



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**CONVERGENCE OF THE NAVAL INFORMATION
INFRASTRUCTURE**

by

James A. Knoll

June 2004

Thesis Advisor:
Second Reader:

William Ray
David Floodeen

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2004	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Convergence of the Naval Information Infrastructure			5. FUNDING NUMBERS	
6. AUTHOR(S) James A. Knoll				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Converging voice and data networks has the potential to save money and is the main reason Voice over Internet Protocol (VoIP) is quickly becoming mainstream in corporate America. The potential VoIP offers to more efficiently utilize the limited connectivity available to ships at sea makes it an attractive option for the Navy. This thesis investigates the usefulness of VoIP for the communications needs of a unit level ship. This investigation begins with a review of what VoIP is and then examines the ship to shore connectivity for a typical unit level ship. An OMNet++ model was developed and used to examine the issues that affect implementing VoIP over this type of link and the results are presented.				
14. SUBJECT TERMS Voice over IP (VoIP), ADNS, OMNET++, Convergence, Satellite Communications, Networks, Simulation			15. NUMBER OF PAGES 279	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CONVERGENCE OF THE NAVAL INFORMATION INFRASTRUCTURE

James A. Knoll
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
June 2004**

Author: James A. Knoll

Approved by: William Ray
Thesis Advisor

David Floodeen
Second Reader

Peter Denning
Chairman, Department of Computer
Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Converging voice and data networks has the potential to save money and is the main reason Voice over Internet Protocol (VoIP) is quickly becoming mainstream in corporate America. The potential VoIP offers to more efficiently utilize the limited connectivity available to ships at sea makes it an attractive option for the Navy. This thesis investigates the usefulness of VoIP for the communications needs of a unit level ship. This investigation begins with a review of what VoIP is and then examines the ship to shore connectivity for a typical unit level ship. An OMNeT++ model was developed and used to examine the issues that affect implementing VoIP over this type of link and the results are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	WHAT IS CONVERGENCE	1
B.	THE CASE FOR VOIP	1
C.	VOIP CONSIDERATIONS	2
D.	US NAVY VOIP	3
II.	BACKGROUND	5
A.	BRIEF HISTORY OF THE PSTN	5
B.	BUSINESS TELEPHONY	7
III.	HOW VOICE OVER INTERNET PROTOCOL (VOIP) WORKS	11
A.	PLACING A CALL	11
B.	NETWORK DATA TRANSPORT	12
C.	VOIP AT THE NODE LEVEL	13
1.	Internet Engineering Taskforce (IETF)	14
2.	International Telecommunications Union (ITU)	15
IV.	CHALLENGES OF VOIP	19
A.	VOICE QUALITY	19
B.	DELAY	20
C.	JITTER	22
D.	LOST PACKETS	22
V.	THE NAVY VOIP IMPLEMENTATION	25
A.	THE AUTOMATED DIGITAL NETWORK SYSTEM	25
B.	TECHNICAL CONSIDERATIONS TO THE VOIP IMPLEMENTATION	26
1.	Delay	26
2.	Jitter	27
3.	Packet Loss	27
C.	TWO POSSIBLE IMPLEMENTATIONS	28
1.	Direct VoIP Implementation	28
2.	An Alternative VoIP Implementation	29
VI.	MODEL DEVELOPMENT	31
A.	TOOL SELECTION	31
B.	MODEL DEVELOPMENT	32
C.	INTERMEADIATE MODELS	36
1.	Frame Size	36
2.	Impact of TCP Congestion Control	38
VII.	RESULTS AND CONCLUSIONS	45
A.	DIRECT IMPLEMENTATION OF VOIP	45
B.	ALTERNATIVE VOIP IMPLEMENTATION	47
C.	CONCLUSIONS	48

D. FUTURE WORK	49
APPENDIX A. GLOSSARY	51
APPENDIX B. SIMULATION CODE	59
A. CHANGES TO IPSUITE SOURCE	59
B. COMPONENTS	69
C. NETWORKS	188
BIBLIOGRAPHY	265
INITIAL DISTRIBUTION LIST	267

LIST OF FIGURES

Figure 1:	Physical Cable Between all Telephone Users (From Davidson & Peters, 2000)	5
Figure 2:	The VoIP Call Process (From Caputo, 2000)	12
Figure 3:	Protocols related to Voice over IP (From Miller, 2002)	15
Figure 4:	H.323 Protocol Stack (After Black, 2000)	16
Figure 5:	Comparison of Compression Techniques (After Caputo, 2000)	20
Figure 6:	Simplified Block Diagram of ADNS (After Buddenburg, 2003)	26
Figure 7:	Current ADNS Ship Configuration (From Casey, 2004)	28
Figure 8:	Black ADNS Ship Configuration (From Casey, 2004)	29
Figure 9:	VoIP network with INEs	36
Figure 10:	Effect of frame size on CODEC performance	37
Figure 11:	TCP Client Network	39
Figure 12:	Data Rate for network with 18 Clients	40
Figure 13:	Data Rate for Network with 3 Clients	41
Figure 14:	Data Rate for Network with 1 Client	42
Figure 15:	Network for Determining VoIP Transition Efficiency	43
Figure 16:	Effects of call cycle on VoIP Implementation	45
Figure 17:	Upgraded 128K INMARSAT	47
Figure 18:	Black Voice Routing	48

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. WHAT IS CONVERGENCE

Convergence is the integration of voice, data, video, or any other imaginable multimedia communication onto a single transmission media. This may seem like a lofty and futuristic goal, but the ideas of convergence are not new. Convergence has been talked about since the 1980's when Integrated Services Digital Network (ISDN) was first introduced for sharing a transmission line between data and voice. Additionally, in the 1990's, the phone companies underwent a major upgrade to their backbone systems. They transitioned to packetized voice on their trunks in order to more efficiently utilize available bandwidth. The potential that VoIP offers to more efficiently utilize the limited connectivity available to ships at sea makes it an attractive option for the Navy. In recent years, a renewed emphasis on convergence has been seen in the form of Voice over Internet Protocol (VoIP). VoIP refers to the transmission of packetized voice traffic on a network traditionally designed for data. VoIP provides Phone-to-Phone, PC-to-PC, PC-to-Phone, Phone-to-PC and fax-to-fax services. VoIP is often used synonymously with the terms Internet telephony, IP telephony and packetized voice.

B. THE CASE FOR VOIP

The number one driving factor behind most new technology is cost savings. The efficiency of VoIP makes it very cost effective for use in industry. Significant savings are realized when toll calls are transported via an internet or the Internet¹. Many organizations, DoD

¹ The term internet (with a lower case i) in general refers to the connection of any two or more separate networks. The term Internet

included, save money by leasing connections used to provide dedicated communications. These leased connections are broken into 64kbit/s ISDN channels. Each channel is dedicated as either voice or data. Given that a normal conversation contains approximately 50% silence, 50% of the bandwidth dedicated to a voice channel is wasted. Data transmission is also 'bursty' in nature. Considerable bandwidth is wasted between data transmissions. By combining the two kinds of traffic, the burst nature of both can be exploited. Both types of traffic can then travel over one line. This can be translated into cost savings by using one dedicated line for both types of traffic vice having one line for voice and another for data.

Further savings come from the reduction of maintenance costs associated with the infrastructure of two disparate networks. In a traditional installation using Plain Old Telephone Service (POTS), separate organizations are required to maintain the data network and the Private Branch Exchange (PBX). Converging the voice and data networks would idealistically eliminate the entire infrastructure associated with the legacy phone system because all phone calls would travel over the data network. In reality, specialty VoIP equipment will be required but still the overall size of the resulting organization will be significantly reduced.

C. VOIP CONSIDERATIONS

In simple terms, convergence is good because it saves money; however, cost savings alone is not always enough to convince industry to fully embrace a new technology. Many

(with a capital I) refers to the specific entity that is publicly accessible and comprised of networks worldwide.

times the quality of the services provided are as important as cost savings. For VoIP to be widely accepted and used, the quality of VoIP service provided must be at least as good as those currently provided by the Public Switched Telephone Network (PSTN). Jitter and delay are often cited as potential problems in the quality of VoIP and need to be addressed. Also, users have grown accustomed to many advanced features provided by the PSTN. These include convenience features such as Call Waiting, Caller ID, and Call Transfer, safety features such as Enhanced 911, and Military Unique Features such as Multi-level Precedence and Preemption (MLPP). All of these must be incorporated as VoIP evolves. Finally, VoIP must be compatible with existing data-over-voice applications such as Modems, Fax, and STU/STE.

D. US NAVY VOIP

For the US Navy, convergence is not an easy task to undertake. In contrast to most other organizations, a good portion of the Navy is unable to communicate with the rest of the world via terrestrial cables. The unique issues associated with shipboard communications while at sea must be considered when designing any system for use by the fleet. Currently, communication for the majority of the fleet is via low bandwidth connections used for both voice and data. The INMARSAT system was introduced with the intent of meeting emerging communications needs of the unit level ships in the fleet. The problem is that applications designed for shore based use, where bandwidth is less of an issue, have been incorporated for use at sea. The current bandwidth needs of the unit level ships exceed the capacity of the INMARSAT system in its current configuration. This

thesis created and developed models used to investigate VoIP in a Navy environment.

Implementing VoIP on a satellite communications system is not an easy task. Problems that affect a high-speed terrestrial network are compounded when a satellite is in the communications path. The delay alone, approximately 500ms for a single trip to and from a satellite, is outside of the conventional norm for voice communications. Therefore, the effects of low bandwidth, high latency communications must be considered in the evaluation of VoIP. This investigation begins with a review of what VoIP is and then examines the ship to shore connectivity for a typical unit level Navy ship. A model is then used to examine several issues associated with implementing VoIP over this type of link and the results are presented.

II. BACKGROUND

VoIP merges the technologies and features of the Public Switched Telephone Network (PSTN) and business telephony systems with computer networking. To truly understand how VoIP evolved, it is important to first review each of these systems. This chapter will begin with a brief history of the PSTN and then covers current types of business telephony systems. The networking aspects of VoIP and the terms used to describe them will be discussed in Chapter III.

A. BRIEF HISTORY OF THE PSTN

In 1876, Alexander Graham Bell made the first voice transmission over an electrical wire. This first transmission was between two locations connected via a single wire. In the early days of the telephone, each user had to be directly connected to every other user. Figure 1 shows the direct connection of eight telephones.

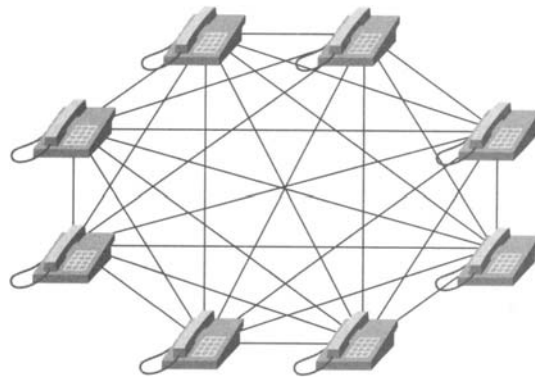


**Figure 1: Physical Cable Between all Telephone Users
(From Davidson & Peters, 2000)**

The number of connections required can be determined by the following equation:

$$\# \text{ of connections} = n(n-1)/2$$

where n is the number of users in the system

For this system with eight users, 28 connections are required. As n increases, this system can quickly become unwieldy and quite costly.

The solution to this problem was to create a switch. All of the physical lines were run to a central location and an operator routed the calls by using a patch cord to physically connect users to each other. Since the switches could be connected to other switches, telephone networks could be scaled up to cover a greater geographic area. In the 1890's, an advance in switching technology enabled switch-to-switch calling without an operator. However, well into the second half of the 20th century, many calls were still patched by hand. (Farley, 2004)

Over the years, many advances have been made to enhance the telephone networks. In 1937, multiplexing of analog signals was introduced. For the first time, multiple calls could be carried on a single transmission line. The impact was as profound as the invention of the switch. This allowed fewer cables to be run and reduced overall system cost. A further enhancement occurred in 1963 with the introduction of digital transmission techniques. These digital techniques are the basis for the infrastructure in use today.

The current state of the telephone industry is mixed. Although operator switched calls are a thing of the past, many analog switches are still used on the periphery of the updated digital backbone. Those areas still using analog switches do not get any of the benefits associated with digital systems.

This digital technology has enabled the modern PSTN to be characterized by advanced digital features such as Caller Id, Call Waiting, Voice Mail, and other services. Audible delays, once common for long distance calls, have been greatly reduced or in most cases eliminated as calls are now transmitted at the speed of light. These services have become commonplace and must be accommodated by any new technology.

B. BUSINESS TELEPHONY

Today's business telephone system is similar in structure yet more plentiful in features than the PSTN. These systems can be classified as one of five types. These are the simple business line, the Centrex line, the Virtual Private Network (VPN), the Private Branch Exchange (PBX), and the Key-system. (Davidson, 2000)

The simplest business telephone system is the business line. Provided by a Local Exchange Carrier (LEC), the business line is usually charged at a higher rate but is essentially the same as a residential line. It is used by small businesses that do not require a large number of features or a large number of users.

Also available from the Local Exchange Carrier (LEC) is the Centrex line. This type of system would be used by a small business that needs additional features not available from a regular business line. The phones are

grouped into a Closed User Group (CUG). This CUG provides the business with features such as call transfer, call waiting and call groups.

A step up from the Centrex is the third type of business system, the Virtual Private Network (VPN). The VPN allows the user to treat geographically dispersed sites as a Closed User Group (CUG). This system is best suited for a medium sized business like a department store where there are several different geographic locations but still not a large volume of calls. It allows separate sites to be connected without the overhead maintenance costs associated with systems that are more complex.

The Private Branch Exchange (PBX) is by far the most common phone system used in business today. The PBX gives the company complete control over the system configuration. A business that has a higher ratio of internal calls to external calls can purchase fewer PSTN trunks (connections). If the internal calls go to separate locations, tie-lines can be purchased to create permanent connections thus reducing long distance charges.

The fifth system is known as the Key-system. It is similar to the PBX but generally used by businesses with fewer than 50 phones. A Key-system costs less than a PBX, in both initial setup and maintenance, but lacks the ability to expand the way a PBX system can. This lack of expansion capability means a business must be fairly stable and able to predict its future needs when purchasing a Key-system.

As previously mentioned, many of the features and functions of the Public Switched Telephone Network (PSTN)

and the business telephony systems used today have contributed to the makeup of VoIP. Users of these systems have expectations for quality that need to be present in VoIP. VoIP, however, is deeply rooted in computer network technology as well. The next chapter explains the basics of how VoIP works in network terms.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HOW VOICE OVER INTERNET PROTOCOL (VOIP) WORKS

Describing how VoIP works is a difficult task. There are two (2) major governing bodies that have published different standards and recommendations on how VoIP should be implemented. The Internet Engineering Task Force (IETF) has addressed the issue from a network communications point of view where the International Telecommunications Union (ITU) has published more along the lines of telephone systems technology. These different approaches do have several overlapping or common components but also have some incompatible parts as well. Placing a VoIP call from a high-level point of view and data transport from the network-level point of view are common to both sets of protocols. Node-level implementation is where the two differ. This chapter will begin by presenting the high level view of placing a VoIP call followed by a description of data transport at the network level. Finally, a description of the two differing node level implementations is presented.

A. PLACING A CALL

When placing a call using VoIP, the dial tone, touch-tone, ringing, and busy signals are all emulated by a terminal or gatekeeper. When a number is dialed, it is mapped to the IP address of the phone to be called. A call setup protocol is then used. The actual set up will depend on which of the two governing bodies' protocols are used. The setup protocol locates the phone to be called and once found, sends a signal to produce a ring. When the receiving handset is picked up, voice is digitized via an analog-to-digital converter (ADC), packetized, encapsulated

into IP datagrams, and sent across the network. At the receiving end, the IP encapsulation is stripped, the data stream is reassembled, and the digital signal is converted to voice via a digital-to-analog converter (DAC). Figure 2 shows this call process.

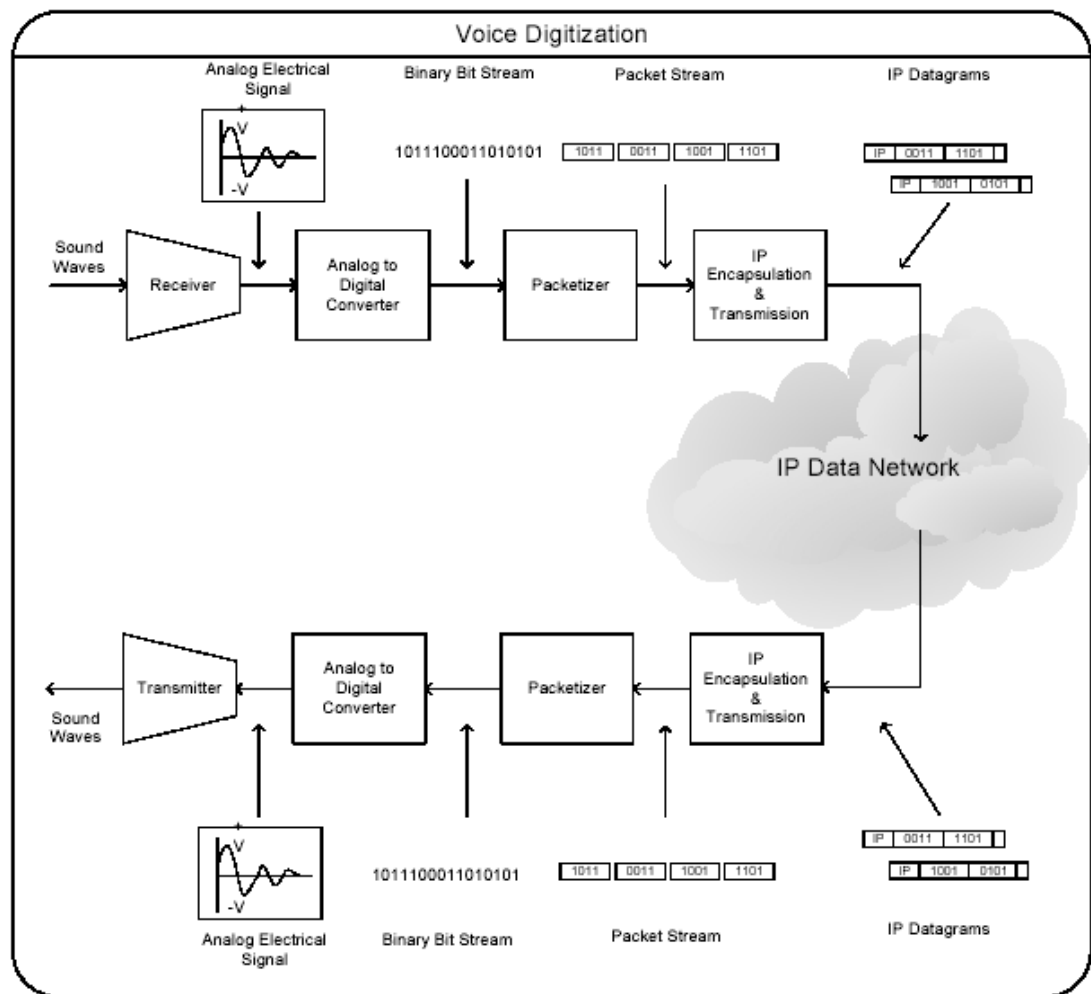


Figure 2: The VoIP Call Process (From Caputo, 2000)

B. NETWORK DATA TRANSPORT

Once a connection is established in the call process, data is then transported across the network. As mentioned above, the method of data transport is the same regardless of which governing bodies' protocols or standards are being

used. Data transport actually begins with packetizing data in accordance with a CODEC. A CODEC or coder/decoder is a standard method for encoding and compressing data. Several different CODECs are currently used for voice transmission. These CODECs are defined in standards published by the International Telecommunications Union (ITU).

The data is then encapsulated in a Real-time Transport Protocol (RTP) (RFC 1889) datagram. RTP is used with other protocols to provide transport for real-time data such as voice or video. The RTP header contains sequencing, time stamping, and content information. This datagram is usually transported via User Datagram Protocol (UDP) (RFC 768). The UDP datagrams are then encapsulated into Internet Protocol (IP) (RFC 791) datagrams that are used to route the information to the desired destination. At the destination, each layer is stripped until the voice data stream can be reassembled.

C. VOIP AT THE NODE LEVEL

Implementing VoIP at the node level is very different depending on which governing bodies' protocols are used by the equipment manufacturer. These different implementations are not compatible with each other so it is important to know which is being used. Some manufacturers of VoIP equipment will include the capability to interface using protocols from either governing body but this is not always the case. This section describes the different sets of protocols from each governing body. A system based on the IETF recommendations is presented first followed by a system described using the ITU standards.

1. Internet Engineering Taskforce (IETF)

The Internet Engineering Task Force (IETF) is the governing body responsible for recommending standards for the Internet. As such the recommendations for VoIP tend to be rooted in networking fundamentals. The following is an example of a typical call using terms from the IETF framework:

A User Agent is the software that interfaces with and acts on behalf of the user. The user agent uses the Session Initiation Protocol (SIP) (RFC 2543 found on Figure 3) to initiate a call. SIP is used to establish, modify, or end a VoIP session. The User Agent will use SIP to contact either a proxy server or a redirect server. The Proxy Server will act on behalf of the User Agent and forward an address request to the next node while the Redirect Server will send the next node information back to the User Agent for further requests. Once the address is resolved, the User Agents negotiate the parameters of the call in Session Description Protocol (SDP) (RFC 2327 found on Figure 3) messages. SDP is used by other protocols as a standard format to describe the elements of a session such as which CODEC will be used. If the call will traverse to a different type of network, the Media Gateway Controller negotiates the call and acts to mediate between the source and destination User Agents during the call. Multi Gateway Control Protocol (MGCP) (RFC 2705 found on Figure 3) establishes the use of Media Gateway Controllers. These controllers govern the operation of various Media Gateways. Media Gateways translate between various types of networks such as the Telco Backbone, a local loop, an Asynchronous Transfer Mode (ATM) network, or a PBX.

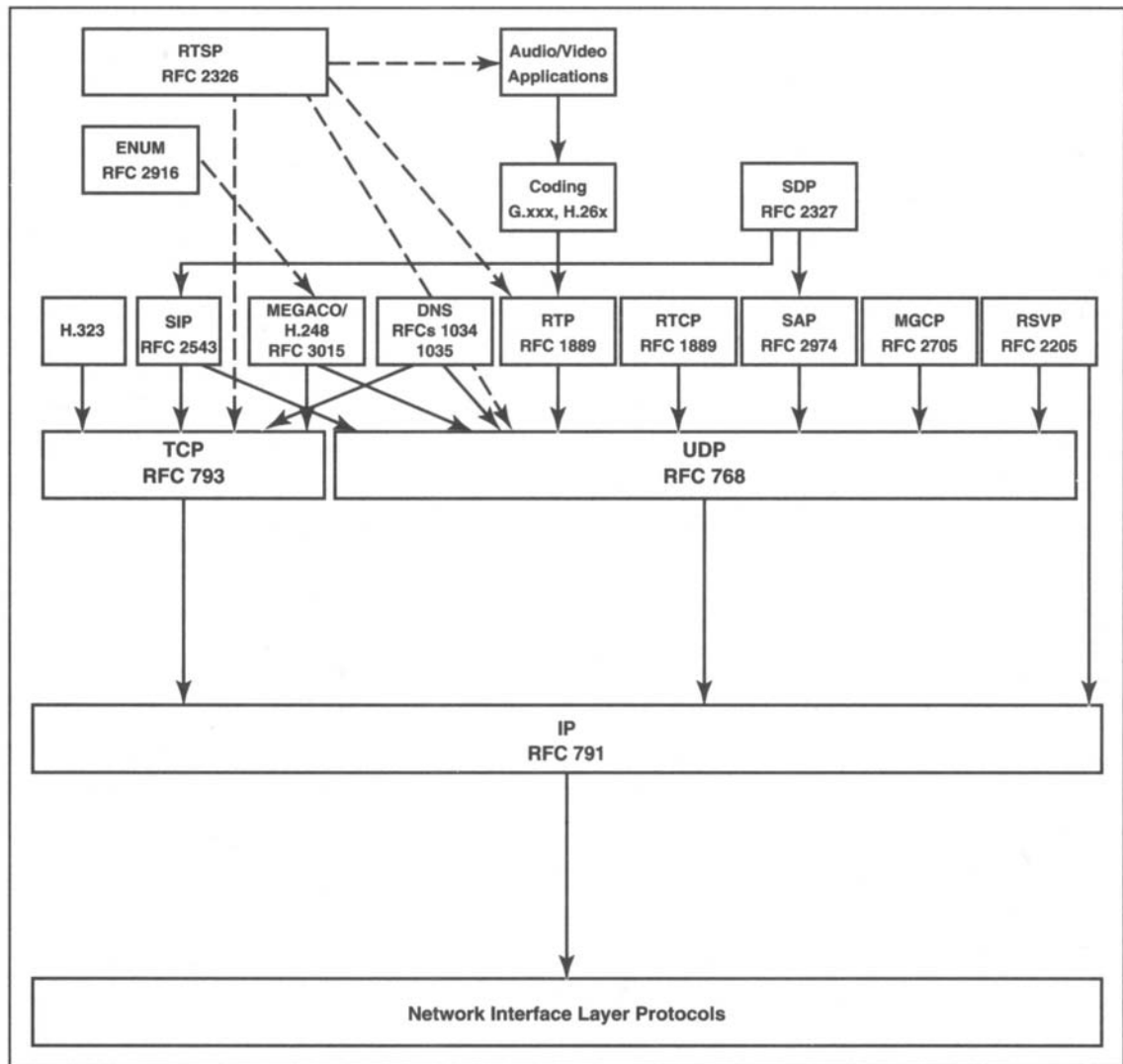


Figure 3: Protocols related to Voice over IP (From Miller, 2002)

2. International Telecommunications Union (ITU)

The International Telecommunications Union (ITU) is a body responsible for establishing global telecommunications standards. The specifications from the ITU for VoIP closely follow other telecommunications standards and specify the working of VoIP in terms of signaling. In contrast to the IETF's collection of protocols that can be used for VoIP, the ITU provides a single specification, H.323. H.323 is an overarching standard for "packet based

multimedia without QoS". H.323 incorporates other protocols such as H.225.0 for terminal to gatekeeper signaling and H.245 for Terminal control. Figure 4 shows the relationship between these protocols and the transport mechanism.

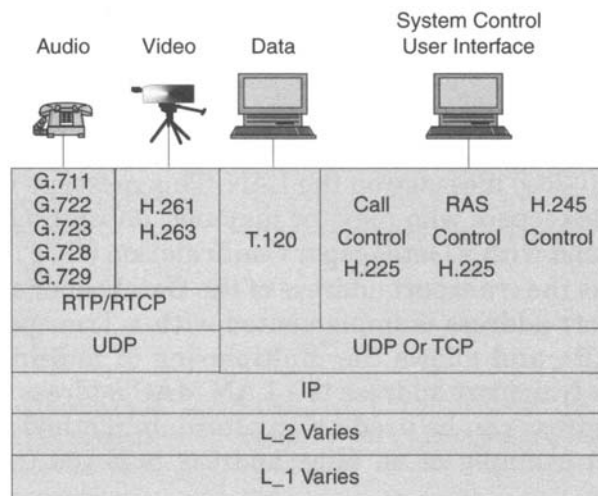


Figure 4: H.323 Protocol Stack (After Black, 2000)

A detailed call progression for systems using H.323 is beyond the scope of this paper. A summarized description follows:

A user's equipment is called a terminal. Before a call can be placed, the terminal must register with a gatekeeper. If the terminal is a part of a data network, the terminal performs the encoding, compression, and encapsulation of the voice sample. If the terminal is not part of a network, this function is performed by the Gateway. A Gatekeeper serves as the overall controller of the VoIP system. It controls access to the network, manages bandwidth, and performs address resolution. The source and destination Gatekeeper actually establish a

call. If the call will traverse a non-IP based network, the Gatekeeper controls the Gateways that perform the required translations. The Gatekeeper uses the previously mentioned Media Gateway Control Protocol (MGCP) or its replacement, Media Gateway Control (MEGACO/H.248) for control of all nodes. MEGACO/H.248 is a joint IETF and ITU standard based on Media Gateway Control Protocol (MGCP).

