Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Proyecto de Grado 2005
de la carrera Ingeniería en Computación

IP over IM

Internet Protocol over Instant Messaging

Ma. Laura Rodríguez Mendaro

Tutores:

Ariel Sabiguero Yawelak

Pablo Rodríguez Bocca

Índice de contenido

Índice de tablas

Índice de ilustraciones

Resumen

La creciente inserción de IM tanto a nivel de usuarios de Internet como a nivel empresarial motivan el presente trabajo, que busca utilizar la mensajería instantánea como canal de comunicación IP.

Este trabajo se basó en la construcción de un prototipo funcional que permitió realizar pruebas sobre el comportamiento de la mensajería instantánea como transporte de datos. Para la construcción de dicho prototipo se utilizaron soluciones open/free, tanto para el transporte de datos como para la mensajería. Las soluciones elegidas: Gaim y OpenVPN.

Se realizaron varias pruebas del prototipo sobre diferentes arquitecturas. Las mismas permitieron obtener resultados sobre la evaluación del prototipo implementado y de la mensajería instantánea como transporte de datos para distintos ambientes. Estos resultados mostraron empíricamente que es posible utilizar los servicios de la mensajería instantánea para transportar cualquier tipo de información.

1 - I N T R O D U C C I Ó N

La mensajería instantánea (Instant Messaging - IM) es una tecnología relativamente nueva de comunicación que ha sido adoptada rápidamente por millones de usuarios. Su contexto de aplicación es tema de investigación, tanto en el área de los negocios como en el mundo de las comunicaciones.
Los servicios de mensajería instantánea están diseñados para transportar mensajes cortos de texto entre dos usuarios con un retardo mínimo, de allí el concepto de instantáneo.

Por otro lado, la demanda de comunicación en cualquier lugar, en cualquier momento y mediante cualquier medio, crece rápidamente con el desarrollo de tecnología de información moderna. Dentro de este contexto se han desarrollado soluciones para permitir establecer canales de comunicación seguros y privados entre PCs. Entre ellos se encuentran los túneles, los cuales establecen conexiones lógicas entre un usuario remoto o host y una red privada, originalmente creados para construir redes virtuales privadas.

Objetivo y Alcance

Este trabajo busca integrar componentes probados y conocidos de tecnología open/free, para construir un prototipo funcional que logre establecer un canal de comunicación IP (Internet Protocol), siendo su transporte la mensajería instantánea. Se busca además conocer la performance y otros parámetros de calidad del canal de comunicación resultante.

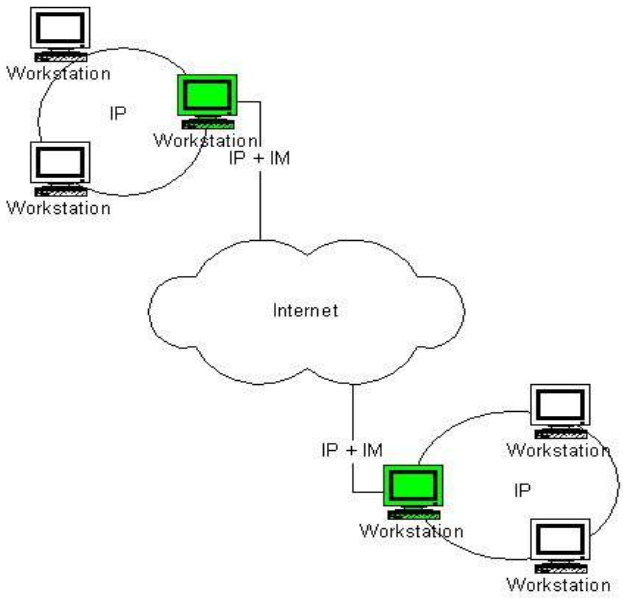


Ilustración 1.- Trabajo a Realizar

El trabajo a realizar consiste en el estudio de los distintos paquetes de software disponibles, para luego construir rápidamente un *prototipo funcional*. Dicho prototipo será utilizado entonces para evaluar y medir los parámetros de calidad deseados.

El prototipo poseerá las restricciones de plataforma propias de los componentes a ser integrados. Uno de los aspectos de considerados en el diseño fue el abarcar la mayor cantidad de plataformas posibles en la solución, cubriendo al menos Linux y Microsoft Windows. Por consiguiente, los componentes a integrar debían ser multiplataforma.

Se estudiaron diferentes alternativas de software disponibles para comunicaciones tanto IP como IM. Los componentes fueron evaluados desde la perspectiva de ser utilizados como "*building blocks"* en la construcción de la solución.

