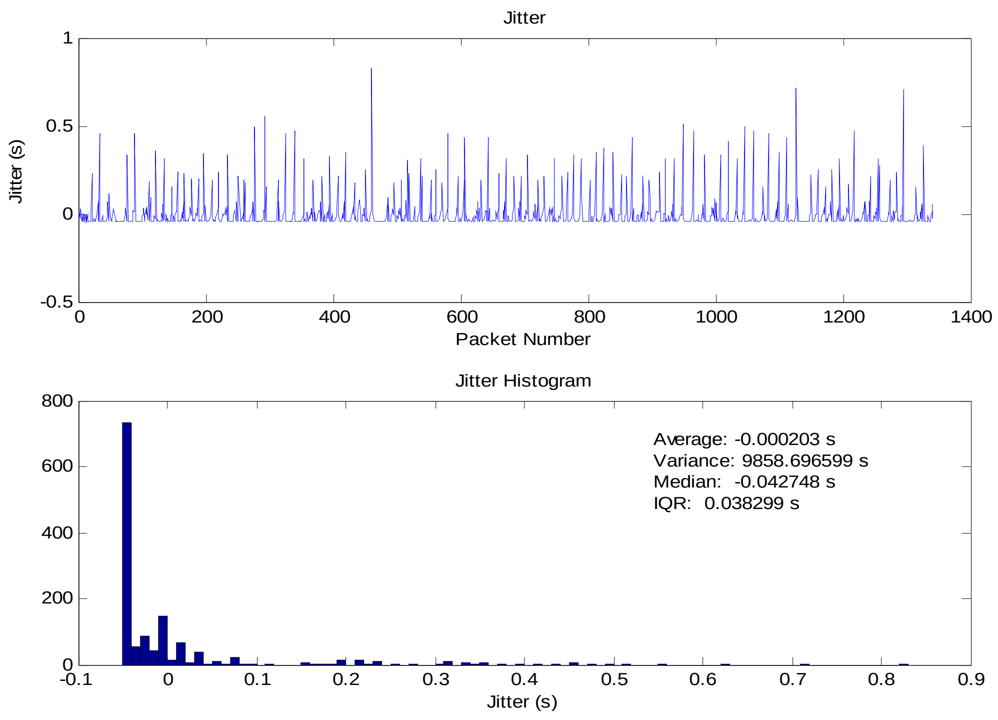


Figure 10.50 – EDGE to EDU Jitter (Upstream)

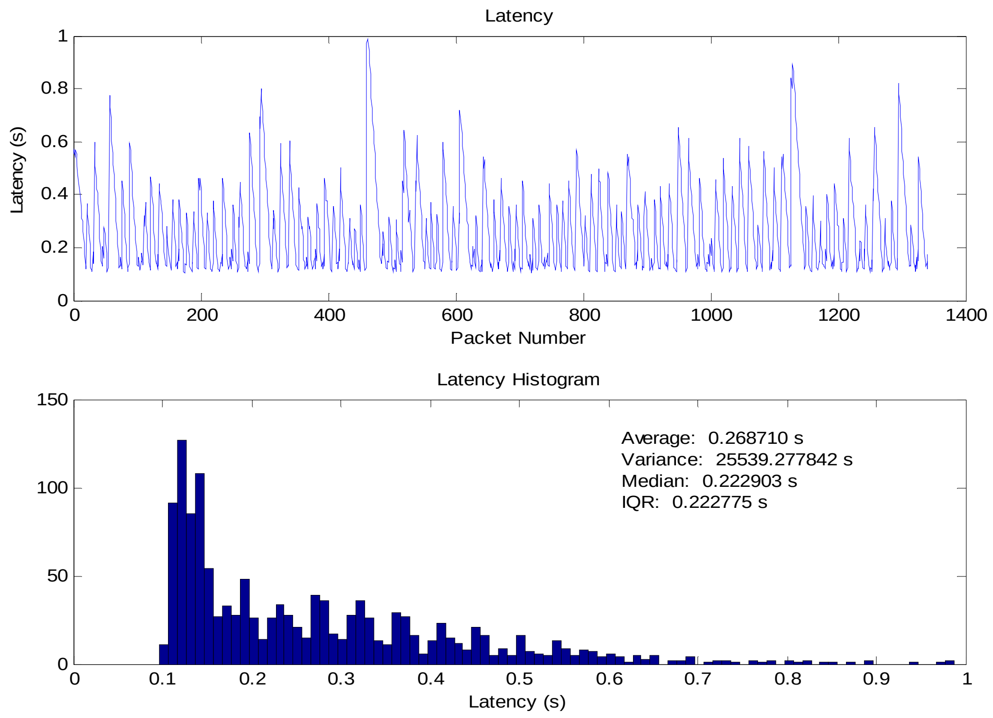


Figure 10.51 – EDGE to EDU Latency (Upstream)

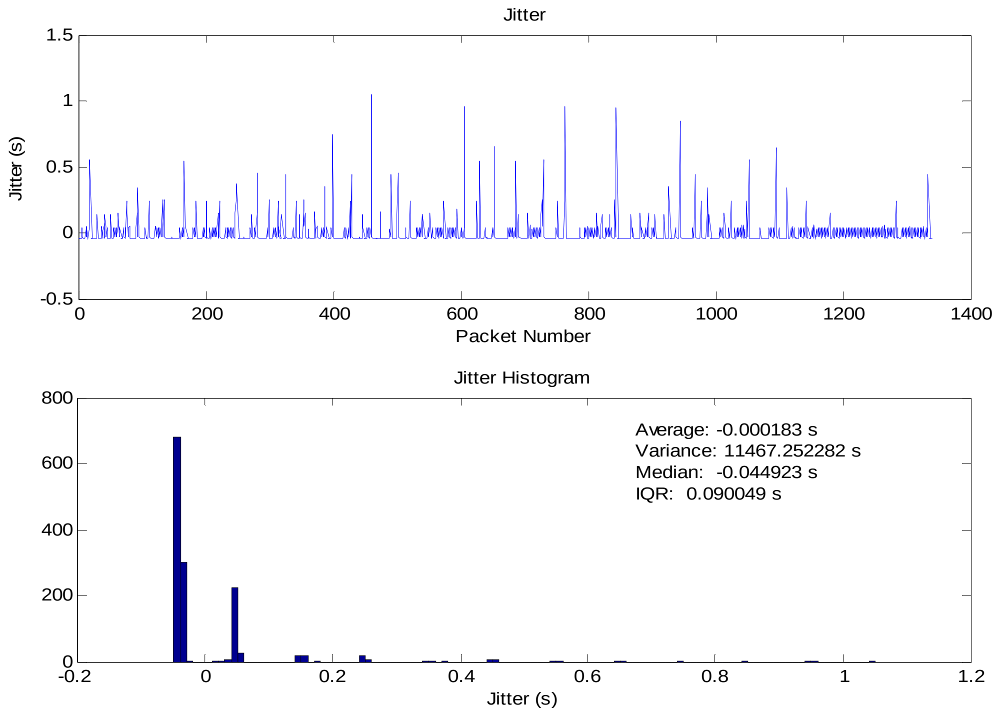


Figure 10.52 – EDU to EDGE Jitter (Downstream)