This section shows VoIP technology is actually governed by two different bodies, the IETF and the ITU. The methods and equipment used by each are different. The differences are seen at the node level but ultimately, both accomplish voice transmission over an IP based network. No matter which type of system is used, specific challenges must be overcome if VoIP is to be successful.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CHALLENGES OF VOIP

For any replacement technology to become widely accepted, the services provided must be comparable with those of the current system. More often, users demand even more from a new technology. In the case of VoIP, this presents several technical challenges. This chapter addresses the four main technical VoIP issues that should be considered when a network is first engineered. They are voice quality, delay, jitter, and packet loss.

A. VOICE QUALITY

Users have come to expect high quality voice communication using current technologies. For VoIP to be successful, it must be able to produce comparable quality voice communication. VoIP voice quality is primarily affected by compression of the voice signal and the type of encoding used in VoIP applications. Compression is important in trying to reap the benefits of VoIP because it reduces the amount of data transmitted. The benefits of compression come with a price because compression affects the quality of the recovered voice signal. The Mean Opinion Score (MOS) is a subjective scoring that rates the quality of a coder/decoder (CODEC) under various conditions such as background noise and multiple encodings. Figure 5, shows an averaged MOS for common CODECs used in VoIP.

Coding	ITU-T Standard	Bit Rate, kbps	MOS	Processing Power, MIPS	Frame Size, ms	Coding Delay, ms
PCM	G.711	64	4.1	0.34	0.125	0.75
ADPCM	G.726	32	3.85	14	0.125	1
LD-CELP	G.728	16	3.61	33	0.625	3-5
CS-ACELP	G.729	8	3.92	20	10	10
CS-ACELP	G.729a	8	3.7	10.5	10	10
MP-MLQ	G.723.1	6.3	3.9	16	30	30
ACELP	G.723.1	5.3	3.65	16	30	30

Figure 5: Comparison of Compression Techniques (After Caputo, 2000)

Each time a voice sample is encoded the MOS decreases. This is important because for each segment of a network that requires a CODEC translation, the resulting MOS will be lower. (Davidson, 2000) This will adversely affect the quality of the received voice signal.

Silence Suppression also adversely affects MOS. Silence Suppression techniques are used to save bandwidth by not transmitting during periods of silence. The problem with these techniques is that clipping of the conversation can occur.

Even though compression and silence suppression reduce the MOS and degrade the quality of the received signal, they are still used by some VoIP applications. Not all VoIP applications do both. VoIP can be tailored, by CODEC selection, to trade voice quality for bandwidth savings as desired.

B. DELAY

Delay is the amount of time it takes a signal to be digitized, transferred, and then converted back into an

analog signal at the receiver. A delay of 250ms or less is the generally accepted threshold for commercial toll quality service. Often, however, longer delays are tolerated. Some overseas phone calls and long distance cellular phone calls have delays exceeding 250ms. Communication via satellite is still possible even with delays in excess of 500ms. The sum of all delays in the system is called the end-to-end delay. End-to-end delay is generally referred to as just the delay or latency of the system. There are three types of delay to consider when discussing VoIP. These are propagation delay, serialization delay, and handling delay.

Propagation delay is the time it takes for a signal to traverse the physical media. For a copper wire, this is about 8 microseconds/mile. For applications involving a few thousand miles this may not be significant but if the network uses a High Earth Orbiting satellite, this delay is on the order of 500ms which is significant.

Serialization delay is characterized by the number of bits that can be transferred per second. This is not to be confused with the data rate of the media. This can more accurately be described as the data rate of the physical interface. This is generally neglected and not an issue for VoIP implementation since it is such a small contribution to the overall delay in the system.

Finally, handling delays incorporate all delays caused by manipulating the data. If 20ms of voice is packaged into a single datagram, the handling delay is this 20ms plus the time to actually encode the data. Additionally there is a delay as each piece of equipment handles the information. Significant delays occur when data is queued.

For most applications, handling delay is the biggest contributor to the end-to-end delay, but is also the one type of delay best controlled through proper engineering of the system.

C. JITTER

Jitter is the variation in the inter-arrival time between packets. Jitter is important because if not accounted for properly it can cause the decoded message to sound choppy. The affects of jitter are usually corrected by implementing a jitter buffer that delays messages on the receiving end longer than the experienced jitter. This allows the information to be replayed at a constant rate. Implementation of the jitter buffer does contribute to the delay but is necessary for maintaining voice quality.

D. LOST PACKETS

Packet loss is not unexpected in any network. This is the reason Transmission Control Protocol (TCP) contains a mechanism for the retransmission of missing packets. The time that it takes for a missing packet to be retransmitted is unacceptable in VoIP. This is the main reason VoIP applications use the User Datagram Protocol (UDP), which does NOT retransmit lost packets. The loss of a single packet can be masked by replaying the previous voice sample. This technique does not work when multiple packets are missing. When multiple packets are lost, the decoded voice signal may contain a pause or sound choppy. Engineering a highly reliable network can mitigate the number of lost packets.

These technical challenges are not insurmountable obstacles but rather items that must be addressed. When engineering a system for VoIP, mechanisms to control voice

quality, delay, jitter, and packet loss must be included. The next chapter will examine the current Navy INMARSAT communication architecture. Later chapters will show how this architecture can benefit from the transition to VoIP.

THIS PAGE INTENTIONALLY LEFT BLANK

V. THE NAVY VOIP IMPLEMENTATION

As previously discussed, VoIP increases efficient use of bandwidth by converging voice and data networks. The Navy currently uses circuit switching for voice communications and the Automated Digital Network System (ADNS) for managing data communications. This chapter will discuss using VoIP to converge these networks. VoIP will be implemented within the ADNS. This chapter will describe the current ADNS and review some of the VOIP technical considerations as they relate to ADNS. Finally, two (2) possible implementation strategies are presented.

A. THE AUTOMATED DIGITAL NETWORK SYSTEM

To manage the increasingly important and complex web of bandwidth limited communications, the Navy developed the Automated Digital Network System (ADNS). ADNS is designed to combine and manage the multiple data communications paths that include UHF, SHF and EHF communications while at sea as well as copper and fiber optic connections when pier-side. ADNS provides continuous data connectivity for the ship. If one communications path becomes inoperative, ADNS is designed to allow another path to handle important traffic. Using this system, most Unit Level ships communicate while underway via a 64kbps INMARSAT leased connection. This leased channel is normally configured as half for data and the other half for Plain Old Telephone System (POTS) connectivity. Figure 6 is a simplified block diagram of the ADNS.

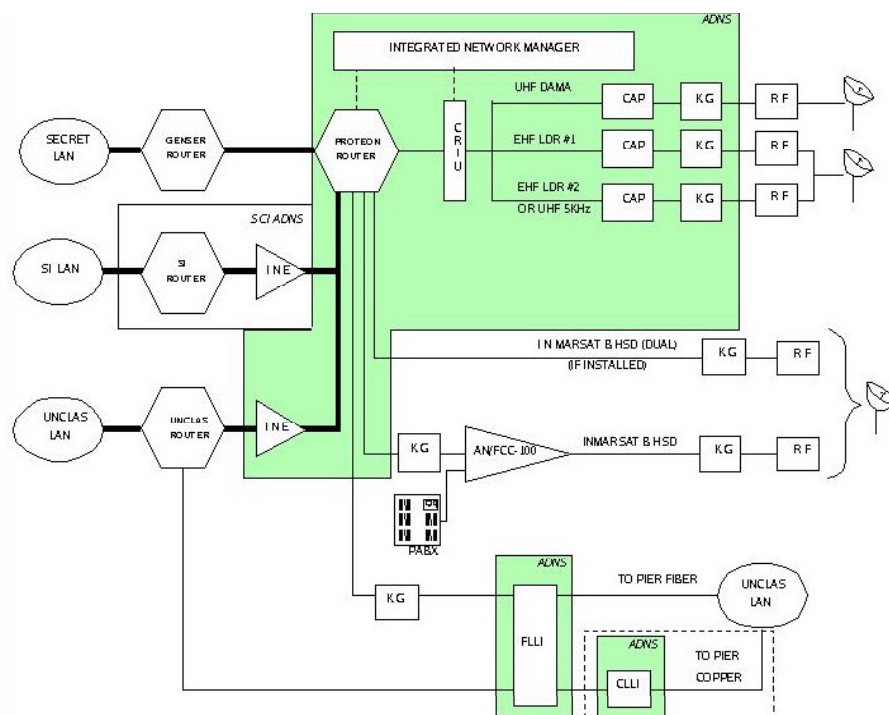


Figure 6: Simplified Block Diagram of ADNS (After Buddenburg, 2003)

Figure 6 shows the system is composed of several security enclaves. These enclaves are merged with the secret enclave at the ADNS router using Inline Network Encryption (INE) to form a common off-ship data stream. The data stream then travels through the Time Division Multiplexing (TDM)/MUX where it is multiplexed with the circuit switched voice communications and sent via satellite connection to shore.

B. TECHNICAL CONSIDERATIONS TO THE VOIP IMPLEMENTATION

1. Delay

As previously stated, there is a need to manage delay in a VoIP implementation. Because INMARSAT uses satellites in a geostationary orbit, the propagation delay is significant, commonly more than 500ms. Although the 250ms goal for toll quality voice is no longer feasible, managing

'handling delays' is still important. The effective data rate of the VoIP system is closely tied to the frame size. Selecting an appropriate frame size is actually an optimization problem. A large frame size can lead to a high efficiency because the fixed overhead associated with transport and encryption has less impact on the effective data rate. But, a large frame size increases handling delays and adversely affects voice quality. The selection of an appropriate frame size must balance efficiency needs with voice quality desires.

2. Jitter

Jitter must also be closely monitored. In a low bandwidth connection, such as INMARSAT, increasing queuing delays for data are likely to occur. This delay will manifest as jitter. For an ADNS implementation of VoIP to succeed, the queuing delay must be controlled. This can be accomplished by implementing a Quality of Service (QoS) mechanism that provides priority handling for VoIP traffic. In the latest version of the ADNS, Class Based Weighted Fair Queuing (CBWFQ) provides QoS. (Barsaleau & Tummala, 2004) CBWFQ can provide guaranteed bandwidth and expedited service for the VoIP traffic and ensure a fair allocation of resources to each ADNS enclave.

3. Packet Loss

A third factor to consider when implementing VoIP in the ADNS is packet loss. The main contributor to packet loss is Bit Error Rate (BER). The BER is the probability that an individual bit will be corrupted during transmission. If a bit is corrupted, the packet is discarded and considered lost. For a terrestrial network, BERs are usually less than 10^{-10} and are not often considered a significant issue. In a Navy system that uses

RF transmissions for data transfer, this is not the case. For a typical INMARSAT connection, BERs in the realm of 10^{-5} - 10^{-7} are common. As frame size increase, the probability of a lost packet increases as well. Additionally, the negative impact on voice quality created by that lost frame also increases. When determining the frame size for an ADNS VoIP implementation, BER should be considered.

C. TWO POSSIBLE IMPLEMENTATIONS

1. Direct VoIP Implementation

The Navy currently uses the secret network as its common ship to shore and shore to ship routing network. All traffic from the unclassified and SCI enclaves are encrypted using an IPsec device, also referred to as an Inline Network Encryption (INE) device. The encrypted traffic is then joined with the secret data traffic in the ADNS router. The INE currently used by ADNS is the Taclane. Figure 7 depicts the current security configuration of ADNS and shows where VoIP traffic will be introduced into the network at the UNCLASSIFIED enclave.

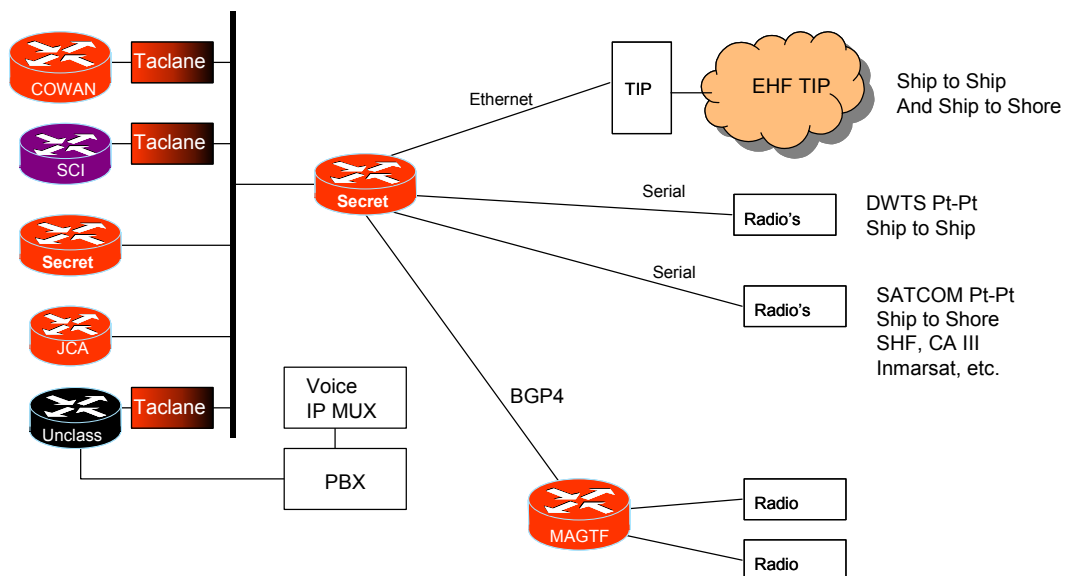


Figure 7: Current ADNS Ship Configuration (From Casey, 2004)

2. An Alternative VoIP Implementation

The impact of the direct implementation presented above is the addition of overhead to the VoIP traffic from the INE. The INE adds a minimum of 58 bytes to the IP datagram. This increases the effective data rate required for VoIP implementation. Eliminating the overhead of the INE from voice traffic will increase the efficiency of the implementation. Figure 8 shows an alternative implementation called a "Black ADNS Ship Configuration" (From Casey, 2004).

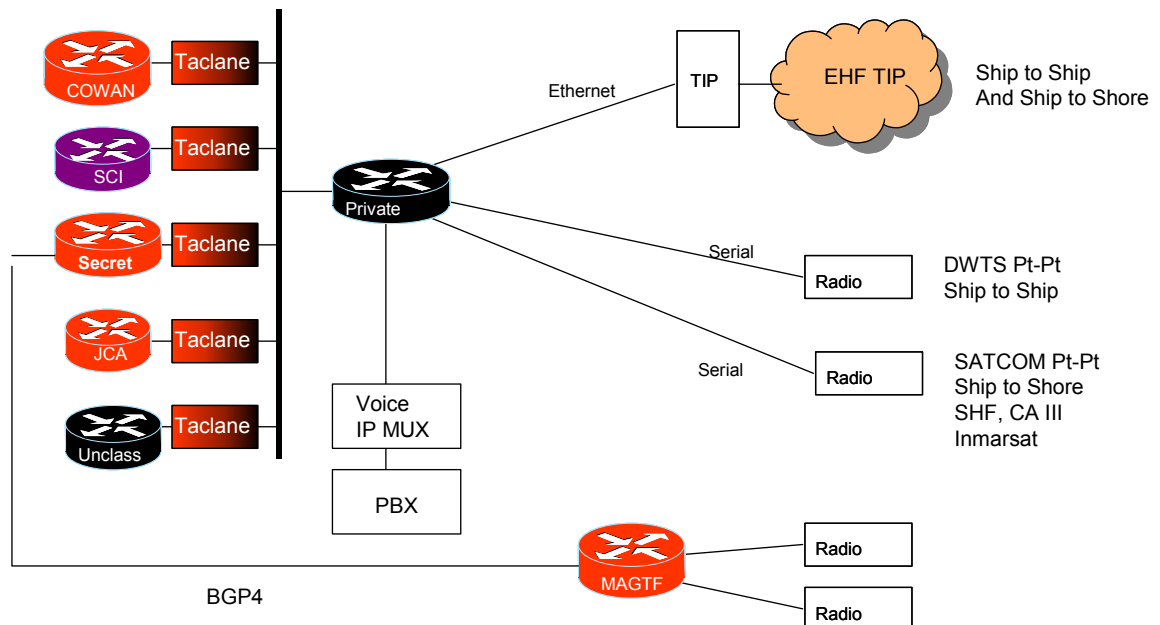


Figure 8: Black ADNS Ship Configuration (From Casey, 2004)

This proposed solution sends the secret enclave through an INE. VoIP traffic is combined with the other network traffic at the ADNS router. However, in this configuration, the VoIP traffic does NOT pass through an INE. There is no added overhead. This will increase the efficiency of the VoIP implementation.

This chapter introduced two possible VoIP implementations within the Automated Digital Network System (ADNS). The thrust of this research was to develop a model that simulates these two scenarios. The next chapter describes the model development.

VI. MODEL DEVELOPMENT

Simulation models are a quick and efficient way to narrow the field of research. Through high level modeling of a proposed network, quick feasibility studies can be conducted and future work can be scoped. A more detailed model can help tune parameters or verify the correctness and optimization of a protocol. All of this can be accomplished without procuring equipment. Hours worth of data can be obtained in minutes worth of runs.

When modeling, it is easy to over analyze a problem in an effort to provide a high fidelity model. In order to scope a project and determine what is important to model, it is necessary to first state the problem as simply as possible. The base question to be answered in this research is: Is it beneficial to pursue the implementation of VoIP on Unit Level ships? Other questions will have to be answered before a final conclusion can be reached, but this question must always be kept in mind. Once the question has been determined, a modeling tool must be selected.

A. TOOL SELECTION

OMNeT++ was chosen because the author was familiar with the package and modification and extensibility of the existing functionality are easy to accomplish. OMNeT++ is a simulation environment whose primary application area is the simulation of communications networks. It is flexible enough to simulate IT systems, queuing networks, hardware architectures and business processes as well. Simulation components are written in C++ and the modules are written in an easy to understand language called NED. OMNeT++ is

easy to learn and use and well suited to this research effort. Appendix B contains a complete listing of the OMNeT++ code written specifically for this research effort.

Once the simulation tool was selected, the next step was to develop the model. The steps used in model development are described below.

B. MODEL DEVELOPMENT

Although it is possible to model every component in the ADNS simplified block diagram (figure 6), every component was not needed to answer the research question. The first step was to determine which nodes and connections were important and to simplify the network to only these nodes and connections.

The last part of the ADNS system, from the ADNS router through the satellite link, was the easiest to simplify. The first simplification was to consolidate the time lag introduced by the KG's and the satellite link into a single delay. Next the bandwidth restrictions in the FCC100 and the satellite were modeled using the most restrictive setting. The voice from the FCC100 was not included because it was already accounted for in the bandwidth restrictions. The delay and data rate were combined into a single channel that was modeled as a 500ms delay and either a 32 kbps or 64 kbps data rate.

The ADNS router was modeled next. The ADNS router performs two primary functions in the model. It both routes the incoming and outgoing messages and provides QoS for the messages traveling via the INMARSAT link. Separating these two functions in our model makes it easier to examine various QoS mechanisms at a later time. Because of this separation, the OMNeT++ IPSuite standard router was

used as the router component and a new component called a WRED Box was created.

The WRED Box was loosely based on the description of Weighted Random Early Drop (WRED) and Class Based Weighted Fair Queuing (CBWFQ) found in (Barceleau, 2004). The component actually used is a scaled down version that adequately represents the configurations needed by this research. The WRED Box queues the incoming messages into either a High Priority Queue (HPQ) or a Low Priority Queue(LPQ). Those messages with a Differentiated Services Code Point (DSCP) marker of 46 were placed into the HPQ. As long as the HPQ contains items but has not yet reached its reserved allotment of bandwidth, the model services the HPQ. The LPQ is serviced when the HPQ is empty or exceeds its reserved allotment. The WRED algorithm for controlling queue depth is implemented on both queues. Because throughput was already calculated for the HPQ, the measurements for system throughput were taken at this point for all types of traffic. The code written to model this component can be found in Appendix B.

The INE was modeled next. It was modeled as a separate element to provide flexibility in the model. Messages that are encrypted can be connected through this node to incur the INE overhead; those that are not, bypass it. The INE was created based on the equation found in (Hucke, et. all, 2003). Rather than actually encapsulating the message as described in (Hucke, 2003), the IP Header field length was modified to save on computing resources when running the simulation. The code written to model this component can be found in Appendix B.

The voice traffic was modeled next. A client was needed that periodically sends a burst of information and then waits for a reply. The reply was modeled after an actual conversation where the listener responds after a reasonable period of inactivity.

The original plan was to create a voice client based on an available RTP implementation. Further research showed the only impact RTP had on the model was the addition of 8 bytes to the packet size. At this point, instead of creating a voice client based on the RTP implementation, the OMNeT++ IPSuite UDP Host was modified to create a VoIP Host. The client application modeled the voice traffic in the following manner: Based on the CODEC rate, frame size, and reply length, a number of messages are sent, modeling a voice burst. An internal timeout was then used to initiate a reply. The timeout was reset with the arrival of each message from the transmitting end. The length of each message is increased by 8 bytes to account for the RTP overhead. A second timeout was added to control the call cycle. This simulates a normal phone being on and off hook. The server side was merged into the client to simplify the reply mechanism. The code written to model this component can be found in Appendix B.

Once the VoIP client was written, an appropriate CODEC had to be selected. The model was built on the premise that the CODEC data rate and frame size were the driving factors in performance. The G.723r53 was selected because it requires the lowest data rate. For the STUIII calls, however, other factors come into play. The STUIII was designed for data over voice and does not perform well with

the lower data rate CODECs. From results in (Hucke, 2003), the G.726r16 was selected as the CODEC used for STU capable conversations.

The background traffic was modeled next. A UDP client was selected because a flow of data could be shaped to provide constant loading to the system. Initially, using a TCP client and server was considered. However, further research showed that managing the proper number of clients to create the desired loading would be difficult. The purpose of the model is not to measure the amount of traffic passed through the network but instead to measure the change in the amount. Therefore, the model could be simplified by combining the clients from the three enclaves. It is the relative change in the aggregate traffic that is of interest to this research. If further work is contemplated on QoS mechanisms, it may be required to separate the types of traffic and identify the source enclave of each.

The standard UDP client was considered, but it did not give enough control over the amount of data that was being sent, therefore this client was also rewritten. The Traffic UDP Host was created to constantly transmit packets based on the desired data rate and message size. On the receiving side of the host, the message is dropped once the desired metrics are recorded. The code written to model this component can be found in Appendix B.

This completed the modeling of the components that make up the overall VoIP model. Before the overall model could be run, two major questions had to be answered. First, what is the optimal frame size? Second, do TCP congestion control methods preclude the use of UDP data

streams as an appropriate abstraction for accurately modeling composite network traffic? The following section describes how these questions were answered.

C. INTERMEDIATE MODELS

1. Frame Size

In order to determine the optimal frame size, two network simulations were built using components already modeled. The code written creating this simulated network can be found in Appendix B. The results of these simulations were used to determine the effects of various frame sizes on the required effective data rate for different CODECs when the IP and INE overheads are applied. Figure 9 shows the network used to test this CODEC efficiency at various frame sizes with an INE.

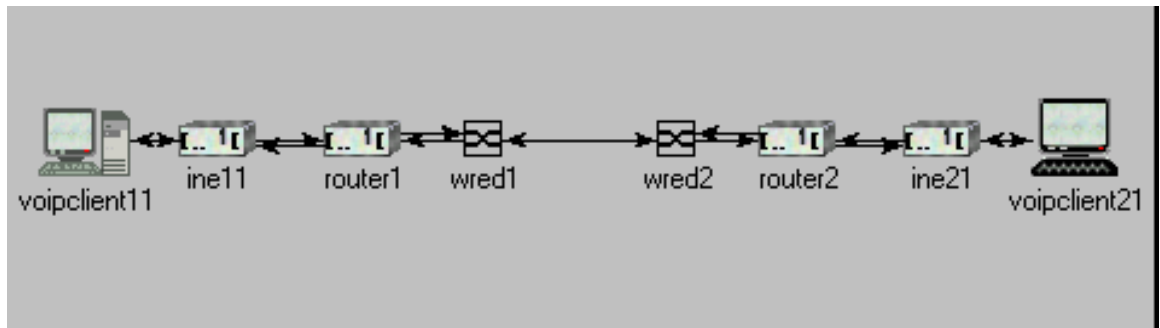


Figure 9: VoIP network with INEs

Runs were conducted at 16 kbps and at 5.3 kbps with frame size varying from 10 to 500 ms. The network was modified to remove the INE and the runs were repeated. Measurements for throughput were taken at the node labeled wred1.

Figure 10 shows a graphical representation of the results. It plots the required effective data rate verses frame size for the four previously described runs. The

lower the data rate required, the more effectively the CODEC performs at that frame size.

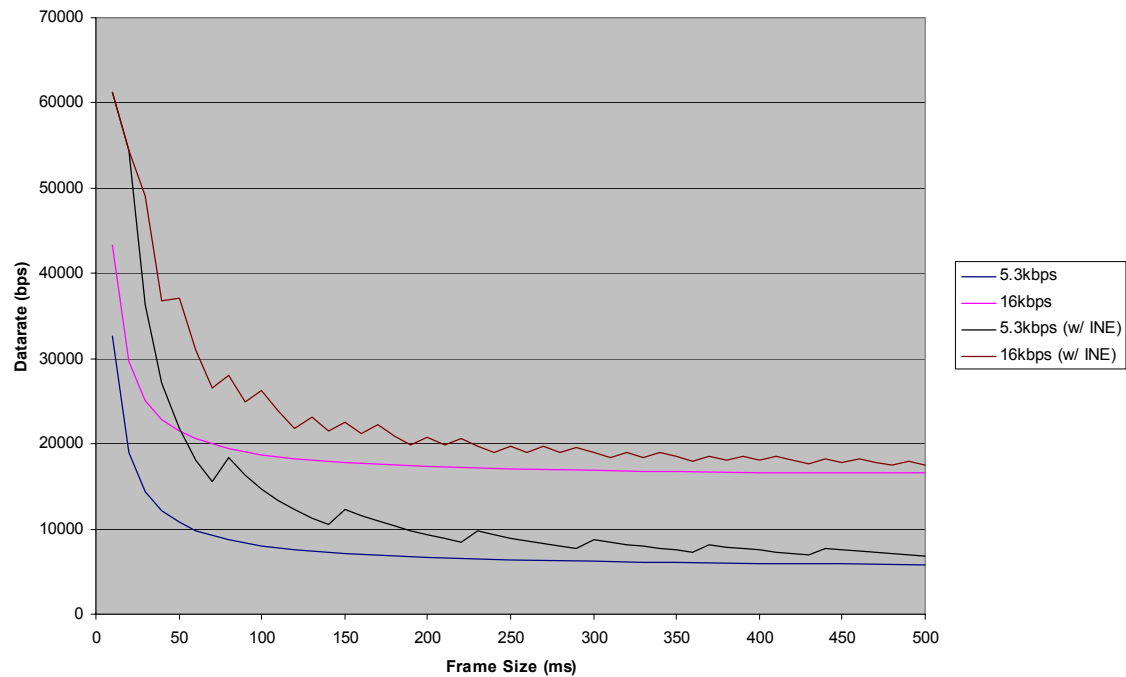


Figure 10: Effect of frame size on CODEC performance

From figure 10, we see the larger the frame size the more effective the CODEC performance as would be expected. As frame size increases above approximately 140 ms the improvement is marginal. Taking into account the earlier discussion of handling delay, the 140 ms frame size is the optimized solution between efficiency and voice quality. The jagged steps in the curves that correspond to the networks with an INE result from the padding introduced by the Taclane. This padding is used to obtain a 48-byte increment needed in the encryption of the packet and implies that the best performance will be achieved where the packet size is near a multiple of 48 bytes. The 140 ms frame size fits this requirement as well.

These results are based upon a generic CODEC. Vendor specific implementations may add look ahead or other mechanisms that increase quality of service but also change the effective CODEC data rate. Therefore, these results should be modified when considering optimal settings for actual CODEC use.

2. Impact of TCP Congestion Control

After initial design considerations were complete, a conversation with Mr. Ed Hucke from SPAWAR PMW 179, the engineers of ADNS, made us question the decision to model the network traffic as UDP packets. Mr. Hucke stated a concern that the TCP Slow Start congestion control mechanism may reduce the amount of traffic that could be transmitted in the periods without voice transmissions due to a lag in resumption of traffic to fill the available bandwidth. (Schilke, 1997) confirms this could be an issue.