El objetivo de este informe es detallar los pasos seguidos para el estudio de viabilidad, construcción del prototipo, y tests realizados.

Contenido

Se estructura el contenido del presente trabajo de la siguiente manera:

Capitulo 2 - Estado del Arte; Descripción del estudio de los conceptos involucrados en este proyecto.

Capitulo 3 - Análisis; En base al estudio anterior se analizan las diferentes soluciones y se estructuran los objetivos planteados.

Capitulo 4 - Implementación; Pasos seguidos para la integración de los conceptos involucrados y la implementación de solución.

Capitulo 5 - Pruebas; Estudio y resultados de performance y otros parámetros de calidad.

Capitulo 6 - Conclusiones y Trabajo Futuro; Consecuencias y posibles aplicaciones de la tecnología desarrollada.

Apéndice A - Implementación; Información técnica y detallada de las implementaciones.

Apéndice B - Instalación; Detalle de instalación de herramientas utilizadas y manual de usuario.

2 – ESTADO DEL ARTE

El estudio realizado detalla los conceptos involucrados en este proyecto; mensajería instantánea y túneles. El enfoque está dado por presentar un panorama general de estos conceptos, y detallar las tecnologías más importantes en cada uno de ellos junto con las características relevantes al presente proyecto.

Con respecto a la mensajería instantánea, se realiza una breve introducción que contiene: qué se conoce bajo esa definición, cómo trabaja generalmente, y su inserción en el ambiente de las comunicaciones.

Luego se estudian en detalle los clientes y protocolos mas conocidos en el mercado. El enfoque dado para el estudio en detalle se basa en conocer, información sobre el desarrollo del producto, el procedimiento que utiliza para la comunicación y las plataformas que abarca, entre otros.
El objetivo de este estudio es presentar un panorama general sobre la mensajería instantánea y utilizar luego el estudio en detalle para decisiones de diseño y solución.

El otro concepto importante en nuestro estudio son los túneles. De éstos nos interesa conocer las distintas implementaciones que se han desarrollado. Se verán en detalle los mas conocidos, describiendo su funcionamiento, sus protocolos de comunicación, y algunas de sus características de seguridad y encriptación.

Para finalizar y continuando con el estudio de túneles, se presenta un relevamiento de algunos productos desarrollados. Estos productos son soluciones open/free que implementan la tecnología de tunelización.

2.1 Mensajería Instantánea

La mensajería instantánea es un servicio de comunicación. Permite a usuarios comunicarse en tiempo real a través de mensajes de texto, con otros usuarios mediante el uso de un protocolo común. Incluye también, lo que se llama “tecnología de presencia”, lo que significa que un usuario puede ver si la persona con la que quiere intercambiar mensajes esta actualmente “logueada” u “on-line” en el sistema.

Es básicamente un “chat room” para dos, y la conversación fluye como telefónicamente, y su tiempo de demora es rara vez más de un segundo o dos. Además de permitir a los usuarios enviar mensajes de texto o de

voz, muchos de los clientes de mensajería permiten también compartir imágenes, sonido, y el envío y/o recepción de archivos.
En lo que respecta a su protocolo de comunicación, la mayoría de los sistemas de mensajería instantánea trabajan de la misma manera. Al iniciar una comunicación, el cliente envía sus datos de usuario al servidor de mensajería, este servidor verifica el nombre de usuario y la contraseña, y "loguea" al cliente. Una vez que está logueado, el cliente envía su identificación, el puerto asignado al servidor, junto con la lista de sus contactos. El servidor crea un archivo de sesión temporaria que contiene la información de la conexión y chequea quienes de la lista están logueados, para luego enviarle al cliente esta información. Cuando toda la información ha sido enviada, se da por iniciada la sesión de mensajería instantánea. Este proceso puede durar aproximadamente 10 segundos.
Para la comunicación entre dos usuarios generalmente se utiliza la misma metodología, en donde los mensajes de un usuario a otro no se transmiten directamente. Cuando un usuario quiere enviar un mensaje a otro usuario, este mensaje es enviado primero al servidor de mensajería y luego este servidor reenvía el mensaje al usuario destinatario.

Haciendo un poco de historia, la primera aproximación a los protocolos de mensajería instantánea fue Internet Relay Chat o IRC (RFC 1459), desarrollado en agosto de 1988. IRC proporciona un medio de comunicación en tiempo real basado en "chatrooms" o canales, donde un usuario puede unirse a uno o varios canales para hablar con todas las personas presentes en dichos canales. También brinda la posibilidad de entablar una conversación entre 2 personas del mismo canal.
IRC lentamente se esparció alrededor del mundo, y para julio de 1990 tenía un promedio de 12 usuarios en 38 servidores, para 1991 ya eran 300 los usuarios concurrentes en IRC.
ICQ fue la primera mensajería instantánea disponible para mensajes persona a persona. Fue creado por Mirabilis y su primer versión estuvo disponible en Internet desde noviembre de 1996. Para junio de 1997 Mirabilis alcanzo los 100,000 usuarios "online" concurrentes.
Luego en 1997 se presentaron Yahoo! en Enero y AOL en Mayo.