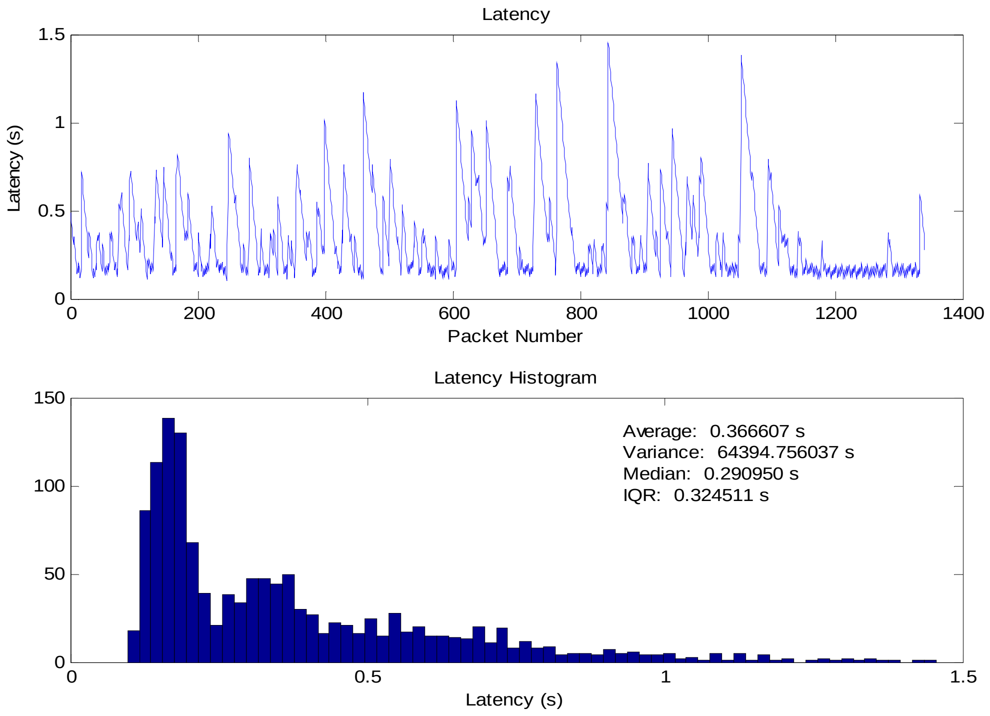


Figure 10.53 – EDU to EDGE Latency (Downstream)

The EDGE tests show large delay spikes just below 1s in the upstream channel and jitter as high as 825ms. The downstream was even worse with delay spikes as large as 10s in some cases and jitter over 1s. With performance like this, the quality of service will be greatly degraded. System will either suffer extreme end-to-end delay or the system will lose audio for a few seconds at a time, or possibly both. As such, it will probably not be a suitable network for VoIP.

Both of the cellular data services tested had poor performance as it pertains to VoIP. They may be useable if it is decided that large delays are permissible. The CDMA data was somewhat better than the EDGE service even though the EDGE service has a higher available bandwidth. In both cases, the upstream had better properties than the downstream which seems counter-intuitive. Perhaps 3G cellular data will be capable of providing the service required by VoIP in the future.

10.4. Concatenated Networks

It is often necessary for data to traverse many networks with differing properties before reaching its destination. In some cases concatenated networks can have effects other than simply additive properties, such as when TCP is used and self-clocking, the regulation of the packet transmission rate by the rate of acknowledgments, adjusts the transmission to the rate capable by the lowest bandwidth link. In the case of UDP and latency, the properties of concatenated networks should be additive. In this section, two configurations of concatenated networks will be discussed. The first is a connection through the AT&T EDGE data network to a computer on a Cox Cable connection. The second is a computer using Bluetooth to access Cox Cable Internet service to communicate with a computer on an educational link.

