UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DISEÑO Y DESARROLLO DE UN SISTEMA EXPERIMENTAL

CUANTIFICADOR DE TRÁFICO DE VOZ SOBRE IP (VOIP).

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar por el título de

INGENIERO EN INFORMÁTICA

REALIZADO POR Daniel Pons Alvarez

PROFESOR GUÍA Miguel Torrealba

FECHA Caracas, Noviembre de 2002

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DISEÑO Y DESARROLLO DE UN SISTEMA EXPERIMENTAL CUANTIFICADOR DE TRÁFICO DE VOZ SOBRE IP (VOIP).

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado:.....19 puntos.....

JURADO EXAMINADOR

Firma:	Firma:	Firma:
Nombre: Miguel Torrealba	Nombre: Wilmer Pereira	Nombre: Saray Hernández

REALIZADO POR	Daniel Pons Alvarez
PROFESOR GUÍA	Miguel Torrealba
FECHA	Caracas, Noviembre de 2002

Resumen General

Este Trabajo Especial de Grado consistió en la construcción de una herramienta de software de código libre en base a la investigación y selección de un protocolo de Voz sobre IP. Esta aplicación fue llamada *Programa Cuantificador de Voz* (PC-Voz). La misma permite el conteo de manera organizada del tráfico de paquetes de audio en una red con servicios de Voz sobre IP.

La metodología aplicada para el desarrollo de este sistema fue el *Modelo del Ciclo de Vida del Desarrollo de Software en Cascada*. Este esquema consiste en establecer un orden sistemático y secuencial en el avance de la solución, dividiendo el problema global en subproblemas. Esta característica permite controlar los tiempos y recursos dedicados a cada actividad.

Este programa fue desarrollado para ser compatible con el Sistema Operativo Linux. En consecuencia, fue estipulado que para la codificación, se utilizarían una serie de herramientas de software de carácter gratuito y de código libre. Las mismas brindan la posibilidad a cualquier persona, con conocimientos suficientes en la materia, modificar el contenido del programa para ajustarlo a sus necesidades.

Con la finalidad de visualizar las mediciones obtenidas y registradas por PC-Voz en una Base de Datos, se codificó un programa cliente. El mismo está en formato Web y debido a esto, puede ser utilizado en diversos tipos de plataformas de hardware/software que cumplan con un mínimo de requisitos.

El protocolo de VoIP seleccionado fue *H.323* de la *International Telecommunication Union – Telecommunication Standardization Sector* (ITU-T). Se concluyó que a pesar de que este protocolo es publicado por esta organización y que tiene un gran soporte comercial, el mismo es complejo y difuso. Se atribuyen estas características al conjunto de recomendaciones de carácter genérico que componen su estructura.

Índice General

Índice de Figuras

Índice de Tablas

CAPÍTULO I Introducción

I.1 Planteamiento del Problema

Sin duda alguna, cada vez existen más teléfonos y redes de Voz sobre IP (VoIP), esto se debe a que estas redes son más económicas que las de telefonía pública, por lo que la tendencia es a que las empresas migren de una tecnología a otra. Sin embargo, esta tecnología no está exenta de problemas: El principal obstáculo es el de garantizar una buena Calidad de Servicio (QoS) y el de cuantificar los beneficios de la implantación de una red que opera con VoIP.

En estos momentos existen varios protocolos para el transporte de voz sobre IP (SIP, H.323, Megaco, etc.), cada uno tiene sus ventajas y desventajas; sin embargo, es el estándar de la ITU-T, H.323 el que sobresale, por la calidad de servicio y la compatibilidad que provee. A pesar de esta aparente ventaja, nadie se ha preocupado por realizar un programa cuantificador de VoIP que sea gratuito a los investigadores y accesible sin restricciones.

La meta a satisfacer en este Trabajo Especial de Grado, es el desarrollo de un sistema experimental básico que permita cuantificar el tráfico generado por el conjunto de llamadas realizadas en una red por medio de VoIP, además de contar con las características suficientes para asegurar la confiabilidad del proceso, tal como es el registro de los paquetes de audio.

El desarrollo de este trabajo es de carácter experimental y pretende hacer un sistema de medición básico que sea parametrizable. Además, será de carácter gratuito, particularmente dirigido a universidades e instituciones de investigación, según permitan las reglas y normas de la UCAB. El mismo podrá ser distribuido por cualquier medio, preferiblemente por la Internet, y convertirse en punto de referencia para profundizar futuros estudios.

I.2 Objetivo General

Diseño y desarrollo de un sistema experimental básico de cuantificación de paquetes de audio sobre IP de tipo genérico y abierto.

I.3 Objetivos Específicos

1. Seleccionar y justificar un (1) protocolo de comunicación para la realización de la investigación.
2. Diseñar un método básico de cuantificación de tráfico de Voz sobre IP.
3. Diseñar y construir una Base de Datos que permita registrar la información obtenida en las mediciones.
4. Diseñar y desarrollar una interfaz básica experimental sobre un módulo de software para captura de paquetes.
5. Diseñar y construir una herramienta experimental básica de software para la cuantificación de paquetes de VoIP.
6. Diseñar y codificar un módulo experimental básico de software de Seguimiento de Actividad.
7. Diseñar y desarrollar un módulo experimental básico de interfaz de software que permita visualizar la información capturada.

I.4 Alcances y limitaciones

Los alcances del proyecto están acotados por los objetivos del mismo, la naturaleza experimental básica del mismo y el hecho de cumplir cabalmente, con las condiciones establecidas en los trabajos de grados de esta carrera. Sin embargo, se puede sintetizar las actividades a realizar de la siguiente forma:

1. Selección analítica de un (1) protocolo de Voz sobre IP.
2. Desarrollo de políticas que permitan cuantificar el tráfico que pasa por la tarjeta de red.
3. Diseño e implantación de un modelo de BD que permita registrar los resultados obtenidos.
4. Desarrollo de interfaces sobre aplicaciones que permitan capturar el tráfico que pasa por la tarjeta de red.
5. Diseño y desarrollo de un programa de usuario que permita visualizar los resultados obtenidos en las mediciones.
6. Codificación del módulo de seguimiento de actividad.

Las principales limitaciones, identificadas hasta esta etapa, se vinculan con recursos técnicos como son:

I.5 Recursos necesarios básicos

- Es necesario disponer de dos computadoras con tarjeta de red de tipo Ethernet y con conexión a la Internet.

- Un ambiente de programación Cliente-Servidor.
- Un sistema de pruebas.

I.6 Justificación

La idea de cuantificar el número de paquetes de datos electrónicos que pasan por un cable de red no es nueva. Es bien conocido que hasta existen funciones del propio sistema operativo que realizan esta tarea. Sin embargo, efectuarlo sobre el protocolo de Voz sobre IP, es algo que sólo ha sido realizado por empresas con fines lucrativos, ya que esta tecnología es relativamente nueva en el mercado. Su importancia principal radica entonces en la posibilidad de ofrecer un servicio y obtener una ganancia en base al uso contable de éste.

Un primer análisis sobre la complejidad de esta actividad revela la necesidad de conocer, entre otras cosas:

La conformación del protocolo de Voz sobre IP. Un conocimiento sólido de sus partes resulta crítico: arquitecturas de red sobre las que corre, seguridad vinculada, estándares, calidad de servicio ofrecida, etc.

Aspectos técnicos de seguridad asociada, para garantizar la continuidad operativa y tener una gran confiabilidad en la información generada por el sistema.

La aplicación de modelos matemáticos para el análisis de los datos, generando así información valiosa para los interesados. Es bien conocido que la ciencia de la computación se apoya sólidamente en los estudios matemáticos y disciplinas afines, a efectos de garantizar precisión y confianza de postulados.

El manejo de convenciones y esquemas comunes usados por los desarrolladores de software, que usan la Internet como su medio principal de distribución, a efectos de poder colocar el producto final como una herramienta abierta y portable.

Por todos los aspectos anteriores, se considera que el desarrollo de este Trabajo satisface los requisitos exigidos por la escuela de Ingeniería Informática para obtener el grado de Ingeniero en Informática en la Universidad Católica Andrés Bello y se presenta ante un jurado calificado, para que confirme esta proposición o emita la opinión académica que juzgue adecuada.

Este documento está organizado en seis secciones. La primera, una parte introductoria, en donde se establecen los objetivos, alcances y limitaciones de este Trabajo Especial de Grado. En la segunda, se detalla el marco teórico con información específica del protocolo de VoIP utilizado. El tercer capítulo, expone la metodología utilizada para el desarrollo de este trabajo y su justificación. El cuarto es el capítulo de desarrollo, donde se explica el progreso secuencial de la realización de este proyecto. La quinta sección presenta los resultados obtenidos, y en el sexto capítulo, se esquematizan las conclusiones y recomendaciones.

CAPÍTULO II Marco Teórico

II.1 Voz Sobre IP (VoIP)

Tradicionalmente, las redes de área local se vienen utilizando para la transmisión de datos, pero conforme las aplicaciones tienden a ser multimedia y los sistemas de comunicaciones se están convirtiendo en servidores de un conjunto más complejo de equipos interconectados, se tiende a transmitir cualquier tipo de información sobre los medios existentes. Así, sobre la LAN corporativa y sobre Internet, unos medios extendidos por la mayor parte de las empresas, mediante la adopción de ciertos estándares y la incorporación de algunos elementos, es posible enviar voz y vídeo, con la gran ventaja y ahorro que supone el utilizar la infraestructura existente. (Huidobro ¶ 1).

La voz y la imagen deben transmitirse en tiempo real, y son especialmente sensibles a problemas como la alteración del orden en que llegan los paquetes, o la pérdida de alguno de ellos. Los protocolos de VoIP requieren por lo tanto, redes que ofrezcan un alto grado de servicio y garanticen el ancho de banda necesario, lo que se consigue en aquellas que son orientadas a la conexión, es decir, que se negocia y establece al inicio de la comunicación la ruta que han de seguir todos y cada uno de los paquetes y se reserva un determinado ancho de banda. En las redes no orientadas a conexión se realiza el llamado "mejor esfuerzo" para entregar los paquetes, pero cada uno y en función del estado de los enlaces puede seguir una ruta distinta, por lo que el orden secuencial se puede ver alterado, lo que se traduce en una pérdida de calidad. (Huidobro ¶ 2)

Existen varios protocolos de VoIP; sin embargo, en este documento se trabaja con la recomendación de la ITU-T H.323 versión 2, ya que es la más soportada y fue la seleccionada para este Trabajo Especial de Grado. Para más información sobre las ventajas de VoIP consúltese el apéndice 1 y para detallar un estudio del ahorro de la telefonía sobre IP en las corporaciones, examínese el apéndice 2.

II.2 La Recomendación ITU-T H.323

La recomendación de la *International Telecommunication Union – Telecommunication Standardization Sector* (ITU-T) H.323 describe los terminales, equipos, servicios y otras entidades para la comunicación multimedia sobre *redes basadas en paquetes* (RBP) que no proveen una garantía en la calidad de servicio. Las entidades H.323 pueden suministrar audio en tiempo real, video y/o comunicación de datos. El soporte para el audio es obligatorio, mientras los datos y video son opcionales. (ITU-T. Recomendación H323 v2. Sección 1)

La red basada en paquetes sobre la cual las entidades H.323 se van a comunicar puede ser de conexión punto a punto, un segmento de red simple, o una red teniendo múltiples segmentos con una topología compleja. (ITU-T. Recomendación H323 v2. Sección 1).

Las entidades H.323 pueden ser integradas en computadoras personales o implementadas en dispositivos independientes como videoteléfonos. En la actualidad existen diversas aplicaciones que manejan dicho estándar, como el proyecto de código abierto multiplataforma *OpenPhone* (http://www.openh323.org) y *Microsoft® NetMeeting* (http://www.microsoft.com/windows/netmeeting/).

La ITU-T ha publicado 4 recomendaciones H.323 las cuales se resumen en la tabla 1:

Versión	Título	Fecha de Publicación
1	Visual Telephone Systems And Equipment For Local Area Networks Which Provide A Non-Guaranteed Quality Of Service.	Mayo 1996
2	Packet-Based Multimedia Communications Systems.	Octubre 1997
3	Packet-Based Multimedia Communications	Septiembre 1999

	Systems.	
4	Packet-Based Multimedia Communications Systems.	Noviembre 2000

Tabla 1: *Versiones de la Recomendación de la ITU-T H.323*

Resumen de http://www.packetizer.com/iptel/h323

II.2.1 La arquitectura de H.323

El alcance de esta recomendación se ve ilustrado en la figura 1. La arquitectura involucra *terminales H.323*, *gateways*, *gatekeepers* y *Unidades Controladoras Multipunto* (MCU). El objetivo principal de H.323 es de hacer posible el intercambio de flujos de contenido multimedia entre *puntos finales H.323*. Donde un punto final H.323 es un terminal, un gateway o un MCU. (Collins, 2001, p.113).

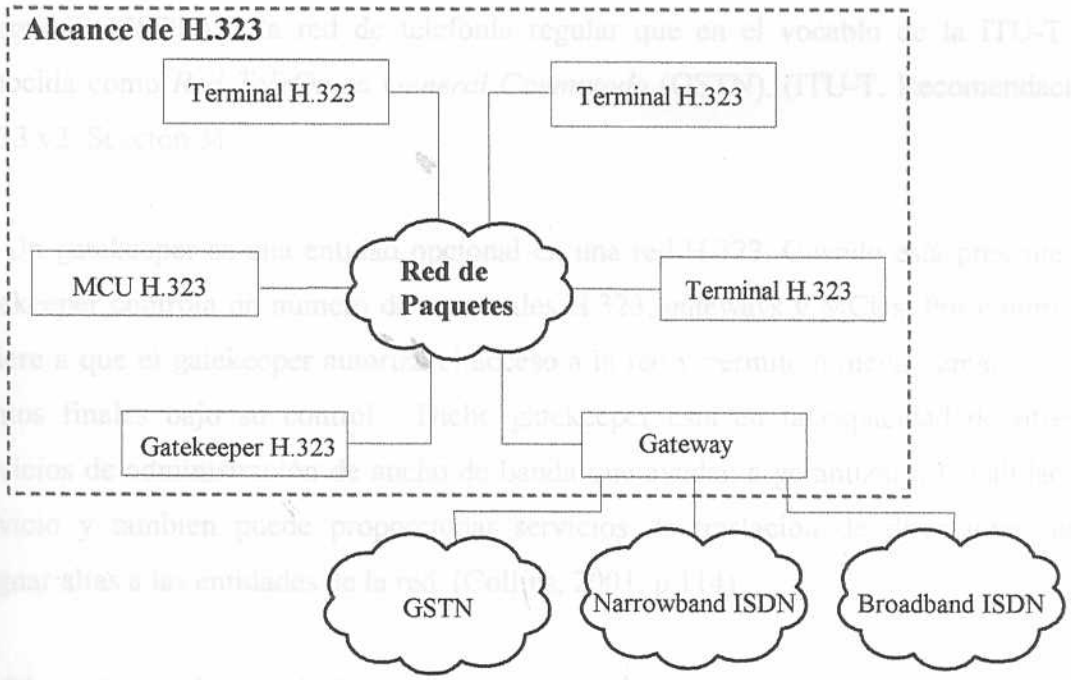

Figura 1: *Alcance de H.323*

Tomado de (Collins, 2001, p.113)

Un punto final es una entidad de hardware o software que puede mandar o recibir y puede generar o terminar flujos de información multimedia. Los puntos finales en H.323 son los terminales, los gateways y los MCUs. (ITU-T. Recomendación H323 v2. Sección 3).

Una entidad en H.323 es un gateway, un gatekeeper, un terminal o un MCU. (ITU-T. Recomendación H323 v2. Sección 3).

Un terminal H.323 es un punto final que ofrece comunicación en tiempo real con otros puntos finales H.323. Típicamente, este terminal es un dispositivo o una aplicación que soporta por lo menos un códec de audio y puede opcionalmente soportar otros códecs de audio y/o video. (ITU-T. Recomendación H323 v2. Sección 3).

Un gateway es un punto final H.323 que provee servicios de translación entre una red H.323 y otro tipo de red distinta, tal es el caso de una *Red Digital de Servicios Integrados* (ISDN) o la red de telefonía regular que en el vocablo de la ITU-T es conocida como *Red Telefónica General Conmutada* (GSTN). (ITU-T. Recomendación H323 v2. Sección 3).

Un gatekeeper es una entidad opcional en una red H.323. Cuando está presente, el gatekeeper controla un número de terminales H.323, gateways y MCUs. Por control se refiere a que el gatekeeper autoriza el acceso a la red y permite o niega llamadas a los puntos finales bajo su control. Dicho gatekeeper está en la capacidad de ofrecer servicios de administración de ancho de banda que ayudan a garantizar a la calidad de servicio y también puede proporcionar servicios de traslación de direcciones para asignar alias a las entidades de la red. (Collins, 2001, p.114).

El conjunto de terminales, gateways y MCUs que un gatekeeper controla se denomina zona. La figura 2 muestra un ejemplo de zona. Una zona puede ser dividida en

redes o subredes y las entidades de una zona no tienen necesariamente que pertenecer a la misma red. (Collins, 2001, p.114).

Una Unidad Controladora Multipunto es un punto final H.323 que maneja las conferencias entre tres o más terminales y/o gateways. Para dichas conferencias, la MCU determina qué tipo de contenido multimedia puede ser compartido entre las entidades y también está en la capacidad de cambiar el tipo de contenido. Una MCU puede ser implementada como una unidad independiente o estar incorporada en el mismo equipo que un gateway, gatekeeper o terminal H.323. (ITU-T. Recomendación H323 v2. Sección 3).

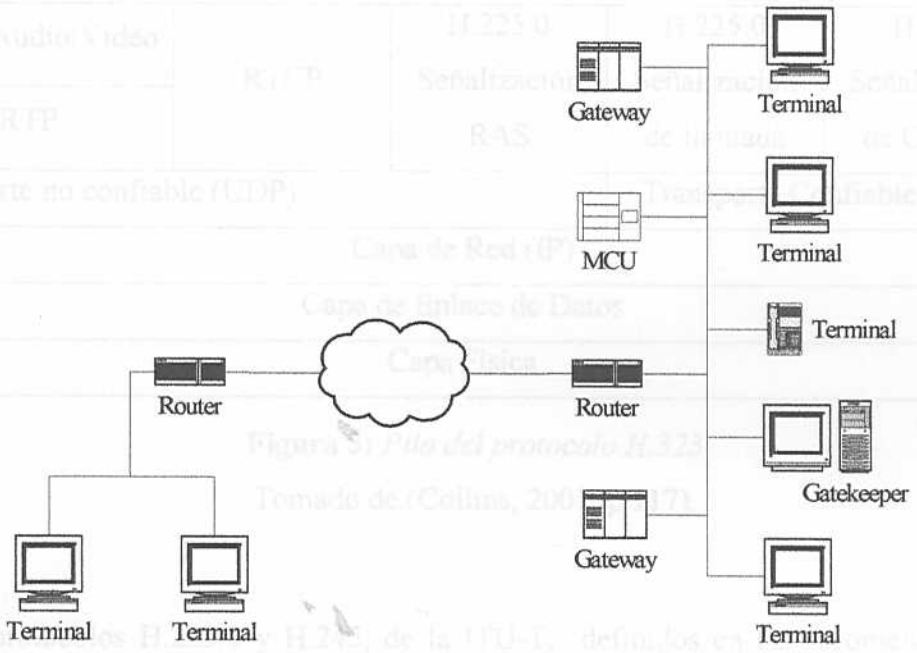

Figura 2: *La zona de un gatekeeper*

Tomado de (Collins, 2001, p.115).

II.2.2 Pila del protocolo H.323

La recomendación H.323 está compuesta por distintos protocolos: En una red IP se usa el *Protocolo de Internet* (IP) para encaminar la información desde un origen a un destino. Para la transmisión confiable de datos se usa el *Protocolo de Control de*

Transmisión (TCP) y para la transmisión no confiable de datos se usa el *Protocolo de Datagramas de Usuario* (UDP). (Collins, 2001, p.116).

El intercambio de contenido multimedia se hace por medio del *Protocolo de Transporte en tiempo Real* (RTP) que se transmite sobre UDP. RTP es controlado por el *Protocolo de Control de RTP* (RTCP). A continuación, en la figura 3, se puede ver gráficamente la pila de H.323. (Collins, 2001, p.116).

<table>
<tr><td>Aplicación de Audio/Video</td><td colspan="4">Terminal/Aplicación de Control</td></tr>
<tr><td>Códec Audio/Video</td><td rowspan="2">RTCP</td><td>H.225.0 Señalización RAS</td><td>H.225.0 Señalización de llamada</td><td>H.245 Señalización de Control</td></tr>
<tr><td>RTP</td><td colspan="3"></td></tr>
<tr><td colspan="3">Transporte no confiable (UDP)</td><td colspan="2">Transporte Confiable (TCP)</td></tr>
<tr><td colspan="5">Capa de Red (IP)</td></tr>
<tr><td colspan="5">Capa de Enlace de Datos</td></tr>
<tr><td colspan="5">Capa Física</td></tr>
</table>

Figura 3: *Pila del protocolo H.323*
Tomado de (Collins, 2001, p.117).

Los protocolos H.225.0 y H.245, de la ITU-T, definidos en las recomendaciones "*Call signalling protocols and media stream packetization for packet-based multimedia communication systems*" y "*Control protocol for multimedia communication*" respectivamente, son los que definen los mensajes que son intercambiados entre puntos finales H.323. Estos protocolos son genéricos y pueden ser utilizados en diversas arquitecturas de red; pero cuando están presentes en una red H.323, la manera en que H.225.0 y H.245 son aplicados está definido en la recomendación H.323. (Collins, 2001, p.116-117).

II.2.3 El protocolo H.225.0

Este protocolo, está compuesto por dos partes: la primera que es una variante de la recomendación de la ITU-T Q.931 de título "*Digital Subscriber Signalling System No. 1 (Dss 1) – Isdn User-Network Interface Layer 3 Specification For Basic Call Control*", una especificación de capa 3 para ISDN y que es utilizado para el establecimiento y la desconexión de llamadas entre puntos finales H.323. Esta parte es conocida como *Señalización de Llamada*. (Collins, 2001, p.117).

La otra parte de H.225.0 es conocida como *Señalización de Registro, Admisión y Estatus* (RAS) y es el medio por el cuál un punto final y un gatekeeper se comunican y es la manera en que un gatekeeper controla los elementos en su zona. (ITU-T. Recomendación H323 v2. Sección 6.2.9).

II.2.3.1 Señalización de Llamada

Como se mencionó con anterioridad, la Señalización de Llamada es utilizada para el establecimiento y la desconexión de llamadas entre puntos finales H.323. Los mensajes utilizados para esto son mensajes Q.931 con algunas modificaciones especificadas en la recomendación H.225.0. (Collins, 2001, p.134).

A continuación se presenta la tabla 2 que resume el conjunto de los mensajes de Señalización de Llamada.

Mensaje	Función	Comentario
Alerting	Enviado por el punto final llamado, indica que el usuario ya ha sido alertado de la existencia de una llamada.	Debe ser soportado.
Call Prodeeding	Un mensaje opcional enviado por el punto final llamado o por un gatekeeper	Debería ser enviado si un punto final llamado

	anterior a un mensaje de Connect.	usa un gatekeeper.
Connect	Es una indicación de que el usuario llamado a aceptado la llamada.	Debe ser soportado.
Progress	Mensaje opcional enviado por el punto final llamado anterior a un mensaje de Connect.	Puede ser usado por un gateway que es llamado en el caso de la existencia de PSTN.
Setup	Es un mensaje inicial utilizado para comenzar el establecimiento de llamada.	Debe ser soportado.
Setup Acknowledge	Un mensaje opcional en respuesta del mensaje de Setup.	Puede ser reenviado por un gateway en el caso de la existencia de PSTN.
Release Complete	Usado para finalizar una llamada.	Debe ser soportado.
User Information	Mensaje opcional usado para mandar información adicional sobre el establecimiento de llamada.	Puede ser usado en señalización superpuesta
Notify	Mensaje opcional que es utilizado para proveer información al usuario.	Puede ser usado en cualquier dirección.
Status	Mensaje respuesta de una petición de Status Inquiry o en respuesta de un mensaje desconocido.	Mensaje opcional, puede ser enviado en cualquier dirección
Status Inquiry	Utilizado para consultar el punto final remoto sobre el estado de la llamada según su perspectiva	Puede ser usado en conjunto con procedimientos de estado RAS.
Facility	Usado para redireccionar una llamada o invocar servicios suplementarios.	Puede ser usado en cualquier dirección.

Tabla 2: *Mensajes de Señalización de Llamada*

Tomado de (Collins, 2001, p.136).

II.2.3.2 Señalización de Registro, Admisión y Estado (RAS)

La señalización RAS es usada entre un gatekeeper y los puntos finales que él controla. Notoriamente un gatekeeper es una entidad opcional en una red H.323, por lo tanto RAS es también un protocolo opcional en estas redes; pero si un punto final quiere usar los servicios de un gatekeeper debe implementar RAS. (Collins, 2001, p.120-121).

Se presenta a continuación la tabla 3 describiendo las funciones de RAS:

Función	Descripción
Gatekeeper Discovery	Permite a un punto final determinar cuál gatekeeper está presente para controlarlo.
Registration	Permite a un punto final registrarse con un gatekeeper en particular y unirse así a la zona del gatekeeper.
Unregistration	Permite a un punto final salir del control del gatekeeper o permite a un gatekeeper cancelar el registro existente de un punto final, removiéndolo así de la zona.
Admission	Usado por un punto final para pedir acceso a una red con el propósito de participar en una llamada. Esta función especifica el ancho de banda solicitado a ser utilizado por el punto final, y el gatekeeper puede escoger entre aceptar o rechazar la petición basándose en el ancho de banda solicitado.
Bandwidth Change	Usado por un punto final para solicitar al gatekeeper que reserve más ancho de banda para él.
Endpoint Location	Utilizado para que el gatekeeper traslade un seudónimo en una dirección de red. Un punto final usa esta función cuando quiere comunicarse con otro punto final del cuál sólo conoce el alias.

Disengage	Efectuado por un punto final para informarle a un gatekeeper que se está desconectando de una llamada en particular. También puede ser usado para que el gatekeeper fuerce a un punto final a terminar una llamada.
Status	Función que se utiliza para pasar información sobre una llamada del punto final al gatekeeper, como por ejemplo el ancho de banda utilizado.
Resource Availability	Empleado para informar al gatekeeper sobre la capacidad actual del gateway de la red y del estado de sus recursos.
Non-Standard	Mecanismo para pasar información propietaria del fabricante, red, etc., entre un punto final y un gatekeeper.

Tabla 3: *Funciones de RAS*

Resumen de (Collins, 2001, p.121-122).

Cada una de las funciones presentadas, está compuesta por uno o varios mensajes RAS, a continuación, en la tabla 4, se presentan los mensajes RAS que serán utilizados en los escenarios de llamada ilustrados en las próximas secciones:

Mensaje	Descripción
Admission Request (ARQ)	Enviado de un punto final a un gatekeeper para pedir permiso para participar en una llamada.
Admission Confirm (ACF)	Usado por el gatekeeer para informar a un punto final que puede participar en determinada llamada.
Disengage Request (DRQ)	Utilizado por un punto final o gatekeeper para pedir la desconexión de una llamada en transcurso de un punto final.
Disengage Confirm (DCF)	Una respuesta afirmativa a DRQ.
Admission Reject (ARJ)	Usado por un gatekeeper para negar a un punto final su participación en una llamada.

Tabla 4: *Mensajes RAS*

Resumen de (ITU-T. Recomendación H323 v2. Sección 7.2)

II.2.4 El Protocolo H.245

El protocolo de *Señalización de Control* H.245 es usado entre los participantes de una llamada para establecer y controlar flujos de contenido multimedia. Este es un mecanismo que tienen los terminales para negociar el formato en que se van a transmitir los datos y los requerimientos de ancho de banda. Para llamadas con audio y video, este protocolo se encarga de multiplexar los flujos de contenido multimedia. (ITU-T. Recomendación H323 v2. Sección 6.2.8).

II.2.4.1 El concepto de Canales Lógicos

H.245 maneja los flujos multimedia por medio de *canales lógicos*. Un canal lógico es un camino de transmisión de datos unidireccional entre dos puntos finales. Generalmente en un entorno IP, podemos ver un canal lógico como una dirección IP y un número de puerto que soporta un tipo particular de flujo multimedia (socket). Cada canal lógico tiene un número asociado que es especificado por la entidad que manda los datos. (Collins, 2001, p.149).

Cada canal lógico es unidireccional desde el emisor hasta el receptor. Por lo tanto en una conversación entre dos entidades, deben existir dos canales lógicos. Dicha separación permite que a un terminal pueda enviar y recibir voz y/o video en distintos formatos. (Collins, 2001, p.149).

Los mensajes H.245 son transmitidos en el c*anal de control H.245*. Cada punto final o gatekeeper debe establecer un canal de control H.245 por cada llamada en la que esté participando. Dicho canal es especial, ya que no se abre y cierra al igual que otros canales. El mismo, se considera permanentemente abierto mientras que un punto final esté involucrado en una llamada. (Collins, 2001, p.149).

II.2.4.2 Procedimientos H.245

Antes de que un canal lógico sea abierto, el punto final emisor de datos debe entender cuáles son las capacidades del punto final receptor. Tratar de enviar voz en un formato que la entidad receptora no puede manejar es inútil. Por lo tanto, H.245 proporciona un procedimiento llamado *Capabilities Exchange* que involucra un conjunto de mensajes que son intercambiados entre los puntos finales para informar a su contraparte sobre sus capacidades respectivas. (Collins, 2001, p.149).

A continuación, se presenta la tabla 5, explicando los procedimientos H.245:

Procedimiento	Descripción
Capabilities Exchange	Permite a un punto final compartir información sobre sus capacidades de recepción y transmisión.
Master-Slave Determination	Dentro de una conferencia, uno de los puntos finales debe ser el maestro, esto con la finalidad de controlar los recursos de la red.
Open Unidirectional Logical Channel	Utilizado para abrir un canal lógico unidireccional. En este procedimiento se especifica el formato de contenido multimedia a ser transmitido e información concerniente a los puertos utilizados por RTP para transmitir los datos de voz y/o video.
Open Bidirectional Logical Channel	Utilizado para abrir un canal lógico bidireccional. Se usa muy poco ya que se considera más eficiente abrir dos canales unidireccionales, uno para el emisor y otro para el receptor.
Close Logical Channel	Utilizado para cerrar un canal lógico; en general un canal lógico sólo puede ser cerrado por la entidad que lo abrió
End session	Después de que se cierra el canal lógico de transmisión, se cierra la sesión H.245.

Tabla 5: *Procedimientos H.245*

Resumen de (ITU-T. Recomendación H323 v2. Sección 6.2.8)

II.2.5 Escenarios de llamada

A continuación, se presentan los posibles escenarios de llamada que en una red H.323 pueden estar presentes, en las imágenes exhibidas, los flujos en rojo representan los mensajes de señalización de llamada H.225.0, los verdes son los mensajes de señalización de control H.245, los negros los mensajes RAS y los azules la transmisión de datos por medio de RTP que es controlado por RTCP.

II.2.5.1 Llamada Básica sin Gatekeepers

Un gatekeeper es una entidad opcional en H.323. En la ausencia de gatekeeper, la señalización de llamada ocurre directamente entre puntos finales. (Collins, 2001, p.140).

La llamada es iniciada con un mensaje de Setup, respondido con el mensaje Connect, se establecen los parámetros de la transmisión por medio de H.245 y se realiza el intercambio de datos con RTP. Posteriormente se cierra la llamada con un mensaje de Release Complete que puede ser enviado por cualquiera de los dos puntos finales.

La figura 4 muestra el establecimiento y la desconexión de una llamada sin gatekeeper.

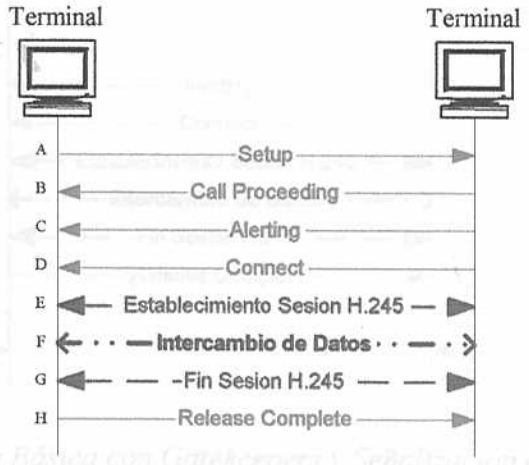

Figura 4: *Llamada Básica sin Gatekeepers*

Tomado de (Collins, 2001, p.141).

II.2.5.2 Llamada Básica con Gatekeepers y Señalización de Llamada Directa entre Puntos Finales

En el caso de que un punto final esté registrado en la zona de un gatekeeper, antes del establecimiento de la señalización de llamada, éste debe obtener permiso del gatekeeper para involucrarse en una llamada. Posteriormente transmite un mensaje de Setup el cual es respondido con un Call Proceeding en el caso de que el punto final remoto haya decidido contestar la llamada. (Collins, 2001, p.141).

Antes de establecer el canal de transmisión de RTP, el punto final que fue llamado debe solicitar el permiso de su gatekeeper para contestar la llamada y proceder así, a mandar dos mensajes, uno de Alerting y otro de Connect para dar paso al establecimiento de la señalización de control H.245.

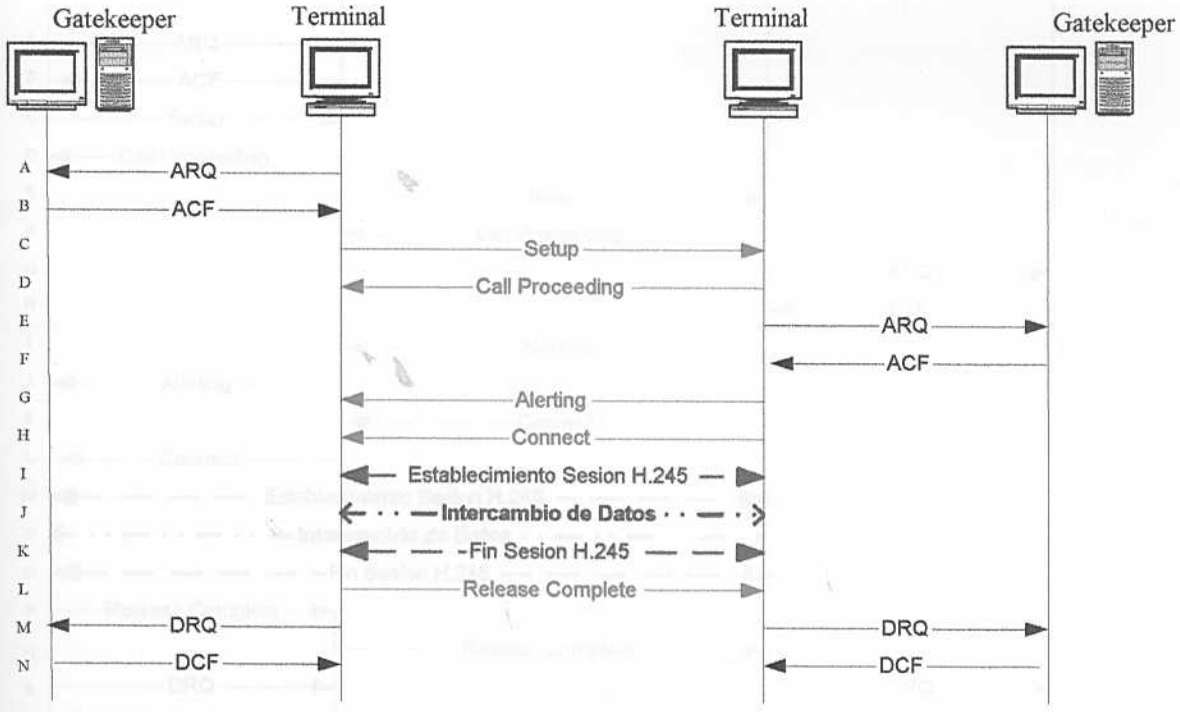

Figura 5: *Llamada Básica con Gatekeepers y Señalización de Llamada Directa entre Puntos Finales*

Tomado de (Collins, 2001, p.142).

Al finalizar la llamada, cada punto final debe notificar a su gatekeeper que se ha terminado la transmisión. Todo esto se ve reflejado en la figura 5, que es un escenario en donde hay diferentes gatekeepers controlando los distintos puntos finales.

II.2.5.3 Llamada Básica con Señalización Encaminada Directamente por el Gatekeeper

En este escenario, expuesto en la figura 6, los puntos finales están conectados a distintos gatekeepers, uno de los gatekeepers encamina la señalización de llamada y el otro deja que lo haga el punto final. (Collins, 2001, p.144).

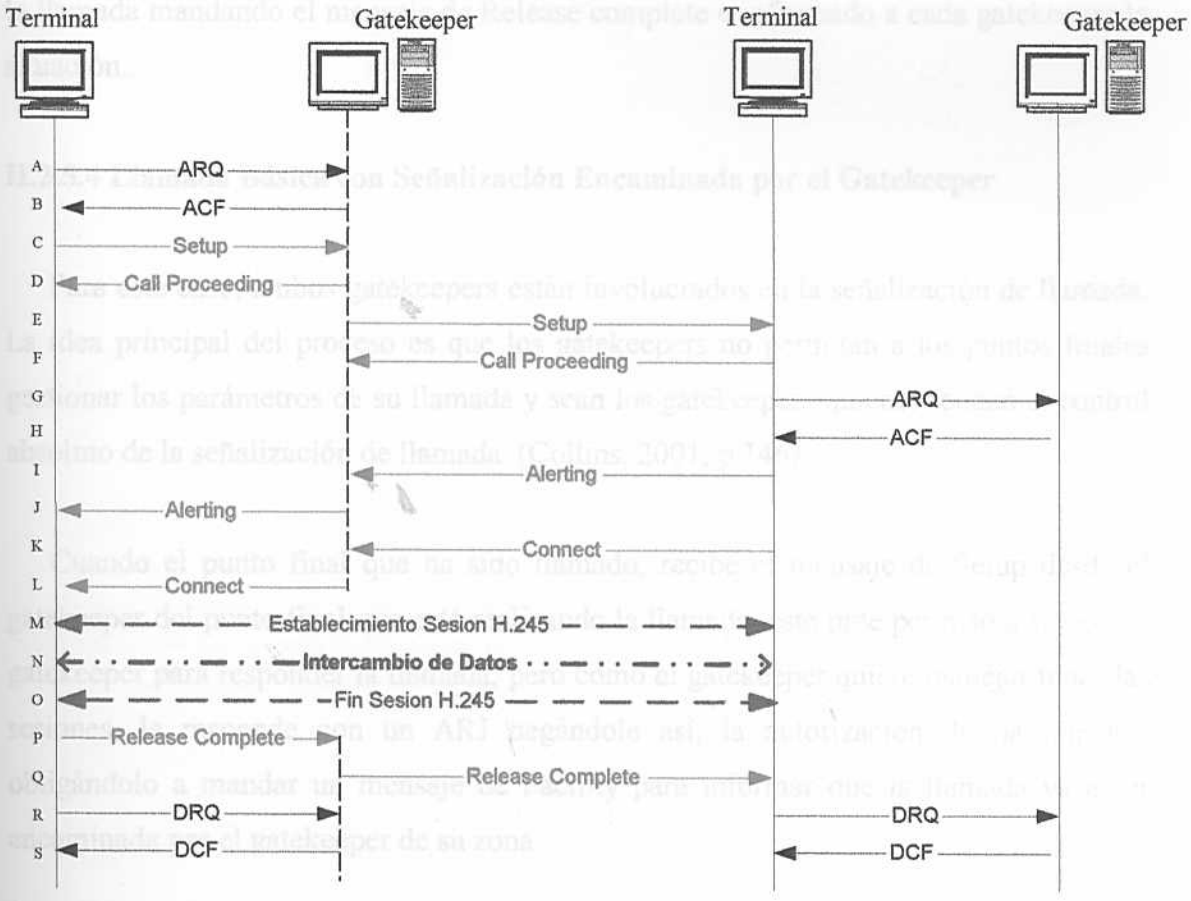

Figura 6: *Llamada Básica con Señalización Encaminada Directamente por el Gatekeeper*

Tomado de (Collins, 2001, p.113).

En el primer mensaje ACF, el gatekeeper le indica al punto final que la señalización de llamada va a ser enviada por medio del gatekeeper, por lo tanto, el punto final manda un mensaje de Setup al gatekeeper, que posteriormente es respondido con un Call Proceeding. Inmediatamente el gatekeeper manda el mensaje de Setup al punto final con el que se desea establecer la comunicación, entonces, este punto final responde con un Call Proceeding y pide autorización a su gatekeeper para participar en la llamada.

Después de recibir la autorización del gatekeeper, el punto final que había sido llamado envía al otro gatekeeper un mensaje de Alerting y otro de Connect que el gatekeeper retransmite al punto final que originó la llamada.