El detalle de estos clientes de mensajería se vera más adelante.

En lo que respecta a su inserción en el mercado, el uso de la mensajería instantánea (IM) tanto en el ambiente de trabajo como en el hogar, se mantiene con un rápido y constante crecimiento.

A continuación veremos investigaciones al respecto:

Según una investigación hecha por Osterman Research Inc.[1] ("IM es hoy lo que fue el e-email en 1994 o '95").

- IM está ahora presente en el 90% de las organizaciones, teniendo el 63% en el 2001.

- 52% utiliza IM como una herramienta de negocios.
- Los tres “grandes” clientes de IM son: AOL Instant Messenger, MSN Messenger y Yahoo! Messenger.
- Las principales preocupaciones sobre el uso de IM son la seguridad en la información enviada mediante IMs, y el riesgo potencial de que entren virus a la red.
- La siguiente figura muestra el avance y la proyección de avance según los años:

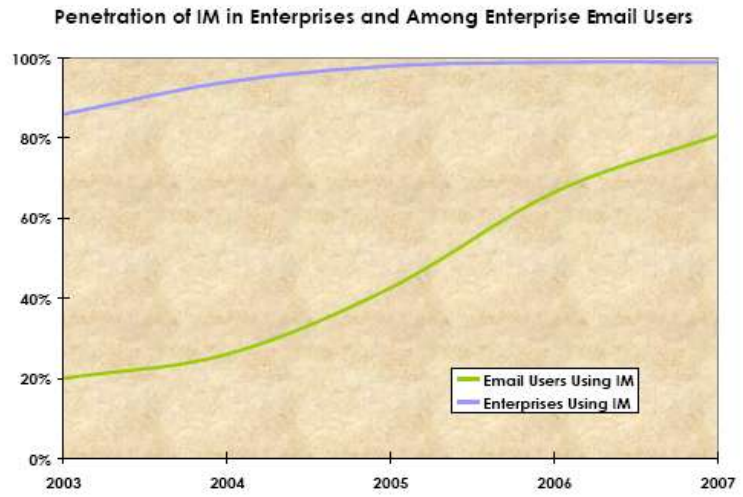


Ilustración 2.- Penetración de IM en Empresas y entre Empresas con usuarios de email

Otro estudio realizado por International Data Corporation [2], determina que en el 2001 veinte millones de personas usaban IM en los negocios prediciendo para fines del 2005, 300 millones y se espera un incremento de mercado para estos servicios de 400 millones de dolares para el año 2008.

Como conclusión a estas investigaciones, la mensajería instantánea es un recurso ampliamente utilizado y en pleno crecimiento tanto en el ámbito laboral como hogareño. En relación con el presente proyecto, los resultados que se obtengan de este trabajo, podrían ser de utilidad y aportar nuevos resultados a las investigaciones en el área.

Dentro de este ámbito vemos a continuación en detalle, los clientes más utilizados, y nos interesamos en sus protocolos, su desarrollo y su disponibilidad.

Antes de comenzar el desarrollo de este estudio, es importante destacar la diferencia entre protocolo de mensajería instantánea y cliente. Los protocolos se refieren al procedimiento de conexión, incluyendo los servidores de mensajería que participan. Los clientes son aplicaciones que implementan los mismos y permiten al usuario utilizar los distintos servicios de mensajería. En la mayoria de los casos, los creadores de los protocolos desarrollan también la aplicacion cliente para utillizar este protocolo.
En primer lugar veremos los clientes que son uniprotocolares (aplicaciones para un sólo protocolo de mensajería instantánea) junto con el detalle del protocolo que utilizan, luego veremos clientes que permiten la utilización de varios protocolos.

2.1.1 Clientes

El objetivo de este estudio es obtener un panorama general de los protocolos y clientes de mensajería instantánea. Se busca conocer y aprender sobre las aplicaciones y protocolos generalmente utilizados, enfocando el estudio en conceptos generales. El resultado de este estudio es utilizado en el resto del trabajo.

AIM (América- On- Line Instant Messenger) [3]

Cliente de mensajería instantánea propietario de América On Line. AIM utiliza un protocolo llamado Oscar [4]. Oscar es un protocolo basado en TCP, utiliza tres servidores para su comunicación.