10.4.1. EDGE and Cable Concatenated Tests

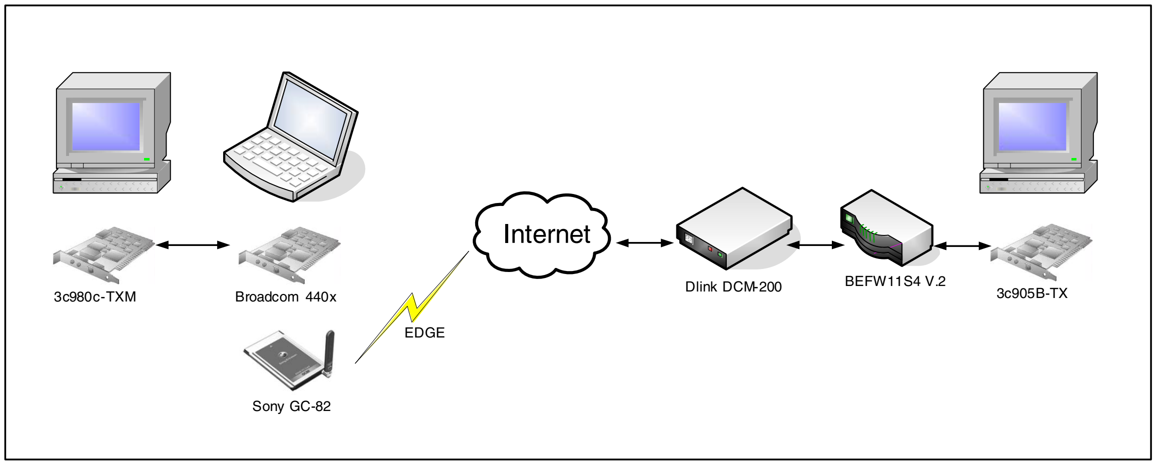


Figure 10.54 – EDGE and Cable Concatenated Test Configuration

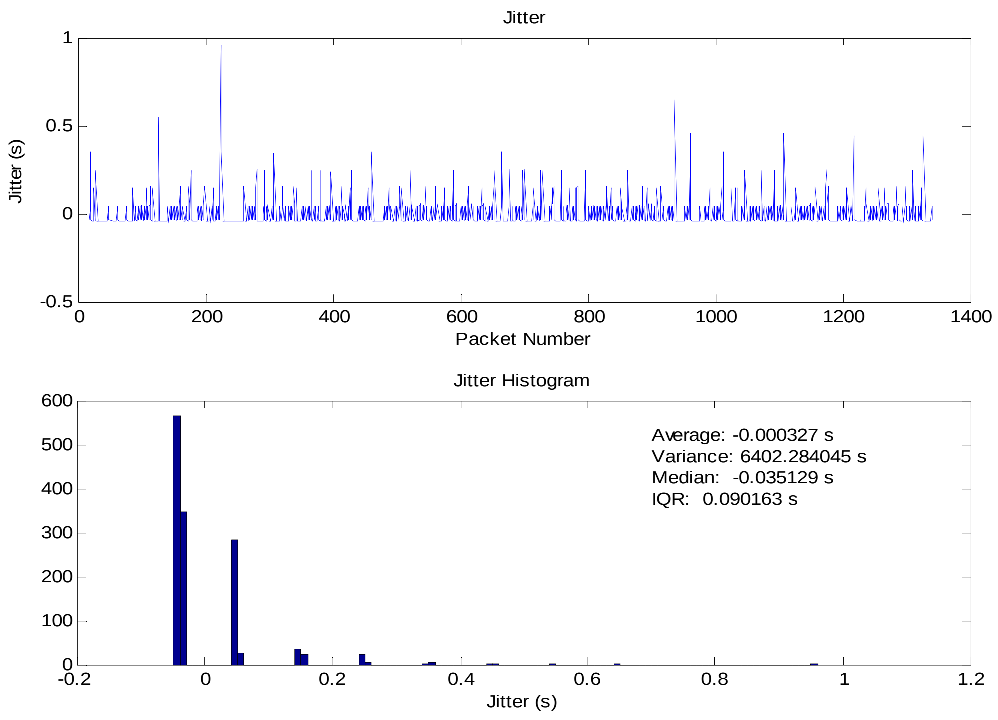


Figure 10.55 – CABLE to EDGE Jitter

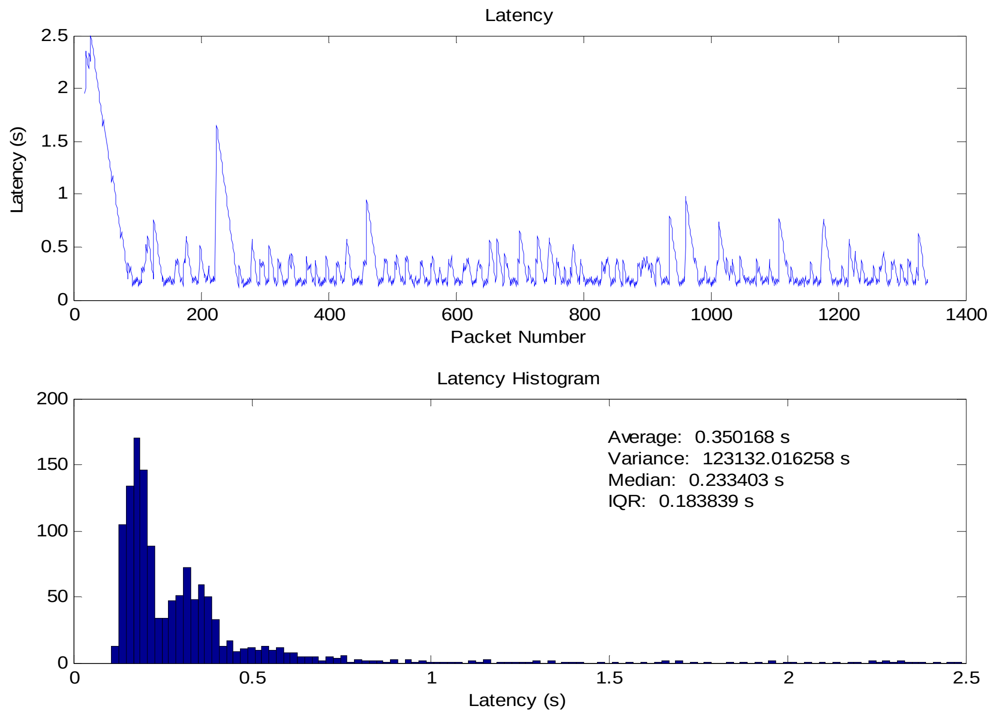


Figure 10.56 – CABLE to EDGE Latency

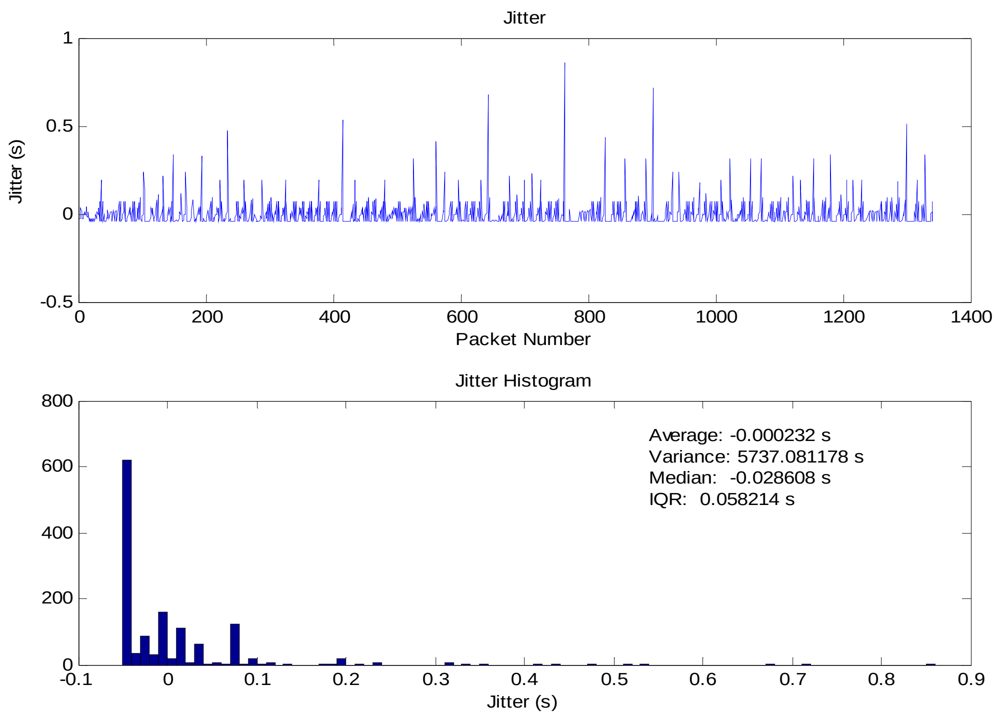


Figure 10.57 – EDGE to CABLE Jitter

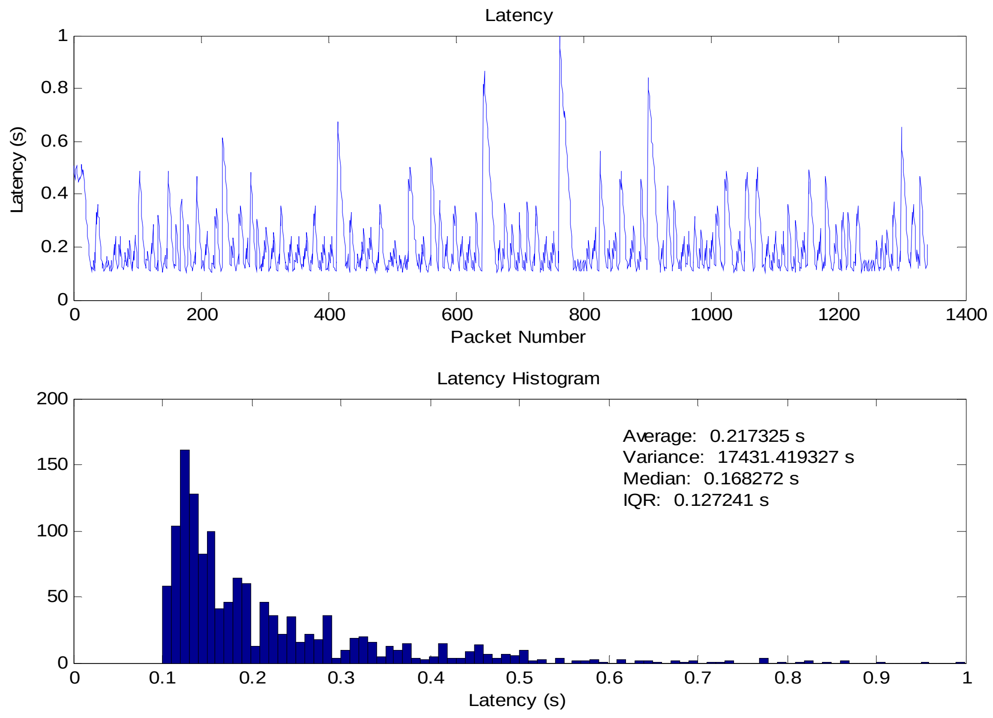


Figure 10.58 – EDGE to CABLE Latency

The tests that measured an EDGE connection to cable Internet essentially replicated the results of EDGE to an educational link. This is because the poor performance of the wireless connection overwhelms the effects of the cable connection. The most drastic is the data path from the EDGE device