Posteriormente se establece el canal de transmisión de datos multimedia y se finaliza la llamada mandando el mensaje de Release complete e informado a cada gatekeeper la situación..

II.2.5.4 Llamada Básica con Señalización Encaminada por el Gatekeeper

Para este caso, ambos gatekeepers están involucrados en la señalización de llamada. La idea principal del proceso es que los gatekeepers no permitan a los puntos finales gestionar los parámetros de su llamada y sean los gatekeepers quienes tengan el control absoluto de la señalización de llamada. (Collins, 2001, p.146).

Cuando el punto final que ha sido llamado, recibe el mensaje de Setup desde el gatekeeper del punto final que está realizando la llamada, éste pide permiso a su propio gatekeeper para responder la llamada; pero como el gatekeeper quiere manejar todas las sesiones, le responde con un ARJ negándole así, la autorización de participar y obligándolo a mandar un mensaje de Facility para informar que la llamada va a ser encaminada por el gatekeeper de su zona.

El gatekeeper del lado que inició la llamada, corta la comunicación con un Release Complete y procede a realizar la llamada por medio el gatekeeper del otro punto final. Entonces, a partir de este instante toda la señalización de llamada queda en manos de los dos gatekeepers como se ve en los mensajes de Connect y Release Complete en la figura 7.

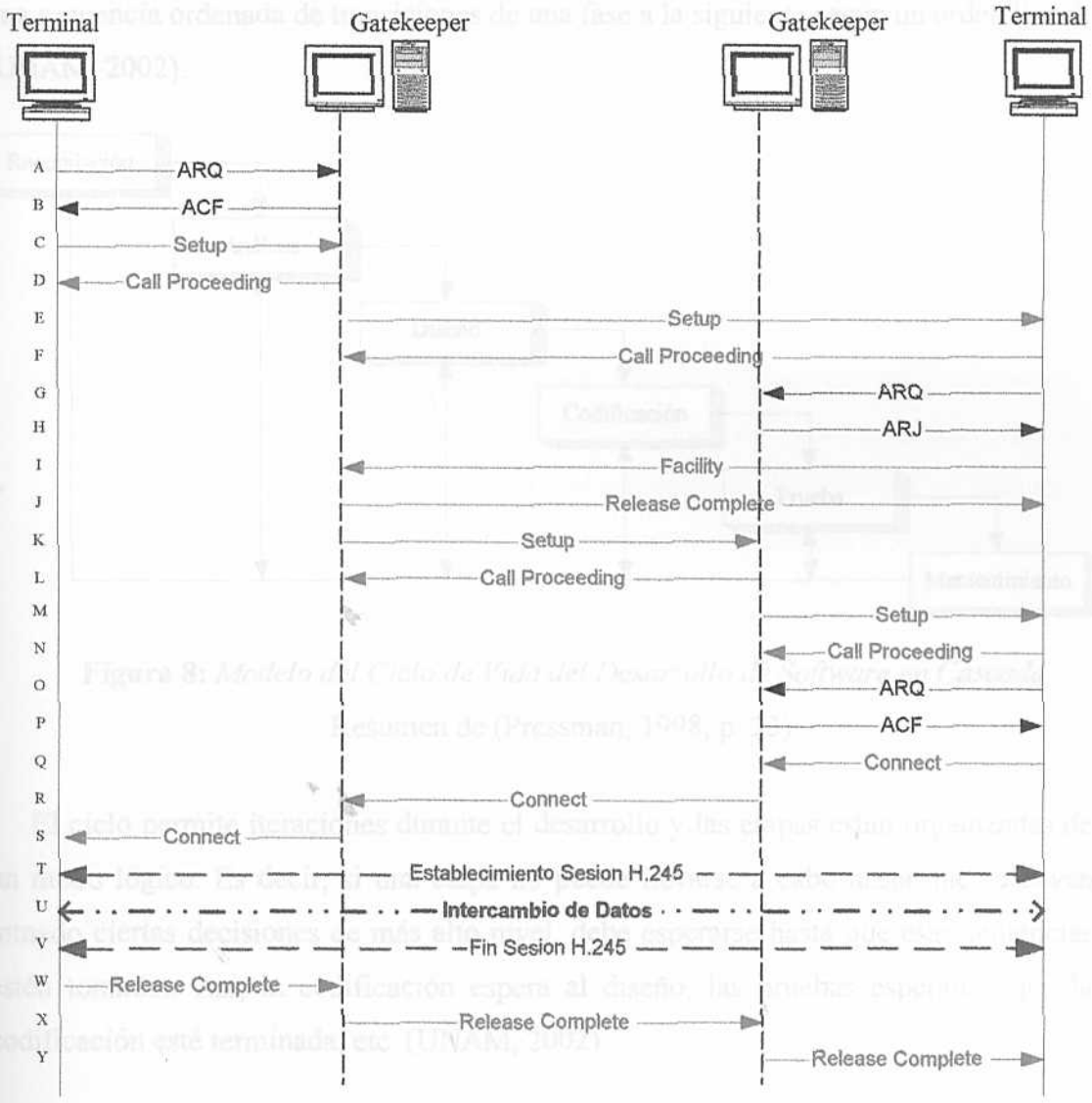

Figura 7: *Llamada Básica con Señalización Encaminada por el Gatekeeper*

Tomado de (Collins, 2001, p.147).

CAPÍTULO III Marco Metodológico

Para el desarrollo de este proyecto, se aplicó el *Modelo del Ciclo de Vida del Desarrollo de Software en Cascada*. Éste se representa tradicionalmente como un simple modelo con forma escalonada de las etapas del software, tal como se ilustra en la figura 8. En este esquema, la evolución del producto de software procede a través de una secuencia ordenada de transiciones de una fase a la siguiente según un orden lineal. (UNAM, 2002).

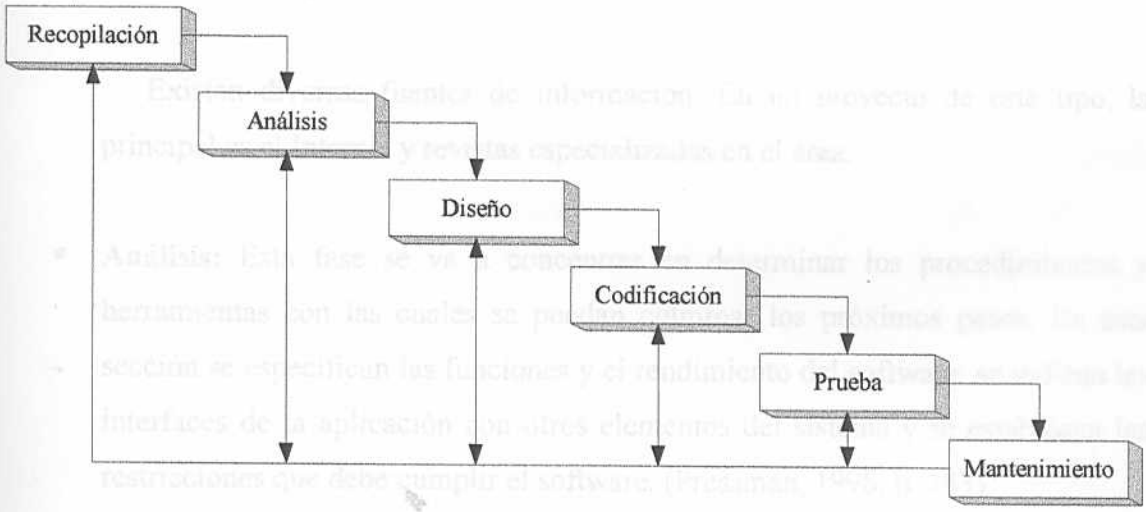

Figura 8: *Modelo del Ciclo de Vida del Desarrollo de Software en Cascada*
Resumen de (Pressman, 1998, p. 23).

El ciclo permite iteraciones durante el desarrollo y las etapas están organizadas de un modo lógico. Es decir, si una etapa no puede llevarse a cabo hasta que se hayan tomado ciertas decisiones de más alto nivel, debe esperarse hasta que esas sentencias estén tomadas. Así, la codificación espera al diseño, las pruebas esperan a que la codificación esté terminada, etc. (UNAM, 2002).

Cada fase incluye cierto proceso de revisión y se necesita una aceptación del producto antes de que la salida de la etapa pueda usarse. Este ciclo de vida está

organizado de modo que se pase el menor número de errores de una etapa a la siguiente. (UNAM, 2002).

A continuación se explican en detalle los pasos de este modelo:

- **Recopilación de Información:** Se va a obtener toda la información y conocimientos posibles para la realización de las partes fundamentales del Trabajo Especial de Grado. Unos ejemplos son: Protocolo de Voz sobre IP, programa de Seguimiento de Actividad, Bases de Datos, etc.

  Existen diversas fuentes de información. En un proyecto de este tipo, la principal es el Internet y revistas especializadas en el área.

- **Análisis:** Esta fase se va a concentrar en determinar los procedimientos y herramientas con las cuales se puedan culminar los próximos pasos. En esta sección se especifican las funciones y el rendimiento del software, se indican las interfaces de la aplicación con otros elementos del sistema y se establecen las restricciones que debe cumplir el software. (Pressman, 1998, p. 183).

- **Diseño:** Se transforman los requisitos a cubrir en representación esquemática de software. Según Taylor (1959, c.p. Pressman, 1998), el diseño es "el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema con suficiente detalle como para permitir su realización física".

  El diseño se sitúa en el núcleo técnico del proceso de ciclo de vida del software: diseño, codificación y prueba. El objetivo del diseño es producir un modelo o representación de una entidad que se va a construir posteriormente.. (Pressman, 1998, p. 229).

- **Codificación:** El diseño se traduce en forma legible para la máquina, se utilizaron varios lenguajes para cubrir las distintas partes del diseño. La codificación consiste en escribir un conjunto de condiciones, bucles, comandos, etc., en algún lenguaje legible al ser humano para que sea pasado a un lenguaje que pueda interpretar la computadora. (Pressman, 1998, p. 23).

- **Pruebas**: Se concentran en los procesos lógicos internos del software, asegurando que todas las sentencias se han ejecutado y en los procesos externos funcionales, es decir, este paso es fundamental para la detección de errores y verificar todas las salidas en el dominio de las entradas. (Pressman, 1998, p. 23).

La creciente inclusión del software como un elemento más de muchos sistemas y la importancia que tienen los costos asociados a un fallo del mismo, son las causas fundamentales que motivan la creación de pruebas minuciosas y bien planificadas. (Pressman, 1998, p. 301).

III.1 Justificación del Modelo del Ciclo de Vida del Desarrollo de Software en Cascada como metodología para la realización de este Trabajo Especial de Grado

Se utilizó este método gracias a que divide el problema en pasos, haciendo más fácil el desarrollo del proyecto y dándole un orden lógico al avance de la solución. Esta división es conocida como "divide y vencerás", la cuál estipula que es más eficiente fragmentar un problema en subproblemas para atacarlos de una manera más sencilla.

Al existir esta partición en subproblemas, es mucho más práctico controlar los tiempos dedicados a cada una de las subtareas y por lo tanto al problema en general.

La mayor ventaja de utilizar este método es que se puede regresar de una fase a alguna anterior para ejecutar el arreglo de algún error o para comenzar a desarrollar

alguna cosa nueva. Por ejemplo, una situación en particular consiste en devolverse de la fase de codificación a la de análisis para solucionar un problema de tipo algorítmico y posteriormente seguir el modelo en cascada hasta su fin o tener que regresarse de nuevo en algún punto ya que los resultados son imprevistos. Este caso se ilustra en la figura 9.

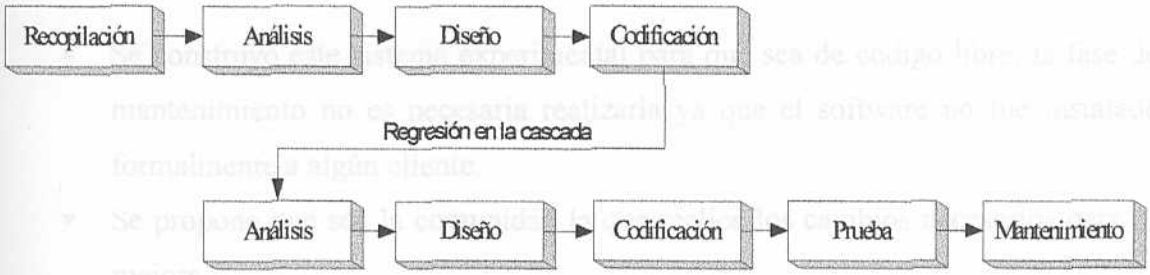

Figura 9: *Regresión en el Modelo en Cascada de la Codificación al Análisis*

Este modelo es un proceso interactivo por naturaleza. Se centra en desarrollar productos operacionales, esto hace que se puedan ver las características del software. Proporciona una plataforma de evaluación y lo más importante, aporta una manera de comprobar si los objetivos se han cumplido.

La crítica más fuerte a este modelo, es que es rígido. Se estipula que es difícil hacer modificaciones al producto gracias a su esquema secuencial. Dicho diseño fue seleccionado, a pesar de este defecto, porque proporciona un paradigma organizado que hace que no se pueda pasar de una fase a la siguiente si no se aprueban todas las variables de aceptación. Esta característica provee calidad a todos los resultados de cada etapa. Además, en un Trabajo Especial de Grado, exísten una serie de objetivos y alcances que deben ser seguidos al pie de la letra, por lo tanto, no pueden haber modificaciones del producto sustanciales. Se concluye entonces, que la hipótesis de que este modelo es rígido no aplicaría para este caso. Por algo es el modelo más utilizado en el desarrollo de software. (Pressman, 1998, p. 23-24).

CAPÍTULO IV Desarrollo

No se utilizaron todas las fases el Modelo del Ciclo de Vida del Desarrollo de Software en Cascada. Las causas de esta modificación fueron las siguientes:

- Se construyó este sistema experimental para que sea de código libre, la fase de mantenimiento no es necesaria realizarla ya que el software no fue instalado formalmente a algún cliente.
- Se propone que sea la comunidad la que realice los cambios necesarios para su mejora.

El carácter experimental del proyecto, permitió flexibilidad al momento de realizar pruebas de software. Sólo fueron aplicadas aquellas pruebas que garantizaran empíricamente un funcionamiento correcto de las aplicaciones desarrolladas. En la Fase de Prueba de este capítulo se entra en detalle sobre el tema.

En la tabla 6, se exponen las fases de desarrollo que cubrieron a los objetivos específicos del Trabajo Especial de Grado.

Nº	Objetivo	Fase
1	Seleccionar y justificar un (1) protocolo de comunicación para la realización de la investigación.	Recopilación Análisis
2	Diseñar un método básico de cuantificación de tráfico de Voz sobre IP.	Recopilación Análisis Diseño
3	Diseñar y construir una Base de Datos que permita registrar la información obtenida en las mediciones.	Recopilación Análisis Diseño Codificación Prueba

4	Diseñar y desarrollar una interfaz básica experimental sobre un módulo de software para captura de paquetes.	Recopilación Análisis Diseño Codificación Prueba
5	Diseñar y construir una herramienta experimental básica de software para la cuantificación de paquetes de VoIP.	Recopilación Análisis Diseño Codificación Prueba
6	Diseñar y codificar un módulo experimental básico de software de Seguimiento de Actividad.	Recopilación Análisis Diseño Codificación Prueba
7	Diseñar y desarrollar un módulo experimental básico de interfaz de software que permita visualizar la información capturada.	Recopilación Análisis Diseño Codificación Prueba

Tabla 6: *Fases del desarrollo que cubrieron los objetivos específicos de la tesis*

A continuación se explican en detalle las tareas llevadas a cabo en cada fase del método aplicado para la realización de este Trabajo Especial de Grado.

IV.1 Fase de Recopilación de Información

Como se especificó en el Marco Metodológico, en esta etapa inicial se realizaron una serie de investigaciones con la finalidad de obtener toda la información y conocimientos posibles para la realización de las partes fundamentales del proyecto.

El primer paso fue la realización de búsquedas en Internet de documentación sobre los distintos protocolos de VoIP existentes. Se consiguieron sitios Web con información sumamente valiosa, incluso se pudieron conocer las plataformas donde existe VoIP integrado como una solución de comunicación.

Posteriormente se realizaron investigaciones en textos de VoIP, redes y telecomunicaciones, en conjunto con la orientación del profesor guía, la cual fue fundamental para hacer la selección del protocolo de VoIP a trabajar.

Para cubrir necesidades de implementación fue necesario recurrir a libros y tutoriales específicos de lenguaje *C* y *Linux* (http://www.tldp.org), además del estudio de algunos segmentos de los *Request for Comments* o RFCs (http://www.ietf.org/rfc.html) y una gran cantidad de información proveniente de los manuales del manejador de la base de datos *MySQL* (http://www.mysql.com).

Posteriormente, la utilización de consultas en Internet a través de buscadores Web para encontrar documentos de VoIP fue disminuyendo y se restringió la información aportada por los sitios técnicos *Information Technologies Papers* (http://www.itpapers.com) y el lugar donde se encuentran todas las recomendaciones del protocolo que fue seleccionado como base para esta tesis, *Packetizer* (http://www.packetizer.com), entre otros.

Diversas lecturas provenientes de diferentes revistas de la *Association for Computing Machinery* (http://www.acm.org) ayudaron a esclarecer la selección del protocolo a trabajar y el método de cuantificación a utilizar. Un documento publicado en la *ACM Computer Communications Review* de título *mmdump: A Tool for Monitoring Internet Multimedia Traffic.* (Véase el anexo 1), fue fundamental para el entendimiento del protocolo H.323.

De gran importancia fue el artículo de título *Analysis of Voice Over IP Traffic*, (http://wand.cs.waikato.ac.nz/wand/publications/jamie_420/final) publicado por J.P. Curtis, J.G. Cleary, A.J. McGregor y M.W. Pearson del Departamento de Ciencias de la Computación de la Universidad de Waikato, Nueva Zelanda, en donde se explica cómo se realizaron mediciones de VoIP en una red H.323.

IV.2 Fase de Análisis

Una vez realizada una investigación generalizada de la arquitectura de VoIP, esta fase se concentró en determinar los procedimientos y herramientas con las cuales se culminaron las próximas etapas

El primer paso, consistió definir mejor los alcances del proyecto. Se estipularon aquellos que se veían directamente ligados a la implementación. Un ejemplo de esto es el caso del sistema operativo, lenguaje de programación, etc.

Posteriormente, se determinó en base a cuál protocolo se iba a realizar la Tesis, los factores utilizados para esto fueron:

- Madurez de implementación.
- Cantidad de documentación accesible.
- Nivel de soporte de los fabricantes.
- Expectativas de crecimiento a futuro.

Se plantearon distintos modelos de desarrollo de software y diversas opciones para realizar la codificación. Los parámetros que fueron evaluados al momento de tomar la decisión son los siguientes:

- Facilidad de implementación.
- Posibilidad técnica de codificación en base al modelo.
- Cantidad de documentación accesible.
- Eficiencia y legibilidad del código.

Después de la selección del sistema operativo y del protocolo a cuantificar, se determinaron algunas de las variables que acotaban los alcances y limitaciones del Trabajo Especial de Grado:

- Librería de captura de paquetes sobre la cuál hacer las capas de aplicaciones.
- Lenguajes de programación.

- Plataforma de hardware necesaria.
- Librerías y módulos necesarios.
- Software manejador de la Base de Datos.
- Aspectos técnicos de programación.
- Método de seguimiento de actividad.

Cabe destacar que fue necesario realizar varias veces regresiones de esta fase a la anterior. Al ir apareciendo nuevas variables se requirió información adicional que no se tenía a disposición. Un caso claro de esto, se presentó al tratar aspectos técnicos de programación, tales como la creación y manejo de procesos hijos.

Al final de esta etapa, se contaba con una precisa idea del procedimiento de codificación del software. Esto debido a que se contaba con una aproximación lógica del modelo estructural del programa. El procedimiento de desarrollo estaba ya definido, en parte, porque un conjunto de preguntas críticas ya estaban respondidas:

- ¿Cómo debe situarse el programa en la red para hacer las mediciones?.
- ¿Cómo se va a crear un ambiente de pruebas para el software?.
- ¿Cómo se va a mantener la integridad de las mediciones?.
- ¿Cómo garantizar la calidad y continuidad del servicio?.
- ¿Cómo se debe hacer la cuantificación del tráfico de VoIP?.

La respuesta a dichas incógnitas se encuentra en el capítulo de Resultados, donde se muestra con detalle el marco técnico que dio paso a la codificación del proyecto.

IV.3 Fase de Diseño

En este paso, se transforman los requisitos a cubrir en representación esquemática de software. Se utilizaron principios de Modularidad para hacer más legible y fácil de implementar el programa.

Meyer (1988, c.p. Pressman, 1998) define: "La modularidad es el atributo del software que permite a un programa ser manejable intelectualmente". Un software monolítico no puede ser entendido con facilidad por un lector. Existen una serie de criterios definidos por este autor, los cuales fueron tomados en cuenta para realizar la modularización del proyecto:

- Capacidad de descomposición modular: Si un método de diseño proporciona un mecanismo sistemático de descomposición del problema en subproblemas, reducirá la complejidad del problema global, consiguiendo, por tanto, una solución modular eficaz.
- Capacidad de empleo de componentes modulares: Si un método de diseño permite ensamblar componentes de diseño existente (reutilizables) en un nuevo sistema, proporcionará una solución modular que no invente nada ya inventado.
- Capacidad de comprensión modular: Si se puede entender un módulo como una unidad por sí sola (sin referencias a otros módulos), será más fácil de construir y cambiar.
- Continuidad modular: Si pequeños cambios a los requisitos del sistema provocan cambios a los módulos individuales, en vez de cambios generalizados en el sistema, se minimizará el impacto de los efectos secundarios de los cambios.
- Protección modular: Si se da una condición aberrante dentro de un módulo y los efectos se restringen dentro de ese módulo, se minimizará el impacto de los efectos secundarios de los errores.

Cabe destacar que también se realizó el diseño del programa pensando en que es un sistema en tiempo real, lo cuál implica consideraciones de (Pressman, 1998, p. 290):

- Tiempo de respuesta.
- Asignación de recursos y manejo de prioridades.
- Sincronización de tareas y comunicación entre ellas.
- Razón de transferencia de datos y tiempo invertido.

IV.4 Fase de codificación

Esta es la sección donde se traduce el diseño en forma legible para la máquina, por medio de algoritmos, estructuras, funciones, etc. La gran mayoría de los requerimientos de hardware y software estaban ya esclarecidos. A continuación se lista el conjunto de soluciones planteadas que satisfacen algunos de los objetivos específicos estipulados en este Trabajo Especial de Grado.

- Se procedió a utilizar el lenguaje C para codificar el *Programa Cuantificador de Voz* (PC-Voz). Sistema solución al problema de cuantificar el tráfico de audio en una red con servicios de VoIP.
- Se implementó la Base de Datos en *Structured Query Language* (SQL). Lugar donde se guardan organizadamente las mediciones de PC-Voz.
- Se realizó el programa cliente a la Base de Datos con *HTML*, *PHP* (http://www.php.net) y *Javascript*. Dicho software, es utilizado como interfaz a los datos capturados y que están registrados en la Base de Datos. El mismo se denomina *Programa Cliente PC-Voz* (PC-PC-Voz).

Algunas de las razones que llevaron a utilizar estos lenguajes de programación fueron:

- En el caso de C:
  - Porque es un lenguaje de propósito general en el cual era posible realizar la codificación sobre la base de los modelos planteados.
  - Viene incluido gratuitamente en el Sistema Operativo. El compilador utilizado fue GNU gcc 3.00 (http://gcc.gnu.org).
  - Cuenta con las librerías necesarias para conectarse a la Base de Datos.
  - Las librerías de captura de paquetes están escritas en C y por lo tanto es mucho más fácil y eficiente hacer las llamadas desde el lenguaje nativo.
  - Es uno de los lenguajes más divulgados y con mejor documentación.

- En el caso de SQL:
  - Es aquel utilizado por el Sistema Manejador de la Base de Datos.
  - Es uno de los lenguajes más divulgados y con mejor documentación.
  - Es un estándar a nivel mundial.
- En el caso de HTML, PHP y Javascript:
  - Los programas desarrollados en este entorno corren en distintos Sistemas Operativos ya que la interfaz es Web.
  - Son lenguajes muy divulgados y con muy buena documentación (http://php.weblogs.com/discuss/msgReader$178).
  - Cuentan con las librerías necesarias para conectarse a la Base de Datos.
  - Vienen integrados con el Sistema Operativo.

IV.5 Fase de prueba

La prueba de software es un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. La misma consiste en hacer revisiones en los procesos lógicos internos del software, asegurando que todas las sentencias se han ejecutado, y en los procesos externos funcionales. (Pressman, 1998, p. 301).

Las pruebas realizadas a PC-Voz se seleccionaron de tal manera que cumplieran los siguientes objetivos (Pressman, 1998, p. 302):

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

A continuación se listan las pruebas realizadas al proyecto:

- Prueba de unidad: Este tipo de prueba se centra en la menor unidad del diseño de software: el módulo. Se prueban los caminos de control importantes, con el fin de descubrir errores dentro del límite del módulo. La complejidad relativa de las pruebas y de los errores descubiertos está limitada por el alcance estricto establecido por la prueba de unidad. (Pressman, 1998, p. 330).

  Myers (1979, c.p. Pressman, 1998), en su libro sobre prueba de software, propone una lista de comprobaciones para probar la interfaz del módulo, los resultados arrojados por la misma se puede ver en la Fase de Prueba del capítulo de Resultados.

- Prueba de integración incremental: Es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción intermodular. El objetivo es asociar los módulos probados con la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. (Pressman, 1998, p. 333).

  Las pruebas de integración incremental se pueden dividir en dos: pruebas de *integración incremental ascendente* y las utilizadas para este proyecto, las pruebas de *integración incremental descendente*, las cuales parten del módulo de control principal hasta los módulos subordinados. (Pressman, 1998, p. 333).

- Prueba de recuperación: Aquellas que se realizan para garantizar que un sistema sea tolerante a fallas; es decir, los fallos del proceso no deben hacer que cese el funcionamiento de todo el sistema. (Pressman, 1998, p. 339).

  Este es un test que fuerza el fallo del software de muchas formas y verifica que la recuperación se lleva a cabo apropiadamente, la recuperación puede ser

automática (realizada por el propio sistema) o puede requerir intervención humana. (Pressman, 1998, p. 339).

Después de una prueba exitosa, pueden ocurrir cuatro cosas:

- Que la persona que realiza la prueba conozca tan bien el código fuente que realice los arreglos necesarios de manera inmediata.
- Que a pesar de haber conseguido un error se desconozca el causante y queda reportado como un bug (error que se espera sea resuelto en el futuro).
- Que se realicen los arreglos de manera empírica: por medio de colocación de segmentos de código fuente en partes específicas de los módulos, tales como banderas, comentarios, etc.
- Que se ejecute una depuración de la aplicación, el cual es un proceso sistematizado y ordenado de detección y corrección de errores. El software que se utilizó para hacer la depuración de la aplicación PC-Voz, se llama *GNU debugger v20010813* (http://sources.redhat.com/gdb).

CAPÍTULO V Resultados

En este capítulo, se presenta la progresión de resultados obtenidos en la realización de este Trabajo Especial de Grado. Los mismos están organizados según la fase metodológica a la cual corresponden.

V.1 Fase de Recopilación de Información

Como se dijo anteriormente, esta fase reunió un conjunto de documentos que fueron la plataforma para la realización de esta Investigación. A continuación se presentan los mismos en forma esquematizada. Cabe destacar que no son todos los instrumentos utilizados; pero son la gran mayoría de los considerados formales según la organización que los publica.

RFCs de la Internet Engineering Task Force (http://www.ietf.org)

- RFC 768. *User Datagram Protocol*. J. Postel. Agosto 1980.
- RFC 791. *Internet Protocol*. J. Postel. Septiembre 1981.
- RFC 793. *Transmission Control Protocol*. J. Postel. Septiembre 1981.
- RFC 894. *Standard for the transmission of IP datagrams over Ethernet Networks*. C. Hornig. Abril 1984.
- RFC 1042. *Standard for the transmission of IP datagrams over IEEE 802 networks*. J. Postel, J.K. Reynolds. Febrero 1988.
- RFC 1180. *TCP/IP tutorial*. T.J. Socolofsky, C.J. Kale. Enero 1991.
- RFC 1889. *RTP: A Transport Protocol for Real-Time Applications. Audio-Video Transport Working Group*, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. Enero 1996.

Recomendaciones de la ITU-T (http://www.itu.int/ITU-T)

- ITU-T Recommendation Q.931: *Digital Subscriber Signalling System No. 1 (Dss 1) – Isdn User-Network Interface Layer 3 Specification For Basic Call Control*. Marzo 1993.
- ITU-T Recommendation H.323, Version 2: *Packet-Based Multimedia Communications Systems*. Octubre 1997.
- ITU-T Recommendation H.225.0, Version 2: *Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems*. Marzo 1997.
- ITU-T Recommendation H.245, Version 3: Control protocol for multimedia communication. Enero 1998.

Publicaciones de la ACM (http://www.acm.org)

- *mmdump: A Tool for Monitoring Internet Multimedia Traffic*. Jacobus van der Merwe, Ramón Cáceres, Yang-hua Chu, Cormac Sreenan. ACM Computer Communication Review. Octubre 2000, Nº 5, Vol 30.
- *Voice Over IP*. Upkar Varshney, Andy Snow,Matt McGivern, and Christi Howard. Communications of the ACM. Enero 2002, Vol 45, Nº 1
- *Packet Types: Abstract Specification of Network Protocol Messages*. Peter J. McCann and Satish Chandra. ACM SIGCOMM 2000.
- *Deriving Traffic Demands for Operational IP Networks: Methodology and Experience*. Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, Fred True. ACM Computer Communication Review. Octubre 2000, Nº 4, Vol 30.

Otras publicaciones

- *Analysis of Voice Over IP Traffic*, J.P. Curtis, J.G. Cleary, A.J. McGregor y M.W Pearson. Departamento de Ciencias de la Computación de la Universidad de Waikato, Nueva Zelanda.
- *Comparison of H.323 and SIP for IP Telephony Signaling*. Kashif Israr Siddiqui, Prof. Muhammad Kamran, Sheikh Fraz Tajammul. Proc. of IEEE 4th International Multitopics Conference Dec 2001, Lahore, Pakistan.
- *Comparison of H.323 and SIP for IP Telephony Signaling*. Ismail Dalgic and Hanlin Fang. Proc. of Photonics East, Boston, Massachusetts, September 20-22, 1999.
- *Voice Over Packet White Paper*. Texas Instruments. Versión 2. Enero 1998.
- *Voice over Internet Protocol*. DSQ Software Ltd.
- *MySQL Reference Manual*. Detron HB y MySQL Finland AB. Febrero 2001.
- *Executing SQL statements in MySQL databases using C Excerpt from Professional Linux Programming*. Neil Matthew, Richard Stones. Febrero 2001.
- *Linux VoIP Howto*. Roberto Arcomano. Junio 2001.
- *GNU C Library Reference Manual*. Sandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram y Ulrich Drepper. Agosto 1999.

V.2 Fase de Análisis

En esta parte era importante determinar sobre qué plataforma tecnológica si iba a trabajar y con qué herramientas se podía contar. El objetivo principal de este Trabajo Especial de Grado es el diseño y desarrollo de un sistema experimental básico de cuantificación de paquetes de audio sobre IP de tipo genérico y abierto. Esta característica condujo a la utilización de software gratuito amparado bajo las Licencias Públicas GNU (http://www.gnu.org/licenses/licenses.html). A continuación se presenta el conjunto de herramientas que fueron seleccionadas en este punto:

- Sistema Operativo gratuito y de código abierto: *Mandrake Linux 8.1* (http://www.mandrake.com). La escogencia de este software fue basada en:
  - Cuenta con casi todas las aplicaciones necesarias para realizar los próximos pasos de la metodología (diseño, codificación, prueba).
  - Es una de las distribuciones de Linux más utilizada y con mejor documentación. (http://www.mandrakeforum.com/article.php?sid=1509).
  - Es considerado muy estable y fácil de instalar por el tesista.
- Librería de Captura de Paquetes: *PCAP* (Integradas en el Sistema Operativo, http://www.tcpdump.org).
- Sistema Manejador de Base de Datos MySQL (Integrado en el Sistema Operativo, http://www.mysql.com) y su respectiva librería codificada en C: *MySQLClient*.
- Lenguaje de programación C (Integrado en el Sistema Operativo, http://www.gnu.org/software/gcc/gcc.html).
- Lenguaje de programación HTML, PHP y Javascript.
- Programa para realizar llamadas H.323 y poder así probar la codificación: OpenPhone (Integrado en el Sistema Operativo, http://www.openh323.org).
- La plataforma de hardware necesaria que se definió en este instante para desarrollar la aplicación: computador de arquitectura Intel Pentium o compatible con unidad de CD-ROM, al menos 32 MB de memoria RAM, un disco duro con una partición de al menos 1.0 GB de espacio libre. Para ver en detalle los requisitos del sistema, véase el apéndice 4.

Teniendo definidos mejor los recursos técnicos, el siguiente paso fue determinar en base a qué protocolo de VoIP se iba a trabajar. Cumpliendo así en su totalidad, el *primer objetivo específico* de este Trabajo Especial de Grado.

Se seleccionó el estándar de la ITU-T H.323 versión 2 por varios aspectos:

- La mayoría de las aplicaciones de software que corren VoIP ejecutan este protocolo. Esto acarrea muchas ventajas para poder realizar las partes de codificación y pruebas.
- Se disponía de toda la documentación necesaria del protocolo.
- Al estudiar la arquitectura (la pila del protocolo y su funcionamiento), se consideró factible realizar la aplicación basándose en este.
- Según Collins (2001), "H.323 versión 2 es ampliamente implementada en soluciones VoIP, y de varias maneras, esta versión es el estándar para VoIP hoy en día".
- Es publicado y mantenido por la International Telecommunication Union–Telecommunication Standarization Sector (ITU-T) desde el año 1996.

A continuación se responden diversas preguntas correspondientes a esta etapa y que son argumentadas fundándose en los documentos conseguidos en la fase de recopilación de Información.

- ¿Cómo debe situarse el programa en la red para hacer las mediciones?.

Para hacer las mediciones se plantearon dos maneras de ubicar la aplicación en la red. La primera es como un *sniffer* (husmeador), también conocida como *medición pasiva.* (http://wand.cs.waikato.ac.nz/wand/publications/jamie_420/final/node9.html). La figura 10 ilustra esta opción.

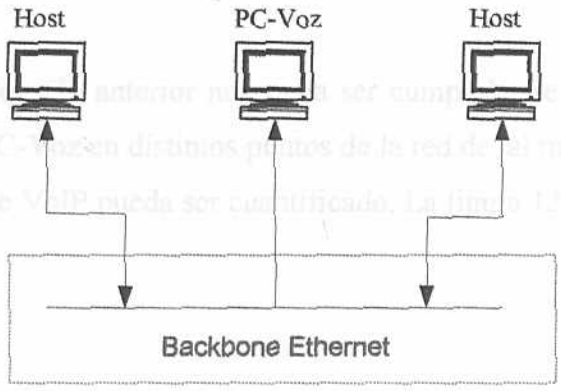

Figura 10: *Medición pasiva de los paquetes de VoIP*

Tomado de http://wand.cs.waikato.ac.nz/wand/publications/jamie_420/final/node13.html

Esta implementación trae numerosas ventajas:

- Al no estar ubicado en una entidad H.323, no se crean cuellos de botella en la red.
- En caso de una caída del sistema PC-Voz, no se ve afectada la efectividad y eficiencia de la red.
- Es fácil de codificar.
- Se pueden ubicar monitores en distintos puntos de la red, subredes, etc.

La ubicación de PC-Voz como un sniffer también trae restricciones:

- Se debe colocar el sistema de cuantificación en un punto de la red por donde pase todo el tráfico, asegurando así que no va a existir reconteo de paquetes. La figura 11 muestra lo planteado.

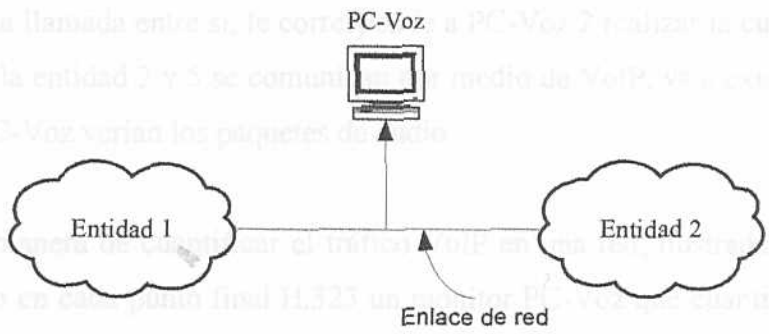

Figura 11: *Medición pasiva en un punto de la red por donde pasa todo el tráfico VoIP*
Tomado de http://wand.cs.waikato.ac.nz/wand/publications/jamie_420/final/node10.html

- En caso de que lo anterior no pueda ser cumplido, se deben colocar varios monitores PC-Voz en distintos puntos de la red de tal manera que la totalidad del tráfico de VoIP pueda ser cuantificado. La figura 12 ilustra un ejemplo de esto.

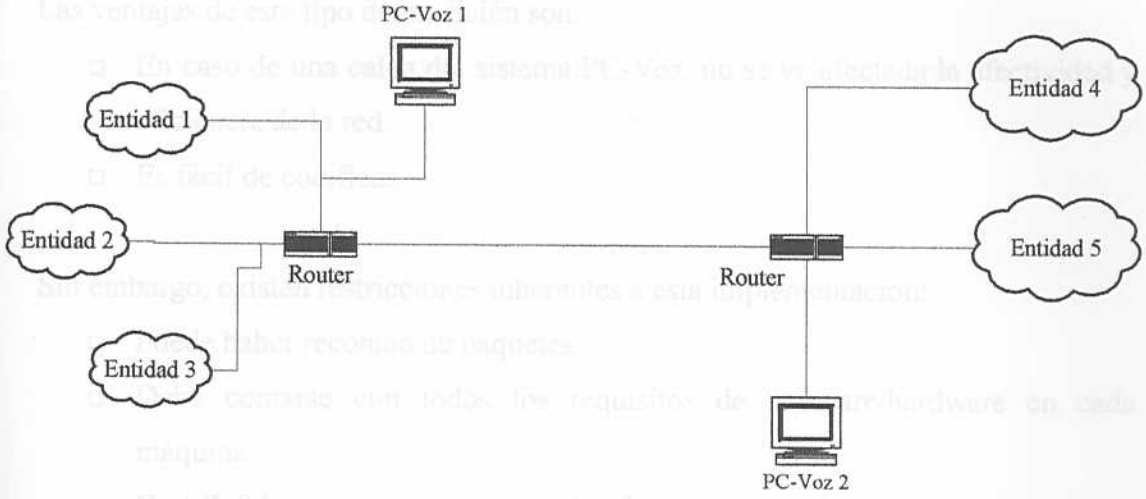

Figura 12: *Medición pasiva en varios puntos de la red por donde pasa todo el tráfico VoIP*

En este escenario, si la entidad 1 y la entidad 3 se involucran en una llamada entre sí, le corresponde a PC-Voz 1 realizar la cuantificación; así mismo si las entidades 4 y 5 se envuelven en una llamada entre sí, le corresponde a PC-Voz 2 realizar la cuantificación. Sin embargo, si la entidad 2 y 5 se comunican por medio de VoIP, va a existir reconteo, ya que ambos PC-Voz verían los paquetes de audio.

La segunda manera de cuantificar el tráfico VoIP en una red, ilustrada en la figura 13, es colocando en cada punto final H.323 un monitor PC-Voz que cuantifique sólo el tráfico de voz que emita o reciba dicha entidad. Se hace referencia a este tipo de medición como *medición individual*.

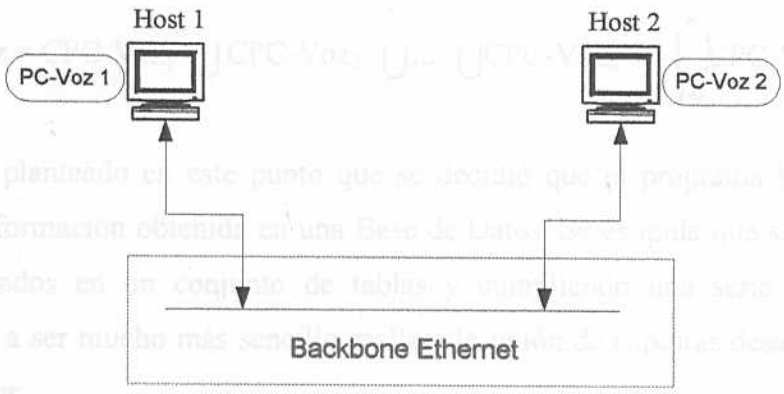

Figura 13: *Medición individual del tráfico VoIP*

Las ventajas de este tipo de medición son:

- En caso de una caída del sistema PC-Voz, no se ve afectada la efectividad y eficiencia de la red.
- Es fácil de codificar.