- El Authorization Server (login.oscar.aol.com) en donde se valida nombre de usuario y contraseña.
- El BOS Server (Basic Oscar Service), esta es la conexión principal en donde se envían y reciben los mensajes.
- Luego es opcional la conexión con el ChatNav (Chat Navigator) que permite crear y unirse a chat rooms.

Utiliza el puerto 5190, su licencia de software es propietaria y tiene distribuciones tanto para Linux como para Windows.

Gabber (The GNOME Jabber Client) [5]

Cliente para el protocolo de mensajería instantánea Jabber [10]. El protocolo Jabber: XMPP (Extensible Messaging and Presence Protocol), basa su comunicación en el protocolo TCP y utiliza la codificación XML, lo cual lo hace extensible.
En el protocolo Jabber, los usuarios utilizan una conexión TCP al servidor durante la sesión, y toda la comunicación es ruteada por los servidores Jabber.

Utiliza los puertos 5222 y 5269, su licencia de software es GPL y tiene distribución sólo para Linux.

ICQ ("I seek you") [6]

Cliente de mensajería instantánea y el primero de su tipo en ser ampliamente utilizado en Internet.
Las primeras versiones utilizaban el protocolo ICQ, éste se basa en los dos protocolos, UDP y TCP. La conexión con el servidor es UDP, y mensajes como "quien esta online" se envía vía UDP. Cuando se tiene una conversación con otro usuario, se conecta usando TCP, y los mensajes se envían directamente.
La versión ICQ99 fue el ultimo cliente oficial ICQ en usar este protocolo.
Con ICQ2000, el cliente oficial ICQ usa una versión modificada del protocolo Oscar, también utilizado por AIM (ver AIM).
Los usuarios de la red ICQ son identificados con un número, el cual es asignado al momento de registrar un nuevo usuario, llamado UIN ("Universal Internet Number" o "numero universal de Internet").

ICQ también cuenta con una versión que proporciona los servicios del cliente de mensajería desde cualquier navegador conectado a Internet. Este servicio es llamado “ICQ2GO!”.

Utiliza el puerto 4000, su licencia de software es propietaria, y tiene distribución sólo para Windows.

Microsoft MSN **[9]**

Bajo esta denominación se engloban realmente tres programas diferentes:

- MSN Messenger: Es el cliente más conocido y su nombre se utiliza para referenciar todos los programas de mensajería de Microsoft.

- Windows Messenger: Se instala con el sistema operativo Microsoft Windows XP y se trata de un cliente de mensajería instantánea básico que no soporta muchas características de éstos (por ej.: imágenes).

- Web Messenger: Versión vía web del cliente aparecida en agosto de 2004. Proporciona características similares al MSN Messenger desde cualquier navegador conectado a Internet.

Los tres programas utilizan distintas versiones del mismo protocolo por lo que muchas de sus características son similares.

El protocolo MSN MSNPX (“X” representa el número de protocolo) ha

sufrido varias revisiones a través de los años. Consiste en una serie de comandos de texto enviados entre el cliente y el servidor (por ej: comando “FLN” indica que un usuario se ha “deslogueado”) .

Un sesión MSN involucra la conexión a un servidor de notificación (Notification Server – NS) el cual provee el servicio de presencia. Éste servidor permite luego la conexión a los servidores de intercambio (Switchboard Servers – SB) los cuales proveen el servicio de mensajería instantánea.
El cliente se conecta al servidor NS a través de una conexión TCP, el cliente y el servidor negocian la versión del protocolo a utilizar.
Cuando un usuario desea comunicarse con otro usuario envía un mensaje al NS. Éste servidor le asigna un servidor SS para esa conversación. Una vez que se ha establecido la conexión entre el usuario y el SS, el usuario “destinatario” de esa comunicación recibe una notificación de su NS para conectarse al mismo SS.

Utiliza los puertos 1863 para la mensajería, 6891 y 6900 para transferencia de archivos, y 6901 para comunicación de voz, su licencia de software es propietaria, y tiene distribuciones para sistemas Windows y Mac OS.

Yahoo! Messenger [7]

Cliente de mensajería instantánea para el protocolo yahoo. Utiliza los protocolos de comunicaciones TCP y HTTP para su comunicación.

Servidores Yahoo: scs.msg.yahoo.com, scsa.msg.yahoo.com, scsb.msg.yahoo.com, scsc.msg.yahoo.com.

Utiliza el puerto 5050, su licencia de software es propietaria y tiene distribución tanto para Windows como para Linux.