to the cable connection where EDGE averages latencies well over 200ms and the cable averages less than 10ms. The jitter histograms of the EDGE to cable and EDGE to EDU are almost identical. This is because, on the scale of the histograms including EDGE, the cable jitter histograms are impulsive. Because of the extent of the delay caused by EDGE, the cable connection's effects can be ignored.

10.4.2. Bluetooth and Cable Concatenated Tests

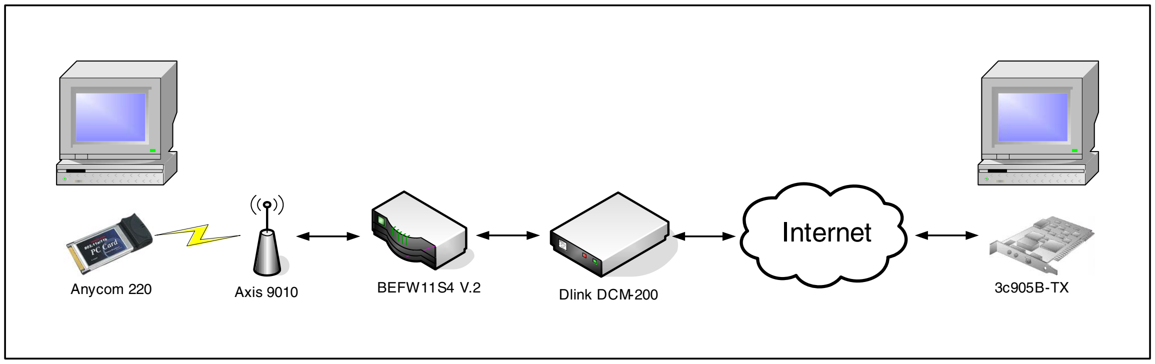


Figure 10.59 – Bluetooth and Cable Concatenated Test Configuration

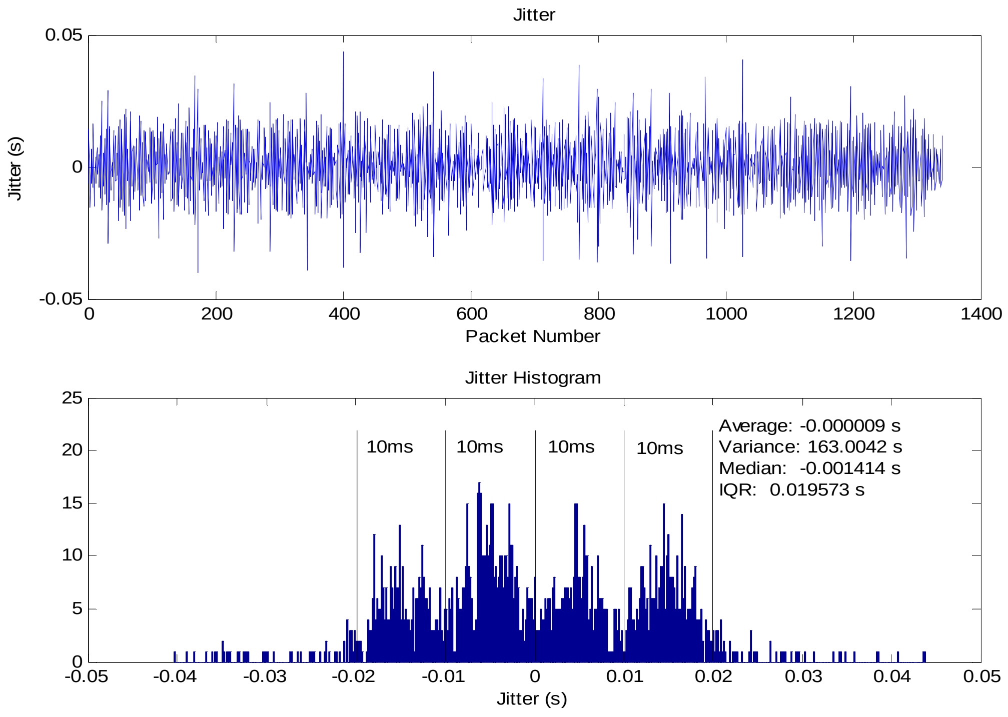


Figure 10.60 – Bluetooth to CABLE to EDU Jitter (Upstream)

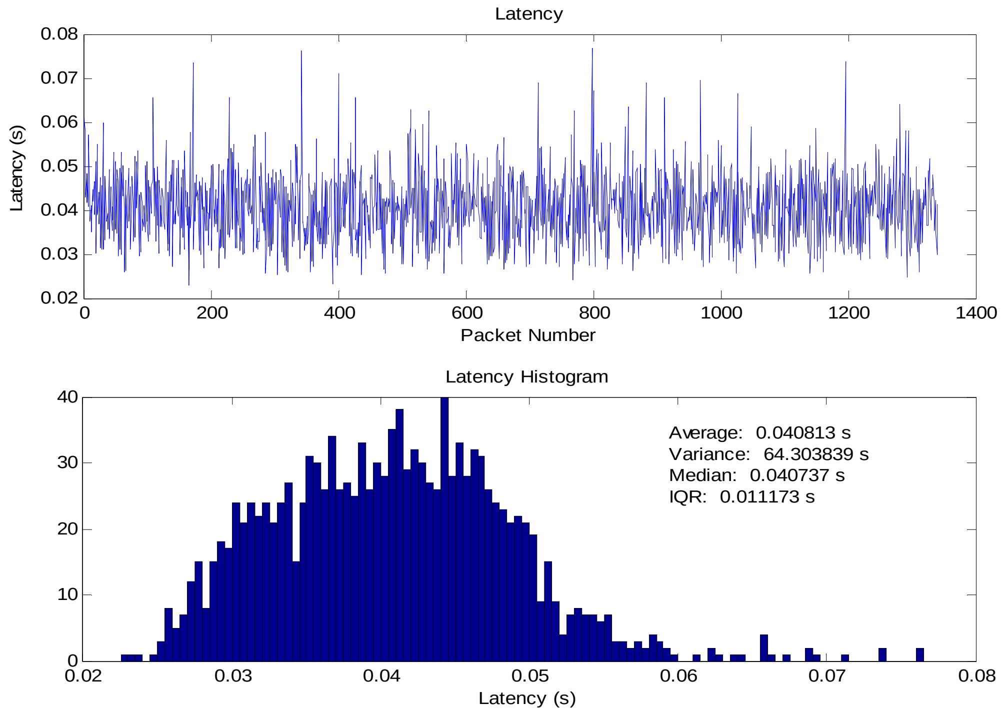


Figure 10.61 – Bluetooth to CABLE to EDU Latency (Upstream)