Sin embargo, existen restricciones inherentes a esta implementación:

- Puede haber reconteo de paquetes.
- Debe contarse con todos los requisitos de software/hardware en cada máquina.
- Es difícil hacer mantenimiento del software.
- Es difícil implementar una buena política de seguridad.
- Debe estar a disposición física la máquina para poder instalar el software.

El sistema PC-Voz es capaz de actuar en ambos modos, tanto realizando mediciones activas e individuales. El problema del reconteo de paquetes se puede solucionar enfocando el problema basándose en los principios de la *unión* de la *teoría de conjuntos*. Busby R., Kolman B. y Ross S. (1997). Para esto, se define la variable $CPC\text{-}Voz_i$ como la cuantificación en un período de tiempo t de la aplicación PC-Voz *i-ésima*, por lo tanto si existen n aplicaciones PC-Voz ejecutándose en un instante t, la cuantificación total de paquetes de esta red de VoIP, asignada a la variable *TPC-Voz*, será determinada por la unión de todas las $CPC\text{-}Voz_i$.

$$TPC\text{-}Voz = CPC\text{-}Voz_1 \cup CPC\text{-}Voz_2 \cup \ldots \cup CPC\text{-}Voz_n = \bigcup_{i=1}^{n} CPC\text{-}Voz_i$$

Es por lo planteado en este punto que se decidió que el programa PC-Voz debía registrar la información obtenida en una Base de Datos. Se estipula que si se tienen los datos organizados en un conjunto de tablas y cumpliendo una serie de reglas de integridad, va a ser mucho más sencillo realizar la unión de capturas desarrollada en el párrafo anterior.

- ¿Cómo se va a crear un ambiente de pruebas para el software?.

El ambiente de pruebas del cuál se dispuso debido a las limitaciones técnicas, fue aquél de dos computadoras conectadas en una red ethernet, tal como se ilustra en la figura 14.

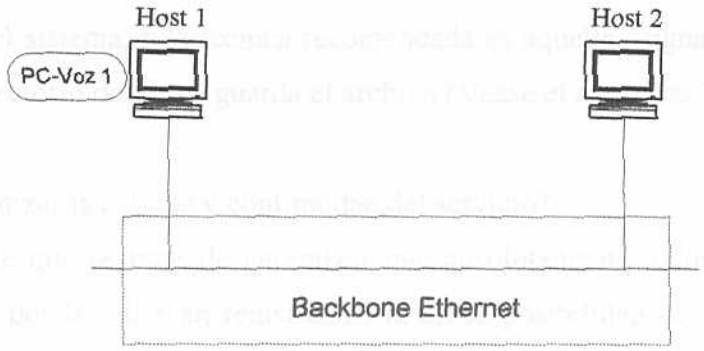

Figura 14: *Ambiente de pruebas para PC-Voz*

El host 1 es una computadora que está ejecutando Linux y por lo tanto puede procesar PC-Voz. El software para realizar y recibir llamadas H.323 en esta máquina es OpenPhone v1.1.4. El host 2 es una computadora ejecutando Microsoft® Windows 98 y, entonces, no puede ejecutar PC-Voz. La existencia de diferentes Sistemas Operativos permiten hacer pruebas cruzadas con distintas aplicaciones, en este caso se utilizó OpenPhone v1.1.4 y Microsoft® NetMeeting 3.01.

- ¿Cómo se va a asegurar la integridad de las mediciones?.

Con anterioridad, se explicó que era necesario registrar los paquetes en una Base de Datos ya que sería mucho más sencillo determinar exactamente cuál era el tráfico de VoIP en una red. Todo Sistema Manejador de la Base de Datos como regla debe mantener la integridad y la recuperación ante fallos de los datos que están registrados en una Base de Datos que este maneja. En el caso de MySQL, según Detron HB y MySQL Finland AB, se utiliza el paradigma de *transacciones atómicas* para garantizar la integridad de los datos y se usa la utilidad *myisamchk* para revisar y recuperar el Sistema manejador de la Base de Datos ante fallos.

También se determinó que era necesario guardar un registro de los paquetes capturados en un archivo log. Esta implementación permite una manera alternativa de recuperar los datos en caso de una falla grave en el Sistema Manejador de la Base de Datos. Esto conlleva a que este archivo debe almacenarse localmente en la máquina que está ejecutando PC-Voz. La seguridad de este archivo debe proporcionarla el administrador del sistema, y la técnica recomendada es aquella asignando los permisos adecuados al directorio donde se guarda el archivo (Véase el apéndice 3).

- ¿Cómo garantizar la calidad y continuidad del servicio?.

En el caso de que se trate de garantizar que absolutamente todos los paquetes de VoIP que pasen por la red sean registrados, la única posibilidad de esto, es poniendo monitores PC-Voz en distintos puntos de la red. Lo planteado establece que si un paquete no puede ser registrado, es debido a que el sistema PC-Voz debe estar sobrecargado de trabajo. En consecuencia, al estar dividida una red en subredes, se puede situar un monitor PC-Voz en cada subred y así cada monitor PC-Voz va a cuantificar el tráfico de su respectiva subred. Obteniendo como resultado, una disminución del número de paquetes que el monitor debe capturar.

Otra medida que fue implementada para garantizar la calidad de servicio, fue la denominada *detección dinámica de los puertos de sesiones*, de tal manera que se establece que la gran mayoría del tráfico que captura PC-Voz son los paquetes de VoIP y no cualquier información que pase por la red. En el anexo 1 se trata el tema en detalle.

Para garantizar la continuidad del servicio se determinó que era necesario implementar un módulo experimental básico de software de Seguimiento de Actividad (MoSAc). El mismo tiene la tarea de monitorear la ejecución de la aplicación PC-Voz. De tal manera, si PC-Voz se detiene, es cargado por MoSAc.

Sin embargo, es necesario que también MoSAc sea continuamente revisado por otro proceso. Esto es debido, a que si llegase a detenerse MoSAc, no habría quién cargara a

PC-Voz ante un eventual fallo, por lo que se implementó el sistema MoSAc como un programa de dos capas. Este diseño se discute en la Fase de Diseño de este capítulo.

- ¿Cómo se debe hacer la cuantificación del tráfico de VoIP?.

La política de cuantificación establece que el conteo debe hacerse en paquetes, no en unidades de tiempo, ni en tamaño, ni en ninguna otra medida. La razón fundamental de esto, es cada paquete es importante, porque todos los mensajes H.225.0 y H.245 se transmiten por medio de un (1) paquete. Es a través de cada uno de éstos paquetes que se pueden detectar los inicios de sesiones, las finalizaciones de llamada, etc.

A pesar de que la unidad de medición es el paquete, se estableció que la duración de la sesión también debería quedar registrada, para realizar las uniones de sesiones en la Base de Datos.

En consecuencia con la política planteada de cuantificación de tráfico de audio por paquetes, se cumple el *segundo objetivo específico* de este Trabajo Especial de Grado.

V.2.1 Modelo Conceptual de la Base de Datos

A continuación, se presenta en la figura 15 el modelo Conceptual de la Base de Datos que fue utilizada para almacenar los datos recogidos.

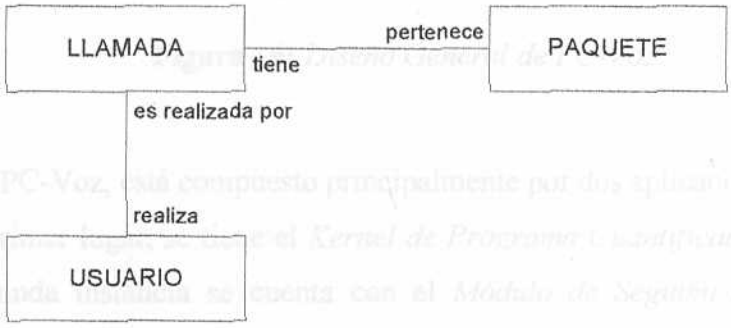

Figura 15: *Modelo Conceptual de la Base de Datos*

Se mantuvo una estructura sencilla en este modelo para garantizar eficiencia en el registro de los resultados al momento de monitorear la red. Al haber una llamada sólo se va a insertar en la tabla de USUARIO y en la de LLAMADA. Mientras que para registrar un paquete sólo se debe hacer una inserción en la tabla de PAQUETE.

V.3 Fase de Diseño

V.3.1 Diseño General

En esta sección se explica la base técnica que precede y sustenta la codificación. Como se mencionó con anterioridad, el sistema PC-Voz está modularizado según una serie de reglas mencionadas en el punto de Desarrollo del capítulo anterior. A continuación se presenta en la figura 16 cómo está diseñado este sistema de manera general.

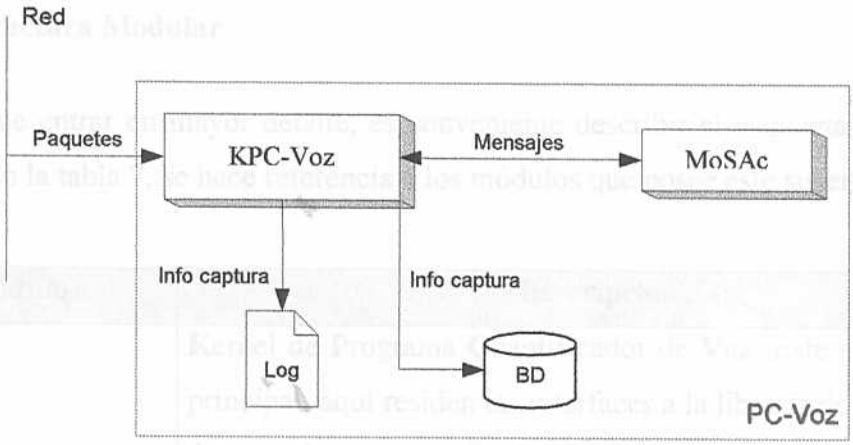

Figura 16: *Diseño General de PC-Voz*

El sistema PC-Voz, está compuesto principalmente por dos aplicaciones corriendo en paralelo. En primer lugar, se tiene el *Kernel de Programa Cuantificador de Voz* (KPC-Voz). En segunda instancia se cuenta con el *Módulo de Seguimiento de Actividad* (MoSAc). El funcionamiento de PC-Voz comienza cuando los paquetes son capturados de acuerdo a un filtro definido en KPC-Voz, este filtro es una cadena de caracteres que

especifican una serie de parámetros que son evaluados por la librería de captura y que restringen el tráfico que es visible para la aplicación.

Posteriormente dichos paquetes son procesados y se convierten en información que es registrada tanto en un archivo de texto (archivo log) como en la Base de Datos. En el archivo de texto se van a almacenar todos los paquetes capturados, mientras que en la Base de Datos se va a registrar de manera ordenada las llamadas, los usuarios involucrados en la llamada, los puntos finales participantes, los paquetes de voz solamente, e información adicional.

También se encuentra el sistema opcional MoSAc, que sigue la actividad de PC-Voz. Entre ellos se pasan periódicamente una serie de mensajes para cumplir con sus funciones.

V.3.2 Estructura Modular

Antes de entrar en mayor detalle, es conveniente describir el esquema modular de PC-Voz. En la tabla 7, se hace referencia a los módulos que posee este sistema.

Módulo	Descripción
KPC-Voz	**K**ernel de **P**rograma **C**uantificador de **Voz**. Este es el módulo principal, aquí residen las interfaces a la librería de captura y se detectan los inicios y finalizaciones de llamadas. En consecuencia con el planteamiento del módulo KPC-Voz, se cumple el *cuarto objetivo específico* de este Trabajo Especial de Grado a nivel de diseño.
MMCQ931	**M**ódulo **M**anejador de **C**abecera **Q.931**. Módulo donde se precisan las estructuras de datos, constantes y procedimientos para el manejo del protocolo Q.931.
MMPG	**M**ódulo **M**anejador de **P**rocedimientos **G**enerales. Aquí se

	definen estructuras de datos, constantes y procedimientos generales; como por ejemplo conversiones entre unidades.
MMRD	**M**ódulo **M**anejador de **R**ecursos **D**inámicos. Módulo donde se establecen las estructuras de datos, constantes y procedimientos sobre los recursos solicitados dinámicamente, tal como es el caso de la memoria.
MAnaP	**M**ódulo de **Aná**lisis de **P**aquetes. En este módulo, se analizan los paquetes para saber de qué tipo son y qué acción tomar, aquí también se van creando los filtros dinámicos.
MMBD	**M**ódulo **M**anejador de la **B**ase de **D**atos. Como su nombre lo indica, es el encargado de manejar la interfaz con la Base de Datos y sus procedimientos.
MPSC	**M**ódulo **P**roveedor de **S**ervicios de **C**ifrado. Módulo que ofrece servicios de cifrado y descifrado trivial de archivos, es usado para guardar los datos de conexión a la base de datos.
MoSAc	**M**ódulo de **S**eguimiento de **Ac**tividad. Es opcional, PC-Voz puede correr sin este módulo ejecutándose; MoSAc es el encargado de garantizar la continuidad del servicio de KPC-Voz.

Tabla 7: *Módulos del sistema PC-Voz*

A continuación, se presenta gráficamente la estructura modular de KPC-Voz en la figura 17. El sistema MoSAc es un sólo módulo principal, por lo tanto no tiene una estructura modular. En secciones posteriores a ésta, se habla en detalle del diseño de MoSAc. Para ver los tipos de datos pasados entre ellos, sus constantes y respectivas funciones, consúltese el anexo 3.

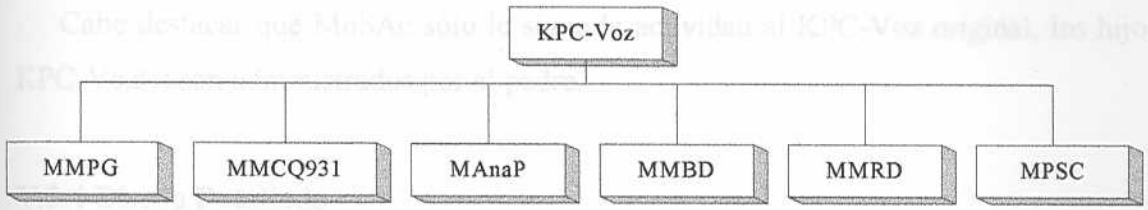

Figura 17: *Módulos del sistema KPC-Voz*

V.3.3 Diseño Funcional

El sistema PC-Voz fue diseñado para ejecutarse concurrentemente. Cada vez que se detecta una llamada nueva, se crea una copia del programa con un filtro específico que va a capturar el tráfico sólo de esa llamada en concreto. Al *hijo* i-ésimo creado para cuantificar una llamada se le denomina *sesión*.

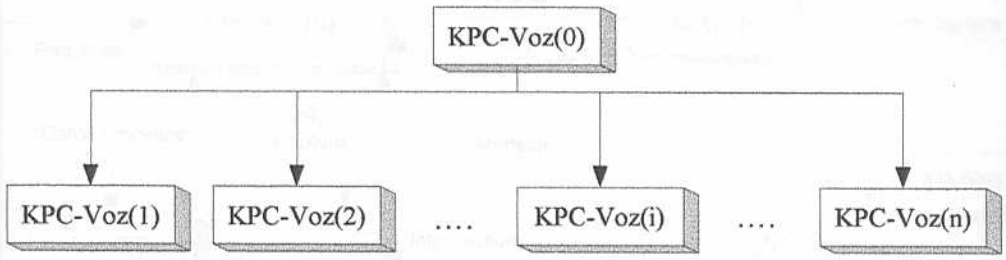

Figura 18: *Diseño funcional de KPC-Voz*

Como se ve en el gráfico 18, existen n llamadas que se están cuantificando, es por eso que hay n + 1 programas KPC-Voz ejecutándose concurrentemente. El KPC-Voz(0) es el original o el *padre*, es el que detecta los inicios y las finalizaciones de las llamadas. Cuando se negocia una llamada, se crea un hijo KPC-Voz(i) de KPC-Voz(0) con un filtro específico que va a cuantificar el tráfico de audio entre dos puntos finales H.323. Al finalizar dicha llamada, KPC-Voz(0) le indica al hijo KPC-Voz(i) que cese la cuantificación y posteriormente finaliza su ejecución.

Cabe destacar que MoSAc sólo le sigue la actividad al KPC-Voz original, los hijos KPC-Voz(i) son administrados por el padre.

V.3.4 Diseño Detallado

Este diseño se muestra en la figura 19. No todos los módulos están incluidos ya que no son utilizados para lo que se trata de explicar en este gráfico. A continuación se hace un recuento de lo que ocurre internamente en KPC-Voz cuando se están ejecutando.

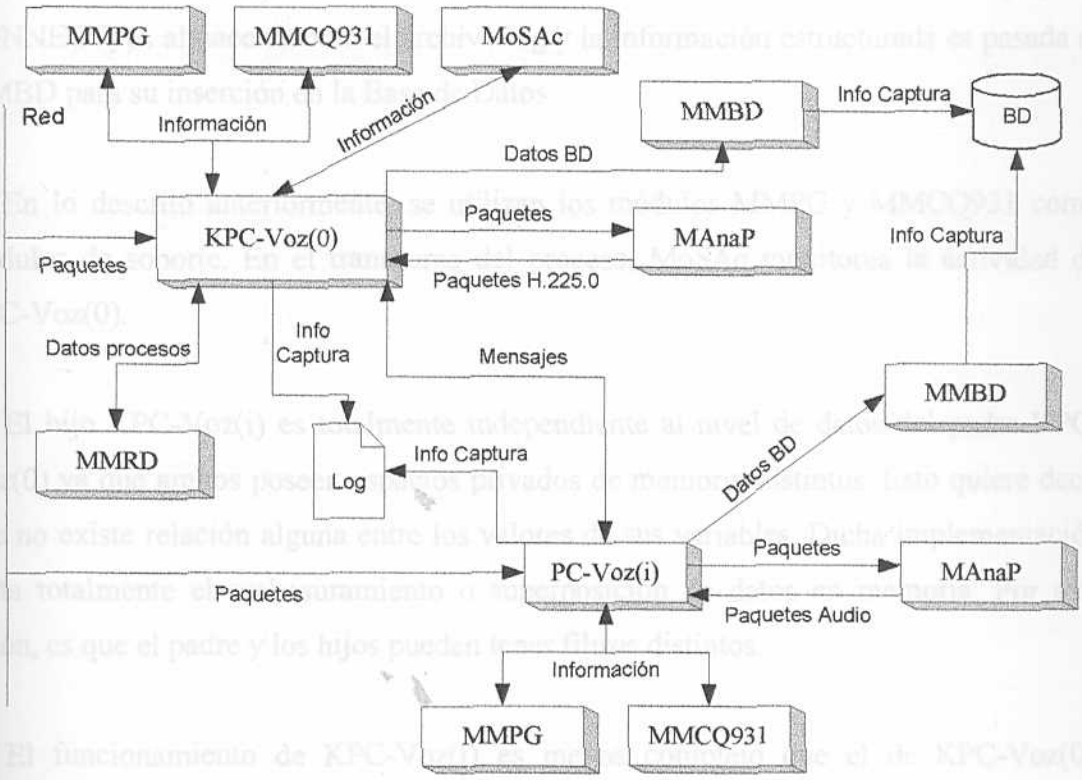

Figura 19: *Diseño en detalle de KPC-Voz*

Los paquetes que cumplen con los requisitos definidos en KPC-Voz(0) son capturados y son pasados a MAnaP, este módulo descompone el paquete y lo transforma en una estructura adecuada para el programa principal. Existen diversos tipos de paquetes H.225.0, pero los que procesa KPC-Voz(0) son los de CONNECT y RELEASE

COMPLETE (véase el Marco Teórico), cualquier otro tipo de paquete es identificado, más no se da soporte para los mismos.

Cuando se detecta un paquete de CONNECT, KPC-Voz(0) crea a partir de los datos arrojados por MAnaP un filtro que contiene los puertos, hosts y un conjunto de datos adicionales que van a servir para cuantificar los paquetes de audio. Estos datos son pasados como parámetros a un nuevo proceso hijo KPC-Voz(i) idéntico al padre. Para llevar un control del estado del programa y sus hijos, se lleva registro de todos los hijos creados por PC-Voz(0) a través de una lista definida en MMRD. Dicho paquete CONNECT, es almacenado en el archivo log y la información estructurada es pasada al MMBD para su inserción en la Base de Datos.

En lo descrito anteriormente, se utilizan los módulos MMPG y MMCQ931 como módulos de soporte. En el transcurso del proceso, MoSAc monitorea la actividad de KPC-Voz(0).

El hijo KPC-Voz(i) es totalmente independiente al nivel de datos del padre KPC-Voz(0) ya que ambos poseen espacios privados de memoria distintos. Esto quiere decir que no existe relación alguna entre los valores de sus variables. Dicha implementación evita totalmente el embasuramiento o superposición de datos en memoria. Por esta razón, es que el padre y los hijos pueden tener filtros distintos.

El funcionamiento de KPC-Voz(i) es menos complejo que el de KPC-Voz(0). Cuando KPC-Voz(i) recibe un paquete de la red, no maneja ninguna lista, ni debe llevar control de los hijos, solamente verifica que dicha información sea de audio sobre IP, comprueba que existe una sesión creada para dicha llamada y posteriormente, es registrada en la Base de Datos y en el archivo log.

En consecuencia con el diseño de PC-Voz, se cumple el *cuarto y quinto objetivo específico* de este Trabajo Especial de Grado a nivel de diseño.

V.3.5 Diseño de PC-PC-Voz

El Programa Cliente PC-Voz, mantiene la estructura jerárquica definida en la figura 20

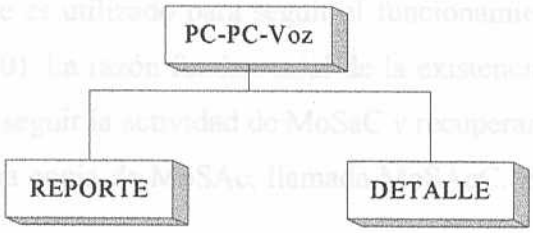

Figura 20: *Diseño de PC-PC-Voz*

A continuación, se hace referencia en la tabla 8 a los módulos que componen dicho programa:

Módulo	Descripción
PC-PC-VOZ	Ventana de inicio del programa. En esta se pide el login, la clave y la dirección del servidor de la Base de Datos al usuario. En dicha página se realiza la autentificación contra el Sistema Manejador de la Base de Datos.
REPORTE	Ventana donde se solicitan los parámetros de búsqueda y se generan los encabezados de los reportes con las llamadas de VoIP.
DETALLE	Ventana detalle de una llamada en específico.

Tabla 8: *Diseño de PC-PC-Voz*

En consecuencia con el diseño de PC-PC-Voz, se cumple el *séptimo objetivo específico* de este Trabajo Especial de Grado a nivel de diseño.

V.3.6 Diseño de MoSAc

MoSAc está diseñado de manera monolítica en dos niveles. Este sistema es un demonio de usuario que es utilizado para seguir el funcionamiento de otro proceso, en este caso de KPC-Voz(0). La razón fundamental de la existencia de dos niveles en este modelo, es que se debe seguir la actividad de MoSaC y recuperarlo en caso de una caída, esta función la tiene una copia de MoSAc, llamada MoSAcC. En la figura 21 se ilustra este procedimiento.

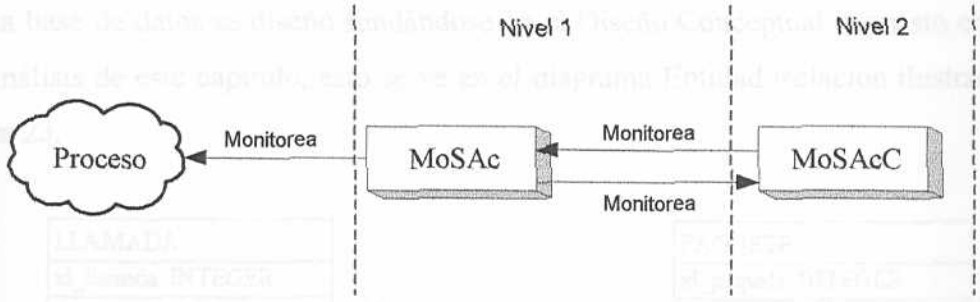

Figura 21: *Diseño de MoSAc*

El sistema de monitoreo debe cargarse primero y posteriormente la aplicación que quiere monitorearse. Cada vez que exista una caída de alguna de las aplicaciones, la entidad que la va a cargar, elimina todos los procesos hijos (zombies) de la que se cayó. En la figura 22 se muestra el diseño de MoSAc en su arranque.

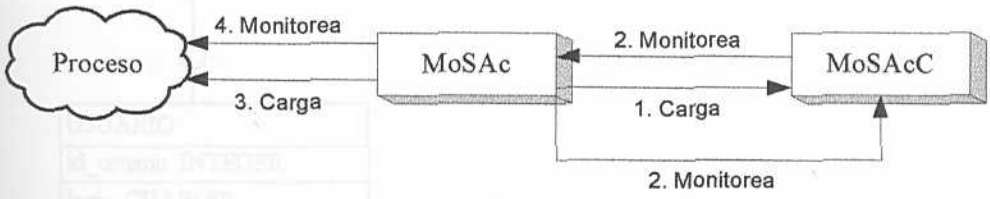

Figura 22: *Diseño de MoSAc en el arranque*

Como se aprecia en la figura anterior, lo primero que hace MoSAc es cargar a MoSAcC, posteriormente, ambas aplicaciones proceden a monitorearse entre sí, para que a continuación MoSAc cargue al proceso y pase a monitorearlo.

En consecuencia con el diseño MoSAc, se cumple el *sexto objetivo específico* de este Trabajo Especial de Grado a nivel de diseño.

V.3.7 Diseño de la Base de Datos

La base de datos se diseñó fundándose en el Diseño Conceptual expuesto en la fase de Análisis de este capítulo, esto se ve en el diagrama Entidad Relación ilustrado en la figura 23.

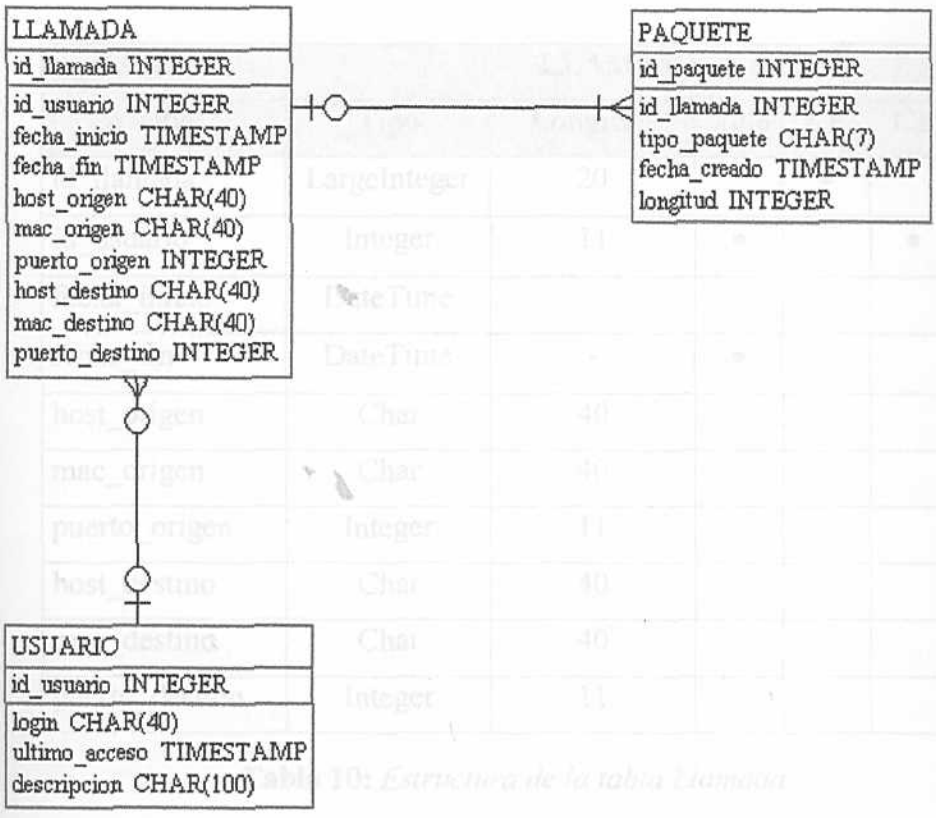

Figura 23: *Modelo ER de la Base de Datos*

A continuación se especifica en el conjunto de tablas 9, 10 y 11, la información detallada de los atributos de cada entidad

Tabla	**USUARIO**				
Atributo	**Tipo**	**Longitud**	**Nulo**	**CP**	**CF**
id_usuario	Integer	11		•	
login	Char	40			
ultimo_acceso	DateTime	-	•		
descripcion	Char	100	•		

Tabla 9: *Estructura de la tabla Usuario*

Tabla	**LLAMADA**				
Atributo	**Tipo**	**Longitud**	**Nulo**	**CP**	**CF**
id_llamada	LargeInteger	20		•	
id_usuario	Integer	11	•		•
fecha_inicio	DateTime	-			
fecha_fin	DateTime	-	•		
host_origen	Char	40			
mac_origen	Char	40			
puerto_origen	Integer	11			
host_destino	Char	40			
mac_destino	Char	40			
puerto_destino	Integer	11			

Tabla 10: *Estructura de la tabla Llamada*

Tabla	PAQUETE				
Atributo	**Tipo**	**Longitud**	**Nulo**	**CP**	**CF**
id_paquete	LargeInteger	20		•	
id_llamada	LargeInteger	20	•		•
tipo_paquete	Char	10			
fecha_creado	DateTime	-			
longitud	Integer	11			

Tabla 11: *Estructura de la tabla Paquete*

En consecuencia con el diseño del modelo de una Base de datos para el registro de paquetes, se cumple el *tercer objetivo específico* de este Trabajo Especial de Grado a nivel de diseño.

V.4 Fase de Codificación

En esta sección se tradujo el diseño en líneas de código fuente estructuradas. En la tabla 12, se hace la equivalencia entre los módulos diseñados explicados en el punto anterior y los módulos implementados escritos en lenguaje C. Cabe destacar que aquí no se representan los módulos estándar de las librerías del compilador. La razón de esto, es que las librerías pueden tener variaciones de acuerdo al compilador y al sistema operativo.

Existe un aproximado de 4500 líneas de código en lenguaje C. Un estimado de 1000 líneas de código en el programa cliente PC-Voz y unas 60 líneas de código en el guión de la Base de Datos.

Módulo Diseño	Módulo (.c)	Cabecera (.h)
KPC-Voz	kpc_voz.c	-
MMCQ931	mmcq931.c	mmcq931.h
MMPG	mmpg.c	mmpg.h
MMRD	mmrd.c	mmrd.h
MAnaP	manap.c	manap.h
MMBD	mmbd.c	mmbd.h
MPSC	mpsc.c	mpsc.h
MoSAc	mosac.c	mosac.h

Tabla 12: Equivalencia diseño codificación sistema PC-Voz

En consecuencia con la codificación de los módulos de PC-Voz, se cumple el *cuarto, quinto y sexto objetivo específico* de este Trabajo Especial de Grado a nivel de desarrollo.

También en esta fase se creó el archivo pc_voz.sql con el script de la Base de Datos para MySQL. Cumpliendo a cabalidad, el *tercer objetivo específico* de este Trabajo Especial de Grado a nivel de construcción.

Se codificó el programa Cliente PC-Voz. Son una serie de páginas Web escritas en HTML, PHP y Javascript para poder visualizar la información capturada. Abarcando así, el *séptimo objetivo específico* de este Trabajo Especial de Grado a nivel de desarrollo.

En el anexo 3, se muestra el código fuente de las cabeceras (.h) de los archivos en C y en el anexo 4, del script de la Base de Datos para MySQL.

Un punto crítico en toda implementación de programas en tiempo real, son los tipos de datos y las estructuras de datos abstractas utilizadas. A continuación se va a realizar un resumen de algunos puntos específicos sobre el funcionamiento de KPC-Voz.

El primer paso, cuando se carga el sistema, es ver si existe el archivo *bd.cfg*, el cuál es creado por KPC-Voz(0) la primera vez que corre y el cuál contiene información cifrada sobre la conexión a la Base de Datos. Dicho archivo es cifrado y descifrado con las funciones del MPSC *Cifrar* y *Descifrar* respectivamente, si dicho archivo no existe, se procede a pedir los datos al usuario por medio de la consola.

Después se detectan las interfaces de red disponibles con la función *Buscar_Interfaces*. Con la salida de la misma, se determina la dirección IP y la máscara de subred a la que pertenece el host con la función *Buscar_IP_y_Mascara*.

Posteriormente. con la información generada por *Buscar_IP_y_Mascara*, se carga el motor de captura KPC-Voz(0) con la función *Capturar_Paquetes*. Este procedimiento recibe una serie de parámetros, de los cuales cabe destacar:

- El filtro generado por el procedimiento *Crear_Filtro*,
- Una variable booleana que define si la captura va a ser pasiva o activa
- Un apuntador a la función *Analizar_Datos* (callback function), que va a ser la encargada de procesar el paquete.

Cuando un paquete es capturado por el motor de captura de paquetes y pasado a la función *Analizar_Datos*, esta reenvía el paquete a la función *Analizar_Paquete* del módulo MAnaP. En el módulo MAnaP, es donde se desglosa un conjunto de datos en forma Unicode Char (u_char) a estructuras de datos dinámicas que almacenan en memoria toda la información de cada capa de la pila del protocolo H.323 (Véase Marco Teórico). Dichas estructuras son:

- const struct ether_header *cabec_ethernet: Registra los datos de Ethernet.

- const struct ip *cabec_ip: Guarda la información de IP.
- const struct tcphdr *cabec_tcp: Almacena la información de TCP.
- const struct udphdr *cabec_udp: Registra los datos de UDP
- const struct q_931hdr *cabec_q_931: Contiene la información de Q.931
- const struct open_logical_channel *openlc: Recopila los datos de H.245.

Posteriormente se obtiene del conjunto de estructuras anteriores, la información de interés que va a ser utilizada para crear sesiones, cuantificar en una determinada sesión, finalizar una sesión, etc. Dicha conversión es almacenada en *struct salida_analisis**, que es una combinación de las estructuras *struct datos_bd *datos* y *struct lista_pids *lista_procesos*, cuya función es respectivamente, mantener en memoria los datos que van a ser guardados en la Base de Datos y registrar en memoria información correspondiente al proceso KPC-Voz(i) que está realizando la cuantificación.

Con los datos que apunta *struct salida_analisis**, existen varias posibilidades, que dicha información corresponda a los mensajes Q.931 CONNECT o RELEASE COMPLETE, que sea un mensaje de H.245, que sea un mensaje de audio o que sea otro tipo de mensaje. En base al tipo de mensaje y su información detallada, se toman determinadas acciones:

- Cuando es un mensaje de CONNECT y no hay una sesión creada para los involucrados (en KPC-Voz(0) se mantiene la variable *struct lista_pids *lista_procs*, que es la encargada de llevar la información de cada uno de los hijos KPC-Voz(i). Esta variable es un apuntador a una lista enlazada simple y es actualizada por un TDA definido en MMRD con la información proveniente de *struct salida_analisis**), se crea dicha sesión, se registra en la base de datos por medio de las función *Insertar_Paquete* definida en el MMBD y se empieza a cuantificar el tráfico de audio.

- Para el mensaje de RELEASE COMPLETE, se cierran las sesiones y se eliminan los hijos con la función *Eliminar_Lista_y_Matar* codificada en MMRD y se registra en la Base de Datos la hora de finalización de la llamada.
- Cuando son mensajes H.245 pertenecientes a una determinada sesión, se capturan los puertos por los cuales va a fluir el tráfico de audio y se genera el filtro con la función *Crear_Filtro*.
- Cuando es un paquete de audio, se registra en la Base de Datos por medio función *Insertar_Paquete* definida en el MMBD.
- Los paquetes distintos a los anteriores son desechados.

Los paquetes registrados en la Base de Datos, pueden ser visualizados por un navegador Web. Dicha implementación es conocida como el Programa Cliente PC-Voz (PC). La estructura modular definida en la Fase de Diseño tiene una correlación a nivel de implementación esquematizada en la tabla 13.

Módulo diseño	Implementación
PC-PC-VOZ	Pcpc_voz.html
REPORTE	Principal.php
DETALLE	Detalle.php

Tabla 13: Equivalencia diseño codificación del programa Cliente PC-Voz

V.5 Fase de Prueba

Como se dijo anteriormente, se realizaron distintos tipos de pruebas, a continuación se muestran los resultados:

- Prueba de unidad: Está prueba se realizó de dos maneras distintas, la primera fue ejecutando el programa y revisando visualmente que el conjunto de salidas en el

dominio de las entradas fueran correctas. Por ejemplo, que la información insertada en la Base de Datos correspondieran con la realidad.

Para garantizar que el sistema PC-Voz capturara y cuantificara correctamente los paquetes que iban pasando por la red, se usó el reconocido programa *tcpdump* (http://www.tcpdump.org). De esta manera se revisó que el proceso se ejecutaba correctamente, garantizando así que el flujo de datos entre los módulos era correcto.

La segunda forma de aplicar este tipo de prueba, se realizó cuando el primer método de prueba era fallida (no se encontraba ningún error). El procedimiento consistió en utilizar el debugger para seguir línea a línea el valor de las variables y los bloques lógicos de control que se ejecutaban.

A continuación se responde el cuestionario de Meyer sobre este tipo de pruebas (1988, c.p. Pressman, 1998):

1. ¿Es igual el número de parámetros de entrada al número de argumentos? Sí.
2. ¿Coinciden los atributos de los parámetros y los argumentos? Sí.
3. ¿Coinciden los sistemas de unidades de los parámetros y de los argumentos? Sí.
4. ¿Es igual el número de argumentos transmitidos a los módulos de llamada que el número de parámetros? Sí.
5. ¿Son iguales los atributos de los argumentos transmitidos a los módulos de llamada y los atributos de los parámetros? Sí.
6. ¿Son iguales los sistemas de unidades de los argumentos transmitidos a los módulos de llamada y de los parámetros? Sí.
7. ¿Son correctos el número de los atributos y el orden de los argumentos de las funciones incorporadas? Sí.
8. ¿Existen referencias a parámetros que no estén asociados con el punto de entrada actual? No.
9. ¿Entran sólo argumentos alterados? No

10. ¿Son consistentes las definiciones de variables globales entre los módulos? No hay variables globales entre módulos.
11. ¿Se pasan las restricciones como argumentos? Sí.

Para el módulo KPC-Voz, se debe hacer el siguiente cuestionario ya que realiza E/S:

1. ¿Son correctos los atributos de los archivos? Sí.
2. ¿Son correctas las sentencias abrir/cerrar? Sí.
3. ¿Coinciden las especificaciones de formato con las sentencias de E/S? Sí.
4. ¿Coincide el tamaño de buffer con el tamaño del registro? Sí.
5. ¿Se abren los archivos antes de usarlos? Sí.
6. ¿Se tienen en cuenta las condiciones de fin de archivo? Sí.
7. ¿Se manejan los errores de E/S? No
8. ¿Hay algún error textual en la información de salida? No.

- Prueba de integración incremental descendente: En este tipo de pruebas, primero se prueba el programa principal (KPC-Voz(0)) y posteriormente se van verificando los módulos subordinados. No fue posible probar el módulo principal en su totalidad de manera aislada. Éste es dependiente del conjunto funciones que están en los módulos subordinados. Esta característica no se presenta en la gran mayoría de los demás módulos.

Aquí también se utilizaron los servicios de tcpdump para corroborar las capturas y de la herramienta de análisis de redes *Analyzer* de la *Universidad Politécnica de Torino* (http://analyzer.polito.it) para confirmar por medio de números hexadecimales los mensajes Q.931 de Señalización de Llamada.

El método que se utilizaba para ir probando cada módulo era por medio del archivo *prueba.c* que llamaba a dicho módulo y mandaba a ejecutar todas sus funciones y flujos lógicos de control. Entonces se revisaban visualmente las salidas y

posteriormente se utilizaba el debugger para seguir línea a línea el valor de las variables y los bloques lógicos de control que se ejecutaban.

- Prueba de recuperación: Esta prueba fue la principal manera de verificar el correcto funcionamiento de MoSAc. El procedimiento consiste en terminar la ejecución de los procesos KPC-Voz(0), MoSAc o MoSAcC de forma manual. Bajo este estado se utilizaba un programa visor de procesos para verificar si el sistema volvía a una fase de ejecución estable.

Se utilizó la técnica de alteración de memoria por medio del debugger para determinar cuál era el comportamiento del sistema. Este procedimiento consiste en asignar valores extraños a las variables de apuntador. Se seleccionaron datos nulos o que produjeran overflow. El resultado de este procedimiento fue la codificación de condiciones lógicas que evaluaran estas situaciones.

CAPÍTULO VI Conclusiones y Recomendaciones

VI.1 Conclusiones

En esta investigación, se hizo un estudio del tema de Voz sobre IP. Se trabajó con el protocolo de la ITU-T H.323 y se entendieron varias características del mismo. Se obtuvieron conocimientos de sus posibles aplicaciones en el mundo de hoy, sus ventajas y desventajas.