To test the theory, the standard TCP client was modified to collect 'goodput'. Goodput is the rate at which unique data arrives at the client. The network simulation shown in Figure 11, was designed to test the effects of the Slow Start algorithm on changes to the bandwidth available for low priority messages. The code written creating this simulated network can be found in Appendix B. The results of this simulation would determine if UDP accurately models aggregate network traffic changes.

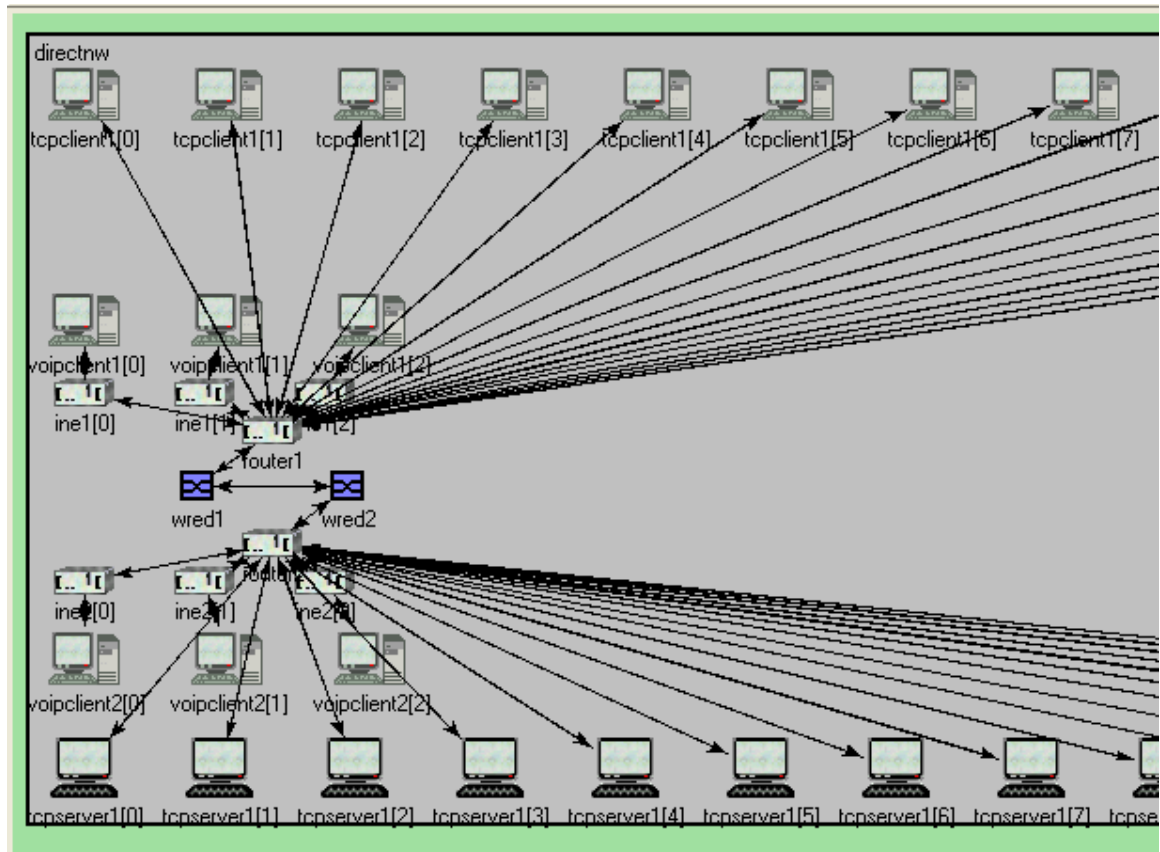


Figure 11: TCP Client Network

The network was configured for a variable number of TCP clients with matching servers and three VoIP Clients. A heavy load, medium load, and light load were set in the configuration by using 18 clients, 3 clients, and 1 client respectively. After each run, the amount of data received from each client was combined in an Excel spreadsheet. The data was sorted by timestamp and the amount of data received by a client was divided by the time difference between this timestamp and the previous timestamp. This calculation provided the network goodput.

Figure 12 shows graphical results of the run under heavy load. It plots the goodput as a data rate verses time. Periods where congestion control effects are potentially affecting the network traffics ability to

respond to cessation of voice transmissions would appear as periods of reduced goodput occurring in the absence of voice transmissions.

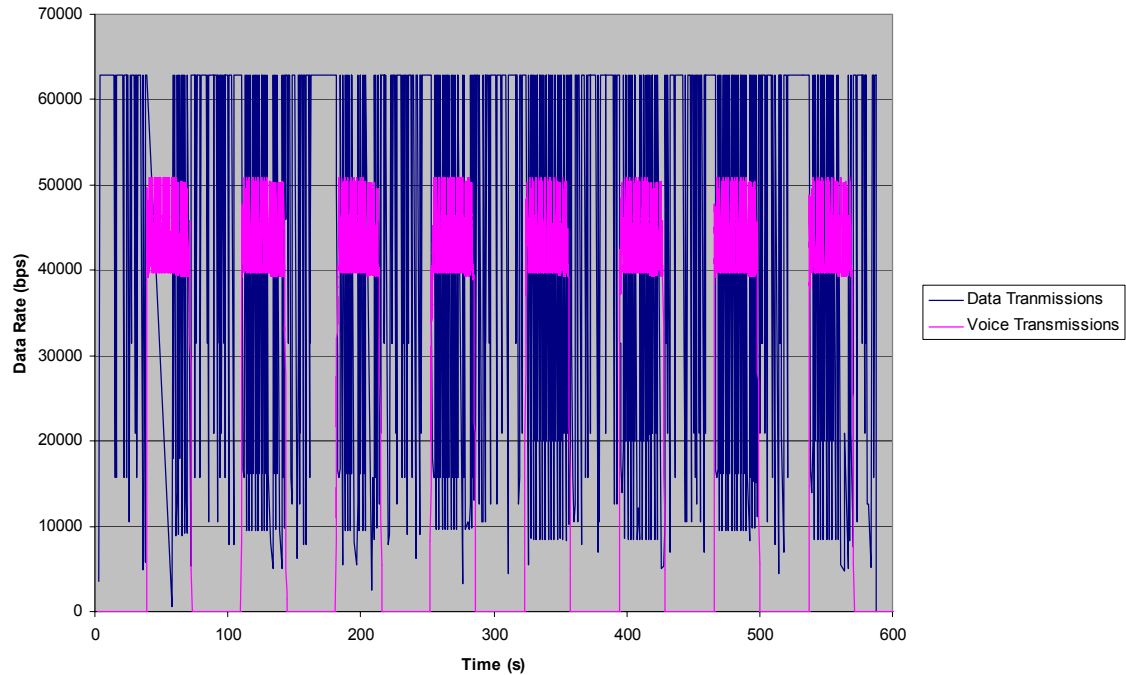


Figure 12: Data Rate for network with 18 Clients

As expected for the heavy load, shown in Figure 12, the amount of data queued and the number of clients receiving data tended to dampen most congestion control effects. There was no evidence that the slow start protocol would cause a problem with modeling the traffic as UDP packets.

When the runs were repeated at a medium and light load, the number of areas where congestion control was potentially affecting the ability to model network traffic using UDP increased. It is not as clear, however, if these are slow start effects after the cessation of voice transmissions. Figure 13 shows that with three clients, some of the periods without data being received by a client

have extended. If this was an issue that affects the use of modeling as UDP, these periods would consistently appear at the end of each voice transmission. Figure 13 clearly shows this is not the case. Therefore, it can be assumed UDP will still accurately model the network traffic in this scenario.

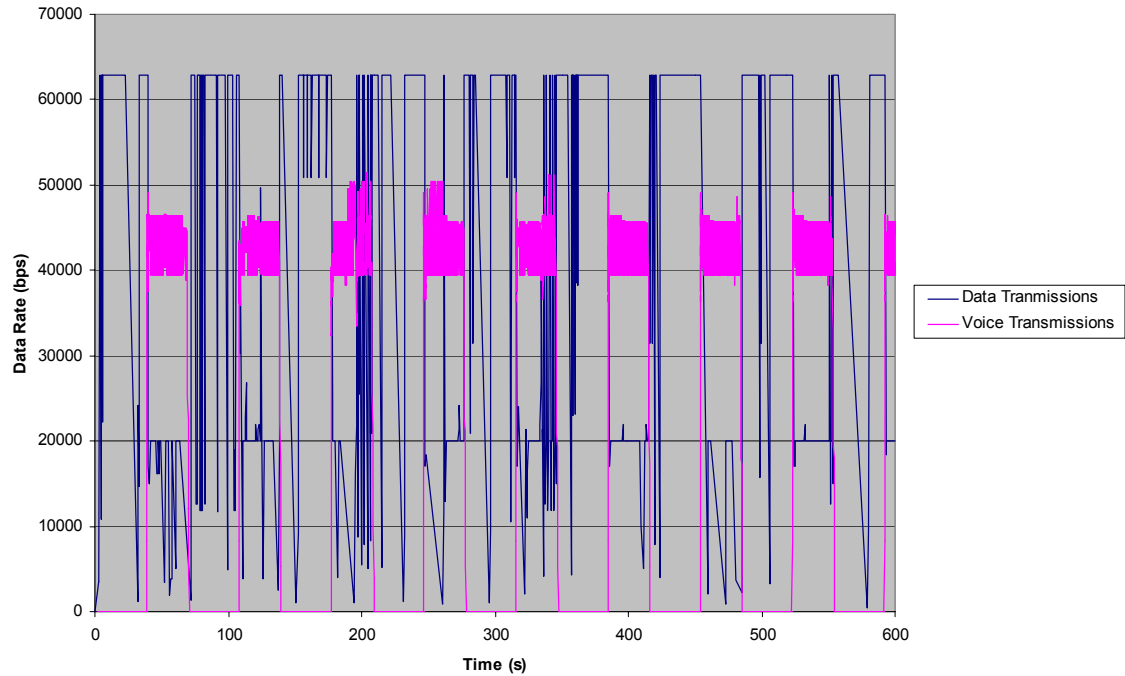


Figure 13: Data Rate for Network with 3 Clients

Figure 14 shows the graphical results of modeling the network with a light load of one client. Once again, extended periods with reduced goodput are present. Again, the lack of consistency in location and duration can only lead to the conclusion that these periods are not affecting transitions from voice transmissions.

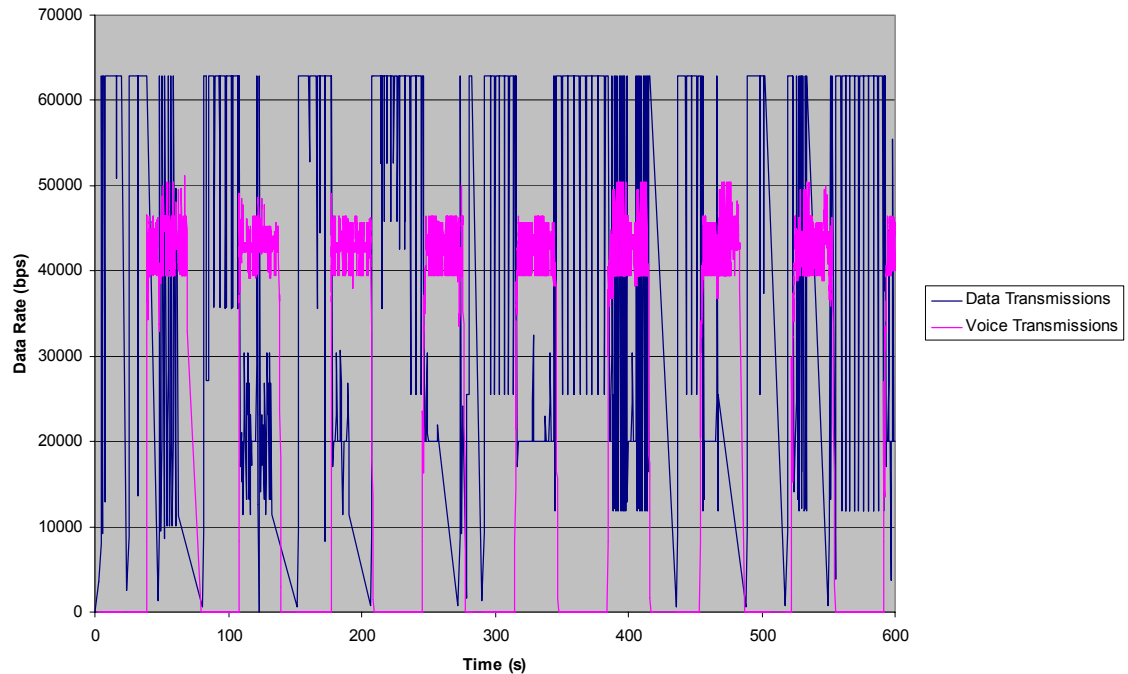


Figure 14: Data Rate for Network with 1 Client

To further ensure that TCP data traffic can be modeled as a UDP data flow, the goodput between voice transmissions for each case was compared with comparable time periods on a simulation run with zero voice clients. The data received on this final run was within 3.5% of the data in each of the previous simulations, further showing that our decision to model using UDP traffic is valid.

Now that the components have been modeled, the optimal frame size determined, and the use of a UDP Host for Traffic verified, the overall models testing the two different VoIP implementations were built. The code written creating this simulated network can be found in Appendix B. The network simulations were configured with a UDP Traffic Client and a variable number of VoIP Clients as shown in Figure 15.

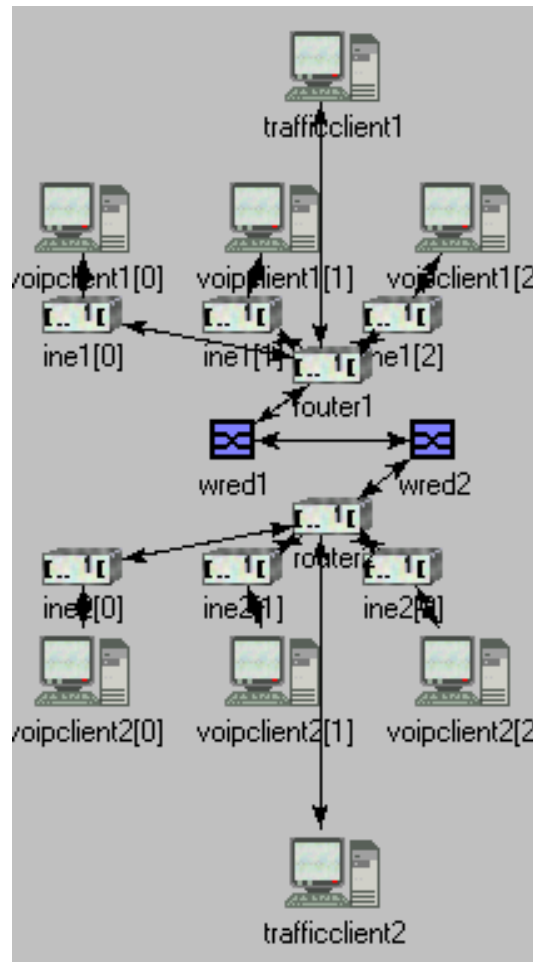


Figure 15: Network for Determining VoIP Transition Efficiency

Each set of runs varied the call cycle while keeping the number and configuration of the clients the same. Detailed results, conclusions, and future work are presented in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. RESULTS AND CONCLUSIONS

Is it beneficial to pursue the implementation of VoIP on Unit Level ships? To answer this question the model described in the previous chapter was run varying the call cycle while keeping the number and configuration of the clients the same. The simulated network was modified to investigate various potential implementation strategies described in Chapter V. Below are the results of those simulations.

A. DIRECT IMPLEMENTATION OF VOIP

The direct VoIP implementation was simulated using varying numbers of VoIP clients that sent data through an INE. Figure 16 shows graphical results obtained from the direct implementation simulation model.

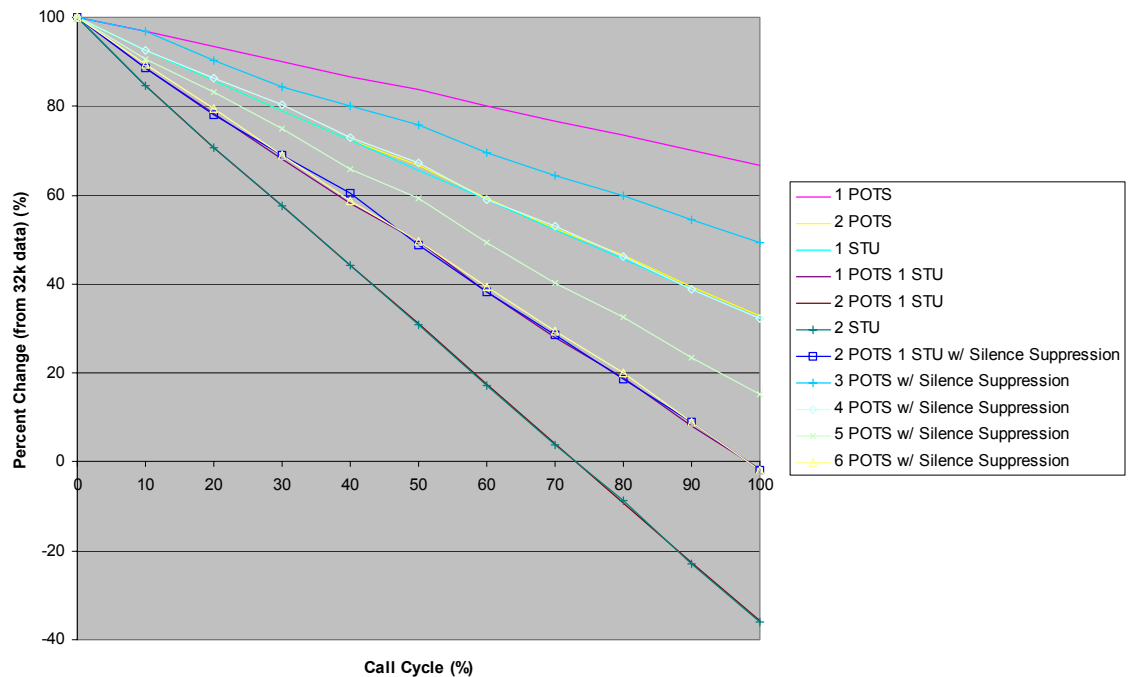


Figure 16: Effects of call cycle on VoIP Implementation

Figure 16 plots percent change in goodput compared to a baseline of 32k data verses the call cycle percentage. A gain is realized when the percent change in goodput is greater than zero. The ideal case would be where the percent change in goodput is greater than zero through 100% call cycle. The different runs represent different possible combination of POTS and STU lines in use simultaneously. Run configurations do not include configurations that will exceed the total available bandwidth.

Figure 16 shows an increase in the throughput for data traffic over the current ADNS configuration for up to a 72% call cycle when implementing VoIP using two (2) POTS lines and one (1) STU line. With silence suppression enabled a throughput gain is seen through close to a 100% call cycle. Another benefit of the transition to VoIP shown by the results of this simulation is the ability to have more POTS lines than are currently available. With silence suppression enabled, six (6) concurrent POTS calls were possible at near 100% call cycle before a decrease in performance is seen compared to current throughput levels.

The current ADNS configuration allows for up to two (2) POTS and two (2) STUs to be operated simultaneously. The Voip implementation simulated above cannot support this configuration and is limited to two (2) POTS and one (1) STU or two (2) STUs. This limitation comes from the 64 kbps bandwidth limitation of the currently fielded INMARSAT configuration.

The direct implementation model was modified to use a potential INMARSAT upgrade to increase bandwidth to 128 kbps, which is commercially available. Figure 17 shows the

ability for a VoIP implementation under these conditions to support up to five (5) STU lines.

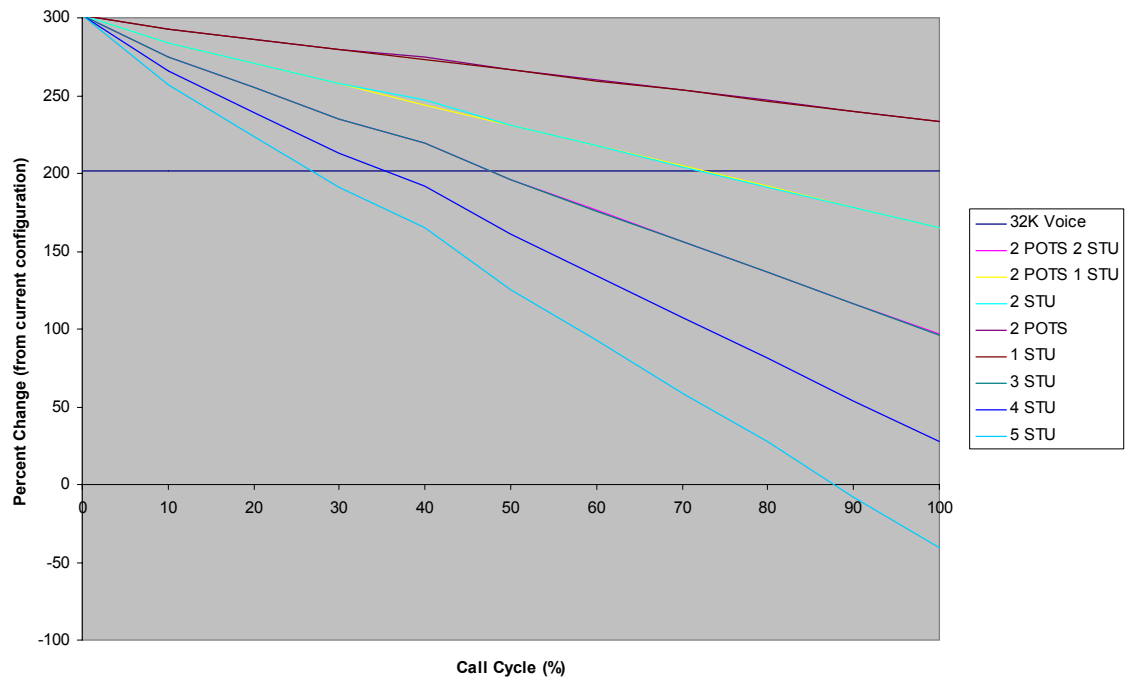


Figure 17: Upgraded 128K INMARSAT

B. ALTERNATIVE VOIP IMPLEMENTATION

The overhead caused by the INE can be eliminated by transitioning to Black Voice routing as discussed in Chapter V. The direct network model was modified by removing the INE module associated with each VoIP Client. Figure 18 shows the results of the simulations run under these conditions. This configuration can support two (2) POTS and two (2) STU lines without upgrading the INMARSAT equipment.

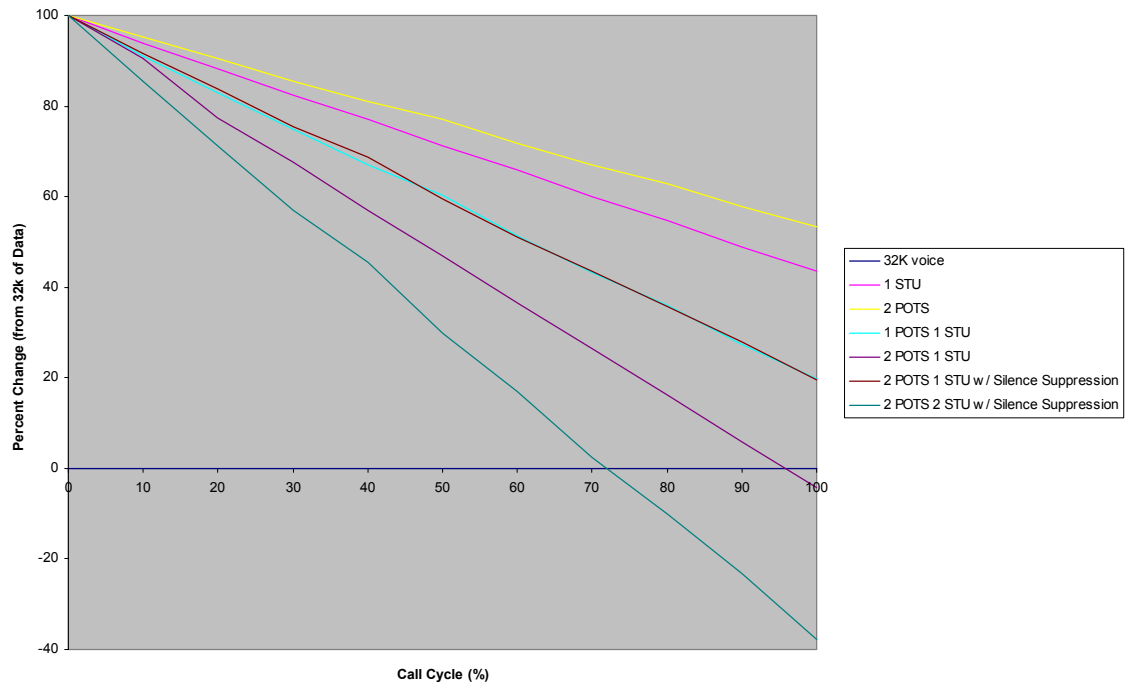


Figure 18: Black Voice Routing

C. CONCLUSIONS

This investigation has shown the benefit of converging the voice and data networks for unit level ships. The Center for Naval Analysis documented the POTS usage for two battle groups during their JTFX's. In their letter CME D0008489.A1 of June 2003, the authors stated that POTS usage for the 18 ships using INMARSAT channels was 8.1 percent. (Hucke, 2003) Using an 8% call cycle as a point of reference, we see approximately an 85% increase in bandwidth available for all configurations. In order to realize these gains it is not necessary to develop a CODEC specifically for the STU line, it is not necessary to transition to a Black Routing paradigm, nor upgrade the INMARSAT connection to 128kbps. This does not mean that any of these endeavors should be abandoned since all will lead to increases in performance that will most likely be

required in the future. As the Navy becomes more NET-CENTRIC WARFARE oriented, additional capacity will be needed. Implementing VoIP and taking advantage of the additional options is one way to meet this future need.

D. FUTURE WORK

This is not the end of development for this model. In its current state, this research has shown the model is able to provide a quick feasibility study. With a refinement of several components, however, it could be used to decide which QoS protocols show the greatest potential benefit and where in the network they are best utilized.

Although it was appropriate to model the background traffic as a single UDP stream in this research effort, many future investigations may need greater fidelity. When the stable release of IPSuite is available, the model should be transitioned and a goodput analysis method developed for TCP.

A fleet demonstration of the direct implementation for VoIP is currently scheduled for the summer of 2004. Results from that demonstration should be used to refine the model for future testing.

A more efficient means of achieving secure voice communications is needed in the form of a native VoIP device that can take advantage of silence suppression and also use lower data rate CODECs.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. GLOSSARY

Analog-to-digital Converter (ADC) - accepts an analog input-a voltage or a current-and converts it to a digital value that can be read by a microprocessor.

Asynchronous Transfer Mode (ATM) - a network technology that is based on transferring information in cells of fixed size. It well suited for converged networks because it creates a channel at the beginning of a data transfer session, allocating a fixed amount of resources to that session.

Automated Digital Network System (ADNS) - a system designed to combine and manage the multiple communications paths to include UHF, SHF and EHF communications as well as copper and optical pier side connections to provide ships force continuous data connectivity for high priority information.

Automatic Digital Network (AUTODIN) - a legacy communications system for ensuring the delivery of message based communications throughout the Department of Defense. For most purposes it has been replaced by DMS.

Bandwidth - traditionally the difference between the upper and lower frequencies of a transmission band. Recently it has also come to mean the amount of data that can be passed along a communications channel in a given period of time measured in bits per second (bps).

Bit Error Rate (BER) - the rate at which data is corrupted expressed as a percentage.

Centrex Line - a service purchased from the local exchange carrier that groups phone lines into a closed user group

(CUG). This provides additional services such as call transfer and call groups without the purchase of a PBX.

Class Based Weighted Fair Queuing (CBWFQ) - provides Quality of Service (QoS) by separating traffic into queues based upon a differentiated Services Code Point (DSCP) and then allocating each queue a share of the bandwidth.

Closed User Group (CUG) - a grouping of business phone lines that allows the phone company to provide PBX services from their office.

CODEC (coder/decoder) - a schema for encoding or decoding information from an analog to digital or digital to analog form.