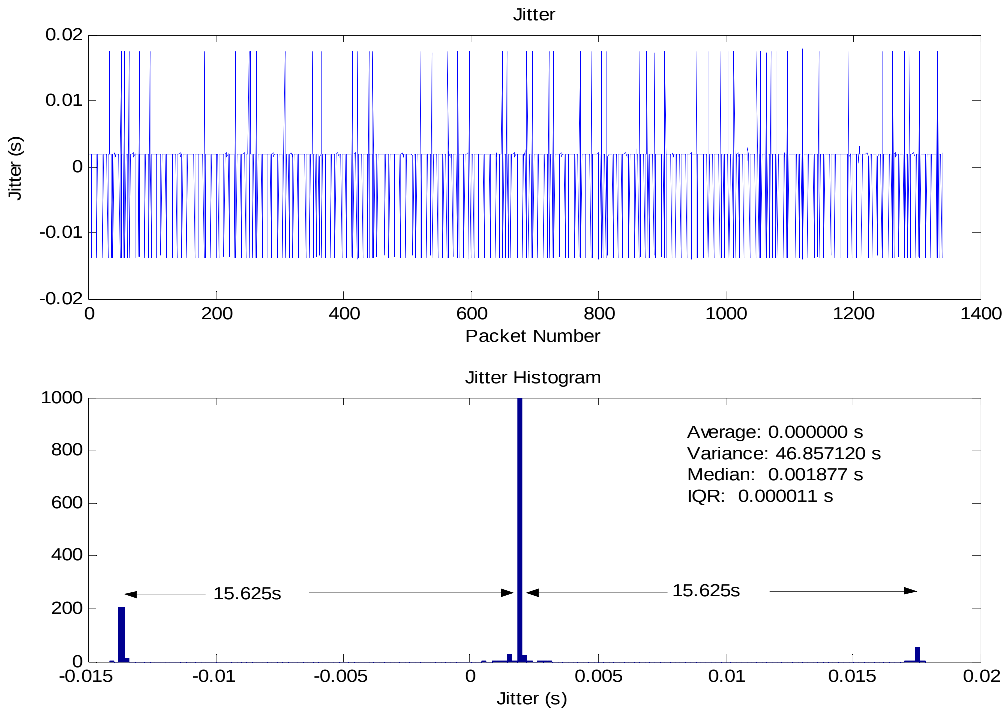


Figure 10.62 – EDU to CABLE to Bluetooth Jitter (Downstream)

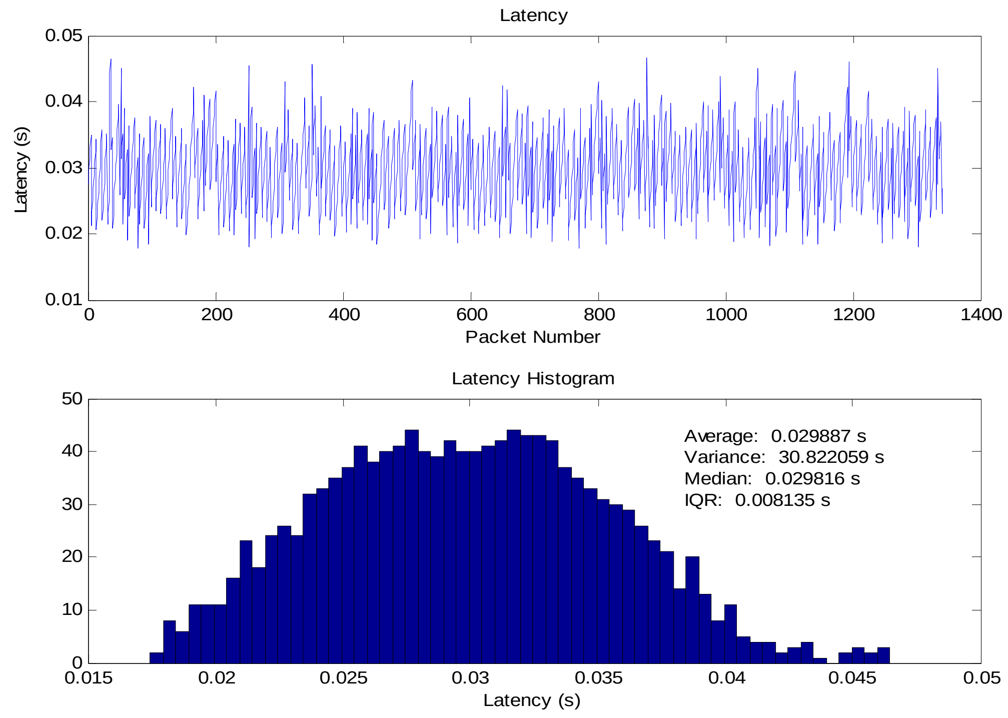


Figure 10.63 – EDU to CABLE to Bluetooth Latency (Downstream)

The concatenated network consisting of cable Internet service and Bluetooth shows very good additive properties. For the upstream case, the average latency is 40.8ms. The independent Bluetooth tests averaged about 22ms and the independent cable internet tests averaged about 18ms. This is very close considering the tests were not done in a controlled environment. The jitter also shows the additive properties of the concatenated network tests. The jitter histogram of the concatenated network should be convolution of the jitter histograms of the independent tests, which appears to be the case. In the downstream tests, the comparison between to independent networks and the concatenated networks agree as well. The average latency in the concatenated test is 30ms. The independent Bluetooth tests averaged about 19ms and the independent cable internet tests averaged about 10ms. These latencies when added closely match those of the concatenated test, even though the tests were not performed at the same time. The convolution of the jitter histograms matches as well and is easy to verify because the jitter histogram for the downstream cable Internet is impulsive (within ±0.5ms) when compared with the jitter histogram of the Bluetooth connection. There is an increase in the jitter spread, though its cause is unknown.

These two examples of concatenated networks support the hypothesis that the effects of latency and jitter are additive. They also demonstrate the repeatability of the system; its ability to provide accurate measurements of jitter and latency.

CHAPTER 11

CONCLUSIONS

From the tests discussed in Chapter 10, the capability that the system has to reveal detailed network properties is apparent. It can not only be used to profile the performance of a network link, but also to detect problems with network configuration that may be otherwise hidden. The testing system is versatile in that it can work with a wide variety of NICs but can also improve accuracy by using a specific NIC.

11.1. System Performance

The error of the system was measured on two machines and shown in Table 11.1. The baseline error is the error floor. The error was always seen to be at or above that level. If these errors are measured for a machine, they can be subtracted from the latencies to improve accuracy. The uncertainty of the measurements, the random part of the error, is how far above the baseline the error was observed. This is the range within which the measurements should not be considered valid. Every measurement should be assumed to have an error within that range.