H.323 es un estándar que a pesar de ser publicado por la ITU-T, a juicio del tesista, es complejo y difuso. La gran cantidad de protocolos que éste abarca, fueron el causante del acto de recurrir a una gran gama de especificaciones. Esta característica puede ser atribuida a que los textos por sí solos son estándares genéricos. Los mismos fueron desarrollados para ser incorporados como parte de otros modelos. H.323 es la muestra de esto.

El principal inconveniente al momento del desarrollo de PC-Voz fue la gran cantidad de información debido al extenso conjunto de protocolos que maneja H.323. El punto clave para haber culminado la codificación exitosamente fue el uso de Internet como medio de investigación. Todos los estándares se consiguieron por este medio. Los ejemplo más claros fueron: Los RFCs de la IETF, las publicaciones de la ACM y las recomendaciones de la ITU-T, entre otros.

La utilización de software de código abierto y la distribución por Internet le brinda a este proyecto un carácter universal. La persona o empresa de cualquier parte del Mundo que quiera modificar el código para ajustarlo a sus necesidades lo puede hacer sin restricciones. Se propone que sea la comunidad la que pula el software.

Se espera, que en un futuro cercano, las redes de VoIP se encuentren frecuentemente en el País. El propósito fundamental de este Trabajo Especial de Grado, fue la creación

de una herramienta de cuantificación de tráfico de VoIP. Es factible una investigación y una profundización en el protocolo y en el sistema PC-Voz, para convertirla así, en una utilidad más poderosa que haga otras cosas.

Uno de los usos de esta aplicación, es servir de herramienta de verificación para desarrolladores de software que necesiten VoIP como medio de transmisión. En la actualidad hay varias compañías que están integrando VoIP como solución de comunicación en sus programas. Ejemplo de esto son los juegos en red para computadora, existen algunos que usan este protocolo para la comunicación entre los usuarios.

La estructura modular de PC-Voz fue diseñada y programada de tal manera, que es posible cuantificar otros protocolos de VoIP. El procedimiento consiste en codificar un módulo sustitutivo de MAnaP que identifique los factores del protocolo en cuestión y que respete las entradas y salidas hacia los demás módulos. Esto hace que este sistema sea muy versátil. Incluso, si se desea registrar los datos en una Base de Datos que no sea MySQL, se puede hacer lo mismo con el MMBD.

VI.2 Recomendaciones

- Este proyecto desarrolló una herramienta muy útil para la cuantificación de paquetes de audio. Es posible extender su funcionalidad profundizando más en los protocolos e incluso cuantificar otros protocolos de VoIP.

- Dado que se desarrolló un sistema experimental, no se verificó que el mismo pudiera ejecutarse en todas las distribuciones de Linux. Las versiones donde se hicieron las pruebas fueron Mandrake Linux 8.1 y Caldera OpenLinux Server 3.1. Se presume que el sistema puede ejecutarse en todos los paquetes Linux que cumplan con los requisitos y que se pueda portar la aplicación a otros tipos de sistemas operativos de esta familia sin muchas variaciones en el código fuente. Algunos sistemas objetivos serían FreeBSD, OpenBSD, NetBSD, Unix, Solaris, entre otros.

- Como se mencionó con anterioridad, el uso de Internet fue crucial para el desarrollo de este Trabajo Especial de Grado. Se recomienda a las instituciones docentes, que proporcionen a su comunidad acceso irrestricto a este medio, en conjunto de sólidas políticas de seguridad. Esto es de gran importancia en los centros educativos que ofrecen cátedras en el área de tecnología.

Referencias

1. Busby R., Kolman B. y Ross S. (1997). *Estructuras de Matemáticas Discretas para la Computación.* (3° ed.). Naucalpan de Juárez, México: Prentice Hall.

2. Collins, D. (2001). *Carrier Grade Voice over IP.* Nueva York, Estados Unidos de América: McGraw-Hill.

3. Detron HB y MySQL Finland AB. (2001). *MySQL Reference Manual.* Finlandia.

4. Fenzi K. y Wreski D. *Linux Security HOWTO versión 1.1.1.* 2000, 17 de Marzo

5. Huidobro, J. M. *H.323. Multimedia sobre redes IP*, [en línea]. Madrid, España. Disponible en: http://www.iies.es/teleco/publicac/publbit/bit109/quees.htm#JMPulido [2002, 4 de Marzo].

6. International Telecommunication Union – Telecommunication Standardization Sector. (1997). *Recommendation H.323: Packet-Based Multimedia Communications Systems, Version 2.* Geneva.

7. J.P. Curtis, J.G. Cleary, A.J. McGregor y M.W. Pearson *Analysis of Voice Over IP Traffic*, [en línea] Waikato, Nueva Zelanda: Universidad de Nueva Zelanda. Disponible en: http://wand.cs.waikato.ac.nz/wand/publications/jamie_420/final [2002, 2 de Marzo].

8. Magenta Tecnología de Redes. *Estudio del Ahorro de Telefonía IP para Corporaciones*, [en línea]. Chile. Disponible en: http://www.magenta.cl/temas/tendencias/ahorro1.asp [2002, 10 de Septiembre].

9. Pressman, R. (1998). *Ingeniería del Software. Un enfoque práctico.* (4° ed.). Madrid, España: McGraw-Hill/Interamericana.

10. Universidad Nacional Autónoma de México (UNAM). *Panoramas de los diferentes Modelos de Ciclos de Vida de los Sietemas.* [en línea]. México. Disponible en: http://www.mcc.unam.mx/~cursos/Objetos/clases3_4.html [2002, 2 de Febrero].

Bibliografía

- Busby R., Kolman B. y Ross S. (1997). *Estructuras de Matemáticas Discretas para la Computación.* (3° ed.). Naucalpan de Juárez, México: Prentice Hall.

- Chen, T. (2001). Increasing the Observability of Internet Behavior. *Communications of the ACM. 44* (1), 93-98.

- Collins, D. (2001). *Carrier Grade Voice over IP*. Nueva York, Estados Unidos de América: McGraw-Hill.

- Detron HB y MySQL Finland AB. (2001). *MySQL Reference Manual.* Finlandia.

- International Telecommunication Union – Telecommunication Standardization Sector. (1987). *Packet-Based Multimedia Communications Systems, Version 2.* Geneva.

- International Telecommunication Union – Telecommunication Standardization Sector. (1993). *Recommendation Q.931: Digital Subscriber Signalling System No. 1 (Dss 1) – Isdn User-Network Interface Layer 3 Specification For Basic Call Control.* Helsinki.

- International Telecommunication Union – Telecommunication Standardization Sector. (1997). *Recommendation H.323: Packet-Based Multimedia Communications Systems, Version 2.* Geneva.

- International Telecommunication Union – Telecommunication Standardization Sector. (1997). *Recommendation H.225.0: Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems, Version 2.* Geneva.

- International Telecommunication Union – Telecommunication Standardization Sector. (1998). *Recommendation H.245: Control Protocol For Multimedia Communication.* Helsinki

- Pressman, R. (1998). *Ingeniería del Software. Un enfoque práctico.* (4° ed.). Madrid, España: McGraw-Hill/Interamericana.

- Tanenbaum, A. (1997). *Redes de Computadoras.* (3° ed.). Naucalpan de Juárez, México: Pearson.

- van der Merwe, J., Cáceres, R., Chu, Y. y Sreenan C. (2000). mmdump: A Tool for Monitoring Internet Multimedia Traffic. *ACM Computer Communication Review. 30* (5), 48-59.

- Varshney, U., Snow, U., McGivern, M. y Howard, C. (2002). Voice Over IP. *Communications of the ACM. 45* (1), 89-96.

- http://www.packetizer.com
- http://www.nlanr.net/NLANRPackets/v1.4/pam2000_voip.html
- http://www.protocols.com
- http://www.iies.es/teleco/publicac/publbit/bit109/quees.htm#JMPulido
- http://www.freesoft.org/CIE/Topics/126.htm
- http://www.commweb.com/article/COM20001127S0008
- http://www.robertgraham.com/pubs/sniffing-faq.html
- http://www.cis.ohio-state.edu/~jain/cis788-99/h323
- http://www3.uji.es/~mmarques/f47/apun/apun.html
- http://www.h323forum.org/why_h323.asp
- http://standards.pictel.com/reference/9706_itca_h323
- http://www.ietf.org/proceedings/98aug/slides/hmibs-bof-kajos-slides-98aug/index.htm
- http://www.cisco.com/global/ES/solutions/sp/techsols/ldvoip_benefits.shtml
- http://www.ctsoluciones.com/VoIP.htm
- http://bochica.udea.edu.co/~amarin/comunica/final/sem-00-1/voz_ip/voip.html

Glosario de Términos

Término	Definición
Algoritmo	Conjunto de reglas claramente definidas para la resolución de una determinada clase de problemas. La escritura de un programa es sencillamente la elaboración de un algoritmo adecuado para la resolución del problema planteado.
Ancho de Banda	Expresa la cantidad de datos que pueden ser transmitidos en determinado lapso de tiempo.
Aplicación	Véase software
Arpanet	Advanced Research Proyects Agency Network – Red de la Agencia de Proyectos de Investigación Avanzada. Red de comunicación desarrollada por ARPA a fines de la década de los 60. Red militar Norteamericana a través de lineas telefónicas de la que posteriormente derivó Internet.
ATM	Asynchronous Transfer Mode. -Transferencia en modo Asíncrona. ATM es una tecnología de conmutación de red que utiliza celdas de 53 bytes, útil tanto para LAN como para WAN, que soporta voz, vídeo y datos en tiempo real y sobre la misma infraestructura. Utiliza conmutadores que permiten establecer un circuito lógico entre terminales, fácilmente escalable en ancho de banda y garantiza una cierta calidad de servicio (QoS) para la transmisión. Sin embargo, a diferencia de los conmutadores telefónicos, que dedican un circuito dedicado entre terminales, el ancho de banda no utilizado en los circuitos lógicos ATM se puede aprovechar para otros usos.

Base de Datos	Es un conjunto de datos entre los que existe una correlación y que se almacenan con criterios independientes con respecto a los programas que los utilizan. Esta agrupación de datos se realiza teniendo en mente su fácil acceso, recuperación y actualización.
Bridge	Puente. Equipo que provee interconexión entre dos redes utilizando la misma estructura de direccionamiento. Un puente filtra paquetes que permanecen en una LAN y envía paquetes que pertenecen a otras LANs. Contrasta con gateway y router.
Calidad de Servicio	Capacidad de un elemento de red (bien una aplicación, un servidor, un router, un conmutador, etc.) de asegurar que su tráfico y los requisitos del servicio previamente establecidos puedan ser satisfechos. Conjunto de tecnologías que permiten a los administradores de red manejar los efectos de la congestión del tráfico usando óptimamente los diferentes recursos de la red
Circuito Virtual	Circuito o conexión que existe lógicamente en vez de uno físicamente. Se establece solo por la duración de la llamada, y puede compartir un circuito físico con otros circuitos virtuales.
Códec	Algoritmos de Compresión/Descompresión. Se utilizan para reducir el tamaño de los datos multimedia, tanto audio como video. Compactan (codifican) un flujo de datos multimedia cuando se envía y lo restituyen (decodifican) cuando se recibe.
Codificar	Planear el procedimiento para llevar a cabo un trabajo. Algunas veces se aplica especialmente a aquella parte del trabajo total que sigue al análisis del problema y que precede a la escritura de los postulados en lenguaje de procedimiento o en instrucciones codificadas en lenguaje de máquina durante la cual se preparan los diagramas de flujo

Código	Es un conjunto de símbolos y reglas que sirven para representar datos de forma que puedan ser reconocidos por una computadora.
Código libre	Software que no es necesario pagarlo para usarlo, distribuirlo o modificarlo. Sin embargo, este tipo de código está amparado por derechos de autor que garantizan al creador su mérito.
Compilador	Es un programa que se utiliza para traducir el texto de otro programa escrito en un lenguaje de programación de alto nivel a código máquina (compilar).
Compilar	Es el método empleado para convertir un programa escrito en un lenguaje de alto nivel a lenguaje máquina, de forma que pueda ser comprendido por la computadora
Concurrencia	N procesos son concurrentes cuando se ejecutan (acción por acción) en el mismo intervalo de tiempo; es decir, para ejecutarlos concurrentemente se requiere tan solo "repartir" el procesador entre ellos a una velocidad tal que, por unidad o intervalo de tiempo, todos reciban su atención (aunque sea sólo parcial).
Cuantificar	Contar, medir.
Cuello de botella	Estancamiento o retraso en la entrega de paquetes en algún segmento o equipo de la red.
Datos	Cualquier información que pueda ser usada para cálculo, comparación u otro procesamiento que requiera ser recordada para un uso futuro. Algunas veces, se usa para referirse a registros u otra información involucrada en un programa a diferencia del programa mismo.
Debug	Véase depuración
Depuración	Conjunto de operaciones de búsqueda y corrección de errores en un programa.
Ethernet	Estándar de facto, desarrollado primero por Xerox. Una red de área local Ethernet utiliza cables coaxiales RG-11, pero puede utilizar

	cable coaxial delgado o cable de par trenzado. Ethernet tiene una topología lineal o de bus y sirve como la base para el estándar 802.3 del IEEE.
Frame Relay	Es un protocolo estándar para interconectar LANs. Proporciona un método rápido y eficiente para transmitir información desde dispositivos de usuario a bridges y routers. Se utiliza el ancho de banda disponible sólo cuando se necesita. Para transmitir la información se divide en paquetes, este método de transmisión resulta eficiente al transmitir comunicaciones de voz, con un adecuado control de la red.
Gatekeeper	Un componente del estándar ITU H.323. Es la unidad central de control que gestiona las prestaciones en una red de Voz o Fax sobre IP, o de aplicaciones multimedia y de videoconferencia. Los Gatekeepers proporcionan la inteligencia de red, incluyendo servicios de resolución de direcciones, autorización, autenticación, registro de los detalles de las llamadas para tarificar y comunicación con el sistema de gestión de la red. También monitorizan la red para permitir su gestión en tiempo real, el balanceo de carga y el control del ancho de banda utilizado. Elemento básico a considerar a la hora de introducir servicios suplementarios.
Gateway	En general se trata de una pasarela entre dos redes. Técnicamente se trata de un dispositivo repetidor electrónico que intercepta y adecua señales eléctricas de una red a otra. En Telefonía IP se entiende que estamos hablando de un dispositivo que actúa de pasarela entre la red telefónica y una red IP. Es capaz de convertir las llamadas de voz y fax, en tiempo real, en paquetes IP con destino a una red IP, por ejemplo Internet. Originalmente sólo trataban llamadas de voz, realizando la

	compresión/descompresión, paquetización, enrutado de la llamada y el control de la señalización. Hoy en día muchos son capaces de manejar fax e incluir interfaces con controladores externos, como gatekeepers, o sistemas de facturación.
H.323	H.323 es la recomendación global (incluye referencias a otros estándares, como H.225 y H.245) de la Unión Internacional de Telecomunicaciones (ITU) que fija los estándares para las comunicaciones multimedia sobre redes basadas en paquetes que no proporcionan una Calidad de Servicio (QoS, Quality of Service) garantizada. Define las diferentes entidades que hacen posible estas comunicaciones multimedia: terminales, gateways, unidades de conferencia multipunto (MCU) y gatekeepers, así como sus interacciones.
Hardware	Es el término que indica todas las partes físicas, eléctricas y mecánicas de una computadora. Significa literalmente "partes duras" y se emplea en contraposición al término software, que significa "partes blandas", es decir, los programas de una computadora.
Host	Anfitrión. Computador conectado a Internet. Computador en general.
HTML	HyperText Markup Languaje - Lenguaje de Marcación de Hipertexto. Lenguaje utilizado para la creación de documentos de hipertexto e hipermedia. Es el estándar utilizado en el www.
HTTP	Hypertext Transport Protocol - Protocolo de Tranferencia de Hypertexto. Es este el protocolo de Internet usado en la Web para servir y

	visualizar documentos en hipertexto. HTTP es un protocolo con la ligereza y velocidad necesaria para distribuir y manejar sistemas de información hipermedia. Es un protocolo genérico orientado a objetos, que puede ser usado para muchas tareas. Una característica de HTTP es la independencia en la visualización y representación de los datos, permitiendo a los sistemas ser construidos independientemente del desarrollo de nuevos avances en la representación de los datos HTTP ha sido usado por los servidores World Wide Web desde su inicio en 1.990.
IEEE	IEEE Institute of Electrical and Electronics Engineers - Instituto de Ingenieros Eléctricos y Electrónicos. Sociedad internacional de profesionales que emiten sus propios estándares y que es miembro de la organización ANSI. Se le atribuyen varios de los protocolos de red más utilizados en la actualidad.
IETF	Internet Engineering Task Force - Grupo de Tareas de Ingeniería de Internet. Asociación de técnicos que organizan las tareas de ingeniería principalmente de telecomunicaciones en Internet.
Integridad	Mantener la integridad de una base de datos es asegurarse de que los datos que contiene son correctos, evitando datos inconsistentes o erróneos de cualquier otro tipo. Un control de integridad o restricciones es aquel que nos permite definir con precisión el rango de valores validos para un elemento

	y/o las operaciones que serán consideraciones validas en la relación de tales elementos.
Interfaz	Se denomina así a todo aquel medio físico que conecta un dispositivo periférico con la computadora; también se le conoce así a todo el software que comunica al usuario con la misma. Medio de enlace entre dos entidades
Internet	La llamada "red de redes" creada de la unión de muchas redes TCP/IP a nivel internacional y cuyos antecedentes están en ARPANet.
IP	Internet Protocol - Protocolo de Internet. Protocolo que provee las funciones básicas de direccionamiento en Internet y en cualquier red TCP/IP.
ISDN	Integrated Services Digital Network - Red Digital de Servicios Integrados. Estándar digital para las compañías telefónicas el cual lleva cualquier tipo de servicios de voz, datos, fax, video digital y servicios especiales de tráfico. El servicio incluye una velocidad básica (Basic rate) de 144 Kbps y una velocidad primaria (Primary rate) de 1.544 Mbps.
ITU-T	International Telecommunications Union - Telecommunication - Unión Internacional de Telecomunicaciones, sector Telecomunicaciones. Antes conocida como CCITT (Comite Consultatif Internationale de Telegraphie et Telephonie). Agencia de la Organización de las Naciones Unidas que trata lo referente a telecomunicaciones: crea estándares, reparte frecuencias para varios servicios, etc. El grupo

	ITU-T recomienda estándares para telecomunicaciones y está en Génova (Suiza). También se encarga de elaborar recomendaciones sobre códecs (compresión/descompresión de audio) y módems.
Javascript	Lenguaje de programación. Formalmente llamado LiveScript. Este lenguaje fue desarrollado por Netscape. Concebido después de JAVA; su principal diferencia radica en que el programa se halla incrustado en un archivo HTML, en lugar de ser un ejecutable que se carga cuando se llama a la página Web.
Lenguaje C	C es un lenguaje maduro de propósitos generales. Su definición aparece en 1978 en el apéndice ``C Reference Manual" del libro The C Programming Language, de Brian W. Kernighan y Dennis M. Ritchie (Englewood Cliffs, Nueva Jersey, Prentice-Hall 1978), pero el estándar recomendable más reciente apareció en junio de 1983, en el documento de los Laboratorios Bell titulado The C Programming Language-Reference Manual, escrito por Dennis M. Ritchie
Lenguaje de programación	Conjunto de caracteres y reglas bien definidas que se utiliza para escribir programas inteligibles por una computadora o un sistema informático en general.
Librería	Es un conjunto de ficheros de datos o fragmentos de programas de funcionamiento totalmente independiente que se pueden utilizar dentro de otros programas formando parte de ellos.
Linux	Sistema operativo gratuito de código libre. El mismo puede venir de distintos desarrolladores, entre los más reconocidos se cuenta con Mandrake Linux, Caldera Linux, Red Hat Linux, Suse Linux, entre otros.
Llamada	Petición de conexión o la conexión resultado de tal petición.
MEGACO	Media Gateway Control.

	MEGACO es un protocolo de VoIP, combinación de los protocolos MGCP e IPDC. Es más sencillo que H.323.
Mensaje	Transmisión completa; a menudo utilizado como sinónimo de paquete, pero un mensaje está comúnmente compuesto de varios paquetes.
Módulo	Término referido a componentes de un programa o sistema que se pueden identificar por separado y a los que es posible dirigirse también separadamente. Es equivalente al término segmento
Multicast	Técnica de transmisión de datos a través de Internet en la que se envían paquetes desde un punto a varios simultáneamente.
Multimedia	Tratamiento informático avanzado de las tecnologías más recientes de sonido e imagen que engloba e integra funciones como la animación gráfica, la manipulación y digitalización de imágenes y sonido.
Multiplexar	Proceso de dividir una señal compuesta en varios canales.
Nodo	Cualquier dispositivo físico, tal como una computadora o impresora a la que se puede acceder en la red. Por definición punto donde convergen más de dos líneas. A veces se refiere a una única máquina en Internet. Normalmente se refiere a un punto de confluencia en una red. Punto de interconexión a una RED.
Paquete	Grupo de bits y que es una cantidad mínima de datos que se transmite en una red o entre dispositivos. Tiene una estructura y longitud distinta según el protocolo al que pertenezca. También llamado trama.
PBX	Private Branch eXchange - Rama Privada de Intercambio. Centralita, central privada. Un sistema telefónico utilizado en

	compañías y organizaciones privadas, por tanto, para manejar llamadas externas e internas. La ventaja es que la compañía no necesita una línea telefónica para cada uno de sus teléfonos. Además, las llamadas internas no salen al exterior y por lo tanto no son facturadas.
PHP	Personal Home Page Tools - Herramientas Personales de Casa. Lenguaje de programación desarrollado originalmente en el año 1994 por Rasmus Lerdorf como un programa escrito en C que permitía la interpretación de un número limitado de comandos. Actualmente es uno de los lenguajes más difundidos para crear páginas web
Política	Conjunto de reglas que garanticen el correcto funcionamiento y privacidad de un sistema de cómputo.
Proceso	Es una secuencia temporal de ejecuciones de instrucciones que corresponde a la ejecución de un programa secuencial. En general se define como la actividad de un sistema informático que está formada por una secuencia de pasos o etapas.
Programar	Véase codificar
Protocolo	Medio utilizado para controlar la comunicación ordenada de información entre estaciones en un enlace de datos o una red de comunicaciones de datos. Un conjunto de formatos y procedimientos acordados que gobiernan la transferencia de información entre dispositivos.
PSTN	Public Switched Telephone Network - Red Pública Telefónica Conmutada. Red telefónica que es compartida entre muchos usuarios, y cualquier usuario puede establecer comunicación con cualquier

	otro usuario, incluyendo servicios de larga distancia.
Recurso	Es la actividad o capacidad de un componente físico de la computadora. También se denomina así a la actividad de un programa que puede ser utilizada por varios programas de modo concurrente o simultáneo.
Red	Sistemas de cómputo, terminales e instalaciones de comunicaciones de datos interconectados; pueden tener cualquier número de enlaces y nodos.
Red de área local	Es un sistema de comunicación entre computadoras situadas en lugares próximos, como la misma oficina o el mismo edificio. También denominada con las siglas inglesas LAN (Local Area Network), gracias a este tipo de redes se pueden compartir diversos periféricos o manejar las mismas informaciones. También se le llama simplemente red local.
RFC	Request For Comment - Petición de Comentarios. Serie de documentos iniciada en 1967 que describe el conjunto de protocolos de Internet. Los RFC son elaborados por la comunidad Internet.
Router	Un dispositivo físico, o a veces un programa corriendo en un ordenador, que reenvía paquetes de datos de una red LAN o WAN a otra. Basados en tablas o protocolos de enrutamiento, leen la dirección de red destino de cada paquete que les llega y deciden enviarlo por la ruta más adecuada (en base a la carga de tráfico, coste, velocidad u otros factores).
RTCP	Real-Time Transport Control Protocol - Protocolo de Control de Trasporte en Tiempo Real. Es el protocolo encargado de mantener la calidad en la transmisión

	de RTP.
RTP	Real-Time Transport Protocol - Protocolo de Trasporte en Tiempo Real. El protocolo estándar en Internet para el transporte de datos en tiempo real, incluyendo audio y video. Se utiliza prácticamente en todas las arquitecturas que hacen uso de VoIP, videoconferencia, multimedia bajo demanda y otras aplicaciones similares. Se trata de un protocolo ligero que soporta identificación del contenido, reconstrucción temporal de los datos enviados y también detecta la pérdida de paquetes de datos.
Servicio	Cualquier dispositivo en una red que puede ser conectada y accesada, tal como una impresora, módem o una computadora remota. Los usuarios de la red pueden ver los servicios disponibles en la red porque los nodos de la LAN que proveen estos servicios los anuncian vía multicast. Conjunto de hardware y software que puede ofrecer un sistema de cómputo para la realización de una tarea.
Servidor	Es un tipo de computadora que suministra espacio en disco, impresoras y otros servicios a otras computadoras conectados con él a través de una red.
SIP	Session Initiation Protocol - Protocolo de Inicio de Sesión. SIP es un protocolo de señalización para conferencia, telefonía, presencia, notificación de eventos y mensajería instantánea a través de Internet. Un estándar de la IETF (Internet Engineering Task Force) definido en la RFC 2543. SIP se utiliza para iniciar, manejar y terminar sesiones interactivas entre uno o más usuarios en Internet. Inspirado en los protocolos HTTP (web) y SMTP

	(email), proporciona escalabilidad, flexibilidad y facilita la creación de nuevos servicios.
Sistema Operativo	Conjunto de programas que forman parte del software de un sistema de cómputo y que tiene la misión de controlar todas las partes de dicho sistema.
SMTP	Simple Mail Transfer Protocol. Protocolo de Transferencia Simple de Correo. Es el protocolo usado para transportar el correo electrónico a través de Internet.
Sniffer	Pequeño programa que busca una cadena numérica o de caracteres en los paquetes que atraviesan un segmento de la red con objeto de conseguir alguna información. Programa que permite visualizar y posiblemente registrar los paquetes que pasan por un segmento de una red.
Software	Programa de computadora o conjunto de programas que se mantienen en un medio de almacenamiento y que se cargan a memoria de lectura/escritura para su ejecución.
SQL	Structured Query Language - Lenguaje de Petición Estructurada. Lenguaje para base de datos desarrollado en 1974 por Donald Chamberlin en los laboratorios de IBM. Actualmente es un estándar universal..
Tarea	Término que define un cometido específico y delimitado en un entorno de multiprogramación.
Tarjeta de red	Dispositivo de hardware que sirve para conectar una computadora a una red.
TCP	Transmission Control Protocol - Protocolo de Control de Transmisión.

	Protocolo de comunicación que permite comunicarse a los ordenadores a través de Internet. Asegura que un mensaje es enviado completo y de forma fiable.
TCP/IP	Transmission Control Protocol/Internet Protocol - Protocolo de Control de Transmisión/ Protocolo de Internet. Protocolo para el enrutamiento interredes y la entrega confiable de mensajes. IP es el protocolo de bajo nivel del conjunto de protocolos TCP/IP. IP provee los servicios de entrega de paquetes entre nodos. TCP es un protocolo orientado a conexión de circuito virtual confiable que corre sobre IP.
Tiempo de respuesta	Tiempo promedio en que tarda un servidor en ejecutar un servicio
Tiempo Real	Computadora en línea que genera salidas casi simultáneamente como recibe entradas.
Tipo de dato abstracto (TDA)	Librería escrita en algún lenguaje de programación que define una estructura de almacenamiento y las operaciones permitidas sobre la misma.
Topología	Arreglo lógico o físico de nodos o estaciones de una red.
Transmisión	Acción de enviar datos desde un lugar y recibirlos en otro diferente.
UDP	User Datagram Protocol - Protocolo de Datagramas de Usuario. El protocolo UDP es un protocolo de transporte muy simple. Se basa en datagramas: Cada operación de envío que efectúa la aplicación se convierte en un datagrama UDP. UDP no es un protocolo fiable: No se garantiza que los datagramas lleguen a su destino y puede producirse desorden en la llegada de

	datagramas
Unix	Es un sistema operativo moderno concebido inicialmente para computadoras de tamaño medio pero que se ha difundido también en computadoras personales. Permite al usuario aumentar las prestaciones de la computadora y está escrito en su mayor parte en lenguaje C.
VoIP	Voice over IP - Voz sobre IP. Término utilizado en la telefonía IP para un conjunto de facilidades para administrar la dotación de voz utilizando el Protocolo Internet (IP).
WAN	Wide Area Network - Red de Área Ancha. Una red de comunicaciones utilizada para conectar ordenadores y otros dispositivos a gran escala. Las conexiones pueden ser privadas o públicas.
Web	Site. Sitio en el World Wide Web. Conjunto de páginas Web que forman una unidad de presentación, como una revista o libro. Un sitio está formado por una colección de páginas Web.

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

DISEÑO Y DESARROLLO DE UN SISTEMA EXPERIMENTAL CUANTIFICADOR DE TRÁFICO DE VOZ SOBRE IP (VOIP).

(Apéndices y Anexos)

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar por el título de

INGENIERO EN INFORMÁTICA

REALIZADO POR Daniel Pons Alvarez

PROFESOR GUÍA Miguel Torrealba

FECHA Caracas, Noviembre de 2002

Índice General

Apéndices

Anexos

Anexo

APÉNDICE A: Continuación de Voz sobre IP

Resumen de: http://www.cisco.com/global/ES/solutions/sp/techsols/ldvoip_benefits.shtml
http://www.ctsoluciones.com/VoIP.htm
http://bochica.udea.edu.co/~amarin/comunica/final/sem-00-1/voz_ip/voip.html

Ventajas de VoIP

Las ventajas que ofrece la convergencia de redes han sido en un principio de índole económica. Tecnologías como VoIP hacen un uso mucho más eficiente del ancho de banda de las redes. Es notorio el ahorro en los costos de gestión y operación que se consigue por el hecho de utilizar una sola red para datos, voz y video. En la siguiente lista se detallan algunos beneficios de una red con VoIP:

- Integración sobre la Intranet de la voz como un servicio más de la red, tal como otros servicios informáticos.
- Las redes IP son la red estándar universal para la Internet, Intranets y Extranets, por lo tanto no es necesario hacer una inversión riesgosa.
- Estándares efectivos (H.323).
- Interoperabilidad de diversos proveedores. Cisco, Radcom, Intel, Nortel son sólo algunas de las empresas que soportan VoIP.
- Uso de las redes de datos existentes, no hay que hacer nuevas inversiones en cableado y reestructuración.
- Independencia de tecnologías de transporte, asegurando la inversión.
- Cada día, existen más aplicaciones que tienden a estar integradas con voz y video.
- Menores costos que tecnologías alternativas (voz sobre TDM, ATM, Frame Relay).
- Menores costos en llamadas en general, en especial las internacionales.

APÉNDICE B: Estudio del Ahorro de Telefonía IP para Corporaciones

Resumen de http://www.magenta.cl/temas/tendencias/ahorro1.asp

Introducción

Una de las características más destacadas de Internet es su capacidad para reducir fuertemente los costos de las comunicaciones al interior de una organización. Si bien esta propiedad fundamental, no ha sido suficientemente difundida en Latinoamérica, en los países donde Internet se encuentra en un estado de desarrollo más avanzado, dicho ahorro de costos ha sido aprovechado con entusiasmo por las corporaciones que han comenzado a emplear Internet como un medio de comunicación sustitutivo de las redes clásicas de voz y video. En este apéndice se muestra un cálculo financiero que determina el valor del minuto sobre redes IP y demuestra el ahorro que logra un usuario corporativo al adquirir sistemas que le permitan emplear Internet como soporte para el transporte de voz.

Parámetros De Evaluación Económica

Los parámetros empleados en la evaluación económica del costo de un minuto de voz IP son:

- Volumen de minutos mensuales generado por el usuario: El volumen de minutos mensuales generados por el usuario corresponde a la cantidad de minutos de tráfico telefónico que emite un usuario del sistema.
- Utilización línea de gateway: Corresponde al número de minutos cursados por cada línea de gateway. Debe tenerse presente que el número total de minutos cursado por línea de gateway está asociado al nivel de servicio y representa una elección de diseño de la calidad del sistema telefónico de la corporación. Para un uso eficiente de la inversión y un grado de servicio aceptable, se recomienda no cursar más de 13.000 minutos al mes (esto no significa que no se pueda cursar

más de 13.000 minutos por mes sino que para valores superiores al anterior, la probabilidad de bloqueo aumenta a niveles poco convenientes).

- Inversión por línea: Representa el costo de instalación de una línea de voz IP. Para el caso de la evaluación se considera que la inversión total es de US$ 2.500 por línea de gateway (incluye tarjeta, PC y software).
- Tasa de interés en depreciación del capital: Es la tasa con que se descuentan los flujos de depreciación de la inversión de capital (básicamente líneas de gateway). La estimación de costos ha sido realizada considerando un interés de 16% anual.
- Período de depreciación de la inversión: Horizonte definido para depreciar completamente la inversión en equipos. Se estima prudente considerar un período de depreciación de 5 años.

Configuraciones De Telefonía Internet De Interés Para Corporaciones

Conexiones teléfono a teléfono

Larga distancia intra-corporación

La conexión teléfono a teléfono al interior de una corporación requiere de 2 gateways, uno en cada punta del enlace, y una red IP (Internet o Intranet) como soporte de la comunicación entre los extremos remotos. Un esquema de este tipo de conexión a nivel nacional se muestra en la figura 1.

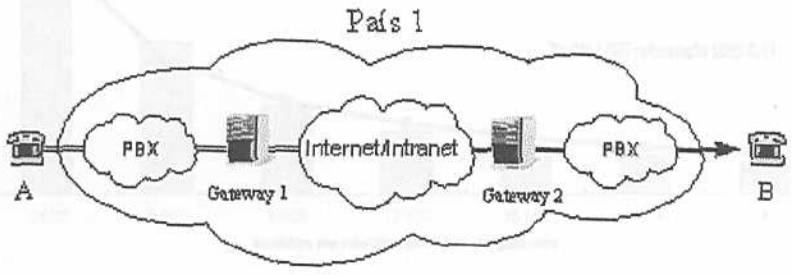

Figura 1: *Esquema de conexión nacional teléfono a teléfono intra-corporación*

Por otra parte, si la comunicación es internacional, el esquema queda dado por.

Figura 2: *Esquema de conexión internacional teléfono a teléfono intra-corporación*

Para apreciar la evolución del costo del minuto IP, es necesario realizar una sensibilización respecto al número de minutos cursados por cada línea de gateway. Se determinó el número de meses necesarios para recuperar completamente la inversión en líneas de gateways. Los gráficos para larga distancia nacional e internacional se muestran a continuación.

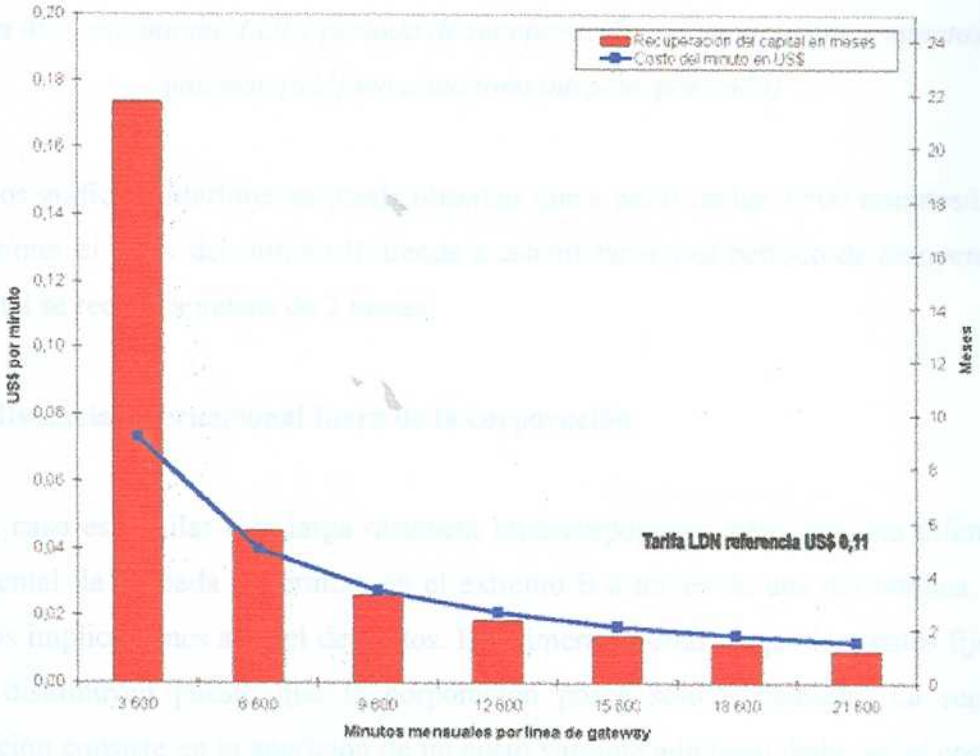

Figura 3: *Costo minuto LDN y período de recuperación del capital versus minutos de gateway (teléfono a teléfono intra-corporación)*

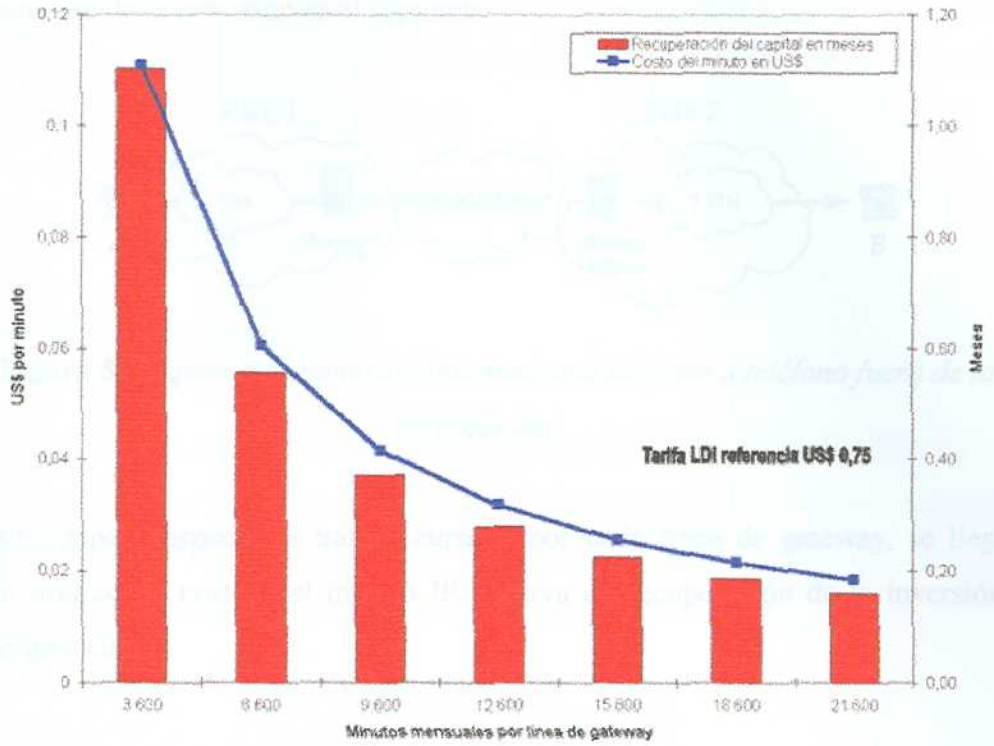

Figura 4: *Costo minuto LDI y período de recuperación del capital versus minutos de gateway (teléfono a teléfono intra-corporación)*

De los gráficos anteriores se puede observar que a partir de los 9.600 minutos/línea-gateway/mes el valor del minuto IP tiende a estabilizarse y el período de recuperación del capital se reduce a menos de 2 meses.

Larga distancia internacional fuera de la corporación

Este caso es similar a la larga distancia intra-corporación pero con una diferencia fundamental: la llamada se termina en el extremo B a través de una red pública. Esto tiene dos implicaciones a nivel de costos. La primera de ellas es que los costos fijos de capital disminuyen puesto que la corporación posee sólo 1 gateway. La segunda implicación consiste en la aparición de un costo variable adicional dado por el costo de terminación del minuto telefónico en el extremo B (denominado costo de corresponsalía).

El esquema de la conexión es el siguiente.

Figura 5: *Esquema de conexión internacional teléfono a teléfono fuera de la corporación.*

Sensibilizando respecto al tráfico cursado por cada línea de gateway, se llega al siguiente gráfico de costos del minuto IP y curva de recuperación de la inversión en líneas de gateways.