Convergence - the combining of multiple networks such as voice data and video into one network.

Datagram - a self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network.

Defense Message System (DMS) - a system based upon email standards to deliver message based communications throughout the Department of Defense. It was designed to replace AUTODIN.

Delay - in VoIP it is the time it takes for speech to transmit from the speakers mouth to the listeners ear.

Differentiated Service (DiffServ) - uses a code in the Type-of-Service (TOS) field of the IP header to determine priority handling.

Digital-to-analog Converter (DAC) - accepts a digital input and converts it to a voltage or current output.

Enhanced 911 - a safety related service that associates location information with an emergency call. Because the information comes from the phone company, systems such as a traditional or VoIP PBX must have a mechanism to provide this information.

Extremely-high Frequency (EHF) - the frequency spectrum from 30 - 300 GHz and is often used for military satellite communications.

FCC100 - a Time Division Multiplexing (TDM)/ Multiplexer (MUX) used in the ADNS system.

Gatekeeper - used in VoIP to control access to the network, manage bandwidth, and serve as the address resolution component.

Gateway - provides the translation functions for the voice / data conversions.

H.323 - an ITU-T standard that offers audio, video and data communications across packet-based network infrastructures. H.323 provides standards for encoding, bandwidth management, admission control, address translation, call control and management, and links to external networks. The H.323 protocol stack comprises a set of protocols that ride on TCP/IP and UDP/IP, where TCP is used for call setup and control, while UDP is used for data transmission and reception.

Inline Network Encryption (INE) - an device to provide payload encryption on a packet by packet basis but leave the IP header information in plain text. The device that is currently in use for ADNS is the KG-194 TACLANE.

IP Security (IPSec) - A protocol that provides security for transmission of sensitive information over unprotected networks such as the Internet.

Integrated Services Digital Network (ISDN) - a set of communications protocols that specify the carrying of voice, video, and data over a single wire that is eventually supposed to replace POTS.

Jitter - the variation in delay between packets.

Key System - a business telephone system that generally is cheaper than a PBX but also contains fewer features. Generally suited for smaller offices.

Latency - see delay.

Mean Opinion Score (MOS) - a subjective scoring system for rating the quality of voice communications. Obtained by having a number of people listen to various voice transmissions and averaging their ratings of between 1 (worst) and 5.

Media Gateway Control (MEGACO/H.248) - a standard developed jointly by the IETF and ITU to recommend controls for gateways between networks.

Multi Gateway Control Protocol (MGCP) - an IETF standard to recommend controls for gateways between networks.

Multi-level Precedence and Preemption (MLPP) - a priority scheme in military communications that give priority to certain calls and specifies timeframes for handling those calls.

Multipoint Control Unit (MCU) - connects three or more terminals in a "conference call".

Packet - a generic term used to describe a unit of data.

Plain Old Telephone System (POTS) - a term used to describe the traditional, analog based, telephone system.

Private Branch Exchange (PBX) - a business telephone system that allows the business complete control over its configuration.

Public Switched Telephone Network (PSTN) - the collection of interconnected systems operated by the various telephone companies and administrations around the world.

Quality of Service (QoS) - a networking term that specifies a guaranteed throughput level.

Radio Frequency (RF) - a frequency in the range within which radio waves may be transmitted, from about 3 kilohertz to about 300,000 megahertz.

Real-Time Transport Protocol (RTP) - provides real-time delivery of data, in particular voice traffic. RTP is typically built on UDP but includes a sequencing system to detect missing packets, as well as information regarding the payload type including the audio and video encoding used.

Real-Time Transport Control Protocol (RTCP) - provides a means to exchange quality of service information between nodes using RTP.

Secure Telephone Equipment (STE) - the replacement for the STU-III.

Secure Telephone Unit - Third Generation (STU-III) - a device designed to enable secure voice communications over an unsecure voice network.

Server - in VoIP this is a general term for the Gatekeepers and Gateways.

Session Description Protocol (SDP) - used by other protocols as a standard format to describe a session.

Session Initiation Protocol (SIP) - considered as the IETF's replacement for H.323, and is a text-based signaling protocol sent over TCP or UDP.

Silence Suppression - a method of conserving bandwidth in a VoIP call by not encoding and sending voice packets during periods of silence.

Super-high Frequency (SHF) - the radio frequencies between 3 - 30 GHz. Well suited for satellite communication, it is the band in which INMARSAT operates.

Tie-line - a communications link between two PBX's.

Time Division Multiplex (TDM) - a type of multiplexing that assigns each voice or data stream its own timeslot.

Transmission Control Protocol (TCP) - a connection-oriented protocol that provides guaranteed delivery of its payload.

Ultra-high Frequency (UHF)

Unit Level Ship - used to contrast with a force level ship (LHA/LHD or CV/CVN). In this paper it generally refers to a DDG or CG.

User Agent - the software that interfaces with and acts on behalf of the user. Sometimes referred to as a terminal.

User Datagram Protocol (UDP) - a connectionless protocol that does not provide guaranteed delivery.

Virtual Private Network (VPN) - a network designed for private information created using a public network to

connect the nodes. Encryption is usually employed to ensure that only authorized users have access to the private network.

Voice Activity Detection (VAD) - see silence suppression.

Voice over Internet Protocol (VoIP) - the transmission of voice over an IP based network.

Weighted Random Early Drop (WRED) - a congestion avoidance mechanism that drops packets before congestion occurs, based upon precedence. Lower priority packets are more likely to be dropped in order to reduce congestion and avoid having to drop higher priority packets.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SIMULATION CODE

The following simulations were written to run using Omnetpp-3.0a3 and IPSuite-20040322 which can be obtained from www.omnetpp.org. Later versions of the software change the mechanisms for creating and sending messages and will require modifications to this code. This appendix begins with changes that were made to the IPSuite source code to fix a few bugs and to allow for data collection in the TCP Client Application. The second section provides the source for the basic components followed by the last section with the source for the networks used for analysis.

A. CHANGES TO IPSUITE SOURCE

IPSuite-20040322\Applications\TCPApp\procserver.cc

Line 110

Change

```
msg = receive(appl_timeout);
```

To

```
goto broken;//application terminates connection
```

Fixes - Problem that the client will continue to wait if the server terminates connection due to a timeout. This fix sends the application into existing code for handling a broken connection.

Line 178

Change

```
msg = receive(appl_timeout);
```

To

```
goto broken;//application terminates connection
```

Fixes - Problem that the client will continue to wait if the server terminates connection due to a timeout. This fix sends the application into existing code for handling a broken connection.

Line 217

Change

```
msg = receive(appl_timeout);
```

To

```
goto broken;//application terminates connection
```

Fixes - Problem that the client will continue to wait if the server terminates connection due to a timeout. This fix sends the application into existing code for handling a broken connection.

IPSuite-20040322\Applications\TCPApp\TCPClient.cc

Line 293

Inserted

```
//added to fix prob with close  
abort = new cMessage("TCP_ABORT", TCP_C_ABORT);  
abort->addPar("src_port") = local_port;  
abort->addPar("src_addr") = local_addr;  
abort->addPar("dest_port") = rem_port;  
abort->addPar("dest_addr") = rem_addr;  
abort->addPar("tcp_conn_id") = tcp_conn_id;
```

```

//no data bits to send
abort->setLength(0);
//no data packets to receive
abort->addPar("rec_pks") = 0;
//make delay checking possible
abort->setTimestamp();
//send "receive" to "TcpModule"
send(abort, "out");

```

Fixes - TCP module waits for the client sends a close message to the client only once and then waits for a reply. For large messages, the client is still processing its queues and does not send the reply expected. This fix will use the existing abort mechanism to continue the process of closing the connection.

IPSuite-20040322\Nodes\IPSuite\TCPUpperLayers.ned

Line 63-64

Change

```

//message_length = input(8000, Number of bits to
be received: ");

```

```

message_length = 8000;

```

To

```

message_length = input(8000, "Number of bits to
be received: ");

```

```

//message_length = 8000;

```

Fixes - This change allows the message length to be specified at run time.

IPSuite-20040322\Transport\TCP\tcpmodule.cc

Line 84

Add

```
        cOutVector *goodput; //jak for recording goodput
        cOutVector *avg_goodput; //jak for recording
goodput
        cOutVector *rec_bits; //jak for recording goodput
```

Fixes - Adds the vectors needed to record goodput calculations.

Line 92

Add

```
        //jak for goodput calculations
        cQueue bw_msg_q; //store messages for goodput
calcs
        simtime_t span; //length of time between messages
for goodput calculation
        double bits; //length of all msgs in bw_msg_q
```

Fixes - Variables needed to calculate goodput.

Line 191

Add

```
~TcpModule();
```

Fixes - When closing OMNeT or rebuilding a network, windows reports a memory access violation for networks that use the TCP module. This adds a destructor that will fix one of the problems causing this error. This fix is from Andras Varga and is included in subsequent releases of IPSuite.

Line 195

```
//Added per Andres to fix error when terminating
application

TcpModule::~~TcpModule()
{
    // clear unused TCB or active connections
    TcbList::iterator iter = tcb_list.begin();
    while (iter != tcb_list.end())
    {
        TcpTcb *tcb_block = (TcpTcb *) iter->second;
        while (tcb_block->tcp_rcv_rec_list.length() >
0) {
            SegRecord* seg_rec = (SegRecord *)
tcb_block->tcp_rcv_rec_list.pop();
            delete seg_rec->pdata;
        }
        delete tcb_block;
        iter ++;
    }
}
```

```

while (!bw_msg_q.empty())
delete((cMessage *)bw_msg_q.pop());
delete tcpdelay;
delete cwnd_size;
delete send_seq_no;
delete rec_ack_no;
delete goodput; //jak for goodput
delete avg_goodput; //jak for goodput
}

```

Fixes - When closing OMNeT or rebuilding a network, windows reports a memory access violation for networks that use the TCP module. This adds a destructor and populates it with code that was in the finish method. This fix will correct one of the problems causing this error. This fix is from Andras Varga and is included in subsequent releases of IPSuite.

Line 238

Add

```

//jak - name vectors for recording goodput
goodput = new cOutVector("Goodput");
avg_goodput = new cOutVector("Avg_Goodput");
rec_bits = new cOutVector("Rec_Bits");
span = strToSimtime("2s"); //jak-time span to
consider in goodput calcs
bits = 0; //jak - initialize

```

```
WATCH(bits); //jak for goodput
```

Fixes - Initialize vectors and variables for goodput calculations.

Line 259

Delete

```
// clear unused TCB or active connections
TcbList::iterator iter = tcb_list.begin();
while (iter != tcb_list.end())
{
    TcpTcb *tcb_block = (TcpTcb *) iter->second;
    while (tcb_block->tcp_rcv_rec_list.length() >
0) {
        SegRecord* seg_rec = (SegRecord *)
tcb_block->tcp_rcv_rec_list.pop();
        delete seg_rec->pdata;
    }
    delete tcb_block;
    iter++;
}
delete tcpdelay;
delete cwnd_size;
delete send_seq_no;
delete rec_ack_no;
delete rec_seq_no;
```

Fixes - When closing OMNeT or rebuilding a network, windows reports a memory access violation for networks that use the TCP module. This adds a destructor that will fix one of the problems causing this error. This fix is from Andras Varga and is included in subsequent releases of IPSuite.

Line 755-776

Uncomment

Fixes - Client application will continue to wait for a closed message from TCP module until a timeout is received and an abort is initiated. This code ends a closed message to the client application even if connection not being aborted. Unknown why it was commented out other than the mechanism does not work if the complete message being received is large.

Line 850

Add

```
//jak - calculate goodput whenever a packet is
received from the server.
```

```
    if (eventsources == FROM_IP)
    {
        //remove messages older than the window of
        interest
        while (!bw_msg_q.empty() && (((cMessage
        *)bw_msg_q.tail())->timestamp())<=(simTime()-span)))
            delete ((cMessage *)bw_msg_q.pop());
```

```

        double qbits = numBitsInQueue(bw_msg_q);
//calc total bits received in window

        if (qbits>0)

            goodput->record((qbits-((cMessage
*)bw_msg_q.tail()->length())/simTime()-((cMessage
*)bw_msg_q.tail()->timestamp()))); //divide the bits
received by the time span - oldest message is removed to
allow the bits to reflect those received in an actual span
of time and make the calculation more accurate.

            avg_goodput->record(bits/simTime());
//average over entire run

            if (!bw_msg_q.empty())

                rec_bits->record((cMessage
*)bw_msg_q.tail()->length()); //record bits received

        }

```

Fixes - Calculates goodput whenever a new message is received from the server.

Line 2255

Add

```

//jak - record bits received

cMessage *bw_msg = (cMessage *) pdata->dup();

bits = bits+bw_msg->length();

bw_msg->setTimestamp(simTime());

bw_msg_q.insert(bw_msg);

```

Fixes - Records the size of the message received to perform goodput calculations.

Line 2261

Add

//jak - flushes duplicated part of message in
current packet

```
flushQueue(bw_msg_q, (tcb_block->rcv_nxt -  
tcb_block->seg_seq), false);
```

```
bits = bits - ((tcb_block->rcv_nxt - tcb_block->  
>seg_seq)*8);
```

Fixes - Flushes that part of a message that has
already been recorded to prevent double counting a message.

Line 2268

Add

//jak - record parts of messages received out of
order for accurate reflection of goodput

```
cMessage *bw_msg = (cMessage *) pdata->dup();
```

```
bits = bits+bw_msg->length();
```

```
bw_msg->setTimestamp(simTime());
```

```
bw_msg_q.insert(bw_msg);
```

Fixes - Records messages that were received out of
order to ensure accurate accounting for goodput
calculations.

IPSuite-20040322\Transport\UDP\UDPProcessing.cc

Line 147

Add

```
        ipIfPacket->setDiffServCodePoint(udpIfPacket->getCodePoint());    //--added by jak to transfer DSCP to IP packet
```

Fixes - Populates the DSCP in the IP Header TOS. This was not previously done even though the mechanism existed.

B. COMPONENTS

```
//-----
```

```
// file: INE.ned
```

```
// author: James Knoll
```

```
//
```

```
// Date: 13 May, 2004
```

```
//
```

```
// This is an implementation of an INE based upon the
```

```
// Taclane description found in Analysis of Quintum
```

```
// Tenor Vocoding for Support for Secure Voice, written by
```

```
// Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis,
```

```
// Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al.
```

```
// It is composed of this file, an encoder, and a decoder.
```

```
// Plain text is sent in the plainIn gate and is encrypted
```

```
// and sent out the cypherOut gate. Encrypted packets are
```

```
// sent into the INE via the cipherIn gate and is decrypted
```

```
// and output through the plainOut gate. The INE was
```

```
// created to change the length of the IP header rather
```

```
// than encapsulating the message in a new message in order
```

```
// to save on resources. As a result the decrypted packets
// still contain padding in the IP header, but are the
// correct length in the UDP header.
```

```
//-----
```

```
import
```

```
    "LinkLayer",
```

```
    "INEEncode",
```

```
    "INEDecode";
```

```
module INE
```

```
    gates:
```

```
        in: plainIn; //unencoded packets in
```

```
        in: cypherIn; //encoded packets in
```

```
        out: plainOut; //decoded packets out
```

```
        out: cypherOut; //encoded packets out
```

```
    submodules:
```

```
        plainProcess: INEEncode; //Encodes the plaintext
```

```
        display: "p=100,60;i=fork";
```

```
        cypherProcess: INEDecode; //Decodes the cyphertext
```

```
        display: "p=160,60;i=fork";
```

```
        plainnetIf : LinkLayer;..//Handles the Link layer
information
```

```
    parameters:
```

```
        NWIName = "PPPModule";
```

```

        display: "p=80,120,row;i=iface";

    cyphernetIf : LinkLayer;    //    Handles    the    Link
layer information

    parameters:

        NWIName = "PPPModule";

        display: "p=120,120,row;i=iface";

connections nocheck:

    // connections to network outside

    plainIn --> plainnetIf.physIn;
    plainnetIf.inputQueueOut --> plainProcess.physIn;
    cyphernetIf.outputQueueIn <-- plainProcess.physOut;
    cypherOut <-- cyphernetIf.physOut;

    cypherIn --> cyphernetIf.physIn;
    cyphernetIf.inputQueueOut --> cypherProcess.physIn;
    plainnetIf.outputQueueIn <-- cypherProcess.physOut;
    plainOut <-- plainnetIf.physOut;

endmodule

//-----
// file: INEEncode.ned
// author: James Knoll
//

```

```

// Date: 13 May, 2004
//
// An implementation of an INE encoder based upon the
// Taclane description found in Analysis of Quintum
// Tenor Vocoding for Support for Secure Voice, written by
// Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis,
// Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al.
//-----

simple INEEncode
    parameters:
        gates:
            in: physIn; //in from network interface
            out: physOut; //out to network interface
endsimple

//-----

// file: INEEncode.cc
// author: James Knoll
//
// Date: 13 May, 2004
//
// An implementation of an INE encoder based upon the
// Taclane description found in Analysis of Quintum
// Tenor Vocoding for Support for Secure Voice, written by

```

```
// Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis,
// Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al.
//-----
```

```
#include <omnetpp.h>
```

```
class INEEncode : public cSimpleModule
```

```
{
```

```
    public:
```

```
        Module_Class_Members(INEEncode, cSimpleModule, 0);
```

```
        virtual void handleMessage(cMessage *msg);
```

```
};
```

```
Define_Module(INEEncode);
```

```
void INEEncode::handleMessage(cMessage *msg)
```

```
{
```

```
    double msg_length = msg->length()/8; //length in Bytes
```

```
    //Calculate encoded length by add 12 bytes of security
```

```
    // information to the message and then pad to a 48 byte
```

```
    // increment. Another 20 bytes of security information
```

```
    // is then added along with a new 20 byte IP header.
```

```
    msg_length = ceil((msg_length+12)/48)*48+40;
```

```

    msg->setLength(msg_length*8); //length in bits

    send(msg, "physOut");
}

//-----
// file: INEDecode.ned
// author: James Knoll
//
// Date: 13 May, 2004
//
// An implementation of an INE decoder based upon the
// Taclane description found in Analysis of Quintum
// Tenor Vocoding for Support for Secure Voice, written by
// Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis,
// Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al.
//-----

simple INEDecode
parameters:
    gates:
        in: physIn; //in from network interface
        out: physOut; //out to network interface
endsimple

```

```
//-----
// file: INEDecode.cc
// author: James Knoll
//
// Date: 13 May, 2004
//
// An implementation of an INE encoder based upon the
// Taclane description found in Analysis of Quintum
// Tenor Vocoding for Support for Secure Voice, written by
// Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis,
// Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al.
//-----
```

```
#include <omnetpp.h>
```

```
class INEDecode : public cSimpleModule
{
    public:
        Module_Class_Members(INEDecode, cSimpleModule, 0);
        virtual void handleMessage(cMessage *msg);
};

Define_Module(INEDecode);
```

```
void INEDecode::handleMessage(cMessage *msg)
```

```

{
    double msg_length = msg->length()/8; //Length in bytes

    //To find the length of the decoded packet, the
    // 20 bytes of IP header and the 20 bytes of security
    // header are first removed. The padding is not
    // removed, but the additional 12 bytes of security
    // header is. For these simulation the additional
    // padding in the IP header is not important since the
    // UDP header will still be the original length.
    msg_length = msg_length-40-12; //does not remove
padding

    msg->setLength(msg_length*8); //length in bits

    send(msg, "physOut");
}

```

```

//-----
// file: trafficUDPHost.ned
// author: James Knoll
//
// Date: 26 Apr, 2004
//
// UDP application created to simulate background network

```

```
// traffic. The client Continuously transmits based upon
// the data rate specified. The server application handles
// incoming messages by recording the desired metrics such
// as delay and then deleting the packet. A UDP
// application was chosen to simulate network traffic since
// it was able to be adjusted to easily provide different
// levels of network saturation and because the metrics
// were easier to obtain from a single application.
//-----
```

```
import
```

```
    "LinkLayer",
    "NetworkLayer",
    "trafficUDPUpperLayers";
```

```
module trafficUDPHost
```

```
    parameters:
```

```
        dest_addr : string, //list of destination addresses
        local_port : numeric const, //client port
        dest_port : numeric, //server port
        msg_length : numeric, // Max length of a message
(bits)
        start_delay : bool, //delay start of transmit?
        traffic_rate : numeric, //rate traffic to be
generated
```

```

        local_addr : string,
        numOfPorts : numeric,    //allows connection to
multiple nodes

        routingFile : string;    //file name of routing file
for this host

    gates:
        in: in[];
        out: out[];

    submodules:
        udpApp: trafficUDPUpperLayers;

        parameters:
            dest_addr = dest_addr,
            local_port = local_port,
            dest_port = dest_port,
            msg_length = msg_length,
            start_delay = start_delay,
            traffic_rate = traffic_rate,
            local_addr = local_addr,
            udpClient1Name = "trafficUDPClientApp",
//specifies app to use
            udpServer1Name = "trafficUDPServerApp";
//specifies app to use

            display: "p=89,68;b=40,24,rect";

        networkLayer: NetworkLayer;

        parameters:

```

```

        IPForward = 0, //this node will not
forward traffic intended for a different host

        numOfPorts = numOfPorts,

        routingFile = routingFile;

    gatesizes:

        physIn[numOfPorts],

        physOut[numOfPorts];

    display: "p=87,155;i=fork";
netIf : LinkLayer[numOfPorts];

    parameters:

        NWIName = "PPPModule"; //specify link
layer to use

        display: "p=80,220,row;i=iface";

connections nocheck:

    // transport connections

    networkLayer.UDPOut --> udpApp.from_ip;
    networkLayer.UDPIn <-- udpApp.to_ip;


    // connections to other nodes
    for i=0..numOfPorts-1 do
        in[i] --> netIf[i].physIn;
        out[i] <-- netIf[i].physOut;
        netIf[i].inputQueueOut
networkLayer.physIn[i];
-->

```

```

        netIf[i].outputQueueIn                                <--
networkLayer.physOut[i];

    endfor;

endmodule

//-----

// file: trafficUDPUpperLayers.ned
// author: James Knoll
//
// Date: 26 Apr, 2004
//
// UDP application created to simulate background network
// traffic. The client continuously transmits based upon
// the data rate and the server discards the messages.
//-----

import "UDPProcessing";
import "trafficUDPApp";

module trafficUDPUpperLayers
    parameters:
        dest_addr    :    string,        //list of destination
addresses
        local_port   :    numeric const, //client port
        dest_port    :    numeric,       //server port

```

```

        msg_length : numeric,    // Max length of a message
(bits)

        start_delay : bool,    //delay start of transmit

        traffic_rate : numeric,    //rate traffic is to be
generated

        local_addr : string,

        udpClient1Name : string,    //client app to use

        udpServer1Name : string;    //server app to use

    gates:

        in: from_ip;

        out: to_ip;

    submodules:

        udpProcessing: UDPProcessing;

        gatesizes:

            from_application[2],

            to_application[2];

        display: "p=94,105;i=fork";

        udpClient1: udpClient1Name like trafficUDPApp;

        parameters:

            dest_addr = dest_addr,

            local_port = local_port,

            dest_port = dest_port,

            msg_length = msg_length,

            start_delay = start_delay,

            traffic_rate = traffic_rate,

```

```

        local_addr = local_addr;

        display: "p=134,43;b=48,32,rect";

udpServer1: udpServer1Name like trafficUDPApp;

    parameters:

        local_port=dest_port;

        display: "p=51,42;b=40,24,rect";

connections nocheck:

    from_ip --> udpProcessing.from_ip;

    to_ip <-- udpProcessing.to_ip;


        udpProcessing.to_application[0]                                -->
udpClient1.from_udp;

        udpProcessing.from_application[0]                            <--
udpClient1.to_udp;


        udpProcessing.to_application[1]                                -->
udpServer1.from_udp;

        udpProcessing.from_application[1]                            <--
udpServer1.to_udp;

        display: "p=10,10;b=157,140,rect";

endmodule


//-----
// file: trafficUDPApp.ned
// author: James Knoll

```

```

//
// Date: 26 Apr, 2004
//
// UDP application created to simulate background network
// traffic. The client continuously transmits based upon
// the data rate and the server discards the messages.
//-----

//
// Peer of trafficUDPServerApp. Sends UDP packets to
// randomly chosen destinations at random intervals.
// Destinations are chosen from the dest_addresses
// parameter.
//
simple trafficUDPClientApp
    parameters:
        local_port : numeric const,
        dest_port : numeric const,    //must match far end
server local port
        msg_length : numeric const,    // Max length of a
message (bits)
        start_delay : bool,    //delay start of transmit
        traffic_rate : numeric,    //rate traffic to be
generated
        local_addr : string,

```

```

        dest_addr: string; // destination IP address
    gates:
        in: from_udp;
        out: to_udp;
endsimple

//
// Peer of trafficUDPClientApp. At the moment just discards
// received packets.
//
simple trafficUDPServerApp
    parameters:
        local_port : numeric const;
    gates:
        in: from_udp;
        out: to_udp;
endsimple

//-----
// file: trafficUDPApp.h
// author: James Knoll
//
// Date: 26 Apr, 2004
//
// UDP application created to simulate background network

```

```

// traffic. The client continuously transmits based upon
// the data rate and the server discards the messages.
//-----

#ifndef __TRAFFICUDPAPP_H__
#define __TRAFFICUDPAPP_H__

#include <vector>
#include <omnetpp.h>

#include "basic_consts.h"
#include "IPInterfacePacket.h"

//
// UDP server app.
//
class trafficUDPServerApp : public cSimpleModule
{
protected:
    int numReceived;    //number of messages received
    cOutVector delay_v; //records delay
    cOutVector receive_v;//records average number of
messages received per second

public:

```

```

        Module_Class_Members(trafficUDPServerApp,
cSimpleModule, 0);

        virtual void initialize();

        virtual void handleMessage(cMessage *msg);
};

//

// UDP client app.

//

class trafficUDPClientApp : public cSimpleModule
{
protected:

    enum MsgKinds    //types of messages
    {

        TRAFFIC,

        VOIP_DATA,

        DATA_COLLECT,

        TIMEOUT_THINK

    };

    std::string nodeName;    //used to determine which
application to use

    int localPort, destPort; //numbers not important as
long as local matches remote dest

    int msgLength;    //length of each message

```

```

    bool startDelay;//delay before starting to transmit?

    double trafficRate;//bits per second to send

    IPAddress localAddr;

    std::vector<IPAddress> destAddresses; //ability to
    randomly send to diff addrs


    double msgInterval; //time between messages


    int numSent; //number of messages sent

    cOutVector send_v; //average number of msgs sent per
    second


    // chooses random destination address

    IPAddress chooseDestAddr();


public:
    Module_Class_Members(trafficUDPClientApp,
    cSimpleModule, 0);

    virtual void initialize();

    virtual void handleMessage(cMessage *msg);
};


#endif


//-----

```

```

// file: trafficUDPApp.cc
// author: James Knoll
//
// Date: 26 Apr, 2004
//
// UDP application created to simulate background network
// traffic. The client continuously transmits based upon
// the data rate and the server discards the messages.
//-----

#include <omnetpp.h>
#include "trafficUDPApp.h"
#include "UDPInterfacePacket_m.h"
#include "StringTokenizer.h"

Define_Module(traffiUDPServerApp);

void traffiUDPServerApp::initialize()
{
    numReceived = 0; //number of messages received
    WATCH(numReceived);

    delay_v.setName("delay_time"); //delay between when
    message sent and received

```

```

        receive_v.setName("receive_rate");    //msg    received
divided by elapsed simTime
}

```

```

//Receive message, record metrics, and discard

```

```

void trafficUDPServerApp::handleMessage(cMessage *msg)

```

```

{

```

```

    //cast msg as UDP Interface Packet and retrieve payload

```

```

    UDPInterfacePacket          *udpIfPacket          =
check_and_cast<UDPInterfacePacket *>(msg);

```

```

    cMessage *payload = udpIfPacket->decapsulate();

```

```

    //get specifics about message and print

```

```

    IPAddress src = udpIfPacket->getSrcAddr();

```

```

    IPAddress dest = udpIfPacket->getDestAddr();

```

```

    int sentPort = udpIfPacket->getSrcPort();

```

```

    int recPort = udpIfPacket->getDestPort();

```

```

    simtime_t sent = payload->creationTime();

```

```

    simtime_t arrive = udpIfPacket->arrivalTime();