Table 11.1 – System Error

	Protocol Baseline	e100b NIC Baseline	Protocol Uncertainty	e100b NIC Uncertainty
Dual-processor	42µs	7µs	40µs	10µs
Single-processor	37µs	12µs	50µs	20µs

11.2. Future Work

It is left as future work to validate the current validation techniques and the system as a whole using DAG cards. After the system is shown to be valid, it would be useful to add capability to store additional meta-data with the measurements, such as what the route of the test was, the NIC driver used, and any comments about the configuration. The route should also be monitored during the test by checking for a change in the time-to-live of the packets that are received. A new user interface would make the system significantly more user-friendly. It would also be useful to provide graphing capabilities within the application. Currently, data is all written to a file and then graphed with Matlab or Excel. In the testing, the packets were transmitted as close to 45ms apart as possible. This could be improved to include the jitter normally introduced by a VOIP system. This would be better than simply queuing the packets in user mode and letting the system perturb the transmission times as it normally would because doing so would not only be unrepeatable, but the computer load of the network test on the system would not accurately represent that of a VoIP system. The system should be expanded to support more than fixed-rate traffic generation so that it is applicable to other applications such as instant messaging.

REFERENCES

[1] "EndRun Technologies White Paper: UTC Time and Frequency Dissemination via the IS-95 CDMA Mobile Telecommunications Infrastructure," November 2000, [on-line], available from http://www.endruntechnologies.com/pdf/PTTI2000_WhitePaper.pdf; Internet; accessed May 28, 2005.

[2] "Guidelines For Providing Multimedia Timer Support," 10-20-2002, [on-line], available from http://www.microsoft.com/whdc/system/CEC/mm-timer.mspx?pf=true#img1; Internet; accessed 05-30-2005.

[3] "Intel I/O Controller Hub 7 (ICH7) Family Datasheet," 05-2005, [on-line], available from http://download.intel.com/design/chipsets/datashts/30701301.pdf; Internet; accessed 05-30-2005.

[4] "Microsoft Windows Driver Development Kits," [on-line], available from http://www.microsoft.com/ddk/; Internet; accessed May 12, 2005.

[5] "The W3IWI/TAPR TAC-2 (Totally Accurate Clock) Project," [on-line], available from http://www.tapr.org/kits_tac2.html; Internet; accessed July 09, 2005.

[6] "Windows Packet Capture Library," 11-04-2004, [on-line], available from http://winpcap.polito.it/; Internet; accessed 03-06-2005.

[7] G. Almes, S. Kalidindi, and M. Zekauskas, "A One-way Delay Metric for IPPM," September, [on-line], available from http://www.ietf.org/rfc/rfc2679.txt; Internet; accessed June 08, 2005.

[8] T. M. Chen and L. Hu, "Internet Performance Monitoring," *Proc. of IEEE, August 2002*, 2002.

[9] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Measurement Considerations for Assessing Unidirectional Latencies," *Internetworking: Research and Experience*, vol. 4 (3), pp. 121-132, 1993.

[10] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson, "Design Principals for Accurate Passive Measurement," *Proc. of the Passive and Active Measurement Workshop*, 2000.

[11] D. Constantinescu, P. Carlsson, and A. Popescu, "One-Way Transit Time Measurements," Blekinge Institute of Technology, Research Report 2004:06.

[12] J. P. Curtis, J. Cleary, A. McGregor, and M. Pearson, "Measurement of Voice over IP Traffic," *Proc. of PAM2000 Workshop on Passive and Active Networking*, 2000.

[13] L. Degioanni, M. Baldi, F. Risso, and G. Varenni, "Profiling and Optimization of Software-Based Network-Analysis Applications," *Proc. of the 15th IEEE*

Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2003), 2003.

[14] C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," [on-line], available from http://www.ietf.org/rfc/rfc3393.txt; Internet; accessed June 08, 2005.

[15] S. Donnelly, I. Graham, and R. Wilhelm, "Passive Calibration of an Active Measurement System," *Proc. of PAM2001, Workshop and Passive and Active Measurements*, 2001.

[16] I. D. Graham, S. F. Donnelly, S. Martin, J. Martens, and J. G. Cleary, "Nonintrusive and Accurate Measurement of Unidirectional Delay and Delay Variation on the Internet," *Proc. of INET'98*, 1998.

[17] G. Harris, [on-line], available from http://www.ethereal.com/lists/ethereal-dev/200406/msg00355.html; Internet; accessed 07-02-2005.

[18] V. Jacobson, C. Leres, and S. McCanne, "libpcap," [on-line], available from http://www.tcpdump.org/; Internet; accessed February 18, 2005.

[19] J. Jeong, S. Lee, and Y. Kim, "Design and Implementation of One-way IP Performance Measurement Tool," *ICOIN 2002*, 2002.

[20] S. Kalindidi and M. J. Zekauskas, "Surveyor: An Infrastructure for Internet Performance Measurements," *Proc. of INET*, 1999.

[21] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," *Proc. of the 1993 Winter USENIX Technical Conference*, 1993.

[22] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," *Proc. of the SIGCOMM IMW*, 2001.

[23] Microsoft Inc., "Basic Operation of the Windows Time Service," [on-line], available from http://support.microsoft.com/kb/q224799/; Internet; accessed.

[24] D. L. Mills, "Network Time Protocol (Version 3): Specification, Implementation and Analysis." Menlo Park, CA: Network Information Center, SRI International, 1992, pp. RFC 1305.

[25] D. L. Mills, "Simple Network Time Protocol (SNTP) Version 4," [on-line], available from http://www.apps.ietf.org/rfc/rfc2030.html; Internet; accessed June 8, 2005.

[26] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and Removal of Clock Skew from Network Delay Measurements," *Proc. of the IEEE INFOCOM Conference on Computer Communications*, 1999.

[27] J. Nilsson, "Implement a Continuously Updating, High-Resolution Time Provider for Windows," [on-line], available from http://msdn.microsoft.com/msdnmag/issues/04/03/HighResolutionTimer/default.aspx; Internet; accessed 03-09-2005.

[28] W. Oney, *Programming the Microsoft Windows Driver Model*, 2 ed: Microsoft Press, 2003.

[29] A. Pàsztor and D. Veitch, "PC Based Precision Timing without GPS," *Proc. of ACM SIGMETRICS*, 2002.

[30] A. Pàsztor and D. Veitch, "A Precision Infrastructure for Active Probing," *Proc. of PAM2001, Workshop and Passive and Active Measurements*, 2001.

[31] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," in *Computer Science*, vol. PhD. Berkeley, CA: University of California, Berkeley, 1997, pp. 409.

[32] V. Paxson, "On Calibrating Measurements of Packet Transit Times," *Proc. of ACM SIGMETRICS*, 1998.

[33] V. Paxson, "Strategies for Sound Internet Measurement," *Proc. of the 4th ACM SIGCOMM on Internet measurement*, 2004.

[34] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP Performance Metrics," May 1998, [on-line], available from http://www.ietf.org/rfc/rfc2330.txt; Internet; accessed June 08, 2005.

[35] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proc. of IEEE INFOCOM*, 1994.

[36] F. Risso and L. Degioanni, "An Architecture for High Performance Network Analysis," *Proc. of the 6th IEEE Symposium on Computers and Communications (ISCC 2001)*, 2001.

[37] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," in *RFC 3550*, 2003.

[38] S. Sharma, "Analysis of 802.11b MAC: A QoS, Fairness, and Performance Perspective," *ArXiv Computer Science e-prints*, 2004.

[39] Symmetricomm Inc., *BC637PCI Time and Frequency Processor - User's Guide - Revision J*, 2003.