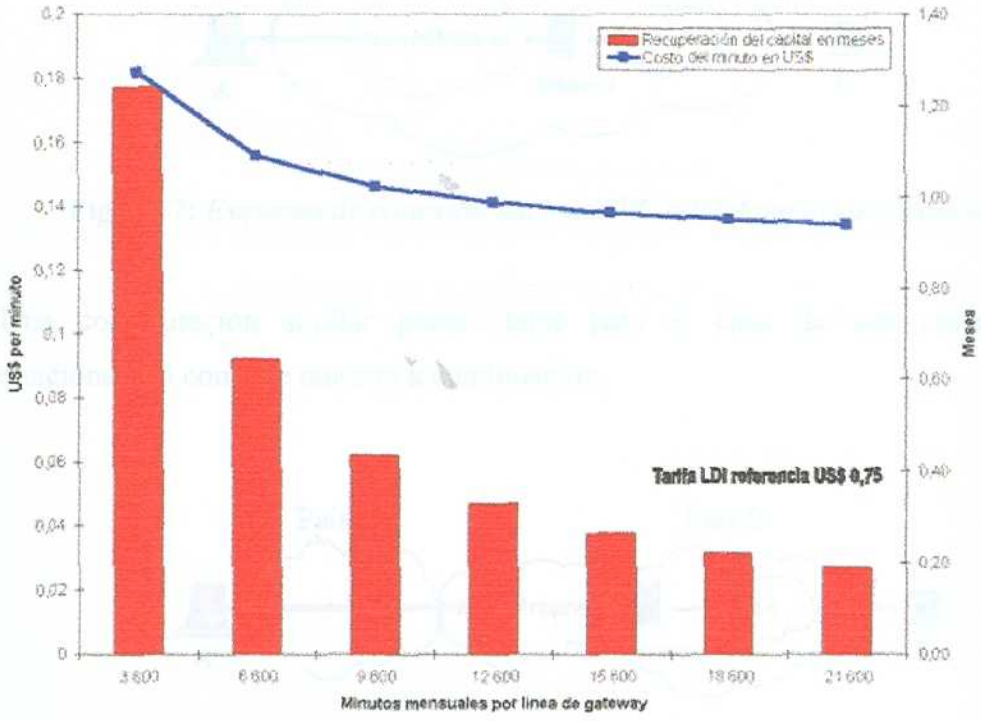

Figura 6: *Costo minuto LDI y período de recuperación del capital versus minutos de gateway (teléfono a teléfono fuera de la corporación)*

Del gráfico anterior se puede observar que a partir de los 9.600 minutos/línea-gateway/mes el valor del minuto IP tiende a estabilizarse y el período de recuperación del capital se reduce a una fracción de mes.

Conexiones PC a teléfono

Larga distancia intra-corporación

La conexión PC a teléfono al interior de una corporación requiere de 1 PC con tarjeta de sonido full dúplex y micrófono, 1 gateway en el extremo B del enlace, y una red IP (Internet o Intranet) como soporte de la comunicación entre los extremos remotos. Un esquema de este tipo a nivel nacional se muestra en la siguiente figura.

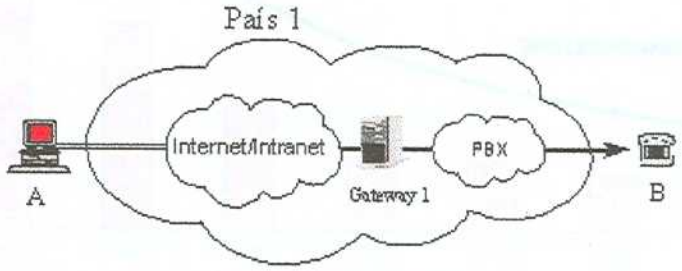

Figura 7: *Esquema de conexión nacional PC a teléfono intra-corporación*

Una configuración similar puede darse para el caso de una comunicación internacional, tal como se muestra a continuación.

Figura 8: *Esquema de conexión internacional PC a teléfono intra-corporación*

La sensibilización de la variable volumen de minutos cursados por línea de gateway permite apreciar su influencia en el costo del minuto IP y en el período de recuperación del capital.

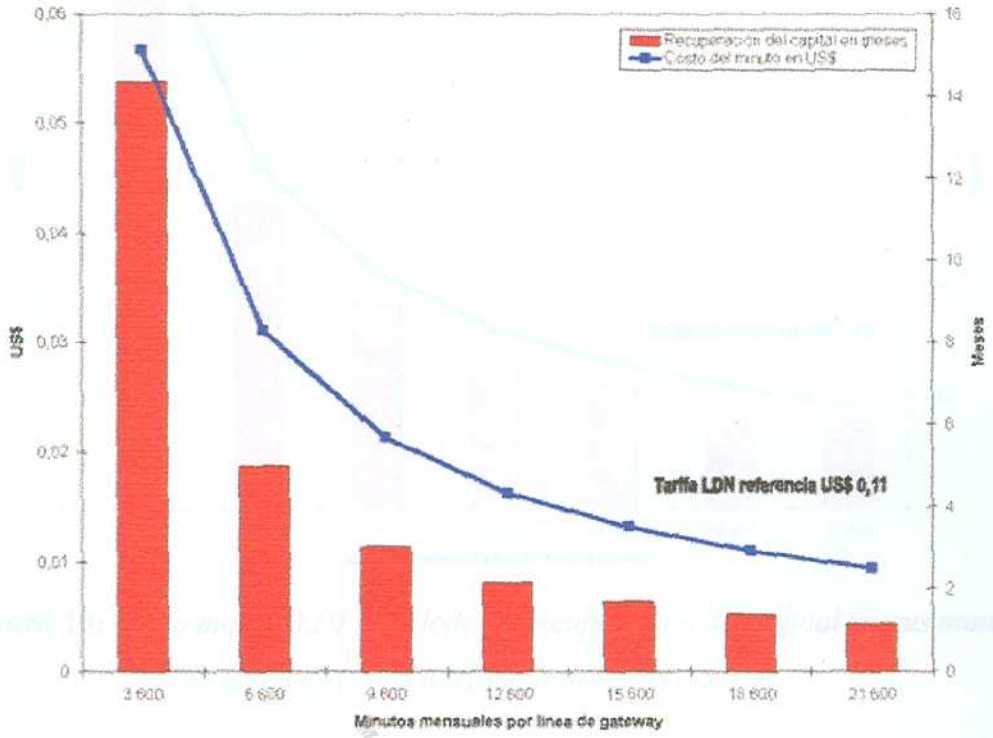

Figura 9: *Costo minuto LDN y período de recuperación del capital versus minutos de gateway (PC a teléfono intra-corporación)*

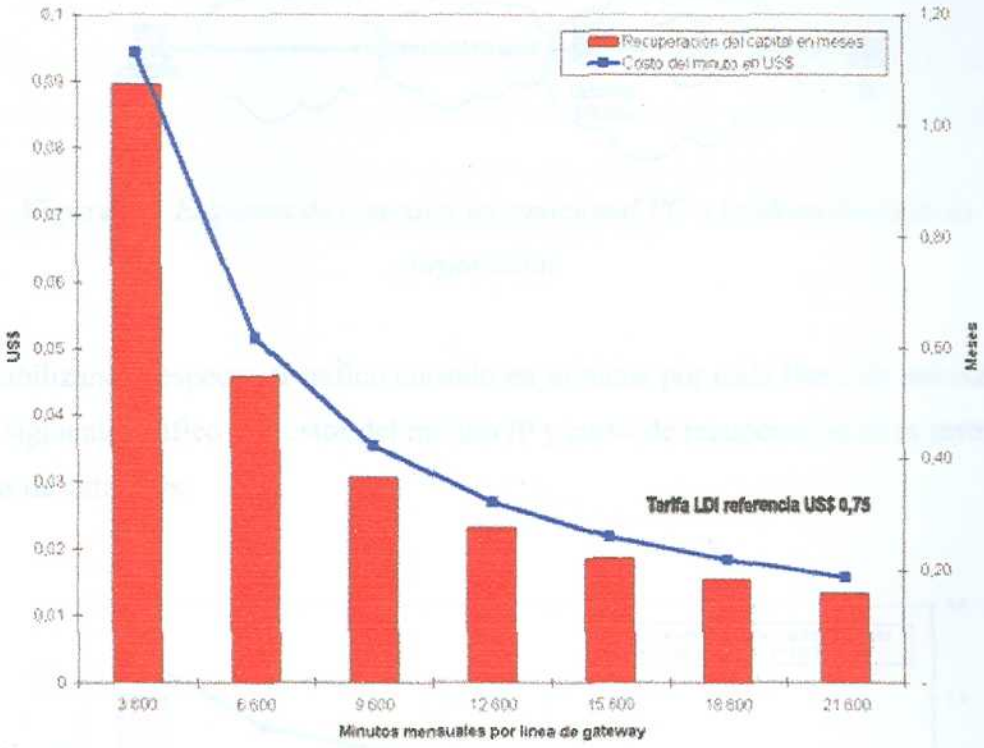

Figura 10: *Costo minuto LDI y período de recuperación del capital versus minutos de gateway (PC a teléfono intra-corporación)*

Larga distancia fuera de la corporación

La diferencia fundamental respecto al caso de larga distancia intra-corporación es que la llamada se termina en el extremo B haciendo uso de una red pública. Esto trae como consecuencia una variación en los costos fijos de capital, puesto que la corporación no posee ningún gateway en el enlace, y la aparición de un costo variable, dado por el gasto de corresponsalía para terminar la llamada a través de una red pública. Un esquema de esta conexión se muestra en la figura 11.

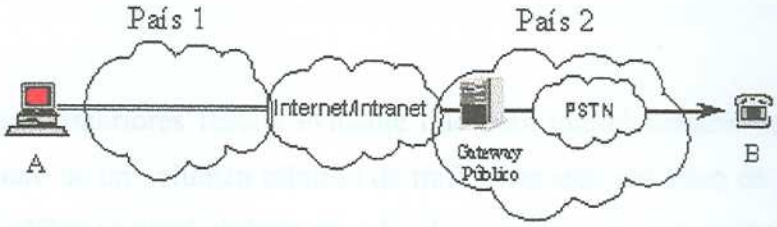

Figura 11: *Esquema de conexión internacional PC a teléfono fuera de la corporación*

Sensibilizando respecto al tráfico cursado en minutos por cada línea de gateway, se llega al siguiente gráfico de costos del minuto IP y curva de recuperación de la inversión en líneas de gateways.

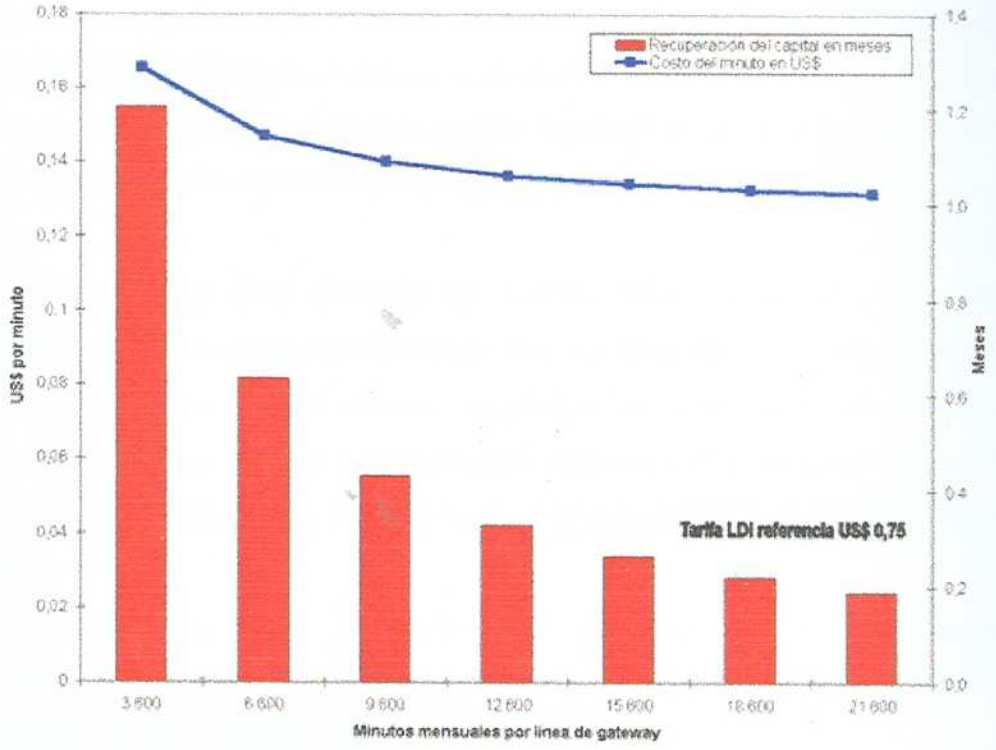

Figura 12: *Costo minuto LDI y período de recuperación del capital versus minutos de gateway (PC a teléfono fuera de la corporación).*

Conclusiones

De los gráficos anteriores resulta evidente que para que el sistema de voz IP sea rentable se requiere de un volumen mínimo de tráfico cursado por línea de gateway. De una inspección gráfica se puede inferir que el valor crítico en el sistema se acerca a los 10.000 minutos cursados al mes por cada línea de gateway.

APÉNDICE C: Permisos de archivos y directorios en Sistemas Operativos Linux
Resumen de (Fenzi y Wreski, 2000).

Los Sistemas Operativos de la familia Unix, tal como el caso de Linux, utilizan un esquema de seguridad de archivos y directorios del tipo Propietario/Permisos. Esta política se puede utilizar para mantener la seguridad de los archivos de PC-Voz de forma trivial. Es un esquema que sirve para proteger los archivos de usuarios no especializados y debe ser acompañado con estrategias de seguridad general.

El árbol de directorios

Una de las características del sistema de almacenamiento de información en sistemas Linux, es que se organiza en un único árbol de directorios. Cada soporte, disco, partición, disquete o CD tiene su propia organización lógica en un sistema de ficheros. Para poder usar uno de estos soportes se debe "montar" en un directorio existente. El contenido de la partición aparecerá como el contenido del directorio.

Un primer criterio para mantener un sistema seguro sería hacer una correcta distribución del espacio de almacenamiento. Esto limita el riesgo de que el deterioro de una partición afecte a todo el sistema. La pérdida se limitaría al contenido de esa partición. No hay unas normas generales aplicables; el uso al que vaya destinado el sistema y la experiencia son las bases de la decisión adecuada. Los expertos dan algunos consejos:

- Si el sistema va a dar servicio a múltiples usuarios que requieren almacenamiento para sus datos privados, sería conveniente que el directorio /home tuviera su propia partición.
- Si el equipo va a ser un servidor de correo, impresión, etc., el directorio /var o incluso /var/spool podrían tener su propia partición.

- Algunos directorios son necesarios en la partición raíz. Contienen datos que son necesarios durante el proceso de arranque del sistema. Son /dev/, /etc, /bin, /sbin, /lib, /boot.
- El directorio /usr/local contiene los programas compilados e instalados por el administrador. Resulta conveniente usar una partición propia para proteger estos programas personalizados de futuras actualizaciones del sistema. Este criterio también se puede aplicar al directorio /opt.

Permisos

Linux, como sistema multiusuario, asigna un propietario y un grupo a cada fichero (y directorio) y unos permisos al propietario, al grupo y al resto de los usuarios. Una forma de comprobar esta característica es usar el comando ls -la. Así aparece el tipo de fichero, el propietario, el grupo, los permisos e información adicional. Por supuesto, el sistema de ficheros tiene que admitir esta característica, como es el caso del sistema de ficheros *ext2* (Linux nativo). En los sistemas de ficheros pensados para entornos monousario, como *msdos* o *vfat*, no se cuenta con esta característica. Por lo tanto, tienden a ser inseguros y su uso no es aconsejable bajo Linux.

Puede que algunas aplicaciones no funcionen bien si algún fichero no tiene el permiso o el propietario correctos. En el caso de PC-Voz esto no ocurre; pero es una norma de seguridad asignar los permisos correspondientes.

Por ejemplo, *inn* (un servidor de noticias de Internet) no funcionará si el propietario de sus ficheros es otro usuario distinto a *news*. Todo esto está perfectamente documentado en cada uno de los programas, por lo que es conveniente la lectura de la documentación que aporta el programa y las páginas del manual.

Permisos de ficheros y directorios

Como se mencionaba con anterioridad, hay que asegurarse de que los ficheros del sistema y los de cada usuario sólo son accesibles por quienes tienen que hacerlo y de la forma en que deben. No sólo hay que proteger el sistema de ataques o miradas importunas, también hay que hacerlo de acciones accidentales. En caso específico de PC-Voz, es imprescindible proteger los archivos binarios, los logs y los datos de la Base de Datos.

En general, cualquier sistema de la familia Unix, divide el control de acceso a ficheros y directorios en tres elementos: propietario, grupo y otros. Tanto el propietario como el grupo son únicos para cada fichero o directorio. A un grupo pueden pertenecer múltiples usuarios. Otros hace referencia a los usuarios que ni son el propietario ni pertenecen al grupo.

Todas estas características se almacenan en el sistema de ficheros. Quedan registrados en un *i-nodo*. Este es un elemento que describe las características de un fichero en disco (salvo su nombre).

Resumen de los permisos Unix:

Propiedad: Qué usuario y grupo posee el control de los permisos del i-nodo. Se almacenan como dos valores numéricos, el uid (user id) y gid (group id).

Permisos: Bits individuales que definen el acceso a un fichero o directorio. Los permisos para directorio tienen un sentido diferente a los permisos para ficheros. A continuación se explican algunas diferencias.

- **Lectura (r)**:
  - **Fichero**: Poder acceder a los contenidos de un fichero

- **Directorio**: Poder leer un directorio, ver qué ficheros contiene
- **Escritura (w)**:
  - **Fichero**: Poder modificar o añadir contenido a un fichero
  - **Directorio**: Poder borrar o mover ficheros en un directorio
- **Ejecución(x)**:
  - **Fichero**: Poder ejecutar un programa binario o guion de shell
  - **Directorio**: Poder entrar en un directorio
- Estos permisos se pueden aplicar a:
  - **usuario (u)**: El propietario del fichero
  - **grupo (g)**: El grupo al que pertenece el fichero
  - **otros (o)**: El resto de los usuarios del sistema

- Existen otros bits de permisos concernientes a funciones de seguridad:

  - **Sticky bit:** El *sticky bit* tiene su significado propio cuando se aplica a directorios. Si el *sticky bit* está activo en un directorio, entonces un usuario sólo puede borrar ficheros que son de su propiedad o para los que tiene permiso explícito de escritura, incluso cuando tiene acceso de escritura al directorio. Esto está diseñado para directorios como /tmp, que tienen permiso de escritura global. El sticky bit aparece como **'t'** en los listados largos de directorios.

    drwxrwxrwt 19 root root 8192 Jun 24 14:40 tmp

  - **Attributo SUID: (Para Ficheros)**: Este bit describe permisos al identificador de usuario del fichero. Cuando el modo de acceso de ID de usuario está activo en los permisos del propietario, y ese fichero es ejecutable, los procesos que lo ejecutan obtienen acceso a los recursos del sistema basados en el usuario que crea el proceso (no el usuario que lo lanza). Por ejemplo, /usr/bin/passwd es un ejecutable propiedad de *root* y con el bit

SUID activo. Esto es debido a que este programa lo puede usar cualquier usuario para modificar su clave de acceso, que se almacena en:

-rw-r--r-- 1 root root 1265 Jun 22 17:35 /etc/passwd

Pero según los permisos que se observan en este fichero, sólo *root* puede escribir y modificar en él. Entonces sería imposible que un usuario pudiera cambiar su clave si no puede modificar este fichero. La solución para este problema consiste en activar el bit *SUID* para este fichero:

-r-s--x--x 1 root root 10704 Apr 14 23:21 /usr/bin/passwd

De forma que cuando se ejecute, el proceso generado por él es un proceso propiedad de *root* con todos los privilegios que ello acarrea.

Este es un gran riesgo para la seguridad del sistema, ya que en este tipo de programas se pueden producir *desbordamientos de búfer* que permitan ejecutar cualquier programa. Como regla general, hay que:

- No asignar el bit *SUID* salvo cuando sea estrictamente necesario.
- Comprobar que cualquier programa con este bit activo no tiene ningún desbordamiento de buffer (conocido).
- No asignarlo jamás si el programa permite salir a la consola.

- **Atributo SGID: (Para ficheros)**: Si está activo en los permisos de grupo, este bit controla el estado de "poner id de grupo" de un fichero. Actúa de la misma forma que SUID, salvo que afecta al grupo. El fichero tiene que ser ejecutable para que esto tenga algún efecto.

- **Atributo SGID: (Para directorios)**: Si se activa el bit SGID en un directorio(con "chmod g+s directorio"), los ficheros creados en ese directorio tendrán puesto su grupo como el grupo del directorio.

A continuación, en la tabla 1, se describen los significados de los permisos de acceso individuales para un fichero. Normalmente un fichero tendrá una combinación de los siguientes permisos:

Permiso	Significado
`-r--------`	Permite acceso de lectura al propietario.
`--w-------`	Permite modificar o borrar el fichero al propietario.
`---x------`	Permite ejecutar este programa al propietario, (los guiones de shell también requieren permisos de lectura al propietario).
`---s------`	Se ejecutará con usuario efectivo ID = propietario
`-------s--`	Se ejecutará con usuario efectivo ID = grupo.
`-rw------T`	No actualiza "instante de última modificación". Normalmente usado para ficheros de intercambio (swap).
`---t------`	No tiene efecto. (antes sticky bit).

Tabla 1: *Permisos para ficheros en Linux*

A continuación, en la tabla 2, se describen los significados de los permisos de acceso individuales para un directorio. Normalmente un directorio tendrá una combinación de los siguientes permisos:

Permiso	Significado
`dr--------`	Permite listar el contenido pero no se pueden leer los atributos.
`d--x------`	Permite entrar en el directorio y usar en las rutas de ejecución completas.
`dr-x------`	Permite leer los atributos del fichero por el propietario.
`d-wx------`	Permite crear/borra ficheros.
`d------x-t`	Previene el borrado de ficheros por otros con acceso de escritura. Usado en /tmp.
`d---s--s--`	No tiene efecto

Tabla 2: *Permisos para directorios en Linux*

Los ficheros de configuración del sistema (normalmente en */etc*) son habituales que tengan el modo 640 (`-rw-r-----`), y que sean propiedad del *root*. Dependiendo de los requisitos de seguridad del sistema, esto se puede modificar.

Nunca se debe dejar un fichero del sistema con permiso de escritura para un *grupo* o para *otros*. Algunos ficheros de configuración, incluyendo */etc/shadow*, sólo deberían tener permiso de lectura por *root*, y los directorios de */etc* no deberían ser accesibles, al menos por *otros*.

SUID Shell Scripts: Los scripts de shell *SUID* son un serio riesgo de seguridad, y por esta razón el núcleo no los acepta. Sin importar lo seguro que se suponga un script de *shell*, puede ser utilizado para que un cracker pueda obtener acceso a un *shell* de *root*.

Normas prácticas para aumentar la seguridad

Es necesario dedicar tiempo a una correcta planificación. Los peligros expuestos por una falla de seguridad pueden ser acomodados con algunos arreglos. Todas las distribuciones de Linux traen unos valores por defecto que son prudentes para cubrir unas necesidades normales.

- **nosuid, nodev, noexec**.

  Salvo casos excepcionales, no debería haber ninguna razón para que se permita la ejecución de programas SUID/SGID en los directorios *home* de los usuarios. Esto se puede evitar usando la opción 'nosuid' en el fichero /etc/fstab para las particiones que tengan permiso de escritura por alguien que no sea el *root*. También puede ser útil usar 'nodev' y 'noexec' en las particiones de los directorios personales de los usuarios y en /var. Estas aplicaciones prohíben la creación dispositivos de bloque o carácter y la ejecución de programas.

- **Sistemas de ficheros en red**

  Si se exporta sistemas de archivos vía NFS, hay que configurar /etc/exports con los accesos lo más restrictivos posibles. Esto significa no usar plantillas, no permitir acceso de escritura a *root* y montar como sólo lectura siempre que sea posible.

- **umask**

  Se debe configurar la máscara de creación de ficheros para que sea lo más restrictiva posible. Son habituales *022*, *033*, y la más prohibitiva *077*, y añadirla a /etc/profile.

  El comando *umask* se puede usar para determinar el modo de creación de ficheros por defecto en el sistema. Es el complemento octal a los modos de fichero deseado. Si los ficheros se crean sin ningún miramiento de estado de permisos, el usuario, de forma inadvertida, podrá asignar permisos de lectura o escritura a alguien que no debería tenerlos. De forma típica, el estado de *umask* incluye *022*, *027* y *077*. Normalmente *umask* se pone en */etc/profile* y por tanto se aplica a todos los usuarios del sistema.

  Hay que asegurar que el valor *umask* de *root* es *077*, lo cual desactiva los permisos de lectura, escritura y ejecución para otros usuarios, salvo que explícitamente se use *chmod.*

- **Limitar recursos**

  Se deben fijar los límites al sistema de ficheros en lugar de 'ilimitado' como está por defecto. Se puede controlar el límite por usuario utilizando el módulo

PAM de límite de recursos y */etc/pam.d/limits.conf*. Por ejemplo, los límites para un grupo 'users' podrían ser estos:

```
@users hard core 0
@users hard nproc 50
@users hard rss 5000
```

Esto significa que se prohíba la creación de ficheros *core*, restringe el número de procesos a 50, y limita el uso de memoria por usuario a 5M.

- **wtmp, utmp**

Los ficheros /var/log/wtmp y /var/run/utmp contienen los registros de conexión de todos los usuarios del sistema. Se debe mantener su integridad, ya que determinan cuándo y desde dónde entró en el sistema un usuario o un potencial intruso. Los ficheros deberían tener los permisos 644, sin afectar a la normal operación del software.

- **Sticky bit**

El sticky bit se puede usar para prevenir borrados accidentales o proteger un fichero para sobreescritura. También previene que alguien cree enlaces simbólicos a un fichero, que ha sido el origen de ataques basados en el borrado de los ficheros /etc/passwd o /etc/shadow.

- **SUID y SGID**

Los ficheros SUID y SGID del sistema son potenciales riesgos de seguridad y deberían ser controlados. Como con estos programas se pueden garantizan privilegios especiales al usuario que los ejecuta, es necesario asegurar que no hay

instalados programas inseguros. Un ataque de los *crackers* es explotar programas con el bit SUID, y entonces dejar un programa SUID como puerta trasera para entrar la próxima vez, incluso aunque el agujero original ya esté tapado.

El administrador tiene la tarea de encontrar todos los programas SUID/SGID del sistema y mantener la pista de lo que son. Se debe estar prevenido de cualquier cambio que podría indicar un potencial intruso. Se puede utilizar el siguiente comando para localizar todos los programas *SUID/SGID*:

```
root# find / -type f \( -perm -04000 -o -perm -02000 \)
```

Incluso se podría crear una base de datos de programas SUID con

```
root# find / -type f \( -perm -04000 -o -perm -02000 \)>/var/suid
```

Y posteriormente verificar si ha aparecido alguno nuevo con el siguiente guión:

```
for fich in `find / -type f \( -perm -04000 -o -perm -02000 \)`
do
if ! grep $fich /var/suid
then
echo "$fich es un nuevo fichero con SUID"
fi
done
echo "Actualiza la base de datos si es necesario"
```

Se pueden eliminar los permisos SUID o SGIG de un programa con chmod, y siempre es factible devolverlo a su estado original si es absolutamente necesario.

- **Permisos de escritura**

Los ficheros con permiso de escritura global, particularmente los del sistema, pueden ser un agujero de seguridad si un cracker obtiene acceso al sistema y los modifica. Además, los directorios con permiso de escritura global son peligrosos, ya que permiten a un cracker añadir y borrar los ficheros que quiera. Para localizar los ficheros con permiso de escritura global, use el siguiente comando:

```
root# find / -perm -2 –print
```

- **Ficheros extraños**

Los ficheros sin propietario también pueden ser un indicio de que un intruso ha accedido al sistema. Se pueden localizar los ficheros del sistema que no tienen propietario o que no pertenecen a un grupo con el comando:

```
root# find / -nouser -o -nogroup –print
```

- **Ficheros peligrosos**

La localización de archivos .rhosts debería ser uno de los deberes regulares de la administración del sistema, ya que estos ficheros no se deberían permitir en Linux. Un cracker sólo necesita una cuenta insegura para potencialmente obtener acceso a toda su red. Puede localizar todos los ficheros .rhosts del sistema con el siguiente comando:

```
root# find /home -name .rhosts –print
```

- **Permisos**

Finalmente, antes de cambiar permisos en cualquier sistema de ficheros. Hay que saber que nunca se debe cambiar permisos de un fichero simplemente porque parezca la forma fácil de hacer que algo funcione. Siempre se debe determinar por qué el fichero tiene esos permisos y propietario antes de modificarlos

APÉNDICE D: Requerimientos y restricciones

Los requisitos y restricciones del Sistema para PC-Voz son los siguientes:

- Red basada en arquitectura Ethernet con IP versión 4
- Sistema Operativo Linux
- Librerías del sistema PCAP y MySQLClient.
- GNU gcc versión 2.96 ó superior para compilarlo.
- Sistema Manejador de Base de Datos MySQL versión 3.23.33 ó superior en el host con el servicio de Base de Datos.
- Procesador Intel Pentium o compatible.
- 32 Mb de memoria RAM.
- El espacio en disco es dependiente del tamaño de los archivos logs y del almacenamiento en la Base de Datos.

Los requisitos y restricciones del Sistema para PC-PC-Voz son los siguientes:

- Servidor Web con el módulo de PHP versión 4.2.2 instalado.
- Navegador Web con soporte para Javascript instalado.

ANEXO 1: mmdump: A Tool for Monitoring Internet Multimedia Traffic

Jacobus van der Merwe, Ramón Cáceres, Yang-hua Chu, Cormac Sreenan. ACM Computer Communication Review. Octubre 2000, Nº 5, Vol 30.

mmdump: A Tool for Monitoring Internet Multimedia Traffic

Jacobus van der Merwe*, Ramón Cáceres†, Yang-hua Chu‡, Cormac Sreenan§
kobus@research.att.com, ramon@vindigo.com, yhchu@cs.cmu.edu, cjs@cs.ucc.ie

Abstract

Internet multimedia traffic is increasing as applications like streaming media and packet telephony grow in popularity. It is important to monitor the volume and characteristics of this traffic, particularly because its behavior in the face of network congestion differs from that of the currently dominant TCP traffic. To monitor traffic on a high-speed link for extended periods, it is not practical to blindly capture all packets that traverse the link. We present *mmdump*, a tool that parses messages from RTSP, H.323 and similar multimedia session control protocols to set up and tear down packet filters as needed to gather traces of multimedia sessions. Unlike tcpdump, dynamic packet filters are necessary because these protocols dynamically negotiate TCP and UDP port numbers to carry the media content. Our tool captures only packets of interest for optional storage and further analysis, thus greatly reducing resource requirements. This paper presents the design and implementation of *mmdump* and demonstrates its utility in monitoring live RTSP and H.323 traffic on a commercial IP network. The preliminary results obtained from these measurements are presented.

1 Introduction

Recent years have seen increasing use of the Internet to send and receive audio and video, including streaming playback of music and news, as well as real-time voice telephony and conferencing. This traffic is expected to continue growing, driven by improvements in PC performance, residential access bandwidth, and media coding algorithms. Whilst the trends and behavior of Web traffic have been studied extensively, multimedia traffic has yet to be studied in detail. Multimedia applications typically use UDP transport, demand relatively large and constant data rates, and react slowly, if at all, to network congestion. As this traffic grows, its impact on network performance may be significant. It is important for network designers to understand the nature of multimedia traffic.

Internet traffic measurements are commonly performed using the *tcpdump* utility, which can be used to monitor packets for a particular application-level protocol by filtering based on the appropriate TCP/UDP port number. Use of *tcpdump* for multimedia traffic is complicated because the majority of multimedia applications use dynamically assigned UDP port numbers. For example, protocols such as the Real Time Streaming Protocol (RTSP) [19], the Session Initiation Protocol (SIP) [6], and H.323 [7] use a well known TCP port number to initiate a multimedia session. Once the session is established, the protocols negotiate other TCP or UDP port numbers dynamically for media control traffic and media data traffic. To address this problem we have created a new utility we call *mmdump* that is based on *tcpdump* but makes use of protocol-specific *parsing modules* to determine the dynamic set of ports that need to be monitored.

In this paper we present the design and implementation of *mmdump*. *mmdump* contains a parsing module for each multimedia control protocol. All traffic received on the well known control port is passed to the parsing module in question. The parsing module identifies individual control sessions in this aggregate control stream, and parses the control messages to extract the dynamically assigned port numbers. The parsing module then dynamically changes the packet filter to allow packets associated with these ports to be captured. Architecturally, *mmdump* departs from *tcpdump* by maintaining state for each multimedia session. This is necessary because *mmdump* needs to associate arriving packets with individual sessions and later report statistics of the sessions. The situation is made worse by issues such as packet loss and asymmetric routing. We present our approach to these problems in the current implementation of *mmdump* and suggest improved approaches.

We also present results obtained using *mmdump* to monitor multimedia traffic in AT&T's commercial IP network. The version of *mmdump* used included RTSP and H.323 parsing modules; we have since developed a rudimentary SIP parser. The varied types of analysis that we present for traffic from different multimedia control protocols highlight the versatility of *mmdump*.

The rest of this paper is organized as follows. Section 2 provides background on *tcpdump* as well as RTSP and H.323. Section 3 explains the structure and operation of *mmdump*. Section 4 presents results demonstrating the use of *mmdump* on live multimedia traffic. Section 5 summarizes related work, and Section 6 concludes the paper.

*AT&T Labs–Research, Florham Park, NJ, USA
†Vindigo, New York, NY, USA
‡Carnegie Mellon University, Pittsburgh, PA, USA
§University College Cork, Cork, Ireland

2 Background Information

Given that *mmdump* is based on and extends *tcpdump*, we give a brief overview of *tcpdump* in this section. In addition, we briefly discuss two example multimedia control protocols that are used to negotiate port numbers for streaming content.

2.1 Structure of *tcpdump*

The *tcpdump* utility provides a popular mechanism for monitoring packet transmissions. *tcpdump* builds on top of the *libpcap* library, which provides two key functions: an abstraction for dealing with different types of network interfaces, and the ability to compile a filter expression for use by a packet filter. The library provides a common interface to different ways of performing packet filtering. For example, on a system with the BSD Packet Filter (BPF) [11], filtering is done in kernel space and *libpcap* simply passes the compiled filter expression to the kernel. *libpcap* can also perform the packet filtering itself (in user space) when required. This is used on systems where the kernel does not support packet filtering, and when *tcpdump* is reading packets from a previously generated raw dump file, rather than directly from the network.

In normal operation, *tcpdump* is run with a command-line expression indicating all packets of interest. The grammar and syntax used for this expression is fairly high level. For example, the expression `host 135.207.26.201 and tcp port 554` indicates an interest in all TCP packets using port 554 that are either originating from or going to the host with the specified IP address. This expression is passed to the *libpcap* library at startup where it is compiled into an intermediary tree structure. The tree is then optimized and the resulting tree is translated into a contiguous filter expression which is installed in the operational packet filter.

For *tcpdump*, all packets that pass through the installed filter will either be logged to file, or be passed to a printing module in the *tcpdump* part of the code. In the latter case, print functions for successively higher layers of the protocol stack typically print out parts of the packet. For example, for a UDP packet carrying a Sun RPC request, the *print-ethernet* function will call *print-ip*, which in turn will call *print-udp* and then *print-sunrpc*. For a more detailed discussion of *tcpdump* please refer to [9, 8, 2].

While extremely popular and successful as a monitoring tool, *tcpdump* is unable to efficiently monitor multimedia traffic, since the majority of multimedia applications use dynamically assigned UDP or TCP port numbers for media transfer. This is the case with popular multimedia protocols such as RTSP and H.323, in which a control interaction using a well known port is used to negotiate the set of dynamic port numbers to be used for media transfers. By their nature, dynamically assigned port numbers cannot be specified from the command line, meaning that *tcpdump* in its current form cannot be used to monitor this type of traffic. If a given multimedia protocol normally picks a port from a small range of port numbers, it is of course possible to statically specify the whole range from the command line and perform post-processing to extract the data of interest. As we will discuss in Section 4.1, using *tcpdump* in this manner is very inefficient in terms of disk space and does not scale to processing packets on fast network links.

2.2 Real Time Streaming Protocol (RTSP)

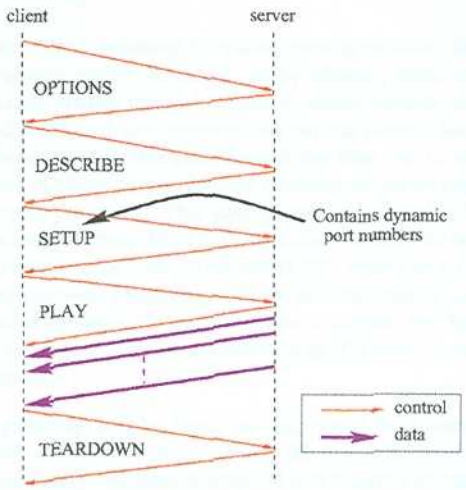

Figure 1: Dynamic port number assignment with RTSP

The Real Time Streaming Protocol (RTSP) is becoming the dominant control protocol for streaming content on the Internet. Figure 1 depicts a sample interaction between an RTSP client and server. RTSP is used to set up and control (pause, forward, etc.) the playback of streaming content across the Internet. RTSP is a classic request-response protocol, but also allows pipelining of messages to reduce latency. Protocol interaction starts with an OPTIONS request/response whereby the client and server establish mutual capabilities. The client then issues a DESCRIBE request for the media stream it is interested in. The response from the server contains media specific information about the stream, e.g. the encoding used, the clip length and the average bit rate. Depending on the particular session, more than one media stream might be described in a single DESCRIBE response message.

After DESCRIBE, the client issues a SETUP request which contains the set of protocols and port numbers (or range of port numbers) on which the client is willing to the receive the media stream. For RTSP this is normally UDP and a dynamically chosen port number, although it is also possible to use RTSP in "interleaved mode" where the data stream is interleaved on the original TCP control connection. This interleaving is typically only used to allow streaming through a firewall. The server selects one of these options and a port number and sends it back to the client in the response message.

Following these exchanges the client can issue a PLAY request to start the streaming and can issue PAUSE and other control requests for the stream. The session normally ends with a TEARDOWN request at which time the TCP connection is also terminated[1].

[1]This summary of the RTSP protocol reflects our monitoring of its usage in practice. It is compliant with the RTSP specification, but the specification allows several variations, for example the use of UDP as the transport mechanism for RTSP and the tearing down of the RTSP control connection without terminating the RTSP session.

2.3 H.323 conferencing control protocols

Conferencing and packet telephony represent another class of applications that make use of a separate control protocol to dynamically negotiate port numbers for media transfer. H.323 is a popular example of such a protocol. In principle H.323 operates in a manner similar to RTSP, but we describe the details here for the sake of completeness. Figure 2 depicts a sample H.323 exchange between caller and callee. Interaction starts with the caller sending a SETUP message on a well known TCP port to the callee. This exchange is on the first of two TCP connections which is called the Q.931 channel. The callee responds with an ALERTING message followed by a CONNECT message. The CONNECT message contains the port number for the second TCP connection between caller and callee which is called the H.245 or conference control channel. At this point the first TCP connection may be disconnected. Interaction on the H.245 channel starts with an exchange of messages to determine terminal capabilities and for determining the master and slave roles between the two terminals. The sender (of subsequent media) then sends an Open Logical Channel message to the receiver. In the Internet environment this message contains the RTCP port number on which the sender wants to receive RTCP reports about the quality of reception [18]. The receiver responds with an Ack message which contains the RTP port number on which the media stream should be received and an RTCP port number on which to receive RTCP sender reports. The Open Logical Channel message always originates from the sender of a data. As indicated in Figure 2 a two-way conversation will therefore require an Open Logical Channel and Ack pair of messages in both directions. The second TCP connection remains in place for the duration of the call and terminates after sending an End Session message.

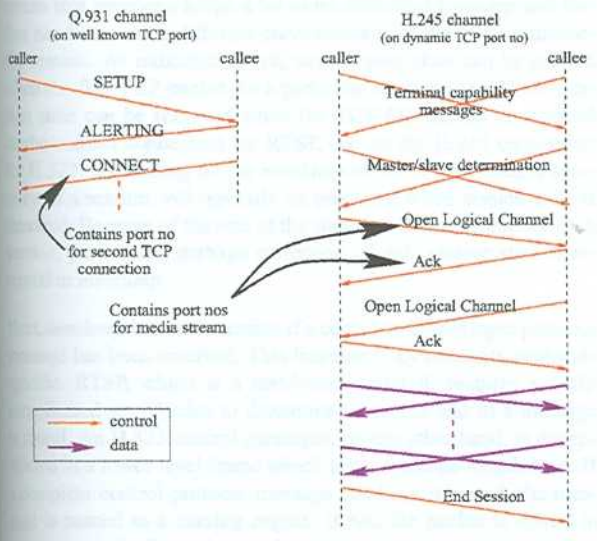

Figure 2: Dynamic port number assignment with H.323

3 Design, implementation and operation of *mmdump*

mmdump extends *tcpdump* by adding parsing modules for multimedia session control protocols and by allowing these parsers to dynamically change the packet filter to accept packets on the dynamically assigned port numbers. As per the normal functioning of *tcpdump*, packets that pass through the filter can be displayed by means of protocol-specific print modules, or can be logged to a file for post processing. This arrangement is depicted in Figure 3 and discussed in more detail in this section. We describe below our parsing modules for both RTSP and H.323. Note that all existing *tcpdump* functions remain in *mmdump* and the latter is therefore a superset of *tcpdump*. However, in order to explain the functionality of *mmdump* it is easier to refer to it as if it were a completely different tool.