```

```

    ev << "Packet received: " << payload << endl;

```

```

    ev << "Payload length: " << (payload->length()/8) << "
bytes" << endl;

```

```

        ev << "Src/Port: " << src << " / " << sentPort << "
";
        ev << "Dest/Port: " << dest << " / " << recPort <<
endl;
        ev << "Sent/Arrive: " << sent << " / " << arrive <<
endl;

```

```

//record delay and average number received

```

```

delay_v.record(simTime()-sent);

```

```

numReceived++;

```

```

receive_v.record(numReceived/simTime());

```

```

//discard msg

```

```

delete udpIfPacket;

```

```

delete payload;

```

```

}

```

```

//=====

```

```

Define_Module(trafficUDPCliientApp);

```

```

void trafficUDPCliientApp::initialize()

```

```

{

```

```

    send_v.setName("send_rate"); //set name of vector

```

```

//get parameters
localPort = par("local_port");
destPort = par("dest_port");
msgLength = par("msg_length");
startDelay = par("start_delay");
trafficRate = par("traffic_rate");
const char *localAddress = par("local_addr");
localAddr =IPAddress(localAddress);

//parse destination addresses
const char *destAddrs = par("dest_addr");
StringTokenizer tokenizer(destAddrs);
const char *token;
while ((token = tokenizer.nextToken())!=NULL)
    destAddresses.push_back(IPAddress(token));

msgInterval = (msgLength/trafficRate);//how fast do we
send messages

//initialize
numSent = 0;
WATCH(numSent);

cMessage *timer = new cMessage("sendTimer"); //self
message for next transmit

```

```

//schedule first message
if (startDelay)
    scheduleAt(msgInterval+dblrand(), timer);
else
    scheduleAt(dblrand(), timer);
}

//handle incoming msgs
void trafficUDPClientApp::handleMessage(cMessage *msg)
{

    scheduleAt(simTime()+msgInterval, msg); //schedule next
message

    char msgName[32];
    sprintf(msgName,"udpAppData-%d", numSent);

    //create payload
    cMessage *payload = new cMessage(msgName);
    payload->setLength(msgLength);

    //header information to be passed on

```

```

        UDPInterfacePacket      *udpIfPacket      =      new
UDPInterfacePacket();

    udpIfPacket->encapsulate(payload);

    IPAddress destAddr = chooseDestAddr();
    IPAddress locAddr = localAddr;
    udpIfPacket->setSrcAddr(locAddr);
    udpIfPacket->setDestAddr(destAddr);
    udpIfPacket->setSrcPort(localPort);
    udpIfPacket->setDestPort(destPort);

    //print header info to user interface
    ev << "Packet sent: " << payload << endl;

    ev << "Payload length: " << (payload->length()/8) << "
bytes" << endl;

    ev << "Src/Port: " << locAddr << " / " << localPort <<
endl;

    ev << "Dest/Port: " << destAddr << " / " << destPort <<
endl;

    send(udpIfPacket, "to_udp"); //send msg

    //record average number of messages sent
    numSent++;

    send_v.record(numSent/simTime());

```

```
}
```

```
//randomly choose a destination
```

```
IPAddress trafficUDPClientApp::chooseDestAddr()
```

```
{
```

```
    int k = intrand(destAddresses.size());
```

```
    return destAddresses[k];
```

```
}
```

```
//-----
```

```
// file: voipUDPHost.ned
```

```
// author: James Knoll
```

```
//
```

```
// Date: 13 Apr, 2004
```

```
//
```

```
// This is a UDP application to send a burst of
```

```
// conversation to the specified address, and then wait for
```

```
// a reply. A conversation should be started by only one
```

```
// node and the delay before replying must be longer than
```

```
// the delay or the nodes will step on each other. Call
```

```
// cycle is accomplished by setting a timer within both
```

```
// nodes to start and stop conversations at a predetermined
```

```
// interval. If random intervals are used, the same value
```

```
// should be passed to both nodes in the conversation since
```

```
// there is not any synchronization mechanism in place.
```

```
//-----
```

```
import
```

```
    "LinkLayer",
```

```
    "NetworkLayer",
```

```
    "voipUDPUpperLayers";
```

```
module voipUDPHost
```

```
    parameters:
```

```
        local_addr : string,
```

```
        dest_addr: string, // Destination IP address
```

```
        local_port : numeric const,
```

```
        dest_port : numeric const, //must match far end
```

```
local port
```

```
        voice_length : numeric const, //length of a voice  
conversation segment
```

```
        initiate : bool, //delay start of transmit on  
receiving end
```

```
        codec_rate : numeric const, //analog to digital  
conversion encoding
```

```
        reply_delay :numeric const, //Time to pause before  
a response begins
```

```
        frame_size :numeric const, //length of a frame
```

```
        talk_cycle:numeric, //percent of off hook time
```

```
        call_length: numeric, //length of a call
```

```

        init_delay: numeric; //amount to delay before the
first conversation

        numOfPorts : numeric const, //allows connection to
multiple nodes

        routingFile : string; /routing file to use
gates:
    in: in[];
    out: out[];
submodules:
    udpApp: voipUDPUpperLayers;
        parameters:
            local_addr = local_addr,
            dest_addr = dest_addr,
            local_port = local_port,
            dest_port = dest_port,
            voice_length = voice_length,
            initiate = initiate,
            codec_rate = codec_rate,
            reply_delay = reply_delay,
            talk_cycle = talk_cycle,
            call_length = call_length,
            init_delay = init_delay,
            frame_size = frame_size,
            udpClient1Name      =      "voipUDPClientApp";
//client app to use

```

```

        display: "p=89,68;b=40,24,rect";
networkLayer: NetworkLayer;
parameters:
    IPForward = 0,          //node does not forward
    numOfPorts = numOfPorts,
    routingFile = routingFile; //routing file
to use

gatesizes:
    physIn[numOfPorts],
    physOut[numOfPorts];
display: "p=87,155;i=fork";
netIf : LinkLayer[numOfPorts];
parameters:
    NWIName = "PPPModule"; //link layer to use
display: "p=80,220,row;i=iface";
connections nocheck:
    // transport connections
    networkLayer.UDPOut --> udpApp.from_ip;
    networkLayer.UDPIn <-- udpApp.to_ip;

    // connections to other nodes
    for i=0..numOfPorts-1 do
        in[i] --> netIf[i].physIn;
        out[i] <-- netIf[i].physOut;

```

```

                netIf[i].inputQueueOut                -->
networkLayer.physIn[i];

                netIf[i].outputQueueIn                <--
networkLayer.physOut[i];

        endfor;

endmodule

```

```

//-----
// file: voipUDPUpperLayers.ned
// author: James Knoll
//
// Date: 13 Apr, 2004
//
// This is a UDP application to send a burst of
// conversation to the specified address, and then wait for
// a reply.
//-----

```

```

import "UDPProcessing";
import "voipUDPApp";

```

```

module voipUDPUpperLayers

```

```

    parameters:

```

```

        local_addr : string, //local IP address

```

```

        dest_addr: string, // Destination IP address

```

```

        local_port : numeric const,
        dest_port : numeric const,    //must match far end
local port

        voice_length : numeric const, //length of a voice
conversation segment

        initiate : bool,    //delay start of transmit on
receiving end

        codec_rate : numeric const, //analog to digital
conversion encoding

        reply_delay :numeric const, //Time to pause before
a response begins

        frame_size :numeric const, //length of a frame
        talk_cycle:numeric, //percent of off hook time
        call_length: numeric, //length of a call
        init_delay: numeric; //amount to delay before the
first conversation

        udpClient1Name : string; //client to use
gates:
    in: from_ip;
    out: to_ip;
submodules:
    udpProcessing: UDPProcessing;
        gatesizes:
            from_application[1],
            to_application[1];
        display: "p=94,105;i=fork";

```

```

udpClient1: udpClient1Name like voipUDPApp;

parameters:
    local_addr = local_addr,
    dest_addr = dest_addr,
    local_port = local_port,
    dest_port = dest_port,
    voice_length = voice_length,
    initiate = initiate,
    codec_rate = codec_rate,
    reply_delay = reply_delay,
    frame_size = frame_size,
    talk_cycle = talk_cycle,
    call_length = call_length,
    init_delay = init_delay;

display: "p=134,43;b=48,32,rect";

connections nocheck:

    from_ip --> udpProcessing.from_ip;
    to_ip <-- udpProcessing.to_ip;

    udpProcessing.to_application[0] -->
udpClient1.from_udp;

    udpProcessing.from_application[0] <--
udpClient1.to_udp;

display: "p=10,10;b=157,140,rect";

```

```
endmodule
```

```
//-----  
// file: voipUDPApp.ned  
// author: James Knoll  
//  
// Date: 13 Apr, 2004  
//  
// This is a UDP application to send a burst of  
// conversation to the specified address, and then wait for  
// a reply.  
//-----
```

```
simple voipUDPClientApp
```

```
    parameters:
```

```
        local_addr : string, //local IP address
```

```
        dest_addr: string, // Destination IP address
```

```
        local_port : numeric const, //local port number
```

```
        dest_port : numeric const, //must match far end  
local port
```

```
        voice_length : numeric const, //length of a voice  
conversation segment
```

```
        initiate : bool, //delay start of transmit on  
receiving end
```

```

        codec_rate : numeric const,    //analog to digital
conversion encoding

        reply_delay :numeric const,    //Time to pause before
a response begins

        frame_size :numeric const,    //length of a frame
        talk_cycle:numeric,    //percent of off hook time
        call_length: numeric,    //length of a call
        init_delay: numeric;    //amount to delay before the
first conversation

    gates:

        in: from_udp;
        out: to_udp;

endsimple

```

```

//-----
// file: voipUDPApp.h
// author: James Knoll
//
// Date: 13 Apr, 2004
//
// This is a UDP application to send a burst of
// conversation to the specified address, and then wait for
// a reply.
//-----

```

```

#ifndef __VOIPUDPAPP_H__
#define __VOIPUDPAPP_H__

#include <vector>
#include <omnetpp.h>

#include "basic_consts.h"
#include "IPInterfacePacket.h"

class voipUDPClientApp : public cSimpleModule
{
protected:
    enum MsgKinds //types of msgs
    {
        TRAFFIC,
        VOIP_DATA,
        DATA_COLLECT,
        TIMEOUT_THINK,
        TIMEOUT_CALL
    };

    int localPort, destPort; //dest port must match remote
    local
    IPAddress localAddr;
    IPAddress destAddr;

```

```
double voiceLength; //length of voice transmission in
seconds
```

```
bool initiate;      //initiate conversation?
double codecRate;   //encoding rate
double replyDelay;  //delay before beginning to speak
double frameSize;   //length of a frame in seconds
double talkCycle;   //off hook to on hook ratio
simtime_t callLength; //length of a call
simtime_t initDelay; //time to delay before beginning
conversation
```

```
int burstCount; //number of msgs left to send
int burstNumber; //number of msgs in a burst
int burstSize;  //size of each msg payload
```

```
//jitter calculation
```

```
double delay;
double old_delay;
double jitter;
```

```
//current state
```

```
bool talk;
bool listen;
bool call_estab;
```

```

//self msgs
cMessage *timeout_think;
cMessage *timeout_call;
cMessage *voip_data;

//metrics
int numSent;
int numReceived;
simtime_t lastRec;
simtime_t lastSend;
cOutVector send_v;
cOutVector delay_v;
cOutVector receive_v;
cOutVector inst_send_v;
cOutVector inst_rec_v;
cOutVector jitter_v;

//handles creating and sending a msg
virtual void sendMessage();

public:
    Module_Class_Members(voipUDPClientApp,    cSimpleModule,
0);
    virtual void initialize();

```

```

        virtual void handleMessage(cMessage *msg);
};

#endif

//-----
// file: voipUDPApp.cc
// author: James Knoll
//
// Date: 13 Apr, 2004
//
// This is a UDP application to send a burst of
// conversation to the specified address, and then wait for
// a reply.
//-----

#include <omnetpp.h>
#include "voipUDPApp.h"
#include "UDPInterfacePacket_m.h"
#include "StringTokenizer.h"

Define_Module(voipUDPClientApp);

void voipUDPClientApp::initialize()
{

```

```

//set vector names
send_v.setName("send_rate");
receive_v.setName("receive_rate");
inst_send_v.setName("inst_send_rate");
inst_rec_v.setName("inst_rec_rate");
jitter_v.setName("jitter");
delay_v.setName("delay_time");

//initialize
old_delay = 0;
jitter = 0;
delay = 0;
lastRec = simTime();
lastSend = simTime();
call_estab = false;
numSent = 0;
numReceived = 0;

//read parameters
localPort = par("local_port");
destPort = par("dest_port");

voiceLength = par("voice_length"); //length of voice
transmission in seconds

```

```
    initiate = par("initiate"); //does this host initiate
conversation
```

```
    codecRate = par("codec_rate"); //kbps of codec
```

```
    replyDelay = par("reply_delay"); //delay before
beginning to speak
```

```
    frameSize = par("frame_size"); //length of a frame in
seconds
```

```
    talkCycle = par("talk_cycle"); //off hook to on hook
ratio
```

```
    callLength = par("call_length"); //length of a call
```

```
    initDelay = par("init_delay"); //time to delay before
beginning conversation
```

```
//convert address strings to IPAddress
```

```
const char *localAddress = par("local_addr");
```

```
localAddr =IPAddress(localAddress);
```

```
const char *destAddress = par("dest_addr");
```

```
destAddr =IPAddress(destAddress);
```

```
    burstNumber = ceil(voiceLength/frameSize); //number of
msgs in a burst
```

```
    burstSize = (frameSize*codecRate)+(12*8); //size of
msg with RTP header
```

```

        //timeout msg creation

        timeout_think                =                new
cMessage("TIMEOUT_THINK",TIMEOUT_THINK);        //if timer
expires before next voice packet received, node will begin
transmitting

        timeout_call                =                new
cMessage("TIMEOUT_CALL",TIMEOUT_CALL); //timer to tell when
to go on and off hook

        voip_data = new cMessage("VOIP_DATA", VOIP_DATA);
//schedules next send


        scheduleAt(simTime()+initDelay,                timeout_call);
//schedule first transmission
}

void voipUDPClientApp::handleMessage(cMessage *msg)
{

    //if msg is from remote destination
    if ( (!(msg->isSelfMessage())) )
    {
        ev<<"Received a message from remote dest \n";

        if (listen) //in listen, reschedule think timer to
begin transmitting
        {

```

```

        if (timeout_think->isScheduled())
        {
            cancelEvent(timeout_think);
        }
        scheduleAt(simTime()+replyDelay,
timeout_think);
    }
    else if (!talk) //enter listen and schedule delay
    {
        listen=true;
        scheduleAt(simTime()+replyDelay,
timeout_think);
        ev<< "Receiving conversation. Enter listen
mode. \n";
    }

    //get payload
    UDPInterfacePacket *udpIfPacket =
check_and_cast<UDPInterfacePacket *>(msg);
    cMessage *payload = udpIfPacket->decapsulate();

    //parse and print
    IPAddress src = udpIfPacket->getSrcAddr();
    IPAddress dest = udpIfPacket->getDestAddr();
    int sentPort = udpIfPacket->getSrcPort();

```

```

int recPort = udpIfPacket->getDestPort();
simtime_t sent = payload->creationTime();
simtime_t arrive = udpIfPacket->arrivalTime();

ev << "Packet received: " << payload << endl;
ev << "Payload length: " << (payload->length()/8)
<< " bytes" << endl;
ev << "Src/Port: " << src << " / " << sentPort << "
";
ev << "Dest/Port: " << dest << " / " << recPort <<
endl;
ev << "Sent/Arrive: " << sent << " / " << arrive <<
endl;

//record metrics
delay= arrive-sent;
delay_v.record(delay);

numReceived++;
receive_v.record(numReceived/simTime());

inst_rec_v.record(payload->length()/(simTime()-
lastRec));
lastRec = simTime();

jitter = jitter+(abs(old_delay-delay)-jitter)/16;

```

```

    old_delay = delay;
    jitter_v.record(jitter);

    //clean up
    delete udpIfPacket;
    delete payload;
}
//if message is to transmit a voip msg
else if ((msg->kind()==VOIP_DATA) && msg-
>isSelfMessage())
{
    if (burstCount>0)
    {
        sendMessage();
    }
    else
        error("No message to send"); //should not
reach here

}
//if msg is to start sending
else if ((msg->kind() == TIMEOUT_THINK))
{
    burstCount = burstNumber;
    talk=true;

```

```

        listen=false;

        if (call_estab)
            sendMessage();
    }
    //if msg is for call cycle
    else if ((msg->kind() == TIMEOUT_CALL))
    {
        if (!call_estab) //start call
        {
            ev<<"Begin call \n";
            call_estab = true;
            if (initiate)    //does this node initiate the
conversation?
            {
                if (timeout_think->isScheduled())//left over
timeout

                    cancelEvent(timeout_think);
                    scheduleAt(simTime()+frameSize,
timeout_think); //schedule first send
                    talk=true;
                    listen=false;
            }
            scheduleAt(simTime()+callLength,    timeout_call);
            //schedule time to terminate call

```

```

    }
    else //end call
    {
        call_estab = false;
        if (timeout_think->isScheduled())
            cancelEvent(timeout_think);
        if (voip_data->isScheduled())
            cancelEvent(voip_data);
        talk=false;
        listen=false;
        scheduleAt(simTime()+      ((callLength/talkCycle*
100)- callLength), timeout_call); //schedule time of next
call
    }
}
else
{
    error("Could not determine origin of message(%d)
(forgot to add timeout?)\n",msg->kind()); //should not get
here
}
}

```

//create and send msg

```
void voipUDPClientApp::sendMessage()
```

```

{
    char msgName[32];
    sprintf(msgName, "udpAppData-%d", numSent);

    //create payload
    cMessage *payload = new cMessage(msgName, VOIP_DATA);
    payload->setLength(burstSize);
    payload->setPriority(46);

    //header info for next layer
    UDPInterfacePacket *udpIfPacket = new
UDPInterfacePacket();
    udpIfPacket->encapsulate(payload);
    udpIfPacket->setSrcAddr(localAddr);
    udpIfPacket->setDestAddr(destAddr);
    udpIfPacket->setSrcPort(localPort);
    udpIfPacket->setDestPort(destPort);
    udpIfPacket->setCodePoint(46);

    //print info about packet
    ev << "Packet sent: " << payload << endl;
    ev << "Payload length: " << (payload->length()/8) << "
bytes" << endl;
    ev << "Src/Port: " << localAddr << " / " << localPort
<< " ";

```

```

    ev << "Dest/Port: " << destAddr << " / " << destPort <<
endl;

    send(udpIfPacket, "to_udp"); //send the message

    //average number sent
    numSent++;
    send_v.record(numSent/simTime());

    //packet by packet send rate in bits
    inst_send_v.record(payload->length()/(simTime()-
lastSend));
    lastSend = simTime();

    // schedule next sending
    if (burstCount>1)
    {
        scheduleAt(simTime()+frameSize, voip_data);
        burstCount--; //keep track of number left to send
    }
    else //done talking, wait for reply
    {
        talk=false;
    }
}

```

```
//-----
// file: wredbox.ned
// author: James Knoll
//
// Date: 24 May, 2004
//
// Application to prioritize VoIP msgs and monitor
// throughput. A combination of CBWFQ and WRED but not a
// complete implementation. This is provided as a separate
// node, but could be integrated into a router. The current
// implementation only recognizes high and low priority
// traffic based upon whether or not a DSCP of 46 is present
// in the TOS field. Throughput is calculated and recorded
// for each queue. The pass in and out gates provide a path
// without
//-----
```

```
simple wredApp
```

```
    parameters:
```

```
        bw_max: numeric,    //maximum bandwidth to allocate
to HPQ
```

```
        win:    numeric,    //time span for bandwidth
calculations
```

```
        hpq_min_thresh: numeric,    //minimum queue size
before implementing WRED
```

```

        hpq_max_thresh: numeric,    //maximum queue size
before implementing WRED

        hpq_mpd: numeric,    //maximum percentage of packets
to drop

        lpq_min_thresh: numeric,    //minimum queue size
before implementing WRED

        lpq_max_thresh: numeric,    //maximum queue size
before implementing WRED

        lpq_mpd: numeric,    //maximum percentage of packets
to drop

        max_q_len: numeric,    //max queue size before tail
drop

        n: numeric;    //weight factor

```

```

    gates:

        in: qIn;

        out: qOut;

endsimple

```

```

module wredBox

```

```

    parameters:

        bw_max: numeric,    //maximum bandwidth to allocate
to HPQ

        win:    numeric;    //time span for bandwidth
calculations

    gates:

```

```

    in: passIn;
    out: passOut;
    in: qIn;
    out: qOut;
submodules:
    wredap: wredApp;
        parameters:
            bw_max = bw_max,
            win = win;

        display: "p=160,60;i=fork";

    netIf1 : LinkLayer;
        parameters:
            NWIName = "PPPModule";
            display: "p=80,120;i=iface";
    netIf2 : LinkLayer;
        parameters:
            NWIName = "PPPModule";
            display: "p=160,120;i=iface";

connections:

    passIn --> netIf2.physIn;
    passOut <-- netIf2.physOut;

```

```

        netIf2.inputQueueOut    -->    netIf2.outputQueueIn;
//pass through
        qIn --> netIf1.physIn;
        qOut <-- netIf1.physOut;
        netIf1.inputQueueOut --> wredap.qIn;
        netIf1.outputQueueIn <-- wredap.qOut;
endmodule

```

```

//-----
// file: wredbox.h
// author: James Knoll
//
// Date: 24 May, 2004
//
// Application to prioritize VoIP msgs and monitor
// throughput. A combination of CBWFQ and WRED but not a
// complete implementation. Provided as a separate node, but
// could be integrated into a router.
//-----

```

```

#ifndef __WREDBOX_H__
#define __WREDBOX_H__

```

```

#include <vector>
#include <omnetpp.h>

```

```

class wredApp : public cSimpleModule
{
    protected:

        double bwMax;      //maximum bandwidth to allocate to
HPQ
        double win;  //time span for bandwidth calculations

        cQueue hpq;        //high priority queue
        cQueue lpq;        //low priority queue
        cMessage *next_send;  //self timing message

        double hpq_bits; //length of all msgs in hpq
        double lpq_bits; //length of all msgs in lpq
        cQueue hpq_bw_q; //stores msgs for bw calcs
        cQueue lpq_bw_q; //stores msgs for bw calcs
        simtime_t old_time; //oldest time to include in bw
calc

        int hpq_min_thresh;  //minimum queue size before
implementing WRED
        int hpq_max_thresh;  //maximum queue size before
implementing WRED
        int hpq_mpd;         //maximum percentage of
packets to drop

```

```

        int lpq_min_thresh;    //minimum queue size before
implementing WRED

        int lpq_max_thresh;    //maximum queue size before
implementing WRED

        int lpq_mpd;           //maximum percentage of
packets to drop

        int max_q_len;         //max queue size before tail
drop

        double hpq_avg_q_len;  //average length of queue
        double lpq_avg_q_len;  //average length of queue
        double n;               //weight factor

        cOutVector hpqsize_v;  //
        cOutVector lpqsize_v;
        cOutVector hpbw_v;
        cOutVector lpbw_v;

        void sendMessage();
        void serviceQueues();

        double bw(); //calculate bandwidth used

        bool drop(int min_thresh, int max_thresh, int mpd,
double avg_q_len); //determine if drop

public:
    Module_Class_Members(wredApp, cSimpleModule, 0);
    virtual void initialize();

```

```

        virtual void handleMessage(cMessage *msg);
};

#endif

//-----
// file: wredbox.cc
// author: James Knoll
//
// Date: 24 May, 2004
//
// Application to prioritize VoIP msgs and monitor
// throughput. A combination of CBWFQ and WRED but not a
// complete implementation. Provided as a separate node, but
// could be integrated into a router.
//-----

#include <omnetpp.h>
#include <math.h>
#include "wredbox.h"
#include "IPDatagram.h"

Define_Module(wredApp);

void wredApp::initialize()

```

```

{
    //parameters
    bwMax = par("bw_max");
    win = par("win");
    hpq_min_thresh = par("hpq_min_thresh");
    hpq_max_thresh = par("hpq_max_thresh");
    hpq_mpd = par("hpq_mpd");
    lpq_min_thresh = par("lpq_min_thresh");
    lpq_max_thresh = par("lpq_max_thresh");
    lpq_mpd = par("lpq_mpd");
    max_q_len = par("max_q_len");
    n = par("n");

    //set vector names
    hpqsize_v.setName("HPQ_size");
    lpqsize_v.setName("LPQ_size");
    hpbw_v.setName("HP_BW");
    lpbw_v.setName("LP_BW");

    //initialize
    hpq_bits = 0;
    lpq_bits = 0;

    hpq_avg_q_len = 0;
    lpq_avg_q_len = 0;

```

```

        //timing message for servicing the queues
        next_send = new cMessage("NEXT_SEND");
    }

void wredApp::handleMessage(cMessage *msg)
{
    //if timer, service queues
    if (msg->isSelfMessage())
        serviceQueues();
    //if new msg
    else
    {
        IPDatagram *ipDatagram = check_and_cast<IPDatagram
*>(msg);

        //if high pri msg, insert in hpq
        if (ipDatagram->diffServCodePoint() == 46)
        {
            hpq_avg_q_len = (hpq_avg_q_len * (1-pow(.5,n)))
+ (hpq.length() * pow(.5,n)); //wred algorithm for
weighting the queue length to damp out transient effects

            //do I drop this msg?