[40] H. Uijterwaal and O. Kolkman, "Internet Delay Measurments using Test Traffic: Design Note," RIPE NCC, Tech Report RIPE-158, June 1997.

[41] D. Veitch, S. Babu, and A. Pàsztor, "Robust Synchronization of Software Clocks Across the Internet," *Proc. of the 4th ACM SIGCOMM on Internet measurement*, 2004.

[42] L. Zhang, Z. Liu, and C. H. Xia, "Clock Synchronization Algorithms for Network Measurements," *Proc. of the IEEE INFOCOM Conference on Computer Communications*, 2002.

APPENDIXES

APPENDIX A

INTRODUCTION TO NETWORKING

Networking between computers has been defined by the general model called the Open Systems Interconnection (OSI) model. This model defines the seven layers shown in Table A.1. These layers separate the network into different logical functionalities that are not necessarily separate in implementation. The layers relevant to this work are discussed in the following sections.

Table A.1 – OSI Model Layers

Layer 7	Application Layer
Layer 6	Presentation Layer
Layer 5	Session Layer
Layer 4	Transport Layer
Layer 3	Network Layer
Layer 2	Data Link Layer
Layer 1	Physical Layer

A.1. Physical Layer

The physical layer simply provides the connection from one piece of hardware to another. The most common is the connection from a Network Interface Card (NIC) in a PC to a port on an Ethernet switch. It defines the electrical standard required to communicate.

A.2. Data Link Layer

The data link layer allows multiple machines to communicate using unicast or broadcast packets. At the data link layer, a NIC for Ethernet is assigned a Medium Access Control (MAC) address by the card manufacturer, but it can typically be overridden by the PC. This address is used to directly communicate with another NIC on the same Local Area Network (LAN). If a packet is sent to the reserved address ff:ff:ff:ff:ff:ff it is transmitted to every NIC on the LAN. A LAN is distinguished from a Wide Area Network (WAN) in that all computers are separated only by switches or passive devices that will not make level-3 (Network Layer) routing decisions and hence will not limit Address Resolution Protocol (ARP) traffic.

A.3. Network Layer

The network layer allows devices to communicate across vast networks having limitless topologies. This is possible because devices called routers direct traffic through various network links in the Internet until the packet arrives at its destination. At the network layer, an Internet Protocol (IP) address can be bound to a NIC. This address is typically a unique address on the Internet assigned by a Dynamic Host Configuration Protocol (DHCP) server. Address ranges for these DHCP servers are assigned by the Internet Assigned Numbers Authority (IANA). The IP address could however be in one of three non-routable address ranges, shown in Table A.2.

Table A.2 – Non-Routable IP Address Ranges

	Class	**Address Range**
Range 1	Class A	10.0.0.0 - 10.255.255.255
Range 2	Class B	172.16.0.0 - 172.31.255.255
Range 3	Class C	192.168.0.0 - 192.168.255.255

Addresses from these three ranges should always be ignored by a router. This allows for addresses in the non-routable ranges to be reused in many independent LANs without causing conflict. Of course, to allow the computers with these non-routable addresses to communicate with other computers on the internet, a process such as Network Address Translation (NAT) must be performed by the router, as is further explained in Section 1.2.2.

A.4. Transport Layer

At the transport layer, a protocol such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), or Real-time Transport Protocol (RTP) organizes and manages the transmission of data from one port on a computer to another port on another computer. TCP is a reliable transport protocol, which means that the data is guaranteed to arrive uncorrupted and in order at the destination. The packet header and the payload are protected by a 32-bit checksum. If the packet arrives in error, the receiver requests that the data be retransmitted. If a packet or its acknowledgment is lost entirely, a timer will expire, causing a retransmission of the missing packet. A fast retransmission can also happen if three duplicate acknowledgments are received, indicating the missing packet.

For a real-time streaming application like Voice-over-IP, reliable transport is undesirable because the data being transported has a lifetime associated with it. Beyond that lifetime, the data is invalid and is a waste of bandwidth. This is discussed further in Chapter 2. UDP is more desirable for Voice-over-IP because it has a small overhead (8-byte header instead of TCP’s 28-byte header) and does not retransmit when an error is detected. In both cases the data is protected by a checksum in the header. RTP is a transport layer protocol that is designed to use UDP and provide additional information in its 14-byte header such as a time stamp, a sequence number, the type of data, the synchronization source, and a list of the contributing sources. Although RTP has more overhead, it provides information to more reliably reconstruct the streams in the receiving application. For the purpose of this work, I will focus only on UDP because it is the base transport protocol used for Voice-over-IP. I will also investigate a slightly modified implementation of UDP, which protects only the header with a checksum.

A.5. Address Resolution Protocol

When attempting to send a packet to an IP address, the network subsystem must have a way to find out what MAC address to send the packet to. The ARP sends a data-link-layer broadcast on the LAN asking for the MAC address of the NIC to which an IP address is bound. If a NIC with that IP address is on the LAN, it will respond with a message stating its MAC address. This information is added to the ARP cache table so that the next time the computer would like to send a packet to that same IP address it can simply look the address up in its table without having to send another ARP request. If there is no response to a broadcast ARP request, it can be assumed that the IP address is not bound to any NIC on the LAN. If this is the case, the packet should be sent to the default gateway. The IP address of the default gateway must now be looked up in the ARP cache or, if not there, a broadcast requesting its MAC address must be sent. The packet is then sent to the MAC address of the default gateway.

APPENDIX B

INTRODUCTION TO ROUTING

Routing is used to direct packets from one LAN to another. There are several routing protocols and many backbone transports that exist across the Internet, but for the purpose of this work, only the immediate routing that must be considered by the network performance application is relevant.

B.1. Default Gateway

When a computer is not located on the same LAN as another computer it is attempting to send a packet to, the sending computer must instead send the packet to its default gateway. The default gateway is responsible for determining the next hop that the packet must make, and the next router for the next hop, etcetera, until the packet reaches its destination. The IP address of the default gateway is typically received from the DHCP server when the computer's IP address is assigned.

B.2. Network Address Translation

In most cases, the computers on a home LAN do not use routable IP addresses. They typically use non-routable IP addresses that are assigned by the DHCP server in a small router, selected from the ranges in Table 1.2. These routers also do Network Address Translation (NAT) when it serves as a gateway between the computers on that

LAN and other computers on the Internet. To perform the NAT, the router will replace the IP address in the header of the packet with its own external IP address and store a record of the packet header. This establishes a route between that remote server and the internal machine based on the source and destination ports. When a packet is received from a server on the external interface of the gateway, the source IP address is compared with all existing routes. If the source IP address, the source port, and the destination port match a route that is stored, the internal IP address that was stored in the route replaces the destination IP address in the header, and the packet is transmitted on the internal LAN to that machine.