When gathering lengthy traces on high-speed links, *mmdump* is commonly used in two stages. During the first stage, only the messages containing dynamically assigned port numbers are parsed, the packet filter is updated and all packets that pass through the filter are dumped into a file for later analysis (this includes all control packets and all data packets). In this mode of operation the parsing modules are only concerned with messages containing the dynamically assigned port numbers. Raw dump files generated in this manner can be post processed again using *mmdump*. In these cases the parsing module might extract information from other messages, e.g. in the case of RTSP the URL of media objects, the type of encoding used and the length of objects might be of interest. It is also possible to use *mmdump* in a one stage process whereby all information of interest is printed online and no packets are logged.

3.1 Structure of *mmdump*

The multimedia control protocols make use of well known port numbers. When started, *mmdump* sets up a default filter to capture all packets that belong to these control connections to bootstrap the monitoring process. This filter is set up to receive all packets for all connections traversing the probe point that *mmdump* is monitoring.

For each of the multimedia applications of interest a parsing module has to be supplied. All packets that arrive on a particular well-known port number are passed to the corresponding parsing module for processing. Figure 4 shows the functionality that each parsing module needs to supply.

mmdump maintains state for each active "session". As shown in Figure 4, the first action required by a parsing module is to do a session lookup. A session is defined as a unique instance of a control protocol interaction, e.g. an RTSP client communicating with a RTSP server, or two H.323 peers communicating. A session lookup therefore involves a matching of source and destination addresses and port numbers in the received packet against the equivalent values in the stored session state. For H.323, the second TCP control connection (the H.245 connection), has to be associated with the first control connection (the Q.931 connection). In this case, the session lookup therefore has to match the incoming packet against both these control connections associated with the same session. While the basic session lookup is fairly generic, i.e. matching IP

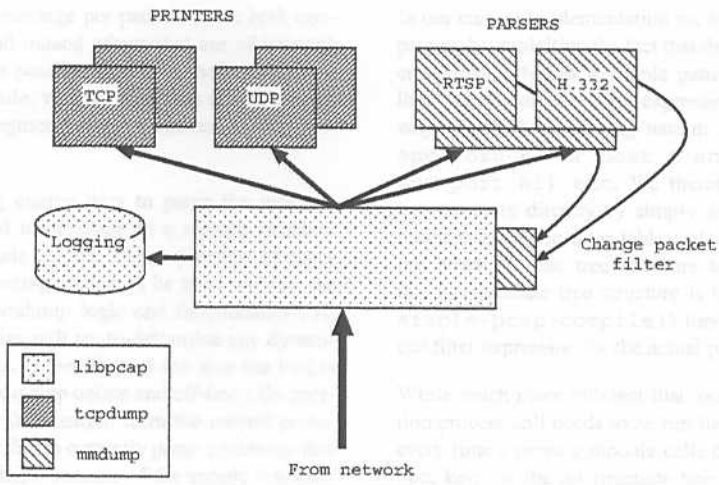

Figure 3: Architecture of *mmdump* in relation to *tcpdump*

addresses and port numbers, protocol-specific variations such as the aforementioned make it awkward to efficiently separate this functionality out in a generic way. If the session lookup was successful, the retrieved session state is used, or if it failed, a new session structure is allocated.

Maintaining state in the tool is a significant departure from the largely stateless operation of *tcpdump*. *tcpdump* does maintain some state, e.g. in order to print out relative rather than absolute sequence numbers for TCP packets. The state maintained by *mmdump* is different in that it keeps track of multimedia sessions, each of which can have more than one TCP or UDP connections. This means that *mmdump* keeps a lot more state than *tcpdump* and further needs to match different connections with the same multimedia session. As indicated above, new session state can be created when the first TCP packet for a particular session is received. Session state can be removed when the TCP FIN packet is received on the control connection for RTSP, and on the H.245 connection for H.323. Depending on the *mmdump* mode of operation, a summary of a session will typically be produced when session state is removed. Because of the size of the state that is maintained for each session, performing garbage collection of stale session state is essential in *mmdump*.

Next, *mmdump* has to determine if a complete higher layer protocol message has been received. This function is by necessity protocol-specific. RTSP, which is a text-based protocol, requires a fairly complicated set of rules to determine when the end of a message is found. An H.323 control messages, on the other hand, is encapsulated in a lower level frame which has a message-length field. If a complete control protocol message has been received, the message is passed to a parsing engine, if not, the packet is stored in a per-session buffer to be used together with subsequent packets received for this session. The current implementation of this per-packet buffer does not take TCP sequence numbers into account and simply treats packets in the order in which they were received. This is clearly problematic in an IP environment where both packet

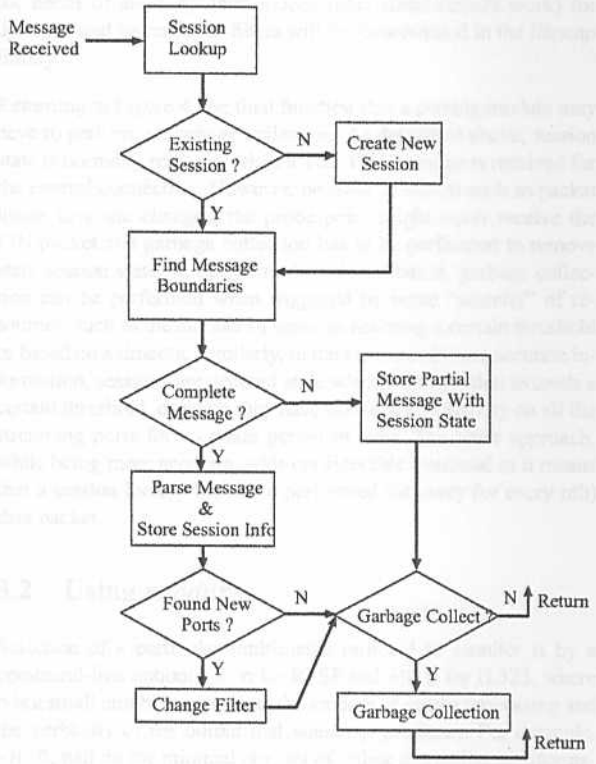

Figure 4: Functioning of a parsing module

reordering and packet loss can happen. Especially for RTSP traffic where different RTSP messages often span several IP packets, or have fragments of different RTSP messages in the same IP packet. For H.323, control messages seem to be contained in one (or two) IP

packets, with a single H.323 message per packet. Since both control protocols in question (and indeed others that are of interest) make use of TCP, it should be possible to extend the implementation with a generic TCP module, which could pass to the parsing modules only in-order TCP segments. We are currently investigating this possibility.

The protocol-specific parsing engine tries to parse the message passed to it, putting extracted information in a session structure supplied by the parsing module proper. This separation of functionality allows a different parsing engine to be used without the need to change any of the mmdump logic and functionality[2]. At the very least the parsing engine will try to determine any dynamically negotiated port numbers. Depending on the way the tool is used (i.e. one-step online, or two-step online and off-line), the parsing engine also extracts other information from the control protocol. Should the parsing engine fail to correctly parse a message that was believed to be intact, perhaps because of the simple reassembly described above, the message is discarded and an event count is incremented.

If the parsing engine was able to extract any negotiated port numbers, this fact is relayed back to the main parsing function by means of a flag in the shared session structure. The parsing module can now invoke new functions exported by the *libpcap* library to dynamically change the filter expression so that packets associated with this port number can also pass through the packet filter.

The interface between the parsers and the packet-filter level is very simple and consists of two function calls: `change-filter()` and `do-filter()`. `change-filter()` allows a parser to either add or delete a port number for a particular address and protocol type to the filter expression. Alternatively a parser can request that all ports associated with a particular address be deleted. Calling `change-filter()` does not result in the immediate update of the real packet filter, rather the requested change is noted at the packet-filter level, and when a parsing module calls `do-filter()`, the actual filter change takes place. This allows the parsing module to make a number of related changes to the packet filter in one go, for example to add both RTP and RTCP port numbers to a specific address. This is desirable, because as explained below, the actual generation of a new packet filter is currently an expensive operation which should not be performed unnecessarily.

As explained in Section 2.1, for regular *tcpdump* a command-line filter expression is compiled once (using the function `pcap-compile()`), into an intermediate tree structure which is then optimized to produce a contiguous filter expression in a form which can be installed in a packet-filter state machine. In our initial proof-of-concept implementation, we made use of this same interface by producing a long ASCII filter expression for input to `pcap-compile()` every time that `do-filter()` was called. Generating the intermediate tree structure is however a very expensive operation and this approach was therefore very inefficient.

[2]For example, a new H.323 library, capable of parsing the Fast-Connect option in the latest version of the specification was recently added to *mmdump* for monitoring voice traffic in a voice over IP trial.

In our current implementation we bypass the standard compilation process by exploiting the fact that the filter expressions that we generate always follow a simple pattern. In particular, a command-line version of the filter expression used by *mmdump* will always be of the following nature: `tcp port X or (host A and port A1 or host A and port A2) or (host B and port B1) etc.` We therefore generate the intermediate tree structure directly by simply walking through the list of current entries in the filter table and AND'ing or OR'ing the building blocks of the tree structure together as needed. As before this intermediate tree structure is then optimized (through a new `simple-pcap-compile()` function) and turned into a contiguous filter expression for the actual packet filter.

While much more efficient than our initial attempt, the optimization process still needs to be run for the complete filter expression every time a parsing module calls `do-filter()`. A better solution, keeping the intermediate tree structure that can be modified based on instructions from the parsers, is not possible with the current *libpcap* implementation, as the intermediate tree structure is "consumed" in the optimization process. We understand that a new version of the *tcpdump* family of tools [2] is being written and that the needs of *mmdump* (and indeed other measurement work) for dynamic and incremental filters will be incorporated in the *libpcap* library.

Returning to Figure 4, the final function that a parsing module may have to perform is garbage collection. As described above, session state is normally removed when a TCP FIN message is received for the control connection. However, because of effects such as packet losses or route changes, the probe point might never receive the FIN packet and garbage collection has to be performed to remove stale session state. In our current implementation, garbage collection can be performed when triggered by some "scarcity" of resources, such as the number of sessions reaching a certain threshold or based on a timeout. Similarly, in the absence of more accurate information, sessions are deemed stale when their duration exceeds a certain threshold, or when they have not seen any activity on all the streaming ports for a certain period of time. The latter approach, while being more accurate, adds considerable overhead as it means that a session lookup has to be performed for every (or every nth) data packet.

3.2 Using *mmdump*

Selection of a particular multimedia protocol to monitor is by a command-line option: `-R n` for RTSP and `-H n` for H.323, where `n` is a small number controlling the amount of online processing and the verbosity of the output that *mmdump* produces. For example, `-H 0`, will do the minimal amount of online extraction of information and is often used in conjunction with the raw-write *tcpdump* option (`-w <filename>`), which saves the packets to disk for later processing when *mmdump* is used in two stages. `-H 1` causes *mmdump* to perform online extraction of protocol specific information and can be used either online or off-line, the latter typically with the *tcpdump* raw-read option (`-r <filename>`). With `n>0`, *mmdump* produces session specific records: For RTSP each session record shows the session related information, such as the start and end time, and the client and server addresses. In addition media-

specific information (e.g. format, size) for each media element (e.g. an audio clip, a background image) is shown including the URL and the clip length of each element. For H.323, each session record contains the IP addresses and phone numbers of the participants, the call duration and information about the audio codec and H.323 vendor whose software was used.

Normal *tcpdump* operation includes the notion of a "snaplen" (snapshop length), the maximum number of bytes from each packet that will be captured. *tcpdump* allows snaplen to be specified from the command-line or uses a default snaplen if none is specified. With *mmdump*, we need to capture the complete control messages in order to parse them correctly. The snaplen should therefore be set to the MTU (Maximum Transmission Unit) on a particular medium. In general, however, there is no need to capture the complete data packets. We have therefore added an option (`-D`), again used in conjunction with the `-w` option, to reduce the snaplen applied to data packets to include only header information of such packets. This dramatically reduces the storage requirements when raw dump files are used.

4 Results

In this section we present preliminary results obtained from our use of the *mmdump* tool. In Section 4.1 we present results of using *mmdump* to monitor RTSP traffic: Section 4.1.1 contains results of a single RTSP presentation in a controlled environment, while Section 4.1.2 presents measurement results from a probe point in AT&T's commercial IP network. In Section 4.2 we present a similar set of results for the use of *mmdump* on H.323 traffic: Section 4.2.1 is for a single H.323 session in a controlled environment and Section 4.2.2 presents results for H.323 traffic from the same probe point in the AT&T network.

The results presented are meant to show some of the possibilities of the tool rather than conclusive results about the use of streaming media in the Internet.

4.1 RTSP Results

4.1.1 Individual session in controlled environment

In this case a single RTSP presentation was viewed by means of a RealPlayer [15] client from a PC running Microsoft Windows. A Linux PC on the same Ethernet segment was running the *mmdump* tool to capture the interaction. The presentation in question was CNN Headline News [4], which was streamed from the Internet.

The CNN Headline news presentation consists of a small video section in the top left corner of the display area. Below the video section is a text window for presenting the latest news in text format (this normally contains a link to the CNN web site), in addition to an advertising section and a hyperlink to provide feedback. The right-hand side of display area consist of hyperlinks to other news-related streaming presentations.

While not visible to the user the presentation in question is served from two separate servers in different domains. This requires two RTSP sessions, the details of which are presented in Tables 1 and 2 and in Figure 5.

Table 1 shows the "base-URL" served by each server as well as the number of RTSP control packets going between the client and each server. Note that because of the location of the *mmdump* machine relative to the client machine, this traffic trace contained packets going in both directions between the client and server machines. Because of asymmetric routing in IP networks this is not the case in general.

Table 2 shows the URL extension, which together with the base-URL presents the complete URL for each object that is part of the presentation. Also shown in the table is the UDP port number chosen by the client, the number of UDP packets used to stream each object to the client, as well as a file-type field.

Figure 5 shows the packet arrival information for all UDP streams on a common timeline. The offset on the y-axis is used to depict the port number used for streaming the media. Each small vertical line on a horizontal line indicates a packet arrival event. Figure 5 clearly shows how the first object "streamed" to the client is a SMIL file [20], index.smi. This object in fact contains a description of the presentation which includes the layout of the presentation display, the various objects associated with each region of of the display, the location of each such object and optionally a timeline indicating when different objects should be be displayed. It is thus from the SMIL file that the client learns that some of its media should be retrieved from a different server. As indicated in Table 2 the actual "interesting" media content is streamed from the cnn.com domain, while several "support objects" like the background and links to other SMIL files are streamed from the real.com domain.

With reference to Figure 5, the presentation starts with a short signature tune (sting.rm) with an accompanying short animation (1.swf). Next the background image (back.jpg), the static advertisement (ad.gif) and text supporting text (left.rt) is streamed and presented. This is followed by an audio/video advertisement clip (ad1-28.rm) and text to make up the hyper-links in the presentation (links.rt). In the mean time more text is streamed (nowplaying_headlinenews.rt) and finally the audio and video clip for the actual headline news is streamed (headlines.rm28.rm).

4.1.2 Sessions in the public Internet

We gathered packet traces from a measurement probe inside the public Internet, more specifically from a T3 private peering link inside AT&T's commercial IP network. This placement of the probe machine means that in general both directions of interaction between two hosts will not necessarily be visible at the probe point due to asymmetrical routing. In all cases the traces were anonymized as soon as they came off the link under study, before writing any packet headers to stable storage. We collected traces on a dedicated 500MHz Alpha workstation that ran Digital UNIX and was attached to the link under study.

The traces analyzed in this section were gathered from our probe point in New York City for the week 15 April 1999 to 22 April 1999. Since at the time *mmdump* was still being tested the traces were gathered with a regular *tcpdump* capturing all packets on TCP port 554 and all UDP packets in the port ranges from 6970 to 7040

Session No	Session base URL	TCP packets	
		Client to server	Server to client
0	albany-b.real.com/showcase/channels/cnn_headlines/gold/	66	75
1	realchannel.cnn.com/	65	47

Table 1: Two RTSP sessions associated with single CNN headline news presentation

Session No	Media Stream	Client Port	URL extension	UDP packets	UDP Bytes	File type
0	0	6970	index.smi	5	1656	SMIL
	1	6972	audio/sting.rm	26	13620	Real Audio/Video
	2	6974	flash/1.swf	18	5357	Shockwave Flash
	3	6976	pix/back.jpg	30	14532	JPEG
	4	6978	text/left.rt	8	2014	Real Text
	5	6980	pix/ad.gif	17	6796	GIF
	6	6982	text/links.rt	11	4032	Real Text
	7	6984	text/feedback.rt	5	472	Real Text
	8	6992	text/nowplaying_headlinenews.rt	4	384	Real Text
1	0	6986	ads/ad1_28.rm	81	23609	Real Audio/Video
	1	6994	channel/headlines.rm28.rm	3991	1578597	Real Audio/Video

Table 2: Eleven media streams associated with single CNN headline news presentation

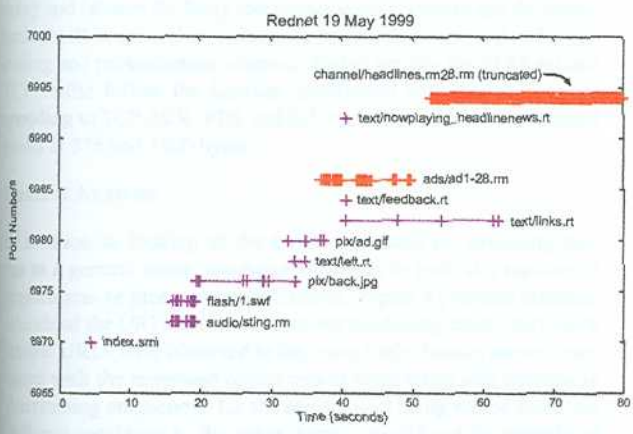

Figure 5: UDP activity for each media stream for CNN headline news from Real Networks (80 seconds shown)

inclusive, and *mmdump* was used exclusively in post processing mode. This proved a useful means to gather data to test *mmdump*, but also served to convince us about the need for a tool like *mmdump*. We captured the whole packet length for all TCP packets, as this is required for the RTSP protocol parsing, but only the first 136 bytes for UDP packets. The trace files for the week resulted in approximately 15 Gbytes worth of gzip'ed files. A new trace file was generated each 30 minutes and typically varied from below 10 Mbytes to well over 100 Mbytes depending on the time of day. Later using *mmdump* to trim these files to the traces it would have created from the original data resulted in a 60% to 80% reduction in required disk space per file.

Traffic Characteristics

One of the main questions we hope to address with this work is to determine the amount of streaming media relative to other Internet traffic and to monitor any changes in the longer term.

The issue of asymmetric routing has been mentioned a number of times in this paper. It turns out that for RTSP-related traffic a very small percentage of traffic at the probe point was in fact visible in both directions between client and server. This can potentially lead to erroneous conclusions about the relationship between control and data traffic for streaming media. For example, the network locality of a popular server might generate a lot of control traffic seen going from clients to servers, without a reciprocal contribution in data streamed from the server if that traffic does not pass the probe point. In Figure 6 we therefore show the control (RTSP/TCP) and data (UDP) traffic volumes (in number of packets) going only from servers to clients. This appears to be a reasonable comparison of the relationship between control and data. As before packet counts were generated for every half hour of the trace data.

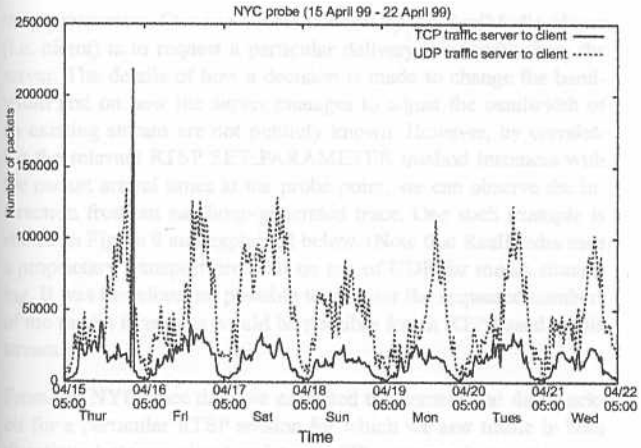

Figure 6: RTSP and related UDP packet counts

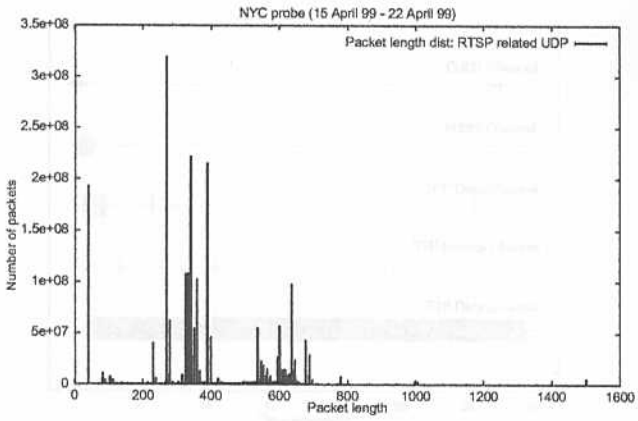

Figure 7: UDP packet length distribution for RTSP related traffic

One observation regarding Figure 6 is that peak hours are drastically shifted towards the late evening hours. This contrasts with aggregate TCP traffic characteristics (not shown) which normally have very clear peaks during office hours. From Figure 6, activity over weekends are not significantly lower than over weekdays, only more evenly spread over all hours.

Figure 7 shows the packet length distribution for RTSP-related (i.e. streaming) UDP traffic. Significant peaks are at packet lengths much shorter than typical Maximum Transmission Unit (MTU) sizes. Some of these can probably be attributed to concerns about delay and latency for fairly low bitrate voice encoders and the distribution will in general be influenced by popular voice and video encoding and packetization schemes. Packet lengths for RTSP-related TCP traffic follow the familiar distribution with 40 bytes corresponding to TCP ACK, FIN, and SIN packets, and two MTU related peaks at 576 and 1500 bytes.

Content Analysis

In addition to looking at the traffic generated by streaming media in a general sense, *mmdump* allows us to look at a number of application- or protocol-specific issues. Figure 8 presents information about the URLs extracted from our week-long trace. Only 3074 unique URLs were observed in the trace. Only domain names combined with the requested object names were taken into account in determining uniqueness. I.e. the same object being served from two different machines in the same domain would not be considered unique. Figure 8 shows the number of references to each of the most popular 1000 objects. This Zipf-like distribution, showing that relatively few objects are extremely popular, has strong implications for caching strategies for multimedia objects.

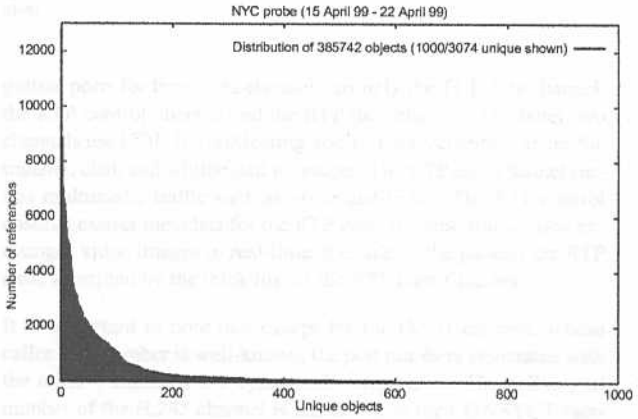

Figure 8: Distribution of URLs

Rate Adaptation

As a final example of the capabilities of *mmdump*, we have investigated the transmission rate of a single media stream and considered its interaction with the application control protocol. The RTSP protocol has a SET_PARAMETER method that can be used to set arbi-

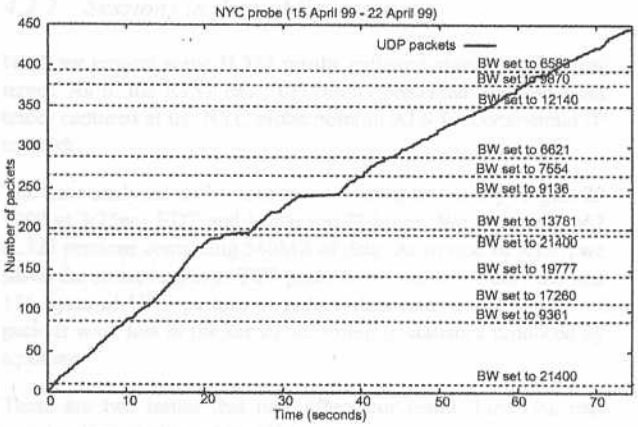

Figure 9: Packet arrivals for a single live UDP stream

trary parameters. One use of this method by the RealMedia player (i.e. client) is to request a particular delivery bandwidth from the server. The details of how a decision is made to change the bandwidth and on how the server manages to adjust the bandwidth of an existing stream are not publicly known. However, by correlating the relevant RTSP SET_PARAMETER method instances with the packet arrival times at the probe point, we can observe the interaction from an *mmdump*-generated trace. One such example is shown in Figure 9 and explained below. (Note that RealMedia uses a proprietary transport protocol on top of UDP for media streaming. It was therefore not possible to monitor the sequence numbers of the media stream as would be possible for an RTP-based media stream.)

From the NYC trace data we extracted the control and data packets for a particular RTSP session for which we saw traffic in both directions between client and server. The session in question was streamed from a live source and contained only a single media stream. In Figure 9 we plot the timestamp of each UDP packet of the media stream as it was captured by *mmdump*. The total duration of the trace is 75 seconds. Time is on the X-axis while the Y-axis reflects the number of the corresponding packet. The slope of this plot is therefore an indication of the rate at which UDP packets where logged by *mmdump* (and the rate at which packets were sent by the source), with a steeper slope corresponding to a higher rate. The slope of the first part of the plot, packets 0 to 200, is clearly steeper than for the final part of the plot, packets 250 to 450.

Superimposed on the plot of UDP packet timestamps, is a number of horizontal dotted lines. Each horizontal line corresponds to the arrival of a SET_PARAMETER method for a bandwidth parameter as seen by the probe point. The value shown is the requested delivery bandwidth in bits per second. The sequence of these parameter requests goes from 21400 to 9361 to 17260 and 1977 in the first part, to 21400 and 13781 in the middle part and ends with a more modest sequence of 9163 to 7554 to 6621 to 12140 to 9870 and 6563 in the final part of the plot. This corresponds with the observed flatter slope of the last part of the plot. (Note that since the probe point is somewhere in the network between the client and the server, there will be a time lag between the time that *mmdump* records a SET_PARAMETER method, and the time that the server will have responded to it.)

4.2 H.323 Results

4.2.1 Individual session in controlled environment

First we first show how *mmdump* captures an H.323 session in a controlled environment. In the lab three machines are connected over a shared Ethernet link. Two Windows PC machines run Microsoft NetMeeting 3.1 and they make a video conferencing session with each other using the H.323 protocol. A third machine runs *mmdump* to capture the session on-line. The session lasts for approximately 35 seconds. Figure 10 shows packet arrival events grouped by channels. Each horizontal line indicates a channel, and there are five channels created in the duration of the session. Each small vertical line on a horizontal line indicates a packet arrival event. The session begins with the establishment of the Q.931 channel, followed by the H.245 channel. Then, NetMeeting uses H.245 to negotiate ports for three data channels, namely the TCP data channel, the RTP control channel and the RTP data channel. The latter two channels use UDP. In NetMeeting, the TCP data channel carries file transfer, chat, and whiteboard messages. The RTP data channel carries multimedia traffic such as voice and video. The RTP control channel carries metadata for the RTP data. Because this session exchanges video images in real-time, the bulk of the packets are RTP data, identified by the thick line of the RTP Data Channel.

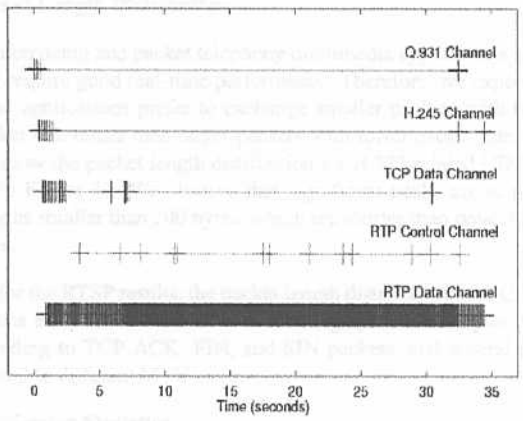

Figure 10: Packet arrival events for each channel of one H.323 session

It is important to note that except for the Q.931 channel, whose callee port number is well-known, the port numbers associated with the other 4 channels are dynamically negotiated. The callee port number of the H.245 channel is embedded in the CONNECT message of a Q.931 packet. The port numbers for the data channels are negotiated by the H.245 Open Logical Channel messages.

4.2.2 Sessions in the public Internet

Next, we present some H.323 results gathered over the public Internet. As in the RTSP case, the results presented here are from traces captured at the NYC probe point in AT&T's commercial IP network.

The trace analyzed in this section was started on Sunday August 22 1999 at 3:25pm EDT and lasted for 72 hours. We captured 2667 H.323 sessions containing 540MB of data. As in case of RTSP, we saved the entire length of TCP packets, but we saved only the first 136 bytes of UDP packets to reduce data size. Less than 1% of packets were lost in the kernel according to statistics produced by *tcpdump*.

There are two issues that may affect our result. First, the current implementation of the H.323 module assumes peer-to-peer communication. It does not work correctly if three or more parties are involved in a session. The module, however, is known to work with various types of H.323-enabled software, including Intel Video Phone, MediaRing GoldenEye, Microsoft NetMeeting, and VocalTec Telephony Gateway. Second, the traffic observed at the

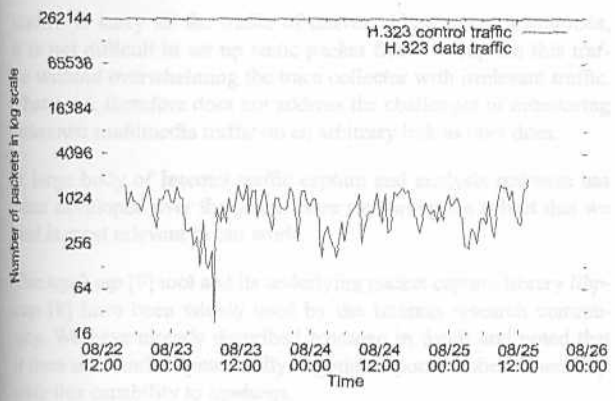

Figure 11: Packet counts for H.323 control versus data traffic

probe point is highly asymmetric. Therefore we incorporate various heuristics to the H.323 module so that it can track a session while seeing traffic flowing in only one direction. For example, if we only see the traffic from caller to callee and not from callee to caller, we will not get the CONNECT message sent by the callee, which contains the callee's port number to follow the subsequent H.245 channel. In this case, we guess that the H.245 port number of the caller is a small increment (one or two) of the caller's Q.931 port number. This appears to work reasonably well in practice.

Traffic Characteristics

Figure 11 shows the amount of aggregated H.323 control traffic (traffic exchanged in Q.931 and H.245 channels) and H.323 data traffic (traffic exchanged in H.323 data channels) over time. Not unexpectedly, the figure shows that the amount of control traffic is significantly lower than the amount of data traffic (note the logarithmic scale on the y-axis) and that there is a positive correlation between the amounts of control traffic and data traffic. Unexpectedly, there does not appear to be any notable pattern in the times of day when sessions occur. This is something we intend to investigate further.

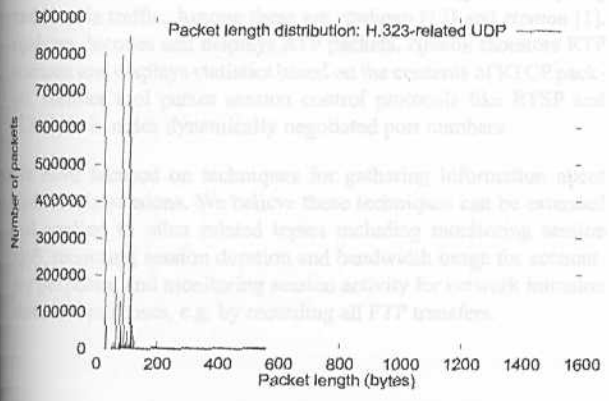

Figure 12: UDP packet length distribution for H.323 related traffic

Packet Length Distribution

Conferencing and packet telephony multimedia applications generally require good real-time performance. Therefore, we expect that these applications prefer to exchange smaller packets with higher packet rate rather than larger packets with lower packet rate. Here we show the packet length distribution for H.323 related UDP traffic in Figure 12. We observe that significant peaks are at packet lengths smaller than 200 bytes, which are shorter than typical MTU sizes.

As for the RTSP results, the packet length distribution for TCP traffic has a familiar distribution with a large peak at 40 bytes corresponding to TCP ACK, FIN, and SIN packets, and several peaks related to different MTU sizes.

Per-Session Statistics

One advantage of using *mmdump* is its ability to track each session individually. We will show an example that derives results based on per-session statistics.

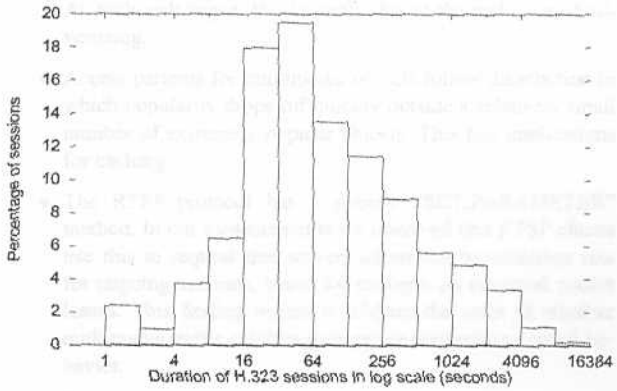

Figure 13: Duration of H.323 sessions

One question of interest is how long an H.323 session lasts. Figure 13 shows a histogram of different ranges of session duration with the percentage of sessions in that range. Here we consider only the subset of sessions for which *mmdump* was able to capture some UDP packets, and discard sessions for which *mmdump* did not capture any UDP packets. The latter can happen if the callee of a session did not answer or had incompatible terminal capabilities with the caller. Session duration is computed as the time between the first packet received (usually the Q.931 SETUP packet) and the last packet received (usually the H.245 FIN packet). The figure shows a majority of calls last between 16 seconds and four minutes. The figure also shows several sessions lasting longer than an hour.

5 Related Work

A recent paper [12] presents a preliminary analysis of streaming media traffic originating from a popular Internet audio service. It is one of the first studies of its kind. However, the set of IP addresses corresponding to the media servers under study was known a priori. In addition, the link under study was close to these servers and was

known to carry all the traffic of interest. Under those conditions, it is not difficult to set up static packet filters to capture this traffic without overwhelming the trace collector with irrelevant traffic. That work therefore does not address the challenges of monitoring unknown multimedia traffic on an arbitrary link as ours does.

A large body of Internet traffic capture and analysis software has been developed over the years. Here we survey the subset that we feel is most relevant to our work.

The *tcpdump* [9] tool and its underlying packet capture library *libpcap* [8] have been widely used by the Internet research community. We have already described *tcpdump* in detail and noted that it does not handle dynamically negotiated port numbers. *mmdump* adds this capability to *tcpdump*.

Online extraction of application specific information, mainly to reduce the volume of generated data, has been reported in [5] and [10]. A software engineering approach similar to our own, is presented in [5] where *tcpdump* has been extended to perform online extraction of HTTP information. A more generic measurement platform, called Windmill, is described in [10]. This platform is meant to run continually providing the means to perform several "experiments" without ever terminating the Windmill instantiation. Since different experiments might be interested in different packet streams, the platform has the ability to dynamically modify the packet-filter expression. This change in packet filter expression is however performed at the time granularity of different experiments, not on the per-multimedia-stream timescales that *mmdump* deals with.

CoralReef [3] is an evolving suite of tools for collecting and analyzing Internet traffic. It is built upon the *libcoral* packet monitoring library and aims for flexibility and high performance. To our knowledge, CoralReef does not yet handle dynamically negotiated port numbers.

Narus [13] and Packeteer [14] have introduced commercial traffic capture and analysis products that reportedly handle dynamically negotiated port numbers. However, we have not had the opportunity to evaluate these products. To our knowledge, their internal details have not been made public and their source code is not available.

There are a number of tools tailored to monitoring and analyzing multimedia traffic. Among these are *rtpdump* [17] and *rtpmon* [1]. *rtpdump* decodes and displays RTP packets. *rtpmon* monitors RTP sessions and displays statistics based on the contents of RTCP packets. Neither tool parses session control protocols like RTSP and H.323, or handles dynamically negotiated port numbers.

We have focused on techniques for gathering information about multimedia sessions. We believe these techniques can be extended and applied to other related topics including monitoring session QoS, recording session duration and bandwidth usage for accounting purposes, and monitoring session activity for network intrusion detection purposes, e.g. by recording all FTP transfers.

6 Conclusions and Future Work

We have presented the design, implementation, and use of a new tool for monitoring multimedia traffic on the Internet. *mmdump* is based on *tcpdump* and further incorporates several novel features that make it practical to monitor multimedia traffic on an arbitrary link. One, it employs protocol-specific parsers to determine which port numbers are dynamically selected for media transport by multimedia session control protocols. Two, it maintains per-session state to record information such as session start/end times, media types, associated traffic. Three, it uses heuristics to deal with incomplete information due to asymmetric routing.

We have been using *mmdump* to monitor traffic from RTSP and H.323 sessions in AT&T's commercial IP network. The tool has already helped uncover a number of interesting features of this traffic:

- Multimedia sessions have a rich structure. We have seen examples of seemingly simple news clip presentations which are composed of more than 10 objects transferred over different port numbers and from multiple servers in different domains. As with web pages, this is partly due to the inclusion of advertising.
- Access patterns for multimedia objects follow distribution in which popularity drops off quickly outside a relatively small number of extremely popular objects. This has implications for caching.
- The RTSP protocol has a generic "SET_PARAMETER" method. In our measurements we observed that RTSP clients use this to request that servers adjust the transmission rate for ongoing sessions, based for example on observed packet losses. This finding begins to address the issue of whether multimedia traffic exhibits appropriate congestion-control behavior.
- The duration of H.323 sessions vary greatly, from a few seconds to over an hour. A majority of sessions last between 16 seconds and four minutes. This distribution of call durations is similar but not identical to that of traditional long-distance telephone traffic.

In terms of ongoing and future work, we have recently added to *mmdump* a different and more complete H.323 parser than the one described in this paper. We are experimenting with using it to monitor the quality of service in a commercial voice-over-IP trial. We have also developed a rudimentary SIP parser to add to the existing RTSP and H.323 parsers. In order to improve the performance of dynamic port processing, we are looking into adopting a modified BPF+ [2] that includes compiler support for incremental filter updates. Finally, we continue to use *mmdump* to monitor multimedia traffic on the public Internet and plan to perform a more thorough analysis of this traffic's growth and characteristics.

Acknowledgements

The RTSP parser was derived from a public-domain RTSP implementation by RealNetworks [16]. The initial H.323 parser was derived from software developed at Columbia University by Christo-

pher Kang. A later version of *mmdump* used the H.323 parser developed in the OpenH323 project (www.openh323.org).

References

[1] BACHER, D., SWAN, A., AND ROWE, L. A. rtpmon: A Third-Party RTCP Monitor. `http://bmrc.berkeley.edu/people/drbacher/projects/mm96-demo/index.html`.

[2] BEGEL, A., MCCANNE, S., AND GRAHAM, S. L. BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture. Proc. ACM SIGCOMM '99, August 1999.

[3] CAIDA. Coralreef. `http://www.caida.org/Tools/CoralReef/`.

[4] CNN. `http://www.cnn.com`.

[5] FELDMANN, A. Continuous online extraction of HTTP traces from packet traces. Proc. W3C Web Characterization Group Workshop, November 1998.

[6] HANDLEY, M., SCHULZRINNE, H., SCHOOLER, E., AND ROSENBERG, J. SIP: Session Initiation Protocol. RFC 2543, March 1999.

[7] Recommendation H.323: Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-guaranteed Quality of Service. ITU-T, 1996.

[8] JACOBSON, V., LERES, C., AND MCCANNE, S. pcap - Packet Capture library. UNIX man page.

[9] JACOBSON, V., LERES, C., AND MCCANNE, S. tcpdump - dump traffic on a network. UNIX man page.

[10] MALAN, G. R., AND JAHANIAN, F. An Extensible Probe Architecture for Network Protocol Performance Measurement. Proc. of ACM SIGCOMM'98, August 1998.

[11] MCCANNE, S. R., AND JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. Proc. 1993 Winter USENIX Technical Conference, January 1993.