```

```

        if ((hpq.length() >= max_q_len) ||
drop(hpq_min_thresh,      hpq_max_thresh,      hpq_mpd,
hpq_avg_q_len))
    {
        delete (ipDatagram); //dropped
        ev<<"Drop from HPQ\n";
    }
    else
    {
        hpq.insert(ipDatagram); //store in the
queue
    }

}
//if low pri msg, insert in lpq
else
{
    lpq_avg_q_len = (lpq_avg_q_len * (1-pow(.5,n))) +
(lpq.length() * pow(.5,n)); //wred algorithm for weighting
the queue length to damp out transient effects

    //do I drop this msg?
    if ((lpq.length() >= max_q_len) ||
drop(lpq_min_thresh,      lpq_max_thresh,      lpq_mpd,
lpq_avg_q_len))
    {

```

```

        delete (ipDatagram); //dropped
        ev<<"Drop from LPQ\n";
    }
    else
    {
        lpq.insert(ipDatagram); //insert into queue
    }
}

//schedule next service of queues
if (!next_send->isScheduled())
    if (parentModule()->gate("qOut")->isBusy())
        scheduleAt(parentModule()->gate("qOut")->transmissionFinishes(), next_send);
    else
        scheduleAt(simTime(), next_send);
}
}

```

```

void wredApp::serviceQueues()
{
    double bw_var = bw(); //determine bw used by hpq

    //service hpq if not over bw allocation
    if (bw_var<bwMax && !hpq.empty())

```

```

{
    //insert in bw calc queue
    cMessage *bw_msg = (cMessage *)((cMessage
*)hpq.tail()->dup());
    bw_msg->setTimestamp(simTime()); //set timestamp
needed for bw calcs
    hpq_bits = hpq_bits+bw_msg->length(); //add to
length of msgs in bw queue
    hpq_bw_q.insert(bw_msg); //store for future calcs

    send((cMessage *) hpq.pop(), "qOut"); //send msg

    hpqsize_v.record(hpq.length()); //record queue size

    //schedule next send
    if (!next_send->isScheduled() && (!hpq.empty() ||
!lpq.empty()))
        if (parentModule()->gate("qOut")->isBusy())
            scheduleAt(parentModule()->gate("qOut")-
>transmissionFinishes(), next_send);
        else
            scheduleAt(simTime(), next_send);
}
//service lpq
else if (!lpq.empty())
{

```

```

        //insert in bw calc queue

        cMessage *bw_msg = (cMessage *)((cMessage
*)lpq.tail()->dup());

        bw_msg->setTimestamp(simTime()); //set timestamp
needed for bw calcs

        lpq_bits = lpq_bits+bw_msg->length(); //add to
length of msgs in bw queue

        lpq_bw_q.insert(bw_msg); //store for future calcs


        send((cMessage *) lpq.pop(), "qOut"); //send msg


        lpqsize_v.record(lpq.length()); //record queue size


        //schedule next send

        if (!next_send->isScheduled() && (!hpq.empty() ||
!lpq.empty()))

            if (parentModule()->gate("qOut")->isBusy())

                scheduleAt(parentModule()->gate("qOut")-
>transmissionFinishes(), next_send);

            else

                scheduleAt(simTime(), next_send);

        }

        else if (!hpq.empty()) //service anyway so that bw not
wasted

        {

            //insert in bw calc queue

```

```

        cMessage *bw_msg = (cMessage *)((cMessage
*)hpq.tail())->dup();

        bw_msg->setTimestamp(simTime()); //set timestamp
needed for bw calcs

        hpq_bits = hpq_bits+bw_msg->length(); //add to
length of msgs in bw queue

        hpq_bw_q.insert(bw_msg); //store for future calcs

        send((cMessage *) hpq.pop(), "qOut"); //send msg

        hpqsize_v.record(hpq.length()); //record queue size

        //schedule next send

        if (!next_send->isScheduled() && (!hpq.empty() ||
!lpq.empty()))

            if (parentModule()->gate("qOut")->isBusy())

                scheduleAt(parentModule()->gate("qOut")-
>transmissionFinishes(), next_send);

            else

                scheduleAt(simTime(), next_send);

    }
}

double wredApp::bw()
{
    double bw_var;

```

```

old_time = simTime()-win;//oldest time to include

bool done = false;

//remove messages older than the window
while (!lpq_bw_q.empty() && !done)
{
    if (((cMessage *)lpq_bw_q.tail())->timestamp())>=
old_time)

        done = true; //done purging messages
    else
    {
        lpq_bits = lpq_bits - ((cMessage
*)lpq_bw_q.tail())->length(); //reduce bits counted
        delete lpq_bw_q.pop(); //delete message
    }
}

//calc and record bw
if (lpq_bits > 0)
    bw_var = lpq_bits/(simTime()-(((cMessage
*)lpq_bw_q.tail())->timestamp()));
else
    bw_var = 0;

lpbw_v.record(bw_var);

```

```

done = false;

//remove messages older than the window
while (!hpq_bw_q.empty() && !done)
{
    if (((cMessage *)hpq_bw_q.tail())->timestamp()>=
old_time)

        done = true; //done purging messages
    else
    {
        hpq_bits = hpq_bits - ((cMessage
*)hpq_bw_q.tail())->length(); //reduce bits in queue
        delete hpq_bw_q.pop(); //delete msg
    }
}

//calc and record bw
if (hpq_bits > 0)
    bw_var = hpq_bits/(simTime()-(((cMessage
*)hpq_bw_q.tail())->timestamp()));
else
    bw_var = 0;

hpbw_v.record(bw_var);

return bw_var;//hpq bw usage

```

```
}
```

```
//determine random drop based on WRED
```

```
bool wredApp::drop(int min_thresh, int max_thresh, int mpd,  
double avg_q_len)
```

```
{
```

```
    bool drop_val;
```

```
    if (avg_q_len > min_thresh)//only drop if over  
min_thresh for queue length
```

```
    {
```

```
        double drop_prob;
```

```
        drop_prob = ((avg_q_len - min_thresh) / (max_thresh  
- min_thresh)) / mpd;    //probability that an individual  
message will be dropped
```

```
        if (drop_prob >= dblrand()) //randomly determine if  
we drop
```

```
            drop_val = true;
```

```
        else
```

```
            drop_val = false;
```

```
    }
```

```
    else
```

```
        drop_val = false;
```

```

        return (drop_val);
    }

# -----

# filename: node1_1.irt
# routing table for node 1
# author: James Knoll
# -----

ifconfig:

# ethernet card 0 to client 2
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.1
MTU: 1500 Metric: 1

ifconfigend.

route:

default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----

# filename: node1_2.irt

```

```

# routing table for node 2
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.2
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----
# filename: node1_3.irt
# routing table for node 3
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.3
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
      ppp0

routeend.

# -----
# filename: node1_4.irt
# routing table for node 4
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.4
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0  
              ppp0
```

```
routeend.
```

```
# -----  
# filename: node1_5.irt  
# routing table for node 5  
# author: James Knoll  
# -----
```

```
ifconfig:
```

```
# ethernet card 0  
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.5  
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
    ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_6.irt
```

```
# routing table for node 6
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0 to client 2
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.6
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
    ppp0
```

```
routeend.
```

```
# -----  
# filename: node1_7.irt  
# routing table for node 7  
# author: James Knoll  
# -----
```

```
ifconfig:
```

```
# ethernet card 0 to client 2  
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.7  
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0  
              ppp0
```

```
routeend.
```

```
# -----  
# filename: node1_8.irt  
# routing table for node 8
```

```

# author: James Knoll
# -----

ifconfig:

# ethernet card 0 to client 2
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.8
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----
# filename: node1_9.irt
# routing table for node 9
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0 to client 2
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.9
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----
# filename: node1_10.irt
# routing table for node 10
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.10
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
              ppp0
```

```
routeend.
```

```
# -----
# filename: node1_11.irt
# routing table for node 11
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.11
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----
# filename: node1_12.irt
# routing table for node 12
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.12
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

```

```
routeend.
```

```
# -----
```

```
# filename: node1_13.irt
```

```
# routing table for node 13
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.0.13
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0  
              ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_14.irt
```

```

# routing table for node 14
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.0.14
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0

routeend.

# -----
# filename: node1_15.irt
# routing table for node 15
# author: James Knoll
# -----

```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.15
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
               ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_16.irt
```

```
# routing table for node 16
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.16
```

MTU: 1500 Metric: 1

ifconfigend.

route:

default: 10.0.1.0 255.255.255.0 G 0
 ppp0

routeend.

filename: node1_17.irt

routing table for node 17

author: James Knoll

ifconfig:

ethernet card 0

name: ppp0 encap: Point-to-Point inet_addr: 10.0.0.17

MTU: 1500 Metric: 1

ifconfigend.

```
route:
default:      10.0.1.0      255.255.255.0      G      0
      ppp0
```

```
routeend.
```

```
# -----
# filename: node1_18.irt
# routing table for node 18
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.18
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
default:      10.0.1.0      255.255.255.0      G      0
      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_19.irt
```

```
# routing table for node 19
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.0.19
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0  
              ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_20.irt
# routing table for node 20
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.0.20
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
              ppp0
```

```
routeend.
```

```
# -----
# filename: node1_21.irt
# routing table for node 21
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.21
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0  
              ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node1_22.irt
```

```
# routing table for node 22
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.22
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
default:      10.0.1.0      255.255.255.0      G      0
              ppp0
```

```
routeend.
```

```
# -----
# filename: node1_23.irt
# routing table for node 23
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.0.23
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.1.0      255.255.255.0      G      0
              ppp0
```

```
routeend.
```

```
# -----
# filename: node1_24.irt
# routing table for node 24
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.24
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
default:      10.0.1.0      255.255.255.0      G      0
      ppp0
```

```
routeend.
```

```
# -----
# filename: node1_25.irt
# routing table for node 25
# author: James Knoll
# -----
```

```
ifconfig:
```

```
# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.0.25
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
default:      10.0.1.0      255.255.255.0      G      0
      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_1.irt
```

```
# routing table for node 1
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.3.1
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_2.irt
```

```
# routing table for node 2
```

```

# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.3.2
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_3.irt
# routing table for node 3
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.3
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_4.irt
# routing table for node 4
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.4
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_5.irt
```

```
# routing table for node 5
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.5
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_6.irt
```

```
# routing table for node 6
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.3.6
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_7.irt
```

```
# routing table for node 7
```

```

# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.3.7
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_8.irt
# routing table for node 8
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.8
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_9.irt
# routing table for node 9
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.9
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_10.irt
```

```
# routing table for node 10
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.10
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_11.irt
```

```
# routing table for node 11
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.3.11
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_12.irt
```

```
# routing table for node 12
```

```

# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.12
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_13.irt
# routing table for node 13
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.13
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_14.irt
# routing table for node 14
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.14
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_15.irt
```

```
# routing table for node 15
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.15
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_16.irt
```

```
# routing table for node 16
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.3.16
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_17.irt
```

```
# routing table for node 17
```

```

# author: James Knoll

# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point   inet_addr: 10.0.3.17
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----

# filename: node2_18.irt
# routing table for node 18
# author: James Knoll
# -----

ifconfig:

```

```

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.18
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_19.irt
# routing table for node 19
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.19
MTU: 1500 Metric: 1

```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_20.irt
```

```
# routing table for node 20
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.20
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_21.irt
```

```
# routing table for node 21
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point inet_addr: 10.0.3.21
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_22.irt
```

```

# routing table for node 22
# author: James Knoll
# -----

ifconfig:

# ethernet card 0
name: ppp0 encap: Point-to-Point    inet_addr: 10.0.3.22
MTU: 1500 Metric: 1

ifconfigend.

route:
default:      10.0.2.0      255.0.0.0      G    0    ppp0

routeend.

# -----
# filename: node2_23.irt
# routing table for node 23
# author: James Knoll
# -----

```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.23
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_24.irt
```

```
# routing table for node 24
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.24
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: node2_25.irt
```

```
# routing table for node 25
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# ethernet card 0
```

```
name: ppp0 encap: Point-to-Point  inet_addr: 10.0.3.25
```

```
MTU: 1500 Metric: 1
```

```
ifconfigend.
```

```
route:
```

```
default:      10.0.2.0      255.0.0.0      G      0      ppp0
```

```
routeend.
```

```
# -----
```

```
# filename: router1.irt
```

```
# routing table 1 for voip networks
```

```
# author: James Knoll
```

```
# -----
```

```
ifconfig:
```

```
# PPP link 0 to router1
```

```
name: ppp0  encap: Point-to-Point  inet_addr: 10.0.1.1
```

```
MTU: 1500 Metric: 1
```

```
# PPP link 1 to node 1
```

```
name: ppp1  encap: Point-to-Point  inet_addr: 10.0.1.2
```

```
MTU: 1500 Metric: 1
```

```
# PPP link 2 to node 2
```

```
name: ppp2  encap: Point-to-Point  inet_addr: 10.0.1.3
```

```
MTU: 1500 Metric: 1
```

```
# PPP link 3 to node 3
```

name: ppp3 encap: Point-to-Point inet_addr: 10.0.1.4
MTU: 1500 Metric: 1

PPP link 4 to node 4

name: ppp4 encap: Point-to-Point inet_addr: 10.0.1.5
MTU: 1500 Metric: 1

PPP link 5 to node 5

name: ppp5 encap: Point-to-Point inet_addr: 10.0.1.6
MTU: 1500 Metric: 1

PPP link 6 to node 6

name: ppp6 encap: Point-to-Point inet_addr: 10.0.1.7
MTU: 1500 Metric: 1

PPP link 7 to node 7

name: ppp7 encap: Point-to-Point inet_addr: 10.0.1.8
MTU: 1500 Metric: 1

PPP link 8 to node 8

name: ppp8 encap: Point-to-Point inet_addr: 10.0.1.9
MTU: 1500 Metric: 1

PPP link 9 to node 9

name: ppp9 encap: Point-to-Point inet_addr: 10.0.1.10

MTU: 1500 Metric: 1

PPP link 10 to node 10

name: ppp10 encap: Point-to-Point inet_addr: 10.0.1.11

MTU: 1500 Metric: 1

PPP link 11 to node 11

name: ppp11 encap: Point-to-Point inet_addr: 10.0.1.12

MTU: 1500 Metric: 1

PPP link 12 to node 12

name: ppp12 encap: Point-to-Point inet_addr: 10.0.1.13

MTU: 1500 Metric: 1

PPP link 13 to node 13

name: ppp13 encap: Point-to-Point inet_addr: 10.0.1.14

MTU: 1500 Metric: 1

PPP link 14 to node 14

name: ppp14 encap: Point-to-Point inet_addr: 10.0.1.15

MTU: 1500 Metric: 1

PPP link 15 to node 15

name: ppp15 encap: Point-to-Point inet_addr: 10.0.1.16

MTU: 1500 Metric: 1

PPP link 16 to node 16

name: ppp16 encap: Point-to-Point inet_addr: 10.0.1.17

MTU: 1500 Metric: 1

PPP link 17 to node 17

name: ppp17 encap: Point-to-Point inet_addr: 10.0.1.18

MTU: 1500 Metric: 1

PPP link 18 to node 18

name: ppp18 encap: Point-to-Point inet_addr: 10.0.1.19

MTU: 1500 Metric: 1

PPP link 19 to node 19

name: ppp19 encap: Point-to-Point inet_addr: 10.0.1.20

MTU: 1500 Metric: 1

PPP link 20 to node 20

name: ppp20 encap: Point-to-Point inet_addr: 10.0.1.21

MTU: 1500 Metric: 1

PPP link 21 to node 21

name: ppp21 encap: Point-to-Point inet_addr: 10.0.1.22

MTU: 1500 Metric: 1

PPP link 22 to node 22

name: ppp22 encap: Point-to-Point inet_addr: 10.0.1.23

MTU: 1500 Metric: 1

PPP link 23 to node 23

name: ppp23 encap: Point-to-Point inet_addr: 10.0.1.24

MTU: 1500 Metric: 1

PPP link 24 to node 24

name: ppp24 encap: Point-to-Point inet_addr: 10.0.1.25

MTU: 1500 Metric: 1

PPP link 25 to node 25

name: ppp25 encap: Point-to-Point inet_addr: 10.0.1.26

MTU: 1500 Metric: 1

PPP link 26 to node 26

name: ppp26 encap: Point-to-Point inet_addr: 10.0.1.27

MTU: 1500 Metric: 1

PPP link 27 to node 27

name: ppp27 encap: Point-to-Point inet_addr: 10.0.1.28

MTU: 1500 Metric: 1

PPP link 28 to node 28

name: ppp28 encap: Point-to-Point inet_addr: 10.0.1.29

MTU: 1500 Metric: 1

PPP link 29 to node 29

name: ppp29 encap: Point-to-Point inet_addr: 10.0.1.30

MTU: 1500 Metric: 1

ifconfigend.

route:

10.0.0.1	*	255.255.255.255	H	0	ppp1
10.0.0.2	*	255.255.255.255	H	0	ppp2
10.0.0.3	*	255.255.255.255	H	0	ppp3
10.0.0.4	*	255.255.255.255	H	0	ppp4
10.0.0.5	*	255.255.255.255	H	0	ppp5
10.0.0.6	*	255.255.255.255	H	0	ppp6
10.0.0.7	*	255.255.255.255	H	0	ppp7
10.0.0.8	*	255.255.255.255	H	0	ppp8
10.0.0.9	*	255.255.255.255	H	0	ppp9
10.0.0.10	*	255.255.255.255	H	0	ppp10
10.0.0.11	*	255.255.255.255	H	0	ppp11
10.0.0.12	*	255.255.255.255	H	0	ppp12
10.0.0.13	*	255.255.255.255	H	0	ppp13
10.0.0.14	*	255.255.255.255	H	0	ppp14

10.0.0.15	*	255.255.255.255	H	0	ppp15
10.0.0.16	*	255.255.255.255	H	0	ppp16
10.0.0.17	*	255.255.255.255	H	0	ppp17
10.0.0.18	*	255.255.255.255	H	0	ppp18
10.0.0.19	*	255.255.255.255	H	0	ppp19
10.0.0.20	*	255.255.255.255	H	0	ppp20
10.0.0.21	*	255.255.255.255	H	0	ppp21
10.0.0.22	*	255.255.255.255	H	0	ppp22
10.0.0.23	*	255.255.255.255	H	0	ppp23
10.0.0.24	*	255.255.255.255	H	0	ppp24
10.0.0.25	*	255.255.255.255	H	0	ppp25
10.0.0.26	*	255.255.255.255	H	0	ppp26
10.0.0.27	*	255.255.255.255	H	0	ppp27
10.0.0.28	*	255.255.255.255	H	0	ppp28
10.0.0.29	*	255.255.255.255	H	0	ppp29
default:	10.0.2.0	255.0.0.0	G	0	ppp0

routeend.

```
# -----
# filename: router2.irt
# routing table 2 for voip networks
# author: James Knoll
# -----
```

ifconfig:

PPP link 0 to router1

name: ppp0 encap: Point-to-Point inet_addr: 10.0.2.1

MTU: 1500 Metric: 1

PPP link 1 to node 1

name: ppp1 encap: Point-to-Point inet_addr: 10.0.2.2

MTU: 1500 Metric: 1

PPP link 2 to node 2

name: ppp2 encap: Point-to-Point inet_addr: 10.0.2.3

MTU: 1500 Metric: 1

PPP link 3 to node 3

name: ppp3 encap: Point-to-Point inet_addr: 10.0.2.4

MTU: 1500 Metric: 1

PPP link 4 to node 4

name: ppp4 encap: Point-to-Point inet_addr: 10.0.2.5

MTU: 1500 Metric: 1

PPP link 5 to node 5

name: ppp5 encap: Point-to-Point inet_addr: 10.0.2.6

MTU: 1500 Metric: 1

PPP link 6 to node 6

name: ppp6 encap: Point-to-Point inet_addr: 10.0.2.7

MTU: 1500 Metric: 1

PPP link 7 to node 7

name: ppp7 encap: Point-to-Point inet_addr: 10.0.2.8

MTU: 1500 Metric: 1

PPP link 8 to node 8

name: ppp8 encap: Point-to-Point inet_addr: 10.0.2.9

MTU: 1500 Metric: 1

PPP link 9 to node 9

name: ppp9 encap: Point-to-Point inet_addr: 10.0.2.10

MTU: 1500 Metric: 1

PPP link 10 to node 10

name: ppp10 encap: Point-to-Point inet_addr: 10.0.2.11

MTU: 1500 Metric: 1

PPP link 11 to node 11

name: ppp11 encap: Point-to-Point inet_addr: 10.0.2.12

MTU: 1500 Metric: 1

PPP link 12 to node 12

name: ppp12 encap: Point-to-Point inet_addr: 10.0.2.13

MTU: 1500 Metric: 1

PPP link 13 to node 13

name: ppp13 encap: Point-to-Point inet_addr: 10.0.2.14

MTU: 1500 Metric: 1

PPP link 14 to node 14

name: ppp14 encap: Point-to-Point inet_addr: 10.0.2.15

MTU: 1500 Metric: 1

PPP link 15 to node 15

name: ppp15 encap: Point-to-Point inet_addr: 10.0.2.16

MTU: 1500 Metric: 1

PPP link 16 to node 16

name: ppp16 encap: Point-to-Point inet_addr: 10.0.2.17

MTU: 1500 Metric: 1

PPP link 17 to node 17

name: ppp17 encap: Point-to-Point inet_addr: 10.0.2.18

MTU: 1500 Metric: 1

PPP link 18 to node 18

name: ppp18 encap: Point-to-Point inet_addr: 10.0.2.19
MTU: 1500 Metric: 1

PPP link 19 to node 19

name: ppp19 encap: Point-to-Point inet_addr: 10.0.2.20
MTU: 1500 Metric: 1

PPP link 20 to node 20

name: ppp20 encap: Point-to-Point inet_addr: 10.0.2.21
MTU: 1500 Metric: 1

PPP link 21 to node 21

name: ppp21 encap: Point-to-Point inet_addr: 10.0.2.22
MTU: 1500 Metric: 1

PPP link 22 to node 22

name: ppp22 encap: Point-to-Point inet_addr: 10.0.2.23
MTU: 1500 Metric: 1

PPP link 23 to node 23

name: ppp23 encap: Point-to-Point inet_addr: 10.0.2.24
MTU: 1500 Metric: 1

PPP link 24 to node 24

name: ppp24 encap: Point-to-Point inet_addr: 10.0.2.25

MTU: 1500 Metric: 1

PPP link 25 to node 25

name: ppp25 encap: Point-to-Point inet_addr: 10.0.2.26

MTU: 1500 Metric: 1

PPP link 26 to node 26

name: ppp26 encap: Point-to-Point inet_addr: 10.0.2.27

MTU: 1500 Metric: 1

PPP link 27 to node 27

name: ppp27 encap: Point-to-Point inet_addr: 10.0.2.28

MTU: 1500 Metric: 1

PPP link 28 to node 28

name: ppp28 encap: Point-to-Point inet_addr: 10.0.2.29

MTU: 1500 Metric: 1

PPP link 29 to node 29

name: ppp29 encap: Point-to-Point inet_addr: 10.0.2.30

MTU: 1500 Metric: 1

ifconfigend.

route:

10.0.3.1	*	255.255.255.255	H	0	ppp1
10.0.3.2	*	255.255.255.255	H	0	ppp2
10.0.3.3	*	255.255.255.255	H	0	ppp3
10.0.3.4	*	255.255.255.255	H	0	ppp4
10.0.3.5	*	255.255.255.255	H	0	ppp5
10.0.3.6	*	255.255.255.255	H	0	ppp6
10.0.3.7	*	255.255.255.255	H	0	ppp7
10.0.3.8	*	255.255.255.255	H	0	ppp8
10.0.3.9	*	255.255.255.255	H	0	ppp9
10.0.3.10	*	255.255.255.255	H	0	ppp10
10.0.3.11	*	255.255.255.255	H	0	ppp11
10.0.3.12	*	255.255.255.255	H	0	ppp12
10.0.3.13	*	255.255.255.255	H	0	ppp13
10.0.3.14	*	255.255.255.255	H	0	ppp14
10.0.3.15	*	255.255.255.255	H	0	ppp15
10.0.3.16	*	255.255.255.255	H	0	ppp16
10.0.3.17	*	255.255.255.255	H	0	ppp17
10.0.3.18	*	255.255.255.255	H	0	ppp18
10.0.3.19	*	255.255.255.255	H	0	ppp19
10.0.3.20	*	255.255.255.255	H	0	ppp20
10.0.3.21	*	255.255.255.255	H	0	ppp21
10.0.3.22	*	255.255.255.255	H	0	ppp22
10.0.3.23	*	255.255.255.255	H	0	ppp23
10.0.3.24	*	255.255.255.255	H	0	ppp24

```

10.0.3.25      *           255.255.255.255 H    0    ppp25
10.0.3.26      *           255.255.255.255 H    0    ppp26
10.0.3.27      *           255.255.255.255 H    0    ppp27
10.0.3.28      *           255.255.255.255 H    0    ppp28
10.0.3.29      *           255.255.255.255 H    0    ppp29
default:      10.0.1.0      255.0.0.0      G    0    ppp0

```

```

routeend.

```

C. NETWORKS

```

//-----
// file: codec.ned
// author: James Knoll
//
// Date: 31 May, 2004
//
// Simple voip configuration to test how codecs respond to
// varying the frame size. The network consists of two
// voip nodes connected with a router and wred box on each
// network. Each set of runs is conducted by varying the
// frame size with a fixed CODEC data rate.
//-----

import
    "voipUDPHost",

```

```

    "wredBox",
    "INE";

module codec
    parameters:
        satrate : numeric;
    submodules:
        voipclient11: voipUDPHost;
        parameters:
            local_addr = "10.0.0.1",
            dest_addr = "10.0.3.1",
            local_port = 100,
            dest_port = 200,
            // Voice parameters
            voice_length = input(30s, "Length of voice
transmission: "),
            initiate      =      input(false,      "Initiate
conversation? "),
            codec_rate    =    input(64000,    "CODEC    stream
rate: "),
            reply_delay   =   input(4s,   "Time   to   delay
before replying to a voice burst: "),
            frame_size    =    input(20ms,    "Length    of    a
frame: "),

            // network parameters
            numOfPorts = 1, //nodes connected to

```

```

        routingFile = "node1_1.irt";
    gatesizes:
        in[1],
        out[1];
    display: "p=45,100;i=pc";
router1: Router;
parameters:
    // network parameters
    numOfPorts = 2, //nodes connected to
    routingFile = "router1.irt";
gatesizes:
    in[2],
    out[2];
    display: "p=160,100;i=ipc";
wred1: wredBox;
parameters:
    win = 1s, //time span for bw calcs
    bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
    display: "p=210,100;i=bwxcon_s";
voipclient21: voipUDPHost;
parameters:
    // UDP parameters
    local_addr = "10.0.3.1",
    dest_addr = "10.0.0.1",

```

```

        local_port = 200,
        dest_port = 100,
        // Voice parameters
        voice_length = input(30s, "Length of voice
transmission: "),
        initiate      =      input(true,      "Initiate
conversation? "),
        codec_rate    =  input(64000, "CODEC  stream
rate: "),
        reply_delay   =  input(4s, "Time  to  delay
before replying to a voice burst: "),
        frame_size    =  input(20ms, "Length  of  a
frame: "),

        // network parameters
        numOfPorts = 1, //nodes connected to
        routingFile = "node2_1.irt";
    gatesizes:
        in[1],
        out[1];
    display: "p=455,100;i=comp";
router2: Router;
    parameters:
        // network parameters
        numOfPorts = 2, //nodes connected to
        routingFile = "router2.irt";
    gatesizes:

```

```

        in[2],
        out[2];
    display: "p=340,100;i=ipc";
wred2: wredBox;
    parameters:
        win = 1s,
        bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
        display: "p=290,100;i=bwxcon_s";

connections nocheck:
    voipclient11.out[0] --> router1.in[1];

    voipclient21.out[0] --> router2.in[1];

    router1.out[0] --> wred1.qIn;
    wred1.qOut --> datarate satrate --> wred2.passIn;
    wred2.passOut --> router2.in[0];

    router2.out[0] --> wred2.qIn;
    wred2.qOut --> datarate satrate --> wred1.passIn;
    wred1.passOut --> router1.in[0];

    router2.out[1] --> voipclient21.in[0];

```

```

        router1.out[1] --> voipclient11.in[0];

        display: "p=10,18;b=345,156";
endmodule

network directnw : codec
endnetwork

# -----
# filename: omnetpp.ini
# ini file for codec.ned
# author: James Knoll
# -----

[General]
preload-ned-files      =      *.ned      ../mynodes/*.ned
@c:/home/IPSuite/nedfiles.lst      ;ned      files      to      load
dynamically
network = directnw
total-stack-kb=7535
sim-time-limit = 10m      ;maximum simulation time to run
simulation
cpu-time-limit= 1h      ;maximum clock time to run simulation
random-seed = 1      ;seed for random numbers
snapshot-file = codec.sna ;file to output snapshots to

```

;output-vector-file = codec.vec ;file to output vectors

[Cmdenv]

runs-to-execute=1-18 ;runs to execute using cmd environment

express-mode = yes ;run in express mode

status-frequency=100000 ;frequency for status messages

[Tkenv]

default-run=1 ;run to execute for TK environment

[OutVectors]

/*.interval = 10s..;delay before starting to record data

#voip and traffic vectors

/*.delay_time.enabled = no

/*.receive_rate.enabled = no

/*.inst_rec_rate.enabled = no

/*.send_rate.enabled = no

/*.inst_send_rate.enabled = no

/*.jitter.enabled = no ;jitter in voip apps

#tcp client vectors

/*.Send No.enabled = no

/*.TCP delay.enabled = no

/*.Rec No.enabled = no

/*.Rec Seq No.enabled = no

/*.Cwnd size.enabled = no

```

*.Goodput.enabled = no
*.Avg_Goodput.enabled = no
*.Rec_Bits.enabled = no
#wred vectors
*.LP_BW.enabled = no
*.HP_BW.enabled = yes
*.HPQ_size.enabled = no
*.LPQ_size.enabled = no

[Parameters]
#connections
*.sat_datarate = 64000 ;data rate of satellite connection
*.sat_error = 0 ;satellite BER
*.sat_delay = 500ms ;delay in satellite link

#traffic
*.msg_length = 11200 ;length of a message in bits
*.traffic_rate = 64000 ;rate of transmission

# voip app configuration
*.voip_clients = 3 ;number of voip clients
*.voice_length = 30s ;length of a voice burst
*.voipclient11.initiate = true ;does this client initiate
the conversation
*.voipclient21.initiate = false

```

```

*.codec_rate = 5300 ;data rate for voip client
*.reply_delay = 4s ;delay before sending a reply
;*.frame_size = 140ms ;size of a frame
*.init_delay = 2s ;delay before first burst
*.talk_cycle = 50 ;percent off hook
*.call_length = 30m ;length of a call