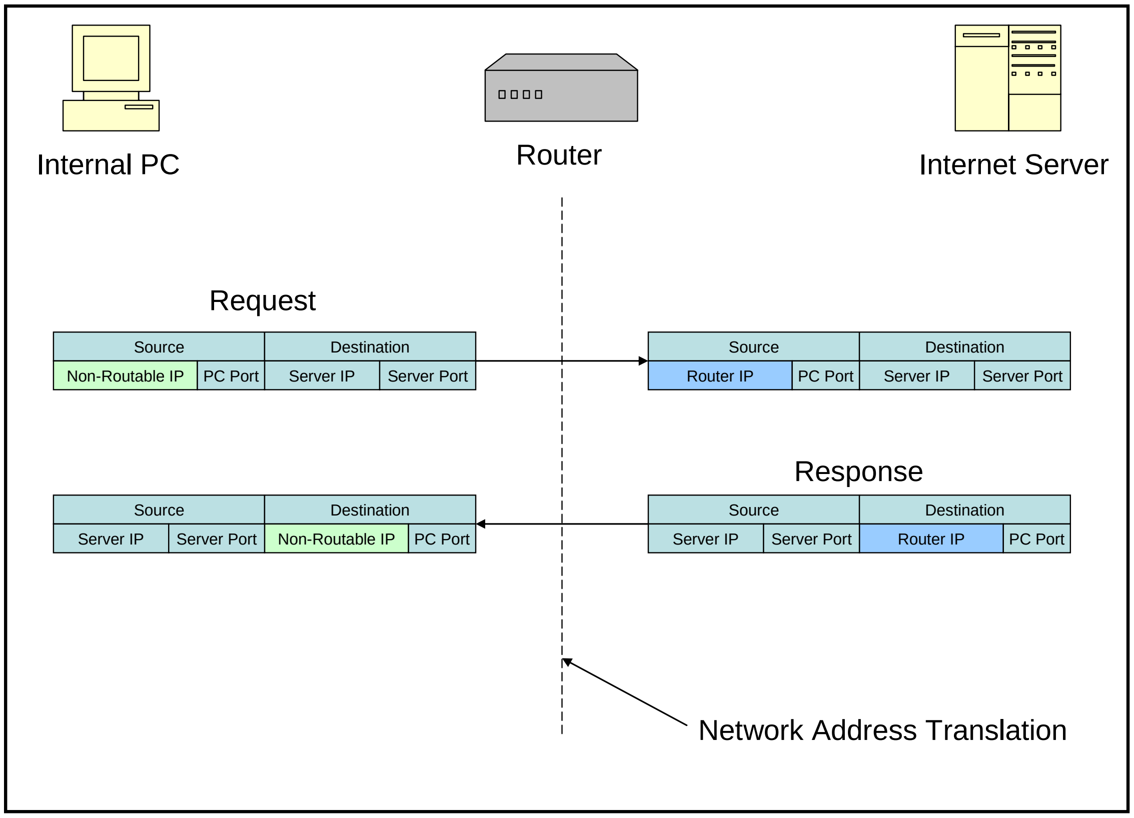


Figure B.1 – Network Address Translation

APPENDIX C

INTRODUCTION TO WINDOWS DEVICE DRIVERS

There are many differences between developing device drivers and developing application software. This section does not attempt to be a reference for device driver development but only to identify those aspects that are required to understand the discussions in Chapter 4. For a full reference, see Walter Oney's book Programming the Microsoft Windows Driver Model [28].

C.1. User Mode versus Kernel Mode

Typical application software runs in user mode. By contrast, most device drivers run in the context of kernel mode along with the operating system. Microsoft Windows 3.0 was the first operating system which made that distinction [28]. Kernel mode software is privileged, trusted software that has the ability to access any memory or resource that it pleases. This means that bugs in a device driver can quite easily be fatal to the operating system. It also means that there are many restrictions which the driver must impose upon itself. Failure to do so will often corrupt the operating system.

C.2. Interrupt Request Level

Within the kernel, software can run at different priorities, called interrupt request levels (IRQL). A program cannot be interrupted by another program running at an equal or lower IRQL. Only two levels are needed discussion here: `PASSIVE_LEVEL` and `DISPATCH_LEVEL`. All user-mode programs run at `PASSIVE_LEVEL` as do many kernel functions. The thread scheduler, which is responsible for preempting threads at the expiration of a time slice, runs at `DISPATCH_LEVEL`. A time slice is an allotted amount of time that a selected thread is allowed to execute before it is preempted and another thread is selected for execution. This allows many programs to play nicely with one another and share CPU time whether they want to or not. Because the thread scheduler runs at `DISPATCH_LEVEL`, any driver code running at `DISPATCH_LEVEL` or higher cannot be preempted. This means that, to maintain system integrity and responsiveness, any software running at `DISPATCH_LEVEL` or higher must voluntarily relinquish control of the CPU as soon as possible.

C.3. I/O Control Operations

One of the primary methods of communicating with a device driver is through I/O Control operations. When a program calls an I/O Control, the system generates an I/O Request Packet (IRP) which contains the control code, the data buffers, and their sizes. The IRP is then passed to the driver which executes the function associated with the control code.

These can either be accessed synchronously or asynchronously. If the I/O Control is accessed synchronously, then in the event that the driver or the hardware it is controlling cannot complete a request immediately, the driver will retain control of the CPU until it is able to complete the request. This means that the software that calls the

I/O Control will be blocked, waiting for control to be returned. If the I/O Control is accessed asynchronously, then in the event of a request that cannot be completed immediately, a `STATUS_PENDING` error code will be returned to notify the calling code that the request is being handled but is not yet complete. The calling program is then free to execute other code and later check the status of the request.

C.4. Direct-Call Interface

A direct-call interface allows a driver to export a standard set of function pointers to be called directly by another driver, instead of creating an IRP and adding it to the driver's queue. This interface has much lower overhead than the I/O Control interface and no restriction on IRQL, though it is less controlled and not available to user-mode applications.

VITA

Joseph Ryan Hershberger

Candidate for the Degree of

Master of Science

Thesis: ACCURATE AND PRECISE NETWORK PERFORMANCE TESTING IN WINDOWS 2000

Major Field: Electrical Engineering

Biographical:

Education: Graduated from Jenks High School, Jenks, Oklahoma in May 1998; received Bachelors of Science Degree in Electrical Engineering Technology from Oklahoma State University in May 2003. Completed the requirements for the Master of Science degree with a major in Electrical Engineering at Oklahoma State University in December, 2005.

Experience: Research Assistant for Dr. Keith Teague at Oklahoma State University from May 2002 through August 2005

Professional Memberships: Institute of Electrical and Electronics Engineers, Society of Automotive Engineers.

Name: Joseph Ryan Hershberger Date of Degree: December 2005

Institution: Oklahoma State University Location: Stillwater, OK

Title of Study: ACCURATE AND PRECISE NETWORK PERFORMANCE TESTING IN WINDOWS 2000

Pages in Study: 142 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: The purpose of this study is to develop a network performance test system which measures jitter and latency of low bit-rate voice over IP traffic. The system is capable of making accurate and precise measurements in a Windows 2000 Operating System environment. Precise, GPS-synchronized timing is achieved with the use of a dedicated bus-level time processing card. Packets are timestamped in a protocol driver for versatility but can be timestamped in a network card driver for more accurate timing.

Findings and Conclusions: The overall error associated with the system's measurement was assessed and was found to be less than 90µs in the worst case and less than 10µs in the best case. Example network tests such as broadband Internet service, wireless LANs, and wireless data service are analyzed. Concatenated networks were tested as well. The work is applicable to not only voice over IP systems, but any real-time, low bandwidth systems.

ADVISER'S APPROVAL: Dr. Keith A. Teague