[12] MENA, A., AND HEIDEMANN, J. An Empirical Study of Internet Audio Traffic. Proc. IEEE Infocom 2000, March 2000.

[13] NARUS. `http://www.narus.com`.

[14] PACKETEER. `http://www.packeteer.com/`.

[15] REALNETWORKS. `http://www.real.com`.

[16] REALNETWORKS. RTSP: Reference Implementation. `http://www.real.com/devzone/library/fireprot/rtsp/index.html`.

[17] SCHULZRINNE, H. rtpdump. `http://www.cs.columbia.edu/~hgs/rtp/rtpdump.html`.

[18] SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.

[19] SCHULZRINNE, H., RAO, A., AND LANPHIER, R. Real Time Streaming Protocol (RTSP). RFC 2336, April 1998.

[20] W3C. SMIL: Synchronized Multimedia Integration Language. `http://www.w3.org/AudioVideo/#SMIL`.

ANEXO 2: Comparison of H.323 and SIP for IP Telephony Signaling

Ismail Dalgic and Hanlin Fang. Proc. of Photonics East, Boston, Massachusetts. 20-22 Septiembre, 1999.

Comparison of H.323 and SIP for IP Telephony Signaling

Ismail Dalgic[a], Hanlin Fang[b]

[a]3Com Corporation
Technology Development Center
5400 Bayfront Plaza, M/S 3219, Santa Clara, CA 95052

[b]Cisco Systems
170 West Tasman Dr., San Jose, CA 95134

ABSTRACT

Two standards currently compete for the dominance of IP telephony signaling: the H.323 protocol suite by ITU-T, and the Session Initiation Protocol (SIP) by IETF. Both of these signaling protocols provide mechanisms for call establishment and teardown, call control and supplementary services, and capability exchange. We investigate and compare these two protocols in terms of Functionality, Quality of Service (QoS), Scalability, Flexibility, Interoperability, and Ease of Implementation. For fairness of comparison, we consider similar scenarios for both protocols. In particular, we focus on scenarios that involve a gatekeeper for H.323, and a Proxy/Redirect server for SIP. The reason is that medium-to-large IP Telephony systems are not manageable without a gatekeeper or proxy server. We consider all three versions of H.323. In terms of functionality and services that can be supported, H.323 version 2 and SIP are very similar. However, supplementary services in H.323 are more rigorously defined, and therefore fewer interoperability issues are expected among its implementations. Furthermore, H.323 has taken more steps to ensure compatibility among its different versions, and to interoperate with PSTN. The two protocols are comparable in their QoS support (similar call setup delays, no support for resource reservation or class of service (CoS) setting), but H.323 version 3 will allow signaling of the requested CoS. SIP's primary advantages are (i) flexibility to add new features, and (ii) relative ease of implementation and debugging. Finally, we note that H.323 and SIP are improving themselves by learning from each other, and the differences between them are diminishing with each new version.

Keywords: IP Telephony, Voice over IP, H.323, SIP, signaling, call control

1. INTRODUCTION

Telephony service today is provided for the most part over circuit-switched networks, which are referred to as Public Switched Telephone Networks (PSTN). This service is known as Plain Old Telephone Service (POTS). A new trend that is beginning to emerge in recent years is to provide telephony service over IP networks, known as *IP telephony*, or *Voice over IP*.

An important driving force behind IP Telephony is cost savings, especially for corporations with large data networks. The high cost of long-distance and international voice calls – thanks to layers of local and international carriers – is the crux of the issue. A significant portion of this cost originates from regulatory taxes imposed on long-distance voice calls. Such surcharges are not applicable to long-distance circuits carrying data traffic; thus, for a given bandwidth, making a data call is much less expensive than making a voice call.

In addition to the cost savings for long-distance voice calls, carrying voice traffic on the data network within a business building or campus also can achieve substantial cost savings, since the operation of today's proprietary PBX setups is relatively cost-inefficient.

There are other very significant motivating factors for carrying voice traffic over data networks as well. A very important benefit of IP Telephony is the integration of voice and data applications, which can result in more effective business processes. Examples of such applications are integrated voice mail and e-mail, teleconferencing, computer-supported collaborative work and automated and intelligent call distribution. Another benefit is the enabling of many new services both for businesses and for customers. The flexibility offered by IP Telephony by moving the intelligence from the network to the end stations, as well as the open nature of IP networks, are the factors that enable new services.

Furthermore, many of the existing services that require a fee today, such as caller-id, call-forwarding, and multi-line presence become trivial to implement; therefore, such services are likely to be offered for free for competitive reasons.

In order for IP Telephony to gain mainstream acceptance and ultimately replace traditional Plain Old Telephone Service (POTS), two conditions have to be met. First, the *quality* of the voice communication must be at least at the same level as POTS. The two primary aspects of voice quality are the end-to-end delay, and the voice clarity (which depends on many factors, including the voice digitization and compression scheme used, and the amount of lost or late-arrived packets). Therefore, the IP network must be designed such that it can meet the delay and packet loss requirements of the telephony application.

The second condition for the acceptance of IP Telephony is the ease of operation and functionality offered to the end-user at least at the same level as in PSTN. This requires the IP Telephony architecture to provide a *signaling infrastructure* that offers at least the same capabilities and features as the Signaling System 7 (SS7) architecture in PSTN[1]. More specifically, the signaling infrastructure must:

- provide the *functionality* required to set up, manage, and tear down calls and connections;
- be *scalable* to support a very large number of registered endpoints (in the order of billions worldwide), and a very large number of simultaneous calls (in the order of millions worldwide);
- support *network management* features for policy control, accounting, billing, etc;
- provide a mechanism to communicate and set up the *Quality of Service* requested by the end points;
- be *extensible* to help with adding new features easily;
- support *interoperability* among different vendors' implementations, among different versions of the signaling protocol, and with different signaling protocols.

Two standards compete for IP Telephony signaling. The older and currently more widely accepted standard is the ITU-T recommendation H.323[2], which defines a multimedia communications system over packet-switched networks, including IP networks. The other standard, Session Initiation Protocol (SIP)[3], comes from the IETF MMUSIC working group. In this paper, we compare these two standards for IP Telephony signaling from the above listed points of view. Other papers exist which compare H.323 and SIP; among them the most notably is the one by Schulzrinne and Rosenberg[4]. Our paper differs from the previous comparisons in the following ways: (i) we consider the new features introduced in H.323 version 3; (ii) we present a comprehensive evaluation, considering many different aspects; and (iii) we focus on the features and characteristics that are relevant to IP telephony.

The remainder of the paper is organized as follows. In Section 2, we provide a brief overview of the two signaling protocols. In Section 3, we describe our comparison in terms of the areas listed above. We present our concluding remarks in Section 4.

2. OVERVIEW OF H.323 AND SIP

2.1. H.323 Overview

Name	The description of protocols
H.323	Specification of the system
H.225.0	Call control (RAS), call setup (Q.931-like protocol), and packetization and synchronization of media stream
H.235	Security protocol for authentication, integrity, privacy, etc.
H.245	Capability exchange communication and mode switching
H.450	Supplementary services including call holding, transfer, forwarding, etc
H.246	Interoperability with circuit-switched services
H.332	For large size conferencing
H.26x	Video codecs including H.261 and H.263
G.7xx	Audio codecs including G.711, G.723, G.729, G.728, etc

Table 1: ITU-T recommendations that are part of the H.323 specification.

H.323 is an ITU-T Study Group 16 recommendation that specifies a system and protocols for multimedia communications over packet-switched networks. In particular, H.323 consists of a set of protocols that are responsible for encoding, decoding, and packetizing audio and video signals, for call signaling and control, as well as for capability exchange. Most of the existing H.323 implementations are based on the second version of the standard, which was decided in February 1998. The third version is expected to be decided in September 1999. In this paper, we discuss both versions of the standard.

H.323 is an umbrella specification, and various aspects of H.323 are specified in different ITU-T recommendations. Table 1 shows the recommendations that are part of the H.323 specification.

2.1.1. H.323 Endpoint Types

H.323 defines four major components for a network-based communication system: Terminals, Gateways, Gatekeepers and Multipoint Control Units (MCUs). (See Figure 1.)

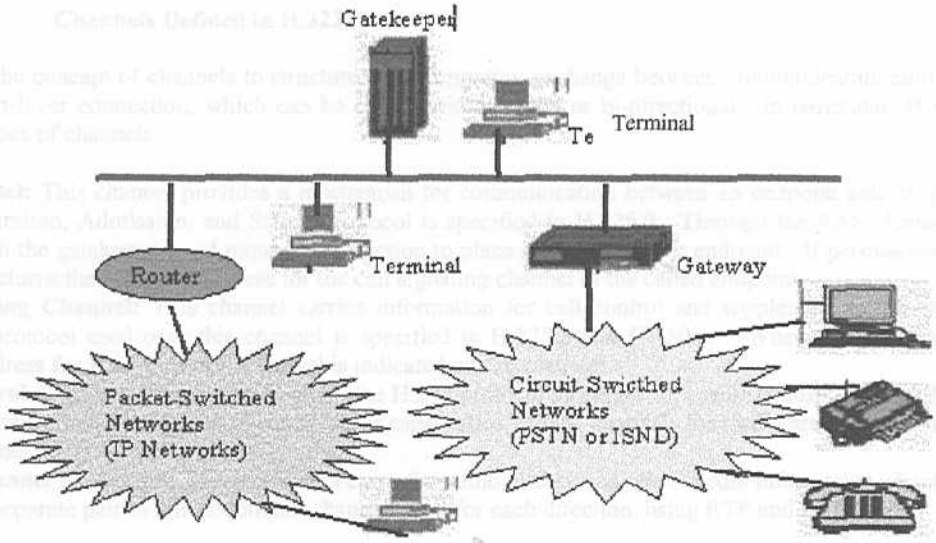

Figure 1: H.323 endpoint types.

Terminals are client endpoints on IP-based networks that provide real-time, two-way communications with other H.323 entities. H.323 terminals are required to support the following three functional parts:

Signaling and Control: H.323 must support H.245, a complex standard for channel usage and capabilities, in addition to a Q.931-like protocol defined in H.225 for call signaling and establishment, as well as Registration/Administration/Status (RAS) protocol defined in H.225 for communication with gatekeepers. All of these protocols use ASN.1 encoding for their messages.
Real-time communication: H.323 terminals must support RTP/RTCP, a protocol for sequencing audio and video packets.
Codecs: Codecs are pieces of software that compress audio/video before transmission and decompress them back after receiving compressed packets. For interoperability purposes, every H.323 terminal is required to support the G.711 audio codec. Other audio and video codecs are optional.

Gateways provide the connection path between the packet-switched network and the Switched Circuit Network (SCN, which can be either public or private). The gateway is not required when there is no connection to other networks. In general, a gateway deflects the characteristics of a LAN endpoint to a SCN endpoint, and vice-versa. Gateways perform call setup and control on both the packet-switched network and on the SCN, and they translate between transmission formats and between communication procedures. Some gateways can also translate between different codec standards

for audio and/or video (referred to as *transcoding*), with the purpose of reducing the bandwidth of the audio/video flow if the SCN bandwidth is limited.

Gatekeepers are optional on an H.323 system, but they have certain mandatory functions if they are present. Gatekeepers perform four required functions: Address Translation (from alias addresses or phone numbers to transport addresses*), Admission Control, Bandwidth Control and Zone Management. Gatekeepers can also support four optional functions: Call Control Signaling, Call Authorization, Bandwidth Management and Call Management. When a gatekeeper is present on an H.323 system, all other types of endpoints are required to register with the gatekeeper and receive its permission prior to making a call.

Multipoint Control Units (MCU) support conferencing between three or more endpoints. The MCU typically consists of a Multipoint Controller (MC) and zero or more Multipoint Processors (MP). MC provides the control functions such as negotiation between terminals and determination of common capabilities for processing audio and video. MP performs the necessary processing on the media streams for a conference. Such processing typically involves audio mixing and audio/video switching.

2.1.2. Channels Defined in H.323

H.323 uses the concept of channels to structure the information exchange between communication entities. A channel is a transport-layer connection, which can be either unidirectional or bi-directional. In particular, H.323 defines the following types of channels:

RAS Channel: This channel provides a mechanism for communication between an endpoint and its gatekeeper. The RAS (Registration, Admission, and Status) protocol is specified in H.225.0. Through the RAS channel, an endpoint registers with the gatekeeper, and requests permission to place a call to another endpoint. If permission is granted, the gatekeeper returns the transport address for the call signaling channel of the called endpoint.

Call Signaling Channel: This channel carries information for call control and supplementary service control. The Q.931-like protocol used over this channel is specified in H.225.0 and H.450.x. When the call is established, the transport address for H.245 Control Channel is indicated on this channel.

H.245 Control Channel: This channel carries the H.245 protocol messages for media control with capability exchange support. After the call participants exchange their capabilities, logical channels for media are opened through the H.245 control channel.

Logical Channel for Media: These channels carry the audio, video, and other media information. Each media type is carried in a separate pair of uni-directional channels, one for each direction, using RTP and RTCP.

H.323 specifies that the RAS channel and the logical channels for media are carried over an unreliable transport protocol, such as UDP. The H.245 control channel is specified to be carried over a reliable transport protocol, such as TCP. H.323 versions 1 and 2 specify that the call signaling channel is carried over a reliable transport protocol. In version 3, this channel can optionally be carried over an unreliable transport protocol.

2.2. SIP Overview

IETF has also specified a multimedia communications protocol suite. In the IETF architecture, the media flows are carried using RTP, just like in H.323. Therefore, the main difference between H.323 and IETF specifications is how the call signaling and control is achieved.

The primary protocol that handles call signaling and control in the IETF specification is SIP. SIP is an application layer control protocol that can establish, modify and terminate multimedia sessions or calls. There are two major architectural elements to SIP: the user agent (UA), and the network server. The UA resides at the SIP end stations, and contains two components: a user agent client (UAC) which is responsible for issuing SIP requests, and a user agent server (UAS), which responds to such requests. There are three different network server types, a redirect server, a proxy server, and a registrar. A basic SIP call does not need servers, but some of the more powerful features depend upon them. To the

---

* When the packet-switched network is an IP network, a transport address is defined as the combination of an IP address and a TCP or UDP port number.

first degree of approximation, the SIP User Agent is equivalent to a H.323 terminal (or the packet-network side of a gateway), and the SIP network servers are equivalent to a H.323 gatekeeper.

The most generic SIP operation involves a SIP UAC issuing a request, a SIP proxy server acting as end-user location discovery agent and a SIP UAS accepting the call. A successful SIP invitation consists of two requests: INVITE followed by ACK. The INVITE message contains session description that informs the called party what type of media the caller can accept and where it wishes the media data to be sent. SIP addresses are referred to as SIP Uniform Resource Locators (SIP-URLs), which are of the form sip:user@host.domain. SIP message format is based on the Hyper Text Transport Protocol (HTTP) message format, which uses a human-readable, text-based encoding.

Redirect servers process an INVITE message by sending back the SIP-URL where the callee is reachable. Proxy servers perform application layer routing of the SIP requests and responses. A proxy server can either be stateful or stateless. A stateful proxy holds information about the call during the entire time the call is up, while a stateless proxy processes a message and then forgets everything about the call until the next message arrives. Furthermore, proxies can either be forking or non-forking. A forking proxy can, for example, ring several phones at once until somebody takes the call. Registrar servers are used to record the SIP address (called a SIP URL) and the associated IP address. The most common use of a registrar server is to register after start-up, so that when an INVITE request arrives for the SIP URL used in the REGISTER message, the proxy or redirect server forwards the request correctly. Note that usually a SIP network server implements a combination of different types of servers.

SIP is used to establish, modify, and terminate multimedia sessions. However, it only handles the communication between the caller and the callee, the endpoint addressing, and user location. There needs to be a description about a multimedia session within a SIP request and response message, as well as an announcement for a session. IETF Session Description Protocol (SDP)[5] is used together with SIP to accomplish all the call signaling functions in IP telephony. Roughly speaking, SIP is the equivalent of RAS and the Q.931-like protocol in H.323. SDP is the equivalent of H.245.

3. COMPARISON OF H.323 AND SIP FOR IP TELEPHONY SIGNALING

We compare H.323 and SIP in terms of Functionality, Quality of Service (QoS), Scalability, Flexibility, Interoperability, Security, and Ease of Implementation. For fairness of comparison, we consider similar scenarios for both protocols. In particular, we focus on scenarios that involve a gatekeeper for H.323, and a Proxy/Registrar server for SIP. The reason is that medium-to-large IP Telephony systems are not manageable without a gatekeeper or a proxy server.

3.1. Functionality

In addition to the basic telephone call service, both SIP and H.323 support some call control services, advanced features, and capability exchange. Roughly, the services they provide are similar but with different approaches. We will discuss the detailed signaling procedure for some services, then summarize their characteristics.

3.1.1. Basic Call Setup and Tear Down

H.323 v2 call setup is based on reliable transport protocol. Therefore, the call setup needs a two-phase connection: TCP connection and call connection. The coming H.323 v3 supports both TCP and UDP, which simplifies the call setup procedure. (See Figures 2, and 3 for call setup in H.323 v2 and v3, respectively.) SIP call setup procedure is similar to H.323 v3. (See Figure 4 for Call setup in SIP.)

The tear down procedure is a reverse of the call setup. Either caller or callee can terminate a call by RELEASE COMPLETE (in H.323) or BYE (in SIP) message.

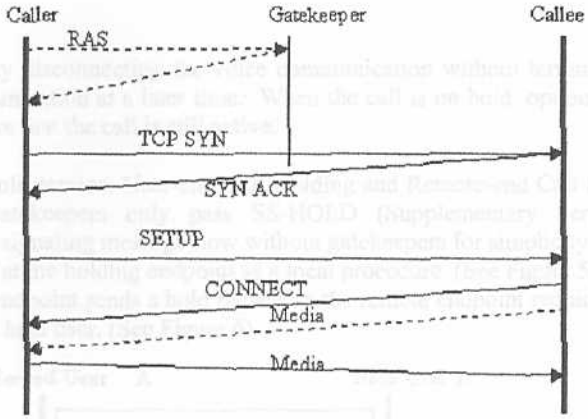

Figure 2: Call Setup in H.323 v2

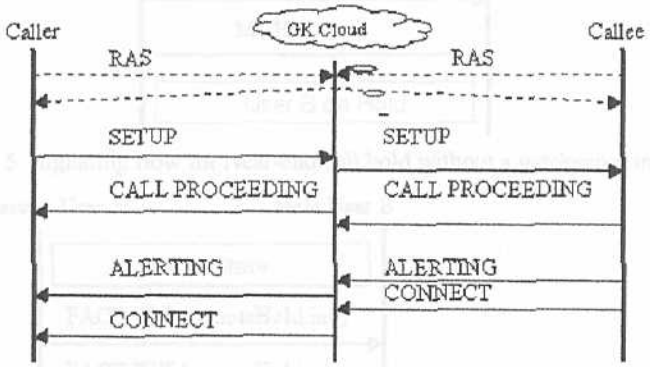

Figure 3: Call Setup, H.323 v3 using UDP (both Endpoints registered, Gatekeper routed call setup)

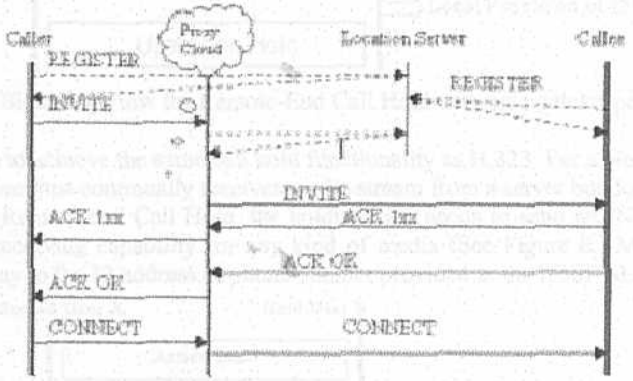

Figure 4: Call Setup with SIP (both endpoints registered, proxy routed call setup)

3.1.2. Call control services

SIP and H.323 both support call hold, call transfer, call forwarding, call waiting, conferencing, and some other supplementary services. In the following, we examine some example supplementary services, namely, call hold, call transfer, call forwarding, and call waiting.

Call Hold

Call Hold is defined as one call party disconnecting the voice communication without terminating the call, with the ability to reestablish the voice communication at a later time. When the call is on hold, optionally some music can be played, so that the party on hold knows that the call is still active.

H.323 defines two scenarios in call hold service: Near-end Call Holding and Remote-end Call Holding. Both can work with or without a gatekeeper. Gatekeepers only pass SS-HOLD (Supplementary Service-HOLD) operation transparently. (Thus, we illustrate the signaling message flow without gatekeepers for simplicity.)
Near-end Call Hold: Hold is invoked at the holding endpoint as a local procedure. (See Figure 5)
Remote-end Call Hold: The holding endpoint sends a hold request to the remote endpoint requiring the held endpoint to provide Music on Hold (MOH) to the held user. (See Figure 6)

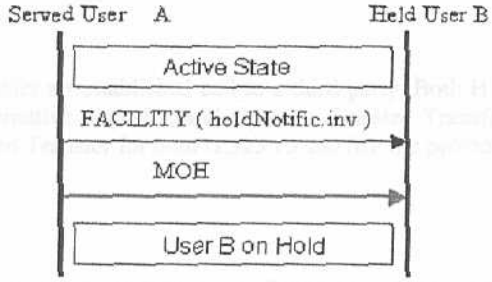

Figure 5: Signaling flow for Near-end call hold without a gatekeeper in H.323

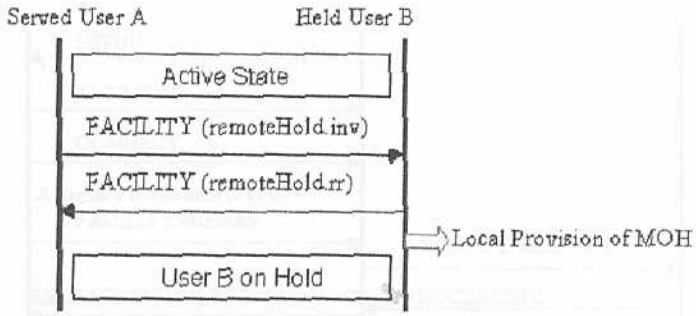

Figure 6: Signaling Flow for Remote-End Call Hold without a gatekeeper in H.323

SIP uses a simpler approach to achieve the same call hold functionality as H.323. For a Near-end Call Hold, no protocol assistance is needed. The client just continually receives media stream from a server but does not generate any response. (See Figure 7.) To achieve Remote-end Call Hold, the holding side needs to send an INVITE message to other side, indicating a NULL set of receiving capability for any kind of media (See Figure 8.) MOH can be implemented by asking an RTSP server to play to the IP address or phone number provided in the RTSP SETUP request.

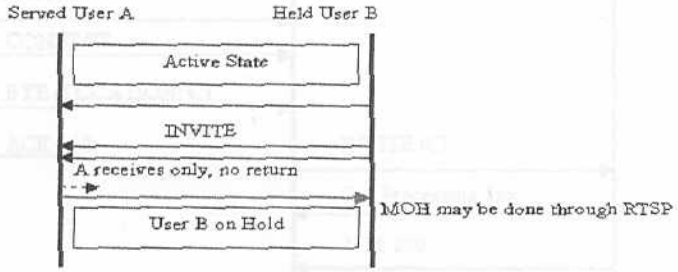

Figure 7: Near-end Call Hold in SIP

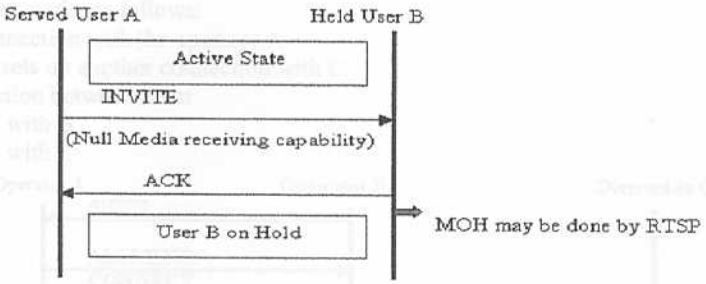

Figure 8: Remote-end Call Hold in SIP

Call Transfer

Call Transfer enables a user to transfer an established call to a third party. Both H.323 and SIP support three types of Call Transfer: Blind Transfer, Alternative Transfer, and Operator-Assisted Transfer. The signaling flow diagrams of Blind Transfer and Operator-Assisted Transfer for both H.323 v3 and SIP are provided in Figures 9, 10, and 11.

Blind Transfer works as follows:
Originator A connects with B
A asks B to connect with C
A simply disconnects with B without any acknowledgement of connection between B and C

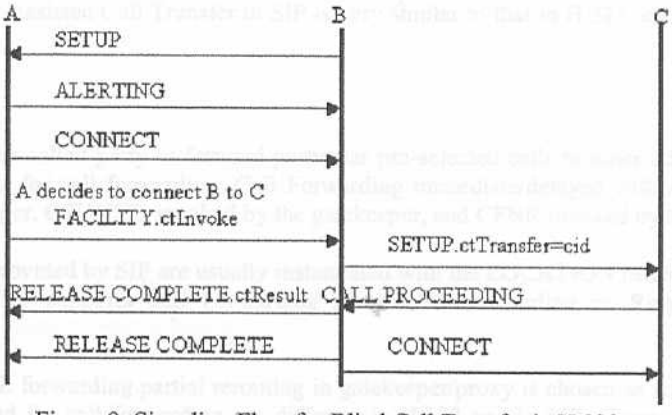

Figure 9: Signaling Flow for Blind Call Transfer in H.323

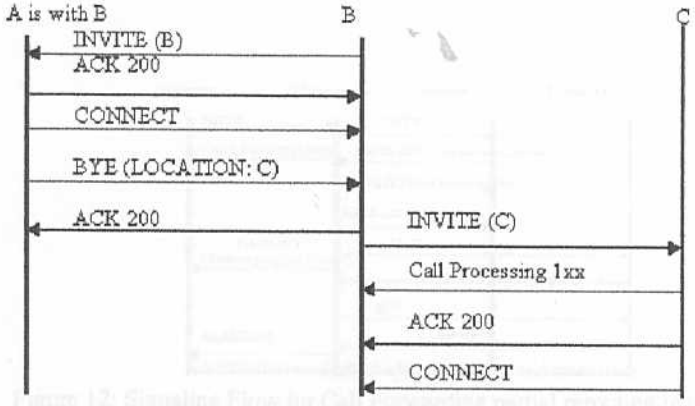

Figure 10: Signaling Flow for Blind Call Transfer in SIP

Operator-Assisted Transfer works as follows:
Originator B sets up a connection with the operator A
A puts B on HOLD, then sets up another connection with C
B and C set up the connection between them
A releases the connection with B
A releases the connection with C

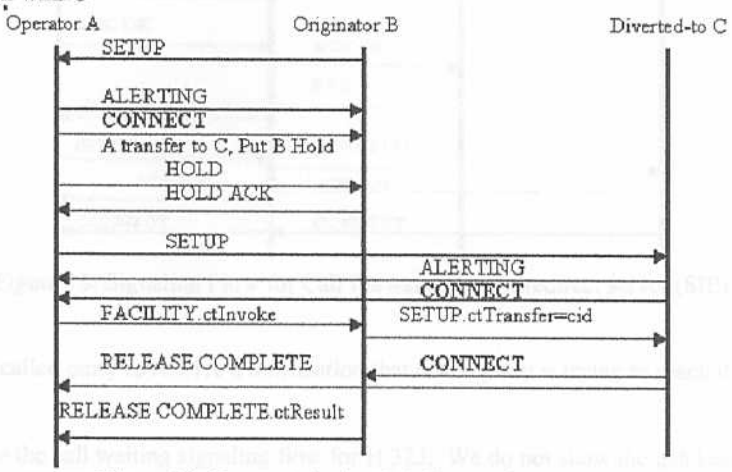

Figure 11: Operator-Assisted Call Transfer in H.323

The procedure of Operator-Assisted Call Transfer in SIP is very similar to that in H.323, except that the equivalent SIP messages are sent out.

Call Forwarding

Call Forwarding permits the called party to forward particular pre-selected calls to other addresses. H.323 defines the following operation models for call forwarding: Call Forwarding immediate/delayed with rerouting, Call Forwarding partial rerouting in gatekeeper, CFU/CFB invoked by the gatekeeper, and CFNR invoked by the gatekeeper.

Call Forwarding services provided by SIP are usually instantiated with the LOCATION header fields, which contain the forwarding destination. SIP supports Call Forwarding Busy, Call Forwarding no Response, and Selective Call Forwarding.

A more general model, Call forwarding partial rerouting in gatekeeper/proxy is chosen as an example. (See Figures 12 and 13) The messages used for call forwarding are different in H.323 and in SIP. However, the call flows are very similar.

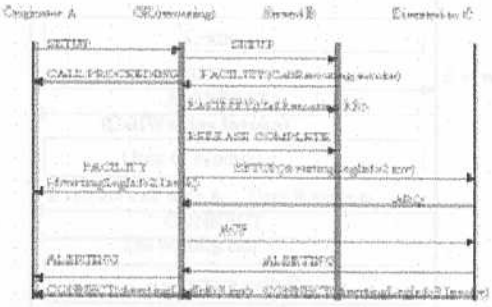

Figure 12: Signaling Flow for Call Forwarding partial rerouting in GK (H.323)

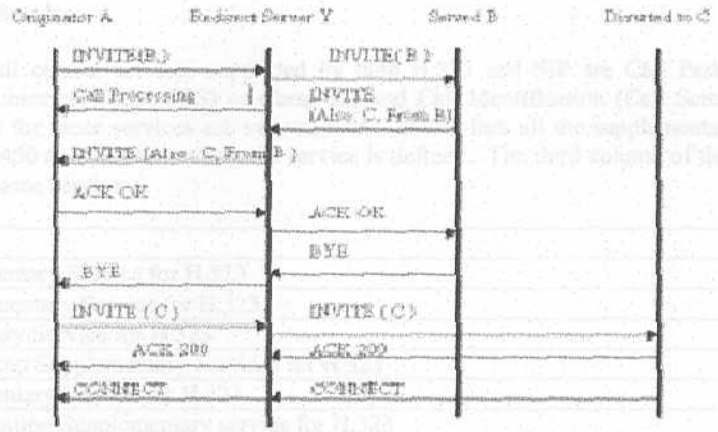

Figure 13: Signaling Flow for Call Forwarding with redirect server (SIP)

Call Waiting

Call Waiting allows the called party to receive a notification that a new party is trying to reach it while it is busy talking to another party.

In Figure 14 we illustrate the call waiting signaling flow for H.323. We do not show the gatekeeper for simplicity, since gatekeepers only pass on SS-CW (Supplementary Service-Call Waiting) operations transparently. We consider that party C calls the party B while B is in another call with A.

Actions at the served endpoint B: B returns an ALERTING message to C. B also optionally starts a timer, and locally provides a call indication to the user. If the served user B likes to accept the waiting call, B stops the timer, and sends a CONNECT message to the calling point.

Action at the calling endpoint C: On receipt of an ALERTING message, the calling endpoint may indicate call waiting to the calling user. Then the calling user may wait until the waiting call gets accepted, release the call, or choose other supplementary services.

SIP can provide call waiting service using the Call-Disposition header field, which allows the UAC to indicate how the server is to handle the call. The following is an example of Call Waiting Service provided by SIP.
The called party B is temporarily unreachable (e.g. it is in another call).
The caller indicates that it wants to have its call queued rather than rejected immediately via a "Call-Disposition: Queue" header field.
If the call is queued, the server returns "181 Queued"
When the callee becomes available, it will return the appropriate status response.
A pending call can be terminated by a SIP BYE request.

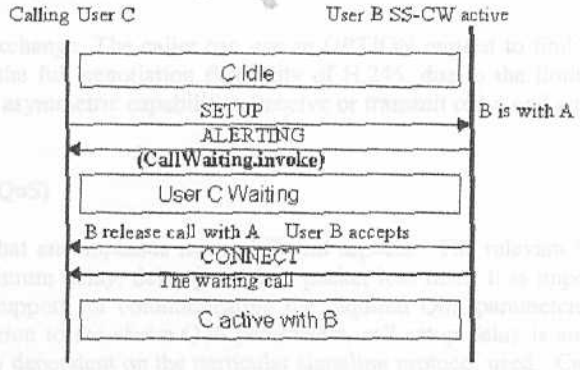

Figure 14: Signaling Flow for Call Waiting (H.323)

Other Supplementary Services

Other supplementary call control services supported by both H.323 and SIP are Call Park and Call Pickup, Call Completion on Busy Subscriber (SS-CCBS) or Camp-on, and Call Identification (Call Screening). H.323 and SIP signaling flow diagrams for those services are very similar. Table 2 lists all the supplementary services supported in H.323 along with the H.450 specification where the service is defined. The third column of the table indicates whether or not SIP supports the same service.

Service Name	H.323	SIP
Call Transfer Supplementary Service for H.323	H.450.2	Yes
Call Diversion Supplementary Service for H.323	H.450.3	Yes
Call Hold Supplementary Service for H.323	H.450.4	Yes
Call Park and Call Pickup Supplementary Services for H.323	H.450.5	Yes
Call Waiting Supplementary Service for H.323	H.450.6	Yes
Massage Waiting Indication Supplementary service for H.323	H.450.7	No
Conference out of Consultation Supplementary Service for H.323	H.450.8	No
Call Completion on Busy Subscriber for H.323	H.450.9	Yes

Table 2: Supplementary services in H.323 and SIP.

3.1.3. Third-Party Control in SIP

Third-Party control is defined as the ability for a party to set up a call between two other parties without necessarily participating in the call. This feature is currently available only in SIP, although work is in progress to add the same functionality to H.323. The flexible and powerful SIP headers, which are very similar to those used in the Hyper-Text Transfer Protocol (HTTP) make the implementation simple. Third-party control is useful for many scenarios, including:

- A secretary dials for a manager
- Auto-dialer hands call to a telemarketer
- Attended call transfer
- Operator service

3.1.4. Capability Exchange

The capability exchange procedures are intended to ensure that multimedia signals that are transmitted can be received and processed appropriately by the receiving terminal.

H.323 uses the H.245 protocol for exchanging capabilities. The complete set of what a terminal can receive and decode is made known to the other terminal by transmission of its capability set. The terminal's total capabilities are described by a set of CapabilityDescriptor structures, each of which is a single SimultaneousCapabilities structure and a capabilityDescriptorNumber[6]. With this structure, very precise information about each terminal's capabilities can be expressed in a relatively compact structure.

SIP uses SDP for capability exchange. The caller can use an OPTION request to find out the capability of the callee. Currently, SIP does not have the full negotiation flexibility of H.245, due to the limited expressiveness of SDP. For example, SIP does not support asymmetric capabilities (receive or transmit only) and simultaneous capabilities of audio & video encoding.

3.2. Quality of Service (QoS)

Quality of Service is a term that encompasses many different aspects. The relevant QoS parameters for multimedia flows are the bandwidth, maximum delay, delay jitter, and packet loss rate. It is important for the call signaling and control protocols to provide support for communicating the required QoS parameters with the goal of meeting the required QoS levels. In addition to the above QoS parameters, call setup delay is another parameter that affects the perceived QoS, and it is highly dependent on the particular signaling protocol used. Call setup delay is also dependent

on the transport protocol that is used to carry the signaling messages, in particular when some signaling messages are lost, and need to be retransmitted.

Therefore, here we first consider the QoS support by the signaling protocols for the multimedia flows. Then, we examine the call setup delay. Since the call setup delay is affected by error detection and error correction mechanisms, we then describe such mechanisms in H.323 and SIP.

3.2.1. QoS Support for Multimedia Flows

Gatekeepers in H.323 provide a rich set of control and management functions, including address translation, admission control, bandwidth control, and zone management. Some optional functions inside gatekeeper are call control signaling, call authorization, bandwidth management and call management. By contrast, SIP does not supply the management or control functions by itself but relies on other protocols.

Admission Control determines whether the network has sufficient resources to support the QoS required for a call, and accepts or rejects the call accordingly. In order to do admission control, the protocol must handle bandwidth management, call management, and bandwidth control. These are supported by H.323 but not by SIP.

In their current stages, neither H.323 nor SIP supports resource reservation by itself. Both of them recommend using external means for resource reservation, such as the Resource Reservation Protocol (RSVP) by the IETF IntServ working group[7]. RSVP addresses the needs of applications that require QoS, promising per-flow service. The reliance of IntServ on Per-flow State and per-flow processing is an impediment to its deployment in the Internet at large networks[8].

Recently, a new Differentiated Services architecture has drawn more attention. Unlike the IntServ model, in which applications explicitly request QoS reservations, Differentiated Services provides a simpler model: Network providers define a set of service levels that is general, simple and application independent. The applications and the users match their needs to specific service levels based on their performance and policy constrains[9].

H.323 v3 can offer some Differentiated Services based on QoS parameter negotiation (Bit rate, delay, and jitter)[10]. Upon initiation of a call, a terminal may request one of three service classes defined: "Guaranteed Service," "Controlled Service," and "Unspecified Service." Neither SIP nor the older versions of H.323 support a similar functionality.

3.2.2. Call Setup Delay

We define call setup delay as the number of round trips needed for establishing audio communication between the call participants. Call setup delay is very large in H.323 v1; it has been reduced significantly with the fast call setup procedure in H.323 v2. SIP and H.323 v3 both have a significantly more efficient call setup, resulting in relatively small delays.

H.323 uses H.225/Q931 signaling procedures to establish a connection between caller and callee. Depending on whether a gatekeeper is being used or not, a H.323 v1 call can take about 6 to 7 round-trip times, including setting up the Q.931 and H.245 TCP connections.

The fast call setup method is an option specified in H.323 v2 that reduces the number of delays up to three roundtrips involved in establishing a call and initial media streams by including H.245 logical channel information in the SETUP and CONNECT messages. With the fast call setup method, only G.711-based voice communication can be established between the two call parties, since the capabilities are not exchanged. If the parties wish to establish other types of media channels, they can optionally perform the H.245 capability exchange procedures after the G.711 channel is established.

In H.323 v3, either UDP or TCP can be used to carry call setup messages. Using UDP has the advantage that there is no roundtrip delay associated with establishing a transport-layer connection. When UDP is used, the call setup delay can be 1.5 or 2.5 round trips, depending on whether or not a gatekeeper is involved.

SIP call setup is very similar to the one in H.323 v3. However, if the UDP call setup fails, H.323 v3 has some advantages over SIP. H.323 v3 sets up a UDP connection and a TCP connection almost simultaneously, and provides an efficient mechanism to close the TCP connection if the UDP set up is successful. If the UDP setup fails, TCP can take over immediately. SIP operates UDP and TCP sequentially. This increases the call setup delay if the UDP connection is not available.

3.2.3. Error detection and correction

Packet Loss

H.323 v1 and v2 are based on the reliable transport protocol. They achieve the reliability based on transport protocol. The use of TCP would simplify the state machine for the call control protocol, since it has its own flow control, window control, and retransmission mechanisms to ensure the reliability.

H.323 v3 specifies its own retransmission policies for both sender and receiver for support TCP and UDP[11]. The sender starts two timers, T1 and T4, after it sends call setup PDU. If T1 expires before the caller received a response from the callee or GK, the sender retransmits the SETUP packet and starts a timer T3. If T3 expires, another retransmission is performed, and T3 is restarted. After a total of N1 transmission, the caller stops retransmission and reverts to the use of TCP call signaling instead of UDP.

With the first transmission of response, the receiver starts a timer T1. If T1 expires, the callee retransmits the packet and restarts a timer T3. If T3 expires, the callee sends another transmission of response and restarts T3. After the response message has been sent for a total N1 times, the callee stops re-transmitting and starts a timer T5. After T5 expires, the callee discards the conference/call identification information and associated state, and regards the setup of this session as failed.

The reliability of SIP is messages is achieved by having the client retransmit requests every 0.5 seconds until either a progress report (1xx) or final status (>200) response has been received. The server simply retransmits the original final response until an ACK is received. The client retransmits an ACK for every final message.

Loop Detection

Another common type of error is call forwarding loops, which may occur especially when multiple gatekeepers or SIP network servers are involved in setting up a call. There is no provision to prevent loops in H.323 v1 and v2. H.323 v3 defines a PathValue field to indicate the maximum number of gatekeepers that signaling message should traverse before being discarded. Using the PathValue field can reduce the rate of loop occurrence, but not as efficiently as the loop detection algorithm used in SIP. When a loop occurs without knowing the names of gatekeepers, the signaling messages will not be stopped until reaching the PathValue. Thus how to define a proper value for PathValue becomes a critical issue. Furthermore, when the network configuration changes, the PathValue may possibly need to be changed.

SIP provides loop detection algorithms similar to the one used in BGP (Border Gateway Protocol) to prevent searching loop[3]. It works through the *via* header field. Before a proxy server redirects any request, it checks the *via* field. If its own name is already in there, then a loop must have occurred. If its name is not on the list of via field, then the proxy server posts its name on the list, then transmits the request to another proxy server or endpoint.

Fault Tolerance

Errors happen in networks for a variety of reasons. The signaling protocol should have the capability to bypass network faults and provide a normal service whenever possible. A client should not notice the error during the services. This capability is called Fault Tolerance.