#wredbox

*.bw_max = 48000 ;48 for 64k and 75 for 128k
*.hpq_min_thresh = 40 ;when to start random drop
*.hpq_max_thresh = 64 ;max drop
*.hpq_mpd = 10 ;percent to drop
*.lpq_min_thresh = 20 ;when to start random drop
*.lpq_max_thresh = 34 ;max drop
*.lpq_mpd = 10 ;percent to drop
*.max_q_len = 64 ;max queue depth
*.n = .01 ;weighting factor

# TCP

;*.clients_net1 = 2 ;number of tcp clients in network 1
*.clients_net2 = 0 ;number of tcp clients in network 2
*.mss=1400 ;maximum segment size
*.tcp.debug=true ;debug on
*.message_length = 64000000 ;length of message to transmit

```

```

# processing delays for all nodes
*.preRouting.procdelay = 0
*.routing.procdelay = 0.2 us
*.localDeliver.procdelay = 1 us
*.send.procdelay = 0.5 us
*.fragmentation.procdelay = 0.1 us
*.icmp.procdelay = 0
*.tunneling.procdelay = 0
*.multicast.procdelay = 0
*.output[*].procdelay = 0.2 us
*.inputQueue.procdelay = 0.1 us
*.networkInterface.procdelay = 0

# hook names
*.qosBehaviorClass = "EnqueueWithoutQoS" ;only hook
currently implemented

#configuration changes between runs

[Run 1]
output-vector-file = codec1.vec
*.frame_size = 10ms

[Run 2]
output-vector-file = codec2.vec
*.frame_size = 20ms

```

[Run 3]

output-vector-file = codec3.vec

*.frame_size = 30ms

[Run 4]

output-vector-file = codec4.vec

*.frame_size = 40ms

[Run 5]

output-vector-file = codec5.vec

*.frame_size = 50ms

[Run 6]

output-vector-file = codec6.vec

*.frame_size = 60ms

[Run 7]

output-vector-file = codec7.vec

*.frame_size = 80ms

[Run 8]

output-vector-file = codec8.vec

*.frame_size = 100ms

[Run 9]

output-vector-file = codec9.vec

*.frame_size = 120ms

[Run 10]

output-vector-file = codec10.vec

*.frame_size = 150ms

[Run 11]

output-vector-file = codec11.vec

*.frame_size = 200ms

[Run 12]

output-vector-file = codec12.vec

*.frame_size = 250ms

[Run 13]

output-vector-file = codec13.vec

*.frame_size = 300ms

[Run 14]

output-vector-file = codec14.vec

*.frame_size = 330ms

[Run 15]

```
output-vector-file = codec15.vec
```

```
*.frame_size = 350ms
```

```
[Run 16]
```

```
output-vector-file = codec16.vec
```

```
*.frame_size = 400ms
```

```
[Run 17]
```

```
output-vector-file = codec17.vec
```

```
*.frame_size = 450ms
```

```
[Run 18]
```

```
output-vector-file = codec18.vec
```

```
*.frame_size = 500ms
```

```
//-----
```

```
// file: codec_w_ine.ned
```

```
// author: James Knoll
```

```
//
```

```
// Date: 31 May, 2004
```

```
//
```

```
// Simple voip configuration with INEs to test how CODECs
```

```
// respond to varying the frame size. It is used with
```

```
// codec.ned to show the effect of the INE on the effective
```

```
// data rate. The network consists of two voip nodes
```

```

// connected with a router and wred box on each network.
// Each set of runs is conducted by varying the frame size
// with a fixed CODEC data rate.
//-----

import
    "voipUDPHost",
    "wredBox",
    "INE";

module codec_w_inet
    parameters:
        satrate : numeric;
    submodules:
        voipclient11: voipUDPHost;
        parameters:
            local_addr = "10.0.0.1",
            dest_addr = "10.0.3.1",
            local_port = 100,
            dest_port = 200,
            // Voice parameters
            voice_length = input(30s, "Length of voice
transmission: "),
            initiate = input(false, "Initiate
conversation? "),

```

```

        codec_rate = input(64000, "CODEC stream
rate: "),

        reply_delay = input(4s, "Time to delay
before replying to a voice burst: "),

        frame_size = input(20ms, "Length of a
frame: "),

        // network parameters

        numOfPorts = 1, //nodes connected to

        routingFile = "node1_1.irt";

    gatesizes:

        in[1],

        out[1];

    display: "p=45,100;i=pc";
ine11: INE;

    display: "p=100,100;i=ipc";
router1: Router;

    parameters:

        // network parameters

        numOfPorts = 2, //nodes connected to

        routingFile = "router1.irt";

    gatesizes:

        in[2],

        out[2];

    display: "p=160,100;i=ipc";
wred1: wredBox;

```

```

parameters:
    win = 1s, //window size for bw calcs
    bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
    display: "p=210,100;i=bwxcon_s";
voipclient21: voipUDPHost;
parameters:
    // UDP parameters
    local_addr = "10.0.3.1",
    dest_addr = "10.0.0.1",
    local_port = 200,
    dest_port = 100,
    // Voice parameters
    voice_length = input(30s, "Length of voice
transmission: "),
    initiate      =      input(true,      "Initiate
conversation? "),
    codec_rate    =      input(64000, "CODEC stream
rate: "),
    reply_delay   =      input(4s, "Time to delay
before replying to a voice burst: "),
    frame_size    =      input(20ms, "Length of a
frame: "),
    // network parameters
    numOfPorts = 1, //nodes connected to
    routingFile = "node2_1.irt";

```

```

    gatesizes:
        in[1],
        out[1];
    display: "p=455,100;i=comp";
ine21: INE;
    display: "p=400,100;i=ipc";
router2: Router;
    parameters:
        // network parameters
        numOfPorts = 2, //nodes connected to
        routingFile = "router2.irt";
    gatesizes:
        in[2],
        out[2];
    display: "p=340,100;i=ipc";
wred2: wredBox;
    parameters:
        win = 1s, //window to use for bw calcs
        bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
    display: "p=290,100;i=bwxcon_s";

connections nocheck:
    voipclient11.out[0] --> ine11.plainIn;
    ine11.cypherOut --> router1.in[1];

```

```
voipclient21.out[0] --> ine21.plainIn;
```

```
ine21.cypherOut --> router2.in[1];
```

```
router1.out[0] --> wred1.qIn;
```

```
wred1.qOut --> datarate satrate --> wred2.passIn;
```

```
wred2.passOut --> router2.in[0];
```

```
router2.out[0] --> wred2.qIn;
```

```
wred2.qOut --> datarate satrate --> wred1.passIn;
```

```
wred1.passOut --> router1.in[0];
```

```
router2.out[1] --> ine21.cypherIn;
```

```
ine21.plainOut --> voipclient21.in[0];
```

```
router1.out[1] --> ine11.cypherIn;
```

```
ine11.plainOut --> voipclient11.in[0];
```

```
display: "p=10,18;b=345,156";
```

```
endmodule
```

```
network directnw : codec_w_ine
```

```
endnetwork
```

```
# -----
```

```

# filename: omnetpp.ini
# ini file for codec_w_ine.ned
# author: James Knoll
# -----

[General]

preload-ned-files      =      *.ned      ../mynodes/*.ned
@c:/home/IPSuite/nedfiles.lst      ;ned      files      to      load
dynamically

network = directnw

total-stack-kb=7535

sim-time-limit = 10m      ;maximum simulation time to run
simulation

cpu-time-limit= 1h      ;maximum clock time to run simulation

random-seed = 1      ;seed for random numbers

snapshot-file = codec.sna ;file to output snapshots to

;output-vector-file = codec.vec ;file to output vectors


[Cmdenv]

runs-to-execute=1-50 ;runs to execute using cmd environment

express-mode = yes      ;run in express mode

status-frequency=100000 ;frequency for status messages


[Tkenv]

default-run=1      ;run to execute for TK environment

```

```

[OutVectors]
;*.interval = 10s ;delay before starting to record data
#voip and traffic vectors
*.delay_time.enabled = no
*.receive_rate.enabled = no
*.inst_rec_rate.enabled = no
*.send_rate.enabled = no
*.inst_send_rate.enabled = no
*.jitter.enabled = no ;jitter in voip apps
#tcp client vectors
*.Send No.enabled = no
*.TCP delay.enabled = no
*.Rec No.enabled = no
*.Rec Seq No.enabled = no
*.Cwnd size.enabled = no
*.Goodput.enabled = no
*.Avg_Goodput.enabled = no
*.Rec_Bits.enabled = no
#wred vectors
*.LP_BW.enabled = no
*.HP_BW.enabled = yes
*.HPQ_size.enabled = no
*.LPQ_size.enabled = no

```

[Parameters]

#connections

*.sat_datarate = 64000 ;data rate of satellite connection

*.sat_error = 0 ;satellite BER

*.sat_delay = 500ms ;delay in satellite link

#traffic

*.msg_length = 11200 ;length of a message in bits

*.traffic_rate = 64000 ;rate of transmission

voip app configuration

*.voip_clients = 3 ;number of voip clients

*.voice_length = 30s ;length of a voice burst

*.voipclient11.initiate = true ;does this client initiate the conversation

*.voipclient21.initiate = false

*.codec_rate = 5300 ;data rate for voip client

*.reply_delay = 4s ;delay before sending a reply

;* .frame_size = 140ms ;size of a frame

*.init_delay = 2s ;delay before first burst

*.talk_cycle = 50 ;percent off hook

*.call_length = 30m ;length of a call

#wredbox

*.bw_max = 48000 ;48 for 64k and 75 for 128k

```

*.hpq_min_thresh = 40      ;when to start random drop
*.hpq_max_thresh = 64      ;max drop
*.hpq_mpd = 10             ;percent to drop
*.lpq_min_thresh = 20      ;when to start random drop
*.lpq_max_thresh = 34      ;max drop
*.lpq_mpd = 10             ;percent to drop
*.max_q_len = 64           ;max queue depth
*.n = .01                  ;weighting factor

# TCP

;*.clients_net1 = 2 ;number of tcp clients in network 1
*.clients_net2 = 0 ;number of tcp clients in network 2
*.mss=1400               ;maximum segment size
*.tcp.debug=true         ;debug on
*.message_length = 64000000 ;length of message to transmit

# processing delays for all nodes
*.preRouting.procdelay = 0
*.routing.procdelay = 0.2 us
*.localDeliver.procdelay = 1 us
*.send.procdelay = 0.5 us
*.fragmentation.procdelay = 0.1 us
*.icmp.procdelay = 0
*.tunneling.procdelay = 0
*.multicast.procdelay = 0

```

```

*.output[*].procdelay = 0.2 us
*.inputQueue.procdelay = 0.1 us
*.networkInterface.procdelay = 0

# hook names

*.qosBehaviorClass = "EnqueueWithoutQoS" ;only hook
currently implemented within IPSuite

#configuration changes between runs

[Run 1]
output-vector-file = codec1.vec
*.frame_size = 10ms

[Run 2]
output-vector-file = codec2.vec
*.frame_size = 20ms

[Run 3]
output-vector-file = codec3.vec
*.frame_size = 30ms

[Run 4]
output-vector-file = codec4.vec
*.frame_size = 40ms

```

[Run 5]

output-vector-file = codec5.vec

*.frame_size = 50ms

[Run 6]

output-vector-file = codec6.vec

*.frame_size = 60ms

[Run 7]

output-vector-file = codec7.vec

*.frame_size = 70ms

[Run 8]

output-vector-file = codec8.vec

*.frame_size = 80ms

[Run 9]

output-vector-file = codec9.vec

*.frame_size = 90ms

[Run 10]

output-vector-file = codec10.vec

*.frame_size = 100ms

[Run 11]

```
output-vector-file = codec11.vec  
*.frame_size = 110ms
```

[Run 12]

```
output-vector-file = codec12.vec  
*.frame_size = 120ms
```

[Run 13]

```
output-vector-file = codec13.vec  
*.frame_size = 130ms
```

[Run 14]

```
output-vector-file = codec14.vec  
*.frame_size = 140ms
```

[Run 15]

```
output-vector-file = codec15.vec  
*.frame_size = 150ms
```

[Run 16]

```
output-vector-file = codec16.vec  
*.frame_size = 160ms
```

[Run 17]

```
output-vector-file = codec17.vec  
212
```

```
*.frame_size = 170ms
```

```
[Run 18]
```

```
output-vector-file = codec18.vec
```

```
*.frame_size = 180ms
```

```
[Run 19]
```

```
output-vector-file = codec19.vec
```

```
*.frame_size = 190ms
```

```
[Run 20]
```

```
output-vector-file = codec20.vec
```

```
*.frame_size = 200ms
```

```
[Run 21]
```

```
output-vector-file = codec21.vec
```

```
*.frame_size = 210ms
```

```
[Run 22]
```

```
output-vector-file = codec22.vec
```

```
*.frame_size = 220ms
```

```
[Run 23]
```

```
output-vector-file = codec23.vec
```

```
*.frame_size = 230ms
```

[Run 24]

output-vector-file = codec24.vec

*.frame_size = 240ms

[Run 25]

output-vector-file = codec25.vec

*.frame_size = 250ms

[Run 26]

output-vector-file = codec26.vec

*.frame_size = 260ms

[Run 27]

output-vector-file = codec27.vec

*.frame_size = 270ms

[Run 28]

output-vector-file = codec28.vec

*.frame_size = 280ms

[Run 29]

output-vector-file = codec29.vec

*.frame_size = 290ms

[Run 30]

output-vector-file = codec30.vec

*.frame_size = 300ms

[Run 31]

output-vector-file = codec31.vec

*.frame_size = 310ms

[Run 32]

output-vector-file = codec32.vec

*.frame_size = 320ms

[Run 33]

output-vector-file = codec33.vec

*.frame_size = 330ms

[Run 34]

output-vector-file = codec34.vec

*.frame_size = 340ms

[Run 35]

output-vector-file = codec35.vec

*.frame_size = 350ms

[Run 36]

```
output-vector-file = codec36.vec  
*.frame_size = 360ms
```

[Run 37]

```
output-vector-file = codec37.vec  
*.frame_size = 370ms
```

[Run 38]

```
output-vector-file = codec38.vec  
*.frame_size = 380ms
```

[Run 39]

```
output-vector-file = codec39.vec  
*.frame_size = 390ms
```

[Run 40]

```
output-vector-file = codec40.vec  
*.frame_size = 400ms
```

[Run 41]

```
output-vector-file = codec41.vec  
*.frame_size = 410ms
```

[Run 42]

```
output-vector-file = codec42.vec  
216
```

```
*.frame_size = 420ms
```

```
[Run 43]
```

```
output-vector-file = codec43.vec
```

```
*.frame_size = 430ms
```

```
[Run 44]
```

```
output-vector-file = codec44.vec
```

```
*.frame_size = 440ms
```

```
[Run 45]
```

```
output-vector-file = codec45.vec
```

```
*.frame_size = 450ms
```

```
[Run 46]
```

```
output-vector-file = codec46.vec
```

```
*.frame_size = 460ms
```

```
[Run 47]
```

```
output-vector-file = codec47.vec
```

```
*.frame_size = 470ms
```

```
[Run 48]
```

```
output-vector-file = codec48.vec
```

```
*.frame_size = 480ms
```

[Run 49]

```
output-vector-file = codec49.vec
```

```
*.frame_size = 490ms
```

[Run 50]

```
output-vector-file = codec50.vec
```

```
*.frame_size = 500ms
```

```
//-----
```

```
// file: slow.ned
```

```
// author: James Knoll
```

```
//
```

```
// Date: 31 May, 2004
```

```
//
```

```
// Simple voip configuration to test if tcp data traffic
```

```
// can be modeled using the UDP application developed. It
```

```
// consists of a variable number of clients with
```

```
// corresponding servers and a variable number of voip
```

```
// conversations. Various loading conditions are
```

```
// accomplished by changing the number of TCP and voip
```

```
// clients in each run. The current limit is 4 voip nodes
```

```
// and up to 25 total nodes per network but can easily be
```

```
// increased if needed.
```

```
//-----
```

```

import
    "Router",
    "TCPClientNode",
    "TCPServerNode",
    "voipUDPHost",
    "INE",
    "wredBox";

module slow
    parameters:
        clients_net1 : numeric const, //number of clients
on network 1
        clients_net2 : numeric const, //number of clients
on network 2
        voip_clients: numeric const,    //number of voip
pairs
        sat_datarate : numeric const, //data rate of
satellite
        sat_error : numeric const,    //BER for satellite
        sat_delay : numeric const;    //delay for satellite

    submodules:
        voipclient1: voipUDPHost [voip_clients];
        parameters:

```

```

        local_port = 100,
        dest_port = 200,
        // Voice parameters
        voice_length = input(30s, "Length of voice
transmission: "),
        initiate      =      input(false,      "Initiate
conversation? "),
        codec_rate    =  input(64000, "CODEC stream
rate: "),
        reply_delay   =  input(4s, "Time to delay
before replying to a voice burst: "),
        frame_size    =  input(20ms, "Length of a
frame: "),

        // network parameters
        numOfPorts = 1; //nodes to connect to
parameters if index==0:
        local_addr = "10.0.0.1",
        dest_addr = "10.0.3.1",
        routingFile = "node1_1.irt";
parameters if index==1:
        local_addr = "10.0.0.2",
        dest_addr = "10.0.3.2",
        routingFile = "node1_2.irt";
parameters if index==2:
        local_addr = "10.0.0.3",
        dest_addr = "10.0.3.3",

```

```

        routingFile = "node1_3.irt";
parameters if index==3:
    local_addr = "10.0.0.4",
    dest_addr = "10.0.3.4",
    routingFile = "node1_4.irt";
gatesizes:
    in[1],
    out[1];
display: "p=40,160,row;i=pc";
ine1: INE [voip_clients];
display: "p=40,200,row;i=ipc";
voipclient2: voipUDPHost [voip_clients];
parameters:
    local_port = 200,
    dest_port = 100,
    // Voice parameters
    voice_length = input(30s, "Length of voice
transmission: "),
    initiate      =      input(false,      "Initiate
conversation? "),
    codec_rate    =      input(64000,      "CODEC  stream
rate: "),
    reply_delay   =      input(4s,      "Time  to  delay
before replying to a voice burst: "),
    frame_size    =      input(20ms,      "Length  of  a
frame: "),

```

```

        // network parameters
        numOfPorts = 1; //nodes to connect to
parameters if index==0:
    local_addr = "10.0.3.1",
    dest_addr = "10.0.0.1",
    routingFile = "node2_1.irt";
parameters if index==1:
    local_addr = "10.0.3.2",
    dest_addr = "10.0.0.2",
    routingFile = "node2_2.irt";
parameters if index==2:
    local_addr = "10.0.3.3",
    dest_addr = "10.0.0.3",
    routingFile = "node2_3.irt";
parameters if index==3:
    local_addr = "10.0.3.4",
    dest_addr = "10.0.0.4",
    routingFile = "node2_4.irt";
gatesizes:
    in[1],
    out[1];
display: "p=40,340,row;i=pc";
ine2: INE [voip_clients];
display: "p=40,300,row;i=ipc";
tcpclient1: TCPClientNode[clients_net1];

```

```

parameters:

    // TCP parameters
    local_addr      =      (10      <<      24)      +
1+voip_clients+index,
    server_addr     =      (10      <<24)      +(3      <<8)+
1+voip_clients+index+clients_net2,

    // network parameters
    numOfPorts = 1; //nodes to connect to
parameters if index+voip_clients==0:
    routingFile = "node1_1.irt";
parameters if index+voip_clients==1:
    routingFile = "node1_2.irt";
parameters if index+voip_clients==2:
    routingFile = "node1_3.irt";
parameters if index+voip_clients==3:
    routingFile = "node1_4.irt";
parameters if index+voip_clients==4:
    routingFile = "node1_5.irt";
parameters if index+voip_clients==5:
    routingFile = "node1_6.irt";
parameters if index+voip_clients==6:
    routingFile = "node1_7.irt";
parameters if index+voip_clients==7:
    routingFile = "node1_8.irt";
parameters if index+voip_clients==8:

```

```
        routingFile = "node1_9.irt";
parameters if index+voip_clients==9:
        routingFile = "node1_10.irt";
parameters if index+voip_clients==10:
        routingFile = "node1_11.irt";
parameters if index+voip_clients==11:
        routingFile = "node1_12.irt";
parameters if index+voip_clients==12:
        routingFile = "node1_13.irt";
parameters if index+voip_clients==13:
        routingFile = "node1_14.irt";
parameters if index+voip_clients==14:
        routingFile = "node1_15.irt";
parameters if index+voip_clients==15:
        routingFile = "node1_16.irt";
parameters if index+voip_clients==16:
        routingFile = "node1_17.irt";
parameters if index+voip_clients==17:
        routingFile = "node1_18.irt";
parameters if index+voip_clients==18:
        routingFile = "node1_19.irt";
parameters if index+voip_clients==19:
        routingFile = "node1_20.irt";
parameters if index+voip_clients==20:
        routingFile = "node1_21.irt";
```

```

parameters if index+voip_clients==21:
    routingFile = "node1_22.irt";
parameters if index+voip_clients==22:
    routingFile = "node1_23.irt";
parameters if index+voip_clients==23:
    routingFile = "node1_24.irt";
parameters if index+voip_clients==24:
    routingFile = "node1_25.irt";
gatesizes:
    in[1],
    out[1];
display: "p=40,40,row;i=pc";
tcpclient2: TCPClientNode[clients_net2];
parameters:
    // TCP parameters
    local_addr = (10 << 24) +(3 <<8)+
1+voip_clients+index,
    server_addr = (10 <<24) +
1+voip_clients+index+clients_net1,
    // network parameters
    numOfPorts = 1;
parameters if index+voip_clients==0:
    routingFile = "node2_1.irt";
parameters if index+voip_clients==1:
    routingFile = "node2_2.irt";

```

```
parameters if index+voip_clients==2:
    routingFile = "node2_3.irt";
parameters if index+voip_clients==3:
    routingFile = "node2_4.irt";
parameters if index+voip_clients==4:
    routingFile = "node2_5.irt";
parameters if index+voip_clients==5:
    routingFile = "node2_6.irt";
parameters if index+voip_clients==6:
    routingFile = "node2_7.irt";
parameters if index+voip_clients==7:
    routingFile = "node2_8.irt";
parameters if index+voip_clients==8:
    routingFile = "node2_9.irt";
parameters if index+voip_clients==9:
    routingFile = "node2_10.irt";
parameters if index+voip_clients==10:
    routingFile = "node2_11.irt";
parameters if index+voip_clients==11:
    routingFile = "node2_12.irt";
parameters if index+voip_clients==12:
    routingFile = "node2_13.irt";
parameters if index+voip_clients==13:
    routingFile = "node2_14.irt";
parameters if index+voip_clients==14:
```

```

        routingFile = "node2_15.irt";
parameters if index+voip_clients==15:
        routingFile = "node2_16.irt";
parameters if index+voip_clients==16:
        routingFile = "node2_17.irt";
parameters if index+voip_clients==17:
        routingFile = "node2_18.irt";
parameters if index+voip_clients==18:
        routingFile = "node2_19.irt";
parameters if index+voip_clients==19:
        routingFile = "node2_20.irt";
parameters if index+voip_clients==20:
        routingFile = "node2_21.irt";
parameters if index+voip_clients==21:
        routingFile = "node2_22.irt";
parameters if index+voip_clients==22:
        routingFile = "node2_23.irt";
parameters if index+voip_clients==23:
        routingFile = "node2_24.irt";
parameters if index+voip_clients==24:
        routingFile = "node2_25.irt";
gatesizes:
        in[1],
        out[1];
display: "p=40,460,row;i=pc";

```

```

tcpserver1: TCPServerNode[clients_net1];

    parameters:
        parameters:
            // TCP parameters
            local_addr    =    (10    <<24)    +(3    <<8)+
1+voip_clients+index+clients_net2,
            // network parameters
            numOfPorts = 1;
        parameters if (index+clients_net2+voip_clients)
==0:
            routingFile = "node2_1.irt";
        parameters if (index+clients_net2+voip_clients)
==1:
            routingFile = "node2_2.irt";
        parameters if (index+clients_net2+voip_clients)
==2:
            routingFile = "node2_3.irt";
        parameters if (index+clients_net2+voip_clients)
==3:
            routingFile = "node2_4.irt";
        parameters if (index+clients_net2+voip_clients)
==4:
            routingFile = "node2_5.irt";
        parameters if (index+clients_net2+voip_clients)
==5:
            routingFile = "node2_6.irt";

```

```

parameters if (index+clients_net2+voip_clients)
==6:
    routingFile = "node2_7.irt";
parameters if (index+clients_net2+voip_clients)
==7:
    routingFile = "node2_8.irt";
parameters if (index+clients_net2+voip_clients)
==8:
    routingFile = "node2_9.irt";
parameters if (index+clients_net2+voip_clients)
==9:
    routingFile = "node2_10.irt";
parameters if (index+clients_net2+voip_clients)
==10:
    routingFile = "node2_11.irt";
parameters if (index+clients_net2+voip_clients)
==11:
    routingFile = "node2_12.irt";
parameters if (index+clients_net2+voip_clients)
==12:
    routingFile = "node2_13.irt";
parameters if (index+clients_net2+voip_clients)
==13:
    routingFile = "node2_14.irt";
parameters if (index+clients_net2+voip_clients)
==14:
    routingFile = "node2_15.irt";

```

```

parameters if (index+clients_net2+voip_clients)
==15:
    routingFile = "node2_16.irt";
parameters if (index+clients_net2+voip_clients)
==16:
    routingFile = "node2_17.irt";
parameters if (index+clients_net2+voip_clients)
==17:
    routingFile = "node2_18.irt";
parameters if (index+clients_net2+voip_clients)
==18:
    routingFile = "node2_19.irt";
parameters if (index+clients_net2+voip_clients)
==19:
    routingFile = "node2_20.irt";
parameters if (index+clients_net2+voip_clients)
==20:
    routingFile = "node2_21.irt";
parameters if (index+clients_net2+voip_clients)
==21:
    routingFile = "node2_22.irt";
parameters if (index+clients_net2+voip_clients)
==22:
    routingFile = "node2_23.irt";
parameters if (index+clients_net2+voip_clients)
==23:
    routingFile = "node2_24.irt";

```

```

parameters if (index+clients_net2+voip_clients)
==24:
    routingFile = "node2_25.irt";
gatesizes:
    in[1],
    out[1];
display: "p=40,400,row;i=comp";
tcpserver2: TCPServerNode[clients_net2];
parameters:
    parameters:
        // TCP parameters
        local_addr      =      (10      <<24)      +
1+voip_clients+index+clients_net1,
        // network parameters
        numOfPorts = 1;
parameters if (index+clients_net1+voip_clients)
==0:
    routingFile = "node1_1.irt";
parameters if (index+clients_net1+voip_clients)
==1:
    routingFile = "node1_2.irt";
parameters if (index+clients_net1+voip_clients)
==2:
    routingFile = "node1_3.irt";
parameters if (index+clients_net1+voip_clients)
==3:

```

```

        routingFile = "node1_4.irt";
parameters if (index+clients_net1+voip_clients)
==4:
        routingFile = "node1_5.irt";
parameters if (index+clients_net1+voip_clients)
==5:
        routingFile = "node1_6.irt";
parameters if (index+clients_net1+voip_clients)
==6:
        routingFile = "node1_7.irt";
parameters if (index+clients_net1+voip_clients)
==7:
        routingFile = "node1_8.irt";
parameters if (index+clients_net1+voip_clients)
==8:
        routingFile = "node1_9.irt";
parameters if (index+clients_net1+voip_clients)
==9:
        routingFile = "node1_10.irt";
parameters if (index+clients_net1+voip_clients)
==10:
        routingFile = "node1_11.irt";
parameters if (index+clients_net1+voip_clients)
==11:
        routingFile = "node1_12.irt";
parameters if (index+clients_net1+voip_clients)
==12:

```

```

        routingFile = "node1_13.irt";
parameters if (index+clients_net1+voip_clients)
==13:
        routingFile = "node1_14.irt";
parameters if (index+clients_net1+voip_clients)
==14:
        routingFile = "node1_15.irt";
parameters if (index+clients_net1+voip_clients)
==15:
        routingFile = "node1_16.irt";
parameters if (index+clients_net1+voip_clients)
==16:
        routingFile = "node1_17.irt";
parameters if (index+clients_net1+voip_clients)
==17:
        routingFile = "node1_18.irt";
parameters if (index+clients_net1+voip_clients)
==18:
        routingFile = "node1_19.irt";
parameters if (index+clients_net1+voip_clients)
==19:
        routingFile = "node1_20.irt";
parameters if (index+clients_net1+voip_clients)
==20:
        routingFile = "node1_21.irt";
parameters if (index+clients_net1+voip_clients)
==21:

```