H.323 v3 provides better fault tolerance than SIP by redundant gatekeepers and endpoints. During registration, a gatekeeper may indicate alternate gatekeepers to the registering endpoint, which may be used in the event of a primary gatekeeper failure. Likewise, an endpoint may indicate a backup, redundant or alternate Transport Address. This allows an endpoint to have a secondary network interface or a secondary H.323 endpoint as a backup.

3.3. Scalability

In a fully operational system, it should be possible for every Internet host to act as an IP telephony client (albeit not all of them participating in a call simultaneously). It is estimated that the worldwide number of Internet users will reach 500 million by the year 2000, with the number increasing exponentially. Therefore, scalability of the current signaling protocols is extremely important. Many different aspects affect a system's scalability. We compare the scalability of H.323 and SIP in terms of their Complexity, Endpoint Location, Server Processing, Inter-Server Communication, Global Addressing, and Multipoint Communication.

3.3.1. Complexity

H.323 is a rather complex protocol. It includes H.225 for call signaling, H.245 for call control, H.332 for large conferences, H.450.x (x=1,2,...9) for supplementary services, H.235 for security and encryption, and H.246 for interoperability with circuit-switched services. Many services require interactions between those sub-protocols, which increases the complexity but decreases the scalability.

On the other hand, SIP and SDP are less complicated. A basic SIP Internet Telephony implementation can be done using four headers (To, From, Call-ID, and Cseq) and three request types (INVITE, ACK, and BYE)[3]. This simplifies programming and maintenance, and better scalability is a consequence.

3.3.2. Server Processing

No connection states are required in UDP. Therefore, large backbone servers based on UDP can operate in a stateless manner. This significantly reduces the memory requirements and improves the scalability.

In SIP, a transaction through servers and gateways can be either stateful or stateless. The stateless model simplifies the memory management, and the stateful model provides the sufficient information to forward the response correctly. H.323 v1 and v2 server processing is stateful, in which TCP was chosen as the transport protocol. Gatekeepers must hold the call states, as well as the TCP connections for the duration of a call. H.323 v3 supports the stateless processing model just as in SIP.

3.3.3. Endpoint Location

The current mechanism for logical addressing and addressing resolution in H.323 standard is to utilize aliases (E.164 or H323ID) and a mapping mechanism supported by gatekeepers. When a client likes to make a connection, the gatekeeper may either return the endpoint's address to the client (in direct call model) or route the SETUP message to the called endpoint (in GK routed model). H.323 v3 defines a mechanism for inter-gatekeeper communication, which aids in locating an endpoint registered in a different zone or administrative domain.

SIP chooses an e-mail-like address, referred to as a *SIP URL*[3]. When a client wishes to send a request, it either sends it to a locally configured SIP proxy server or a SIP redirect server, independent of the Request-URL, or sends it to the IP address and port corresponding to the Request-URL. In the former case, SIP redirect or proxy server obtains information about a callee's possible location(s) from SIP location servers. SIP does not specify a means to locate endpoints registered in other administrative domains, and suggests the use of external mechanisms such as DNS.

3.4. Flexibility

A well-defined protocol should have the capability to extend the current functionality for further development, and should allow implementers to customize sub-components depending on individual interest. In this section, H.323 and SIP will be evaluated in terms of those aspects.

3.4.1. Extensibility of Functionality

IP telephony technology is not yet mature. It is likely that new signaling capabilities and functionality need to be added. Also, different vendors may want to support additional features.

H.323 chooses vendor-defined NonStandardParam field in ASN.1 as its extension mechanism. NonStandardParam consists of vendor codes and an opaque code for that particular vendor. This approach has some limitations, as only the NonStandardParam field can be extended.

SIP offers a more flexible mechanism by providing a hierarchical namespace of feature names and hierarchically organized numerical error codes. The client inputs feature information in a SIP Require header. If there are some required features that the server does not support, the server sends back a hierarchical error code to the client. New features can either be registered with Internet Assigned Numbers Authority (IANA) or hierarchically derived from the feature owner's Internet domain name depending on whether the new feature can be derived directly.

3.4.2. Ease of customization

To customize the services, H.323 requires more interactions between its sub-protocols. For example, if the conference size changes from small to big, then a different protocol, H.332, has to be used. Also, since H.323 requires the full compatibility between each version, the customization will definitely increase the size of code. SIP uses its relatively simple header fields to handle those interactions. The text-based encoding makes the customization in SIP much easier than that in H.323.

3.5. Interoperability

An IP Telephony signaling protocol needs to cooperate with different versions, implementations, and other signaling protocols crossing worldwide networks. This capability is defined as interoperability.

3.5.1. Interoperability among Versions

The fully backward compatibility in H.323 enables all implementations based on different H.323 versions to be seamlessly integrated. This is important for customers. If a customer has bought client side products implemented by H.323 v2, he or she does not have to change anything when server side is upgraded to H.323 v3. The new server supports every function the customer had before with the older server.

In SIP, a newer version may discard some old features that are not expected to be implemented any more. This approach saves code size and reduces protocol complexity, but loses some compatibility between different versions. Some products implemented on the older version may not be supported in the new version.

3.5.2. Interoperability among Implementations

Different vendors on the market may implement the same protocol with different approaches. It is possible that endpoints in a large-scale system use different vendors' products, and simply complying with the standard does not guarantee interoperability among the different products.

H.323 provides an implementers' guide, which clarifies the standard and helps towards interoperability among different implementations. Furthermore, International Multimedia Teleconferencing Consortium (IMTC) iNOW! Activity Group defined an interoperability profile, which combines, clarifies and complements existing standards to provide a complete IP telephony interoperability protocol[12]. IMTC also organizes interoperability events, where different vendors can test their implementations against each other.

SIP, being in an earlier stage of development, thus far has not provided an implementation agreement. The interoperability between different implementations is still uncertain, although the first interoperability tests began recently.

3.5.3. Interoperability with Other Signaling Protocols

To support traditional telephony services, the VoIP signaling protocols have to support ISDN Signaling System 7 (SS7). SS7 performs out-of-band signaling in support of the call-establishment, billing, routing, and information-exchange functions of the public switched telephone networks (PSTN). There are two signaling specifications available in SS7 for

different interfaces: Q.931 used for User-to-Network Interface (UNI) and ISUP used for Network-to-Network Interface (NNI).

H.323 embraces the more traditional circuit-switched approach based on the ISDN/Q.931 protocols. Q.931-like signaling messages are used in H.323 procedures, which makes it easier to interoperate with ISDN/Q.931. However, the call setup messages of H.323 are only a subset of those in SS7/ISUP. Because there is no established standard for relaying of SS7/ISUP messages over an H.323 network, H.323 can only translate a portion of SS7 messages in the conversion.

The H.32x family of recommendations offers specific standards to interoperate with other circuit-switched networks; for example, H.320 for ISDN and B-ISDN, H.324 for GSTN. Within those standards, the interoperability by gateways is well defined.

For SIP, no translation function for SS7 signaling messages is provided. Although there is an Internet draft on the subject, "A Functional Description of SIP-PSTN Gateway,"[13] the detailed information about signaling information transportation is not yet worked out.

3.6. Ease of Implementation

H.323 signaling messages are binary encoded using ASN.1 PER (Packet Encoding Rules)[14]. Within a system, the information represented using an abstract syntax must be mapped into some form for presentation to human users. Similarly, this abstract syntax must be mapped into some local format for storage. Such mappings require a special parser, which makes implementation and debugging more complicated.

SIP messages are text-based, using ISO 10646 in UTF-8 encoding[15]. Text-based encoding allows easy implementation in languages such as JAVA, Tcl and Perl, and easy debugging.

4. CONCLUSIONS

In Table 3, we present a summary of the comparison presented in this paper. Our major conclusions are as follows. In terms of functionality and services that can be supported, H.323 version 2 and SIP are very similar. However, supplementary services in H.323 are more rigorously defined. Therefore, fewer interoperability issues are expected among its implementations. Furthermore, H.323 has better compatibility among its different versions and better interoperability with PSTN. The two protocols are comparable in their QoS support (similar call setup delays, no support for resource reservation or class of service (QoS) setting), but H.323 version 3 will allow signaling of the requested QoS.

SIP's primary advantages are its flexibility to add new features and its relative ease of implementation and debugging. Finally, we note that H.323 and SIP are improving themselves by learning from the other side, and the differences between them are diminishing with each new version.

REFERENCES

1. Bell Atlantic, "Signaling System 7 (SS7) Tutorial", http://www.webproforum.com/bellatlantic2/topic01.html
2. ITU-T Recommendation H.323, "Packet-Based Multimedia Communications Systems", February 1998.
3. M. Handley, H. Schulzrinne, E. Scholler, J. Rosenberg, " SIP: Session Initiation Protocol", RFC2543, IETF, March 1999.
4. H. Schulzrinne, J. Rosenberg, "A Comparison of SIP and H.323 for Internet Telephony", Network and Operating System Support for Digital Audio and Video (NOSSDAV), Cambridge, England, July 1998.
5. V. Jacobson and M. Handley," SDP: Session Description Protocol", RFC2327, IETF, April 1998, work in process.
6. ITU-T Recommendation H.245, "Control protocol for multimedia communication," February 1998.
7. Braden, R., Zhang, L., Berson, S., herzog, S. And Jamin, S.," Resource Reservation Protocol (RSVP) - version 1 Functional Specification," RFC 2205, proposed standard, September 1997.
8. Y. Bernet, R.Yavatkar, P. Ford, F. Baker, L, Zhang, " A Framework for End-to-End QoS Combining RSVP/Intserv and Differentiated Services ," Internet Draft, March, 1998.
9. S. Blake, et. al, "An Architecture for Differentiated Services," RFC 2475, December 1998.

10. "H.323 Differentiated Services and Their Protocol Architectures," ITU Telecommunication Standardization Sector, APC-1492, February, 1999
11. ITU-T Standardization Sector, "H.323 Annex E: call signaling over UDP", APC 1441, September, 1998
12. International Multimedia Teleconferencing Consortium, "iNOW!™ Standards-Based IP Telephony Interoperability Profile," February 23, 1999
13. Steve Donovan, Mattew Cannon, "A Functional Description of SIP-PSTN Gateway", Internet Draft, IETF, November 1998.
14. ITU-T Recommendation X.691, "Information Technology - ASN.1 encoding rules – Specification of Packed Encoding Rules (PER)," Dec. 1997.
15. ISO/IEC 10646-1:1993, "Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane", 1993.

	H.323 v1	H.323 v2	H.323 v3	SIP
FUNCTIONALITY				
CALL CONTROL SERVICES:				
Call Holding	No	Yes	Yes	Yes
Call Transfer	No	Yes	Yes	Yes
Call Forwarding	No	Yes	Yes	Yes
Call Waiting	No	Yes	Yes	Yes
ADVANCED FEATURES:				
Third Party Control	No	No	No	Yes
Conference	Yes	Yes	Yes	Yes
Click-for-Dial	Yes	Yes	Yes	Yes
Capability Exchange	Yes&Better	Yes &Better	Yes &Better	Yes
QUALITY OF SERVICE				
Call Setup Delay	6~7 RT	3~4 RT	2~3 RT	2~3 RT
RELIABILITY:				
Packet Loss Recovery	Through TCP	Through TCP	Better	Better
Fault Detection	Yes	Yes	Yes	Yes
Fault Tolerance	N/A	N/A	Better	Good
MANAGEABILITY				
Admission Control	Yes	Yes	Yes	No
Policy Control	Yes	Yes	Yes	No
Resource Reservation	No	No	No	No
SCALABILITY				
Complexity	More	More	More	Less
Server Processing	Stateful	Stateful	Stateful or stateless	Stateful or Stateless
Inter-Server Communication	No	No	Yes	Yes
FLEXIBILITY				
Transport Protocol Neutrality	TCP	TCP	TCP/UDP	TCP/UDP
Extensibility of Functionality	Vendor Specified			Yes, IANA
Ease of Customization	Harder	Harder	Harder	Easier
INTEROPERABILITY				
VersionCompatibility	N/A	Yes	Yes	Unknown
SCN Signaling Interoperability	Better	Better	Better	Worse
EASE OF IMPLEMENTATION				
Protocol Encoding	Binary	Binary	Binary	Text

Table 3: Comparison summary

ANEXO 3: Código fuente de las cabeceras (.h) de PC-Voz

MMBD

```
/*****************************************************************************/
/* mmbd.h                                                                    */
/*                                                                           */
/*      Cabecera del Módulo Manejador de la Base de Datos (MMBD).            */
/* Encargado de realizar las operaciones contra un Servidor de Base de Datos */
/*                                                                           */
/*****************************************************************************/
/* Hecho por:          Daniel Pons A.                                        */
/* Email:              daniel_pons_1@ucab.edu.ve                             */
/*                                                  U.C.A.B. Caracas. 2002   */
/*****************************************************************************/
/* Comentarios:                                                              */
/*  El Sistema Manejador de Base de Datos MySQL viene incluido en la gran    */
/*  mayoría de distribuciones de Linux en la actualidad. También se puede    */
/*  descargar de manera gratuita en http://www.mysql.com                     */
/*                                                                           */
/*  DBMS -> Sistema Manejador de la Base de Datos                            */
/*                                                                           */
/*****************************************************************************/

#ifndef _MMBD_H

# define _MMBD_H      1

/*****************************************************************************/
// Archivos de cabecera a incluir

#include <stdio.h>
#include <stdlib.h>
#include <mysql/mysql.h>
#include <string.h>
```

```
#include "mmcq931.h"

// Fin de archivos de cabecera a incluir
/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/
// Variables de ámbito global

MYSQL conexion_BD;

// Fin de variables de ámbito global
/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/
// Definición de los prototipos de las funciones

/*****************************************************************************
 función:       Conectar_BD
 descripción:   Función que se conecta a la BD, mantiene una conexión
                persistente
 entradas:      const char *host -> host donde está corriendo MySQL
                 const char *usuario -> usuario con el que se va a autentificar
                 contra el DBMS
                const char *clave -> clave del usuario en cuestión
                const char *BD -> nombre de la base del usuario
 salidas:       int -> devuelve 0 al no poder conectarse, cualquier otro
                valor positivo significa una conexión exitosa
```

```
comentarios:    El usuario debe estar registrado en el DMBS y debe tener esa
                clave asociada. Véase la documentación de MySQL (http://www.mysql.com)
/*************************************************************************/
int Conectar_BD(const char *host, const char *usuario, const char *clave, const char *BD);
/*************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*************************************************************************
función:        Insertar_Paquete
descripción:    Inserta un paquete en la BD, puede ser que inserte:
                        - Una llamada
                        - Una finalización de llamada
                        - Un paquete de audio
entradas:       const struct datos_bd *datos -> datos del paquete
salidas:        float -> identificador en la BD del registro insertado.
                Si hay error se retorna -1
comentarios:
/*************************************************************************/
float Insertar_Paquete(const struct datos_bd *datos);
/*************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*************************************************************************
función:        Esta_DBMS_Montado
descripción:    Verifica que el DBMS esté montado y prestando servicio
entradas:       void
salidas:        int -> si esta variable es -1 no está montado. Cualquier otro
                valor, significa que el DBMS está montado
comentarios:    Véase la documentación de MySQL (http://www.mysql.com)
/*************************************************************************/
int Esta_DBMS_Montado();
```

```
/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************
 función:       Ultimo_ID
 descripción:   Retorna el último ID de la BD del tipo autoincrement
 entradas:      void
 salidas:       float -> último autoincrement insertado
 comentarios:   Véase la documentación de MySQL (http://www.mysql.com)
/*****************************************************************************/
float Ultimo_ID();

// Fin de definición de los prototipos de las funciones
/*****************************************************************************/


#endif


/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/
// Fin de archivo -- mmbd.h
```

MMRD

```
/*************************************************************************/
/*  mmrd.h                                                               */
/*                                                                       */
/*       Este es la cabecera del  Módulo que Maneja Recursos Dinámicos como */
/*  apuntadores a memoria RAM, archivos en disco, etc. Contiene un TDA de */
/*  manejo de listas  simplemente enlazadas                              */
/*                                                                       */
/*************************************************************************/
/* Hecho por:     Daniel Pons A.                                         */
/* Email:         daniel_pons_1@ucab.edu.ve                              */
/*                                                  U.C.A.B. Caracas. 2002 */
/*************************************************************************/
/* Comentarios:                                                          */
/*                                                                       */
/*************************************************************************/

#ifndef _MMRD_H

# define _MMRD_H       1

/*************************************************************************/
// Archivos de cabecera a incluir

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <signal.h>

#include "mmpg.h"

// Fin de archivos de cabecera a incluir
/*************************************************************************/
```

```
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************/
// Variables de ámbito global

struct lista_pids
{
        int tipo;
        float id_llamada;
        pid_t id_proc;
        char *id_filtro;
        char *host_origen;
        char *host_destino;
        struct lista_pids *sig;
};
// Fin de variables de ámbito global
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************/
// Definición de constantes generales

#define  SETUP          1
#define  CONNECT        2
#define  RELEASE        3
// Fin de definición de constantes generales
/*****************************************************************************/
```

```
// Definición de los prototipos de las funciones

/*****************************************************************************
 función:       Crear_Lista
 descripción:   Crea en memoria una lista simple enlazada
 entradas:      void
 salidas:       struct lista_pids * -> lista enlazada en memoria, en caso de
                algún error, esta función devuelve NULL
 comentarios:
*****************************************************************************/
struct lista_pids *Crear_Lista();
/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************
 función:       Insertar_Lista
 descripción:   Inserta un elemento en el principio de la lista
 entradas:      struct lista_pids *lista -> lista en donde insertar
                pid_t id_proc -> PID del proceso
                const char *id_filtro -> filtro de sesión
                int tipo -> tipo mensaje Q.931
                float id_llamada -> identificador de la llamada en la BD
                const char *host_origen -> IP host origen
                const char *host_destino -> IP host destino
 salidas:       struct lista_pids * -> lista enlazada en memoria, en caso de
                algún error, esta función devuelve la lista inicial sin modificaciones
 comentarios:   No verifica si el objeto a insertar está ya en la lista
*****************************************************************************/
struct lista_pids *Insertar_Lista(struct lista_pids *lista, pid_t id_proc, const char *id_filtro, int tipo,
float id_llamada, const char *host_origen, const char *host_destino);
/*****************************************************************************/
                                /******/
                                /******/
```

```
                                        /******/
/*****************************************************************************
 función:       Eliminar_Lista
 descripción:   Elimina el primer elemento encontrado en la lista según el
                filtro
 entradas:      struct lista_pids *lista -> lista en donde eliminar
                const char *id_filtro -> filtro con el cuál se va a hacer la comparación
 salidas:       struct lista_pids * -> lista enlazada en memoria con el ítem insertado al
                principio de la misma, en caso de algún error, o de no encontrarse
                el filtro, esta función retorna la lista inicial sin modificaciones
 comentarios:
*****************************************************************************/
struct lista_pids *Eliminar_Lista(struct lista_pids *lista, const char *id_filtro);
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************
 función:       Eliminar_Lista_y_Matar
 descripción:   Elimina el primer elemento encontrado en la lista según el
                filtro y mato al proceso que tiene ese filtro
 entradas:      struct lista_pids *lista -> lista en donde eliminar
                const char *id_filtro -> filtro con el cuál se va a hacer la
                 comparación
 salidas:       struct lista_pids * -> lista enlazada en memoria a la cuál le
                fue extraído el ítem que encajó con el filtro. En caso de
                algún error, o de no encontrarse el filtro, esta función retorna
                 la lista inicial sin modificaciones
 comentarios:
*****************************************************************************/
struct lista_pids *Eliminar_Lista_y_Matar(struct lista_pids *lista, const char *id_filtro);
/*****************************************************************************/
                                        /******/
                                        /******/
```

```
                                        /******/
                                        /******/
/*****************************************************************************
 función:        Imprimir_Lista
 descripción:    Dispone los datos de la lista por la consola
 entradas:       struct lista_pids *lista -> lista a imprimir
 salidas:        void
 comentarios:
*****************************************************************************/
void Imprimir_Lista(const struct lista_pids *lista);
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************
 función:        Contar_Filtros
 descripción:    Retorna el número de filtros encontrados en la lista y que
                 son iguales a id_filtro
 entradas:       struct lista_pids *lista -> lista en donde buscar
                 const char *id_filtro -> filtro el cuál se quiere contar en
                 la lista
 salidas:        int -> número de filtros encontrados
 comentarios:
*****************************************************************************/
int Contar_Filtros(const struct lista_pids *lista, const char *id_filtro);
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************
 función:        Devolver_ID_Llamada
 descripción:    Devuelve el identificador de llamada de la BD que tenga el
                 ítem con filtro igual a id_filtro
```

```
entradas:       struct lista_pids *lista -> lista en donde buscar
                 const char *id_filtro -> filtro el cuál se quiere buscar el
                ID en la lista
salidas:        float -> ID de la BD. Se devuelve -1 al no encontrase el ID
comentarios:
/*****************************************************************************/
float Devolver_ID_Llamada(const struct lista_pids *lista, const char *id_filtro);
/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************
función:        Liberar_Lista
descripción:    Limpia la memoria que ocupa una lista, con la intención de que
                pueda ser reutilizada
entradas:       struct lista_pids *lista -> lista a borrar
salidas:        int -> Devuelve 1 si la lista fue liberada correctamente y
                retorna 0 en caso contrario
comentarios:
/*****************************************************************************/
int Liberar_Lista(struct lista_pids *lista);

// Fin de definición de los prototipos de las funciones
/*****************************************************************************/

#endif

/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************/
// Fin de archivo -- mmrd.h
```

MPSC

```
/*****************************************************************************/
/* mpsc.h                                                                    */
/*                                                                           */
/*      Cabecera de la implementación del Módulo Proveedor de Servicios de Cifrado.  */
/*                                                                           */
/*****************************************************************************/
/* Hecho por:          Daniel Pons A.                                        */
/* Email:              daniel_pons_1@ucab.edu.ve                             */
/*                                                  U.C.A.B. Caracas. 2002   */
/*****************************************************************************/
/* Comentarios:                                                              */
/*   Usa un cifrado sumamente trivial, es por sustitución. Permite protección */
/*   ante usuarios inexpertos                                                */
/*                                                                           */
/*****************************************************************************/


#ifndef _CIFRADO_TRIVIAL_H

# define _CIFRADO_TRIVIAL_H 1

/*****************************************************************************/
// Archivos de cabecera a incluir

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mmpg.h"

// Fin de archivos de cabecera a incluir

/*****************************************************************************/
```

```
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************/
// Definición de constantes generales

#define SLASH_N                 10
#define NOMBRE_ARCHIVO_BD       "bd.cfg"
#define HOST                    0
#define USUARIO                 1
#define PASSWORD                2
#define DESPLAZO                12

// Fin de definición de constantes generales
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************/
// Definición de los prototipos de las funciones

/*****************************************************************************
 función:       Cifrar
 descripción:   Cifra un string y lo almacena en un archivo
 entradas:      const char *texto -> string a cifrar
                int desplazo -> desplazo de cifrado
 salidas:       int -> se setea esta variable a 1 si se cifra exitosamente y
                en 0 si no se cifra
 comentarios:
*****************************************************************************/
int Cifrar(const char *texto, int desplazo);
/*****************************************************************************/
```

```
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************
 función:       Descifrar
 descripción:   Descifra un archivo y devuelve el string plano
 entradas:      int desplazo -> desplazo de cifrado
 salidas:       char * -> string plano descifrado. Se retorna NULL si se
                encuentra un error
 comentarios:
*****************************************************************************/
char *Descifrar(int desplazo);
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************
 función:       Ver_Datos
 descripción:   Utilizada para organizar en una estructura los datos de la
                conexión a la BD
 entradas:      char *texto -> string al cual extraerle los datos de conexión
 salidas:       struct info_conexion -> Devuelve una estructura con información
                específica de la conexión a la BD. Esto ocurre cuando el archivo
                cifrado es un archivo de conexión a la BD. Se retorna NULL
                 cuando hay algún error
 comentarios:
*****************************************************************************/
struct info_conexion *Ver_Datos(char *texto);
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
```

```
/*****************************************************************************
 función:        Preparar_Texto
 descripción:    Dada una estructura de conexión a la BD, esta función la pasa
                  a string
 entradas:       const struct info_conexion *entrada -> estructura de conexión
                  a la BD
 salidas:        char * -> arreglo de caracteres con los datos de la estructura
                 de conexión. El formato es: host'\n'usuario'\n'clave'\0'. Se
                 devuelve NULL al haber un error
 comentarios:
*****************************************************************************/
char *Preparar_Texto(const struct info_conexion *entrada);

// Fin de definición de los prototipos de las funciones
/*****************************************************************************/


#endif

/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/
// Fin de archivo -- mpsc.h
```

MMPG

```
/*****************************************************************************/
/*  mmpg.h                                                                   */
/*                                                                           */
/*      Cabecera del Módulo Manejador de Procedimientos Generales. En este se*/
/* definen rutinas, constantes y tipos de ámbito general.                    */
/*                                                                           */
/*****************************************************************************/
/* Hecho por:        Daniel Pons A.                                          */
/* Email:            daniel_pons_1@ucab.edu.ve                               */
/*                                          U.C.A.B. Caracas. 2002           */
/*****************************************************************************/
/* Comentarios:                                                              */
/*   Aquí se definen tipos, funciones y constantes de uso general            */
/*****************************************************************************/


#ifndef _GENERAL_H

# define _GENERAL_H  1

/*****************************************************************************/
// Archivos de cabecera a incluir

#include <stdlib.h>
#include <stdio.h>

// Fin de archivos de cabecera a incluir
/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
```

```
                                        /******/
/*****************************************************************************/
// Definición de constantes generales

#define ERROR_TXT                       "-1"
#define ERROR_NUM                       -1
#define BIEN_TXT                        "1"
#define BIEN_NUM                        1
#define PAQUETE_TCP                     6
#define PAQUETE_IP                      8
#define PAQUETE_UDP                     17
#define VERSION_SISTEMA                 "1.00"
#define ENTERO                          1
#define TEXTO                           2
#define DEFECTO_Q931                    1720
#define DEFECTO_PROMISC                 ERROR_NUM   // Modo promíscuo
#define DEFECTO_NO_BD                   ERROR_NUM   // No registro en la BD
#define SI_REGISTRO_BD                  BIEN_NUM    // Registro en la BD
#define DEFECTO_EXPLICITO               0
#define TCP_ID                          1
#define RTP_ID                          2
#define UDP_ID                          3
#define MAX_FILTRO                      100
#define PAQUETE                         20
#define LLAMADA                         30
#define NOMBRE_BD                       "PC-Voz"
#define MAX_CHAR_LONG                   40

// Fin de definición de constantes generales
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
```

```
/*****************************************************************************/
// Definición de estructuras de datos

// la estructura mensaje es definida para pasar mensajes entre funciones.

typedef struct
{
        char *salida;     // Texto descriptivo
        int respuesta;    // Valor booleano
}mensaje;


typedef struct
{
        struct ether_header     *mi_ethernet;
        struct ip               *mi_ip;
        struct tcphdr           *mi_tcp;
        struct q_931hdr         *mi_q931;

}pila_q931;


struct datos_bd
{
        int tipo_datos;
        // Para la llamada
         float id_llamada; // Sólo necesario cuando
                           // se va a insertar una llamada
        char *host_origen;
        char *mac_origen;
        int puerto_origen;
        char *host_destino;
        char *mac_destino;
        int puerto_destino;
```

```
    // Para el paquete
    char *tipo_paquete;
    int longitud;

};

struct info_conexion
{
        int no_BD;
        char *host;
        char *usuario;
        char *clave;
};

struct salida_analisis
{
        struct datos_bd *datos;
        struct lista_pids *lista_procesos;

};

// Fin de definición de estructuras de datos
/*************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*************************************************************************/
// Definición de los prototipos de las funciones

/*************************************************************************
```

```
función:     Existe_Archivo
descripción: Verifica la existencia en disco de un archivo
entradas:    const char * nombre_archivo -> nombre del archivo a verificar
salidas:     int -> variable entera con un 1 si el archivo existe y un 0
             si el archivo no existe
comentarios:
/*************************************************************************/
int Existe_Archivo(const char * nombre_archivo);
/*************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*************************************************************************
función:     Longitud_Archivo
descripción: Calcula el número de caracteres de un archivo
entradas:    const char *archivo -> nombre del archivo
salidas:     int -> retorna el número de caracteres de un archivo. Devuelve
             0 al no conseguir el fichero
comentarios:
/*************************************************************************/
int Longitud_Archivo(const char *archivo);

// Fin de definición de los prototipos de las funciones
/*************************************************************************/

#endif

/*************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*************************************************************************/
// Fin de archivo -- mmpg.h
```

MMCQ931

```
/*****************************************************************************/
/*  mmcq931.h                                                                */
/*                                                                           */
/*        Cabecera de la librería para el manejo de los mensajes Q.931       */
/*                                                                           */
/*****************************************************************************/
/* Hecho por:              Daniel Pons A.                                    */
/*                                                  U.C.A.B. Caracas. 2002   */
/*****************************************************************************/
/* Comentarios:                                                              */
/*   Define la cabecera usada en Call Signaling del protocolo H.323, la misma*/
/*   está definida en las recomendaciones de la ITU-T, H.225.0 v.2 y Q.931 v.2.*/
/*****************************************************************************/

#ifndef _Q_931_HEADER_H

# define _Q_931_HEADER_H    1


/*****************************************************************************/
// Archivos de cabecera a incluir

#include <stdlib.h>
#include "mmpg.h"

// Fin de archivos de cabecera a incluir
/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/
```

```
// Definición de constantes generales

#define PROTOCOL_DISC      8      // Protocol discriminator

// Fin de definición de constantes generales
/*****************************************************************************/
                                  /******/
                                  /******/
                                  /******/
                                  /******/
/*****************************************************************************/
// Definición de constantes de tipos de mensajes

// Mensajes de establecimiento de llamada
#define ALERTING           1
#define CALL_PROCEEDING    2
#define PROGRESS           3
#define SETUP              5
#define CONNECT            7
#define SETUP_ACKNOWLEDGE  13
#define CONNECT_KNOWLEDGE  15


// Mensajes de limpieza y finalización de llamada
#define DISCONNECT         69
#define RESTART            70
#define RELEASE            77
#define RESTART_ACKNOWLEDGE 78
#define RELEASE_COMPLETE   90

// Mensajes de transmisión de información de llamada
#define USER_INFORMATION   32
#define SUSPEND_REJECT     33
#define RESUME_REJECT      34
```

```
#define SUSPEND                    37
#define RESUME                     38
#define SUSPEND_ACKNOWLEDGE        45
#define RESUME_ACKNOWLEDGE         46


// Mensajes misceláneos
#define SEGMENT                    96
#define NOTIFY                    110
#define STATUS_ENQUIRY            117
#define CONGESTION_CONTROL        121
#define INFORMATION               123
#define STATUS                    125

// Fin de definición de constantes de tipos de mensajes
/*****************************************************************************/
                              /******/
                              /******/
                              /******/
                              /******/
/*****************************************************************************/
// Definición de constantes del elemento identificador de información

#define SENDING_COMPLETE          161
#define BEARER_CAPABILITY         4
#define EXTENDED_FACILITY
#define FACILITY
#define NOTIFICATION_INDICATOR    39
#define DISPLAY                   40
#define KEYPAD_FACILITY           44
#define SIGNAL                    52
#define CALLING_PARTY_NUMBER      108
#define CALLING_PARTY_SUBADDRESS  109
#define CALLED_PARTY_NUMBER       112
#define CALLED_PARTY_NUMBER       113
```

```
#define USER_TO_USER

// Fin de definición de constantes del elemento identificador de información
/*****************************************************************************/
                        /******/
                        /******/
                        /******/
                        /******/
/*****************************************************************************/
// Definición de estructuras de datos

struct msg_long_variable
{
        u_int8_t info_id;           // information element identifier
        u_int8_t long_data;         // longitud de la data del mensaje
};


struct msg_long_byte
{
        u_int8_t info_id;           // information element identifier
};


// Cabecera incompleta de los mensajes Q.931, definida en  4/Q.931,
// sólo es utilizada para verificar el protocol discriminator.
struct q_931hdr
{
        u_int8_t basura[4];         // basura de 4 bytes al final
                                    // de TCP
        u_int8_t prot_disc;         // protocol discriminator
        u_int8_t long_call_ref;     // longitud de call reference

};
```

```
struct msg_8
{
        u_int8_t basura[4];             // basura de 4 bytes al final
                                        // de TCP
        u_int8_t prot_disc;             // protocol discriminator
        u_int8_t long_call_ref;         // longitud de call reference
        u_int8_t call_ref;              // call reference de 2 bytes
        u_int8_t msg_type;              // tipo de mensaje
};


struct msg_16
{
        u_int8_t basura[4];             // basura de 4 bytes al final
                                        // de TCP
        u_int8_t prot_disc;             // protocol discriminator
        u_int8_t long_call_ref;         // longitud de call reference
        u_int8_t call_ref[2];           // call reference de 2 bytes
        u_int8_t msg_type;              // tipo de mensaje
        /////// Hasta aquí es una longitud fija

};


struct user_to_user
{
        u_int8_t usr_usr;
        u_int8_t long_usr_usr[2];
        u_int8_t prot_disc;
        u_int8_t prot_id;
        u_int8_t h245_address;
        u_int8_t src_address;
        u_int8_t src_info;
        u_int8_t dst_address;
```

```
        u_int8_t dst_signal_address;
        u_int8_t dst_extra_call_info;
        u_int8_t dst_extra_crv;
        u_int8_t active_mc;
        u_int8_t conference_id;
        u_int8_t conference_goal;
        u_int8_t call_services;
        u_int8_t call_type;
        u_int8_t src_signal_address;
        u_int8_t rmt_ext_address;
        u_int8_t call_id;
        u_int8_t h245_sec_cap;
        u_int8_t tokens;
        u_int8_t crypto_tokens;
        u_int8_t fast_start;
        u_int8_t media_wait_connect;
        u_int8_t overlap_send;

};

struct open_logical_channel
{
        u_int16_t forward_lc;

};

// Fin de definición de estructuras de datos
/*****************************************************************************/
                                        /******/
                                        /******/
                                        /******/
                                        /******/
/*****************************************************************************/
// Definición de los prototipos de las funciones
```

```
/*****************************************************************************
 función:        Ver_Tipo_Mensaje
 descripción:    Sirve para devolver el tipo de mensaje Q.931 en texto
 entradas:       int tipo_msg -> valor del campo Message Type, éstos valores
                 están definidos en Q_931_header.h en la sección "constantes
                 de tipos de mensajes"
 salidas:        char* -> valor en texto del equivalente de tipo_msg, en caso
                 de que no se consiga el equivalente, se devuelve "-1".
 comentarios:    Los formatos de los mensajes y sus valores están definidos en
                 la recomendación ITU-T 7/H.225.0 y 4/Q.931
*****************************************************************************/
char *Ver_Tipo_Mensaje(int tipo_msg);
/** ***************************************************************************/

// Fin de la definición de los prototipos de las funciones
/*****************************************************************************/


#endif


/*****************************************************************************/
                                    /******/
                                    /******/
                                    /******/
                                    /******/
/*****************************************************************************/

// Fin de archivo -- mmcq931.h
```

MANAP

```
/*************************************************************************/
/*  manap.h                                                              */
/*                                                                       */
/*       Cabecera del Módulo de Análisis de paquetes                     */
/*                                                                       */
/*************************************************************************/
/* Hecho por:            Daniel Pons A.                                  */
/* Email:                daniel_pons_1@ucab.edu.ve                       */
/* Ultima modificación:  14/03/02, 16:30                                 */
/*                                          U.C.A.B. Caracas. 2002       */
/*************************************************************************/
/* Comentarios:                                                          */
/*   Define la cabecera usada en Call Signaling del protocolo H.323, la misma  */
/*   está definida en las recomendaciones de la ITU-T, H.225.0 v.2 y Q.931 v.2.  */
/*************************************************************************/


#ifndef _MANAP_H

# define _MANAP_H     1

/*************************************************************************/
// Archivos de cabecera a incluir


#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <net/ethernet.h>
#include <stdlib.h>
#include <pcap.h>
#include <signal.h>
```

```
#include "mmcq931.h"
#include "mmrd.h"
#include "mmbd.h"

// Fin de archivos de cabecera a incluir
/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************/
// Variables de ámbito global

struct datos_generales
{
        //int proto;            // protocolo
        int tipo_msg;           // tipo de mensaje
        char *dir_mac_ori;      // dirección MAC origen
        char *dir_mac_dest;     // dirección MAC destino
        char *dir_ip_ori;       // IP origen
        char *dir_ip_dest;      // IP destino
        int puerto_ori;         // puerto origen
        int puerto_dest;        // puerto destino

};

// Fin de variables de ámbito global
/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************/
```

```
// Definición de los prototipos de las funciones

/*****************************************************************************
función:        Analizar_Paquete
descripción:    Función de análisis de paquetes, organiza los datos del paquete
                y los pone en memoria
entradas:       const struct pcap_pkthdr *cabecera -> cabecera (header) del paquete
                const u_char *paquete -> paquete (header + data) residente
                en memoria que fue capturado por el kernel de captura de paquetes
                u_char *args -> argumentos opcionales de la función
salidas:        struct salida_analisis * -> estructura que almacena todos los
                datos de manera organizada del paquete
comentarios:
*****************************************************************************/
struct salida_analisis *Analizar_Paquete(const struct pcap_pkthdr *cabecera,
const u_char *paquete, u_char *args);

// Fin de definición de los prototipos de las funciones
/*****************************************************************************/

#endif

/*****************************************************************************/
                                /******/
                                /******/
                                /******/
                                /******/
/*****************************************************************************/
// Fin de archivo -- manap.h
```

ANEXO 4: Código fuente del script de la Base de Datos

```
# CREATE DATABASE PC-Voz;
# USE PC-Voz;


#
# Table structure for table 'llamada'
#

# DROP TABLE IF EXISTS llamada;
CREATE TABLE `llamada` (
  `id_llamada` bigint(20) NOT NULL auto_increment,
  `id_usuario` int(11) default NULL,
  `fecha_inicio` datetime NOT NULL default '0000-00-00 00:00:00',
  `fecha_fin` datetime default NULL,
  `host_origen` char(40) NOT NULL default '',
  `mac_origen` char(40) NOT NULL default '',
  `puerto_origen` int(11) NOT NULL default '0',
  `host_destino` char(40) NOT NULL default '',
  `mac_destino` char(40) NOT NULL default '',
  `puerto_destino` int(11) NOT NULL default '0',
  PRIMARY KEY  (`id_llamada`),
  KEY `hosts_y_fechas_llamadas`
     (`host_origen`,`host_destino`,`fecha_inicio`,`fecha_fin`)
);
```

```
#
# Table structure for table 'paquete'
#

# DROP TABLE IF EXISTS paquete;
CREATE TABLE `paquete` (
  `id_paquete` bigint(20) NOT NULL auto_increment,
  `id_llamada` bigint(20) default NULL,
  `tipo_paquete` set('TCP','UDP') NOT NULL default 'UDP',
  `fecha_creado` datetime NOT NULL default '0000-00-00 00:00:00',
  `longitud` int(11) NOT NULL default '0',
  PRIMARY KEY  (`id_paquete`),
  KEY `fechas_paquetes` (`fecha_creado`)
);

#
# Table structure for table 'usuario'
#

# DROP TABLE IF EXISTS usuario;
CREATE TABLE `usuario` (
  `id_usuario` int(11) NOT NULL auto_increment,
  `login` char(40) NOT NULL default '',
  `ultimo_acceso` datetime default NULL,
  `descripcion` char(100) default NULL,
  PRIMARY KEY  (`id_usuario`)
);
```

ANEXO 5: Manual de instalación del sistema PC-Voz

El primer paso consiste en verificar que se cumplen con todos los requisitos para compilar y ejecutar PC-Voz:

- Red basada en arquitectura Ethernet con IP versión 4
- Sistema Operativo Linux
- Librerías del sistema PCAP y MySQLClient.
- GNU gcc versión 2.96 ó superior.
- Sistema Manejador de Base de Datos MySQL versión 3.23.33 ó superior en el host con el servicio de Base de Datos.
- Procesador Intel Pentium o compatible.
- 32 Mb de memoria RAM.
- El espacio en disco es dependiente del tamaño de los archivos logs y del almacenamiento en la Base de Datos.

Posteriormente se debe crear un directorio, por ejemplo */pc_voz* y copiar el código fuente en esa carpeta.

En caso de ser necesario, se deben cambiar los permisos del script llamado *comp* para que pueda ser ejecutado. Esto puede hacerse con el comando *chmod a=rwx comp*.

Debe ejecutarse el guión *comp* con el comando *./comp*. Este script compila y crea el archivo ejecutable *pc_voz*.

Para ejecutar pc_voz correctamente, se debe crear el directorio *logs* dentro de la carpetacreada inicialmente (*/pc_voz* en este caso).

La instalación está finalizada y se puede cargar el programa con el comando *./pc_voz*. Para más ayuda puede escribir *./pc_voz –h*.