```

        routingFile = "node1_22.irt";
parameters if (index+clients_net1+voip_clients)
==22:
        routingFile = "node1_23.irt";
parameters if (index+clients_net1+voip_clients)
==23:
        routingFile = "node1_24.irt";
parameters if (index+clients_net1+voip_clients)
==24:
        routingFile = "node1_25.irt";
gatesizes:
        in[1],
        out[1];
display: "p=40,100,row;i=comp";

router1: Router;
parameters:
        // network parameters
        numOfPorts =
1+voip_clients+clients_net1+clients_net2,
        routingFile = "router1.irt";
gatesizes:
in[1+voip_clients+clients_net1+clients_net2],
out[1+voip_clients+clients_net1+clients_net2];

```

```

        display: "p=140,220;i=ipc";
router2: Router;
parameters:
    // network parameters
    numOfPorts =
1+voip_clients+clients_net1+clients_net2,
    routingFile = "router2.irt";
gatesizes:

in[1+voip_clients+clients_net1+clients_net2],

out[1+voip_clients+clients_net1+clients_net2];
    display: "p=140,280;i=ipc";

wred1: wredBox;
parameters:
    win = 2s, //window size for bw calcs
    bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
    display: "b=16,15;p=100,250;i=bwxcon_s";
wred2: wredBox;
parameters:
    win = 2s, //window size for bw calcs
    bw_max = input(42000, "Max amount of bw to
allocate to high pri traffic: ");
    display: "b=16,15;p=180,250;i=bwxcon_s";

```

```

connections nocheck:

for i=1..voip_clients do //network 1
    voipclient1[i-1].out[0] --> ine1[i-1].plainIn;
    ine1[i-1].cypherOut --> router1.in[i];
    router1.out[i] --> ine1[i-1].cypherIn;
    ine1[i-1].plainOut --> voipclient1[i-1].in[0];
endfor;

for i=1..voip_clients do //network 2
    voipclient2[i-1].out[0] --> ine2[i-1].plainIn;
    ine2[i-1].cypherOut --> router2.in[i];
    router2.out[i] --> ine2[i-1].cypherIn;
    ine2[i-1].plainOut --> voipclient2[i-1].in[0];
endfor;

for i=1..clients_net1 do //network 1
    tcpclient1[i-1].out[0] -->
router1.in[i+voip_clients];
    tcpserver1[i-1].out[0] -->
router2.in[i+clients_net2+voip_clients];
    router1.out[i+voip_clients] --> tcpclient1[i-
1].in[0];
    router2.out[i+clients_net2+voip_clients] -->
tcpserver1[i-1].in[0];

```

```

endfor;

for i=1..clients_net2 do //network 2
    tcpclient2[i-1].out[0] -->
router2.in[i+voip_clients];
    tcpserver2[i-1].out[0] -->
router1.in[i+clients_net1+voip_clients];
    router2.out[i+voip_clients] --> tcpclient2[i-
1].in[0];
    router1.out[i+clients_net1+voip_clients] -->
tcpserver2[i-1].in[0];
endfor;

router1.out[0] --> wred1.qIn;
    wred1.qOut --> datarate sat_datarate error
sat_error delay sat_delay --> wred2.passIn;
    wred2.passOut --> router2.in[0];

router2.out[0] --> wred2.qIn;
    wred2.qOut --> datarate sat_datarate error
sat_error delay sat_delay --> wred1.passIn;
    wred1.passOut --> router1.in[0];

endmodule

network directnw : slow

```

endnetwork

filename: omnetpp.ini

ini file for slow.ned

author: James Knoll

[General]

preload-ned-files = *.ned ../mynodes/*.ned
@c:/home/IPSuite/nedfiles.lst ;ned files to load
dynamically

network = directnw

total-stack-kb=7535

sim-time-limit = 10m ;maximum simulation time to run
simulation

cpu-time-limit= 30m ;maximum clock time to run simulation

random-seed = 1 ;seed for random numbers

snapshot-file = tcpip.sna ;file to output snapshots to

;output-vector-file = tcpip.vec ;file to output vectors

[Cmdenv]

runs-to-execute=1-4 ;runs to execute using cmd environment

express-mode = yes ;run in express mode

status-frequency=100000 ;frequency for status messages

[Tkenv]

default-run=1 ;run to execute for TK environment

[OutVectors]

;*interval = 10s ;delay before starting to record data

#voip and traffic vectors

*.delay_time.enabled = no

*.receive_rate.enabled = no

*.inst_rec_rate.enabled = no

*.send_rate.enabled = no

*.inst_send_rate.enabled = no

*.jitter.enabled = no ;jitter in voip apps

#tcp client vectors

*.Send No.enabled = no

*.TCP delay.enabled = no

*.Rec No.enabled = no

*.Rec Seq No.enabled = no

*.Cwnd size.enabled = no

*.Goodput.enabled = no

*.Avg_Goodput.enabled = no

*.Rec_Bits.enabled = no

#wred vectors

```
*.LP_BW.enabled = no
*.HP_BW.enabled = yes
*.HPQ_size.enabled = no
*.LPQ_size.enabled = no
```

[Parameters]

#connections

```
*.sat_datarate = 64000 ;data rate of satellite connection
*.sat_error = 0 ;satellite BER
*.sat_delay = 500ms ;delay in satellite link
```

#traffic

```
*.msg_length = 11200 ;length of a message in bits
*.traffic_rate = 64000 ;rate of transmission
```

voip app configuration

```
*.voip_clients = 3 ;number of voip clients
*.voice_length = 30s ;length of a voice burst
;*.voipclient1[0].initiate = true ;does this client
initiate the conversation
*.voipclient2[0].initiate = false
;*.voipclient1[1].initiate = true
*.voipclient2[1].initiate = false
;*.voipclient1[2].initiate = true
*.voipclient2[2].initiate = false
```

```

*.voipclient1[3].initiate = false
*.voipclient2[3].initiate = false
*.voipclient1[0].codec_rate = 5300 ;data rate for voip
client
*.voipclient2[0].codec_rate = 5300
*.voipclient1[1].codec_rate = 5300
*.voipclient2[1].codec_rate = 5300
*.voipclient1[2].codec_rate = 16000
*.voipclient2[2].codec_rate = 16000
*.voipclient1[3].codec_rate = 16000
*.voipclient2[3].codec_rate = 16000
*.reply_delay = 4s ;delay before sending a reply
*.frame_size = 140ms ;size of a frame
*.init_delay = 4s ;delay before first burst
*.talk_cycle = 50 ;percent off hook
*.call_length = 30m ;length of a call

```

```

#wredbox

```

```

*.bw_max = 48000 ;48 for 64k and 75 for 128k
*.hpq_min_thresh = 40 ;when to start random drop
*.hpq_max_thresh = 64 ;max drop
*.hpq_mpd = 10 ;percent to drop
*.lpq_min_thresh = 20 ;when to start random drop
*.lpq_max_thresh = 34 ;max drop
*.lpq_mpd = 10 ;percent to drop

```

```

*.max_q_len = 64      ;max queue depth
*.n = .01             ;weighting factor

# TCP

;*.clients_net1 = 2 ;number of tcp clients in network 1
*.clients_net2 = 0    ;number of tcp clients in network 2
*.mss=1400            ;maximum segment size
*.tcp.debug=true      ;debug on
*.message_length = 64000000 ;length of message to transmit

# processing delays for all nodes
*.preRouting.procdelay = 0
*.routing.procdelay = 0.2 us
*.localDeliver.procdelay = 1 us
*.send.procdelay = 0.5 us
*.fragmentation.procdelay = 0.1 us
*.icmp.procdelay = 0
*.tunneling.procdelay = 0
*.multicast.procdelay = 0
*.output[*].procdelay = 0.2 us
*.inputQueue.procdelay = 0.1 us
*.networkInterface.procdelay = 0

# hook names

```

```
*.qosBehaviorClass    =    "EnqueueWithoutQoS"    ;only    hook  
currently implemented in IPSuite
```

```
#configuration changes between runs
```

```
[Run 1]
```

```
*.clients_net1 = 3  
*.voipclient1[0].initiate = false  
*.voipclient1[1].initiate = false  
*.voipclient1[2].initiate = true  
output-vector-file = tcpip1.vec
```

```
[Run 2]
```

```
*.clients_net1 = 18  
*.voipclient1[0].initiate = false  
*.voipclient1[1].initiate = false  
*.voipclient1[2].initiate = true  
output-vector-file = tcpip2.vec
```

```
[Run 3]
```

```
*.clients_net1 = 1  
*.voipclient1[0].initiate = true  
*.voipclient1[1].initiate = true  
*.voipclient1[2].initiate = true  
output-vector-file = tcpip3.vec
```

```

[Run 4]

*.clients_net1 = 3

*.voipclient1[0].initiate = false
*.voipclient1[1].initiate = false
*.voipclient1[2].initiate = false
output-vector-file = tcpip4.vec

//-----
// file: trades.ned
// author: James Knoll
//
// Date: 31 May, 2004
//
// A simple voip network to test the amount of data
// throughput with varying voip configurations. The UDP
// application provides network traffic that can be
// adjusted with the data rate. The number of voip nodes
// is currently limited to 12, but this can easily be
// expanded. Runs are configured to vary the call cycle
// for each configuration to examine how the configuration
// compares against the standard 32k of data.
//-----

import

    "Router",

```

```

    "TCPClientNode",
    "TCPServerNode",
    "voipUDPHost",
    "INE",
    "wredBox";

module trades
    parameters:
        voip_clients: numeric const, //number of voip pairs
        sat_datarate : numeric const, //satellite data rate
        sat_error : numeric const,    //satellite BER
        sat_delay : numeric const;    //satellite delay

    submodules:
        voipclient1: voipUDPHost [voip_clients];
        parameters:
            local_port = 100,
            dest_port = 200,
            // Voice parameters
            voice_length = input(30s, "Length of voice
transmission: "),
            initiate     = input(false, "Initiate
conversation? "),
            codec_rate   = input(64000, "CODEC stream
rate: "),

```

```

        reply_delay = input(4s, "Time to delay
before replying to a voice burst: "),
        frame_size = input(20ms, "Length of a
frame: "),

        // network parameters
        numOfPorts = 1;
parameters if index==0:
    local_addr = "10.0.0.2",
    dest_addr = "10.0.3.2",
    routingFile = "node1_2.irt";
parameters if index==1:
    local_addr = "10.0.0.3",
    dest_addr = "10.0.3.3",
    routingFile = "node1_3.irt";
parameters if index==2:
    local_addr = "10.0.0.4",
    dest_addr = "10.0.3.4",
    routingFile = "node1_4.irt";
parameters if index==3:
    local_addr = "10.0.0.5",
    dest_addr = "10.0.3.5",
    routingFile = "node1_5.irt";
parameters if index==4:
    local_addr = "10.0.0.6",
    dest_addr = "10.0.3.6",

```

```
        routingFile = "node1_6.irt";
parameters if index==5:
    local_addr = "10.0.0.7",
    dest_addr = "10.0.3.7",
    routingFile = "node1_7.irt";
parameters if index==6:
    local_addr = "10.0.0.8",
    dest_addr = "10.0.3.8",
    routingFile = "node1_8.irt";
parameters if index==7:
    local_addr = "10.0.0.9",
    dest_addr = "10.0.3.9",
    routingFile = "node1_9.irt";
parameters if index==8:
    local_addr = "10.0.0.10",
    dest_addr = "10.0.3.10",
    routingFile = "node1_10.irt";
parameters if index==9:
    local_addr = "10.0.0.11",
    dest_addr = "10.0.3.11",
    routingFile = "node1_11.irt";
parameters if index==10:
    local_addr = "10.0.0.12",
    dest_addr = "10.0.3.12",
    routingFile = "node1_12.irt";
```

```

        gatesizes:
            in[1],
            out[1];
        display: "p=40,160,row;i=pc";
ine1: INE [voip_clients];
        display: "p=40,200,row;i=ipc";
voipclient2: voipUDPHost [voip_clients];
        parameters:
            local_port = 200,
            dest_port = 100,
            // Voice parameters
            voice_length = input(30s, "Length of voice
transmission: "),
            initiate      =      input(false,      "Initiate
conversation? "),
            codec_rate    =    input(64000,    "CODEC    stream
rate: "),
            reply_delay   =   input(4s,   "Time   to   delay
before replying to a voice burst: "),
            frame_size    =    input(20ms,    "Length    of    a
frame: "),

            // network parameters
            numOfPorts = 1;
        parameters if index==0:
            local_addr = "10.0.3.2",
            dest_addr = "10.0.0.2",

```

```
        routingFile = "node2_2.irt";
parameters if index==1:
    local_addr = "10.0.3.3",
    dest_addr = "10.0.0.3",
    routingFile = "node2_3.irt";
parameters if index==2:
    local_addr = "10.0.3.4",
    dest_addr = "10.0.0.4",
    routingFile = "node2_4.irt";
parameters if index==3:
    local_addr = "10.0.3.5",
    dest_addr = "10.0.0.5",
    routingFile = "node2_5.irt";
parameters if index==4:
    local_addr = "10.0.3.6",
    dest_addr = "10.0.0.6",
    routingFile = "node2_6.irt";
parameters if index==5:
    local_addr = "10.0.3.7",
    dest_addr = "10.0.0.7",
    routingFile = "node2_7.irt";
parameters if index==6:
    local_addr = "10.0.3.8",
    dest_addr = "10.0.0.8",
    routingFile = "node2_8.irt";
```

```

parameters if index==7:
    local_addr = "10.0.3.9",
    dest_addr = "10.0.0.9",
    routingFile = "node2_9.irt";
parameters if index==8:
    local_addr = "10.0.3.10",
    dest_addr = "10.0.0.10",
    routingFile = "node2_10.irt";
parameters if index==9:
    local_addr = "10.0.3.11",
    dest_addr = "10.0.0.11",
    routingFile = "node2_11.irt";
parameters if index==10:
    local_addr = "10.0.3.12",
    dest_addr = "10.0.0.12",
    routingFile = "node2_12.irt";
gatesizes:
    in[1],
    out[1];
display: "p=40,340,row;i=pc";
ine2: INE [voip_clients];
display: "p=40,300,row;i=ipc";

trafficclient1: trafficUDPHost;
parameters:

```

```

        local_addr = "10.0.0.1",
        dest_addr = "10.0.3.1",
        local_port = 400,
        dest_port = 500,
        msg_length = input(12000, "Maximum payload
length(bits): "), //1500 bytes
        start_delay = false,
        traffic_rate = input(64000, "Traffic stream
rate: "),

        // network parameters
        numOfPorts = 1,
        routingFile = "node1_1.irt";
    gatesizes:
        in[1],
        out[1];

    display: "p=140,100,row;i=pc";
    trafficclient2: trafficUDPHost;
    parameters:
        // UDP parameters
        local_addr = "10.0.3.1",
        dest_addr = "10.0.0.1",
        local_port = 400,
        dest_port = 500,
        msg_length = input(1500, "Maximum payload
length: "),

```

```

        // traffic parameters
        start_delay = false,
        traffic_rate = input(64000, "traffic stream
rate: "),

        // network parameters
        numOfPorts = 1,
        routingFile = "node2_1.irt";
    gatesizes:
        in[1],
        out[1];
    display: "p=140,420,row;i=pc";

```

```

router1: Router;
    parameters:
        // network parameters
        numOfPorts = 2+voip_clients,
        routingFile = "router1.irt";
    gatesizes:
        in[2+voip_clients],
        out[2+voip_clients];
    display: "p=140,220;i=ipc";

```

```

router2: Router;
    parameters:
        // network parameters
        numOfPorts = 2+voip_clients,

```

```

        routingFile = "router2.irt";
    gatesizes:
        in[2+voip_clients],
        out[2+voip_clients];
    display: "p=140,280;i=ipc";

wred1: wredBox;
    parameters:
        win = 2s,
        bw_max = input(48000, "Max amount of bw to
allocate to high pri traffic: ");
        display: "b=16,15;p=100,250;i=bwxcon_s";
wred2: wredBox;
    parameters:
        win = 2s,
        bw_max = input(48000, "Max amount of bw to
allocate to high pri traffic: ");
        display: "b=16,15;p=180,250;i=bwxcon_s";

connections nocheck:

for i=1..voip_clients do //network 1
    voipclient1[i-1].out[0] --> ine1[i-1].plainIn;
    ine1[i-1].cypherOut --> router1.in[i+1];
    router1.out[i+1] --> ine1[i-1].cypherIn;

```

```

        ine1[i-1].plainOut --> voipclient1[i-1].in[0];
    endfor;

    for i=1..voip_clients do //network 2
        voipclient2[i-1].out[0] --> ine2[i-1].plainIn;
        ine2[i-1].cypherOut --> router2.in[i+1];
        router2.out[i+1] --> ine2[i-1].cypherIn;
        ine2[i-1].plainOut --> voipclient2[i-1].in[0];
    endfor;

    trafficclient1.out --> router1.in[1];
    trafficclient1.in <-- router1.out[1];

    trafficclient2.out --> router2.in[1];
    trafficclient2.in <-- router2.out[1];

    router1.out[0] --> wred1.qIn;
    wred1.qOut --> datarate sat_datarate error sat_error
    delay sat_delay --> wred2.passIn;
    wred2.passOut --> router2.in[0];

    router2.out[0] --> wred2.qIn;
    wred2.qOut --> datarate sat_datarate error sat_error
    delay sat_delay --> wred1.passIn;
    wred1.passOut --> router1.in[0];

```

```
endmodule
```

```
network directnw : trades
```

```
endnetwork
```

```
# -----
```

```
# filename: omnetpp.ini
```

```
# ini file for trades.ned
```

```
# author: James Knoll
```

```
# -----
```

```
[General]
```

```
preload-ned-files      =      *.ned      ../mynodes/*.ned  
@c:/home/IPSuite/nedfiles.lst      ;ned files to load  
dynamically
```

```
network = directnw
```

```
total-stack-kb=7535
```

```
sim-time-limit = 1h      ;maximum simulation time to run  
simulation
```

```
cpu-time-limit= 30m ;maximum clock time to run simulation
```

```
random-seed = 1          ;seed for random numbers
```

```
snapshot-file = trades.sna ;file to output snapshots to
```

```
;output-vector-file = trades.vec ;file to output vectors
```

[Cmdenv]

runs-to-execute=1-10 ;runs to execute using cmd environment

express-mode = yes ;run in express mode

status-frequency=100000 ;frequency for status messages

[Tkenv]

default-run=1 ;run to execute for TK environment

[OutVectors]

*.interval = 1000s ;delay before starting to record data

#voip

*.delay_time.enabled = no

.voipclient1[0]..receive_rate.enabled = no

.voipclient1[1]..receive_rate.enabled = no

.voipclient1[2]..receive_rate.enabled = no

.voipclient1[3]..receive_rate.enabled = no

.voipclient2[0]..receive_rate.enabled = no

.voipclient2[1]..receive_rate.enabled = no

.voipclient2[2]..receive_rate.enabled = no

.voipclient2[3]..receive_rate.enabled = no

.voipclient1[4]..receive_rate.enabled = no

.voipclient1[5]..receive_rate.enabled = no

.voipclient1[6]..receive_rate.enabled = no

.voipclient1[7]..receive_rate.enabled = no

```

*.voipclient2[4].*.receive_rate.enabled = no
*.voipclient2[5].*.receive_rate.enabled = no
*.voipclient2[6].*.receive_rate.enabled = no
*.voipclient2[7].*.receive_rate.enabled = no
*.trafficclient1. *.receive_rate.enabled = no
*.trafficclient2. *.receive_rate.enabled = yes
*.inst_rec_rate.enabled = no
*.send_rate.enabled = no
*.inst_send_rate.enabled = no
*.jitter.enabled = no                ;jitter in voip apps
#tcp
;*.Send No.enabled = no
;*.TCP delay.enabled = no
;*.Rec No.enabled = no
;*.Rec Seq No.enabled = no
;*.Cwnd size.enabled = no
;*.Goodput.enabled = no
;*.Avg_Goodput.enabled = no
#wred
*.LP_BW.enabled = no
*.HP_BW.enabled = no
*.HPQ_size.enabled = no
*.LPQ_size.enabled = no

```

[Parameters]

```

#connections

*.sat_datarate = 64000    ;data rate of satellite connection

*.sat_error = 0           ;satellite BER

*.sat_delay = 500ms ;delay in satellite link


#traffic

*.msg_length = 11200      ;length of a message in bits

*.traffic_rate = 64000    ;rate of transmission


# voip app configuration

*.voip_clients = 1        ;number of voip clients

*.voice_length = 3m        ;length of a voice burst

;*.voipclient1[0].voice_length = 30s    ;length when
silence suppression enabled

;*.voipclient1[1].voice_length = 3m

;*.voipclient1[2].voice_length = 3m

;*.voipclient1[3].voice_length = 3m

;*.voipclient2[0].voice_length = 30s

;*.voipclient2[1].voice_length = 3m

;*.voipclient2[2].voice_length = 3m

;*.voipclient2[3].voice_length = 3m

*.voipclient1[0].initiate = true    ;does this client
initiate the conversation

*.voipclient2[0].initiate = false

*.voipclient1[1].initiate = true

```

```

*.voipclient2[1].initiate = false
*.voipclient1[2].initiate = true
*.voipclient2[2].initiate = false
*.voipclient1[3].initiate = true
*.voipclient2[3].initiate = false
*.voipclient1[4].initiate = true
*.voipclient2[4].initiate = false
*.voipclient1[5].initiate = true
*.voipclient2[5].initiate = false
*.voipclient1[6].initiate = true
*.voipclient2[6].initiate = false
*.voipclient1[7].initiate = true
*.voipclient2[7].initiate = false
*.voipclient1[0].codec_rate = 16000    ;data rate for voip
client
*.voipclient2[0].codec_rate = 16000
*.voipclient1[1].codec_rate = 16000
*.voipclient2[1].codec_rate = 16000
*.voipclient1[2].codec_rate = 5300
*.voipclient2[2].codec_rate = 5300
*.voipclient1[3].codec_rate = 5300
*.voipclient2[3].codec_rate = 5300
*.voipclient1[4].codec_rate = 5300
*.voipclient2[4].codec_rate = 5300
*.voipclient1[5].codec_rate = 5300

```

```

*.voipclient2[5].codec_rate = 5300
*.voipclient1[6].codec_rate = 5300
*.voipclient2[6].codec_rate = 5300
*.voipclient1[7].codec_rate = 5300
*.voipclient2[7].codec_rate = 5300
*.reply_delay = 4s ;delay before sending a reply
*.frame_size = 140ms ;size of a frame
*.init_delay = 0s ;delay before first burst
;*.talk_cycle = 50 ;percent off hook
*.call_length = 30m ;length of a call

#wredbox

*.bw_max = 48000 ;48 for 64k and 75 for 128k
*.hpq_min_thresh = 40 ;when to start random drop
*.hpq_max_thresh = 64 ;max drop
*.hpq_mpd = 10 ;percent to drop
*.lpq_min_thresh = 20 ;when to start random drop
*.lpq_max_thresh = 34 ;max drop
*.lpq_mpd = 10 ;percent to drop
*.max_q_len = 64 ;max queue depth
*.n = .01 ;weighting factor

# TCP

;*.clients_net1 = 2 ;number of tcp clients in network 1
*.clients_net2 = 0 ;number of tcp clients in network 2

```

```

*.mss=1400          ;maximum segment size
*.tcp.debug=true    ;debug on
*.message_length = 64000000 ;length of message to transmit

# processing delays for all nodes
*.preRouting.procdelay = 0
*.routing.procdelay = 0.2 us
*.localDeliver.procdelay = 1 us
*.send.procdelay = 0.5 us
*.fragmentation.procdelay = 0.1 us
*.icmp.procdelay = 0
*.tunneling.procdelay = 0
*.multicast.procdelay = 0
*.output[*].procdelay = 0.2 us
*.inputQueue.procdelay = 0.1 us
*.networkInterface.procdelay = 0

# hook names
*.qosBehaviorClass = "EnqueueWithoutQoS" ;only hook
currently implemented in IPSuite

#configuration changes between runs

[Run 1]
*.talk_cycle = 100
output-vector-file = trades1.vec

```

[Run 2]

*.talk_cycle = 90

output-vector-file = trades2.vec

[Run 3]

*.talk_cycle = 80

output-vector-file = trades3.vec

[Run 4]

*.talk_cycle = 70

output-vector-file = trades4.vec

[Run 5]

*.talk_cycle = 60

output-vector-file = trades5.vec

[Run 6]

*.talk_cycle = 50

output-vector-file = trades6.vec

[Run 7]

*.talk_cycle = 40

output-vector-file = trades7.vec

[Run 8]

*.talk_cycle = 30

output-vector-file = trades8.vec

[Run 9]

*.talk_cycle = 20

output-vector-file = trades9.vec

[Run 10]

*.talk_cycle = 10

output-vector-file = trades10.vec

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

- Black, Uyless. *Voice over IP*. Upper Saddle River, New Jersey: Prentice Hall PTR, 2000.
- Barsaleau, Dean A. & Tummala, Murali. *QoS Testing of the ADNS Increment II Architecture*. Presentation at May, 2004 Navy Quality of Service (QoS) Working Group.
- Buddenburg, Rex. *Radio-WAN Building*. May 2003. Available from http://web1.nps.navy.mil/~budden/lecture.notes/r-wan/radio-WAN_building.html (Last accessed June 2004)
- Caputo, Robert. *Cisco Packetized Voice & Data Integration*. New York: McGraw-Hill, 2000.
- Casey, Rodger. *Black Routing Configuration For IPv4 And Transition Approach Rev 2*. February, 2004. Retrieved from [https://vpo.spawar.navy.mil/pd-17/pmw-179-2/adns/documents.nsf/titlelibrarydoc/black-core+transition/\\$file/Black+](https://vpo.spawar.navy.mil/pd-17/pmw-179-2/adns/documents.nsf/titlelibrarydoc/black-core+transition/$file/Black+) (Last accessed June 2004)
- Davidson, Jonathan & Peters, James. *Voice over IP Fundamentals*. Indianapolis, Indiana: Cisco Press, 2000.
- Deering, S. & Hinden, R. *RFC 2460: IPv6 Specification*. December 1998 Available from <http://www.ietf.org/rfc/rfc2460.txt> (Last accessed June 2004)
- Defense Information Systems Agency/Joint Interoperability Test Center (DISA/JITC) *APPENDIX 3 GENERIC SWITCHING CENTER REQUIREMENTS (GSCR) 08 SEP 03 DSN VOICE OVER INTERNET PROTOCOL (VOIP) REQUIREMENTS*. Retrieved from http://jitc.fhu.disa.mil/tssi/cert_pdfs/gscr_apdx3_dec03.pdf (Last accessed June 2004)

- Farley, Tom. *Tom Farley's Telephone History Series*. 1998-2004
Available from <http://www.privateline.com/TelephoneHistory/History1.htm> (Last accessed June 2004)
- Gomaa, Hassan. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. New York: Addison-Wesley, 2000.
- Hucke, Ed, Nguyen, Quang, Teng, Weden, Goodrich, Callis, Bart, Ron, Wadler, Andrew, Arendale, Ron, Et. al. *Analysis of Quintum Tenor Vocoding for Support for Secure Voice*. November 2003. Retrieved from [https://vpo.spawar.navy.mil/pd-17/pmw-179-2/adns/documents.nsf/titlelibrarydoc/voip+trunking+w/secure+voice/\\$file/Quinti](https://vpo.spawar.navy.mil/pd-17/pmw-179-2/adns/documents.nsf/titlelibrarydoc/voip+trunking+w/secure+voice/$file/Quinti) (Last accessed June 2004)
- Miller, Mark A. *Voice over IP Technologies: Building the Converged Network*. New York, NY: M&T Books, 2002.
- Schilke, Andreas. (1997, June). *TCP over Satellite Links*. Seminar ``Broadband Networking Technology'' <http://www.tkn.tu-berlin.de/curricula/ss97/bnt97/schilke.html> (Last accessed June 2004)

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Deborah Goldsmith
Navy QoS WG
San Diego, California
4. Jessie Rubalcava
Automated Digital Networking System (ADNS)
San Diego, California