Otros

Existen muchos otros clientes y protocolos de mensajería instantánea, algunos relativamente conocidos como ser Napster, Skype y Google Talk entre otros.

2.1.2 Clientes Multi- Protocolos

Los clientes multi- protocolos, son aplicaciones que permiten a través de una única interfase, utilizar varios protocolos de mensajería para la comunicación.

Kopete (el mensajero instántaneo de KDE) [8]

Es un programa libre para mensajería instantánea, disponible desde el 2001. Flexible y sistema multi-protocolo extensible, puede comunicarse con protocolos como ICQ, AIM, Jabber, MSN Messenger y Yahoo! Messenger, entre otros.
Kopete es parte del proyecto KDE y con una excelente integración con este entorno gráfico.

Su licencia de software es GPL y tiene distribución sólo para Linux.

Gaim [11]

Es un programa libre para mensajería instantánea desde 1998. Es multi-protocolo y extensible, funciona con muchos protocolos de mensajería instantánea comúnmente usados, incluyendo:
AOL Instant Messenger, ICQ, MSN Messenger, Jabber, Yahoo! Messenger entre otros.

Recientemente, Gaim ha sufrido una división entre la interfaz gráfica de usuario y el motor, de forma que ahora es posible escribir programas cliente usando diferentes interfaces. El motor se llama ahora libgaim.

Su licencia de software es GPL y tiene distribuciones tanto para Windows como para Linux.

Otros

Existen también otros clientes multi-protocolos como CenterICQ, EB-lite y Miranda.

2.2 Túneles

Al referirnos a “utilizar la mensajería instantánea como transporte de datos”, nos referimos a transportar paquetes IP sobre IM. En términos más corrientes, esto significa que la comunicación es transportada sobre los protocolos de mensajería instantánea y la información contenida en esa comunicación son paquetes IP.

Generalmente para establecer una comunicación entre dos puntos (redes privadas, hosts, usuarios remotos, etc) se utiliza la tecnología de “tunneling” y ésta es usada para muchos tipos de soluciones. Se le llama túnel a la conexión lógica a través de la cual se encapsulan los datos. El túnel es un vínculo seguro y privado entre un usuario remoto o un host

y una red privada, en donde normalmente se encapsulan y cifran los datos.

Las tecnologías de túneles fueron originalmente creadas para construir redes virtuales privadas (VPN - Virtual Private Networks), y eso es para lo cual se usan actualmente.
Hasta no hace mucho tiempo, las diferentes sucursales de una empresa podían tener, cada una, una red local que operaba aislada de las demás. Cada una de estas redes locales tenía su propio esquema de nombres, su propio sistema de email, e inclusive usaban protocolos que diferían de los usados en otras sucursales. Es decir, en cada lugar existía una configuración totalmente local, que no necesariamente debía ser compatible con alguna o todas las demás configuraciones de las otras sucursales dentro de la misma empresa. Surge entonces la necesidad de comunicar las diferentes redes locales de cada sucursal.
Además de la comunicación entre diferentes sucursales, surgió también la necesidad de proveer acceso a los usuarios móviles de la empresa. Mediante Remote Access Services(RAS), este tipo de usuario puede conectarse a la red de la empresa y usar los recursos disponibles dentro de la misma.

Una Virtual Private Network (VPN) es un sistema para simular una red privada sobre una red pública, por ejemplo Internet. Como se muestra en la figura siguiente, la idea es que la red pública sea “vista” desde dentro de la red privada como un cable lógico que une las dos o más redes que pertenecen a la red privada. Las VPNs también permiten la conexión de usuarios móviles a la red privada.

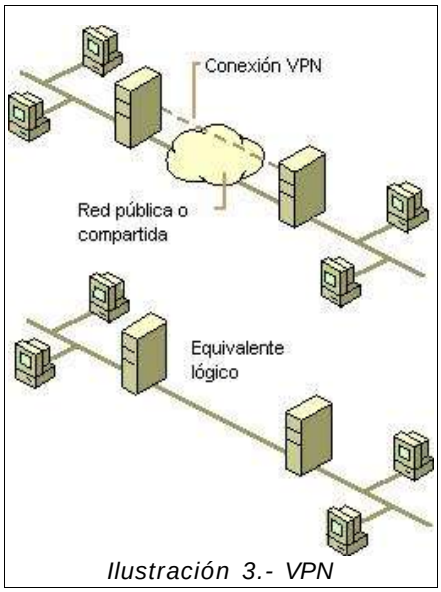


Ilustración 3.- VPN

La forma de comunicación entre las partes de la red privada a través de la red pública se hace estableciendo túneles virtuales entre dos puntos

para los cuales se negocian esquemas de encriptación y autenticación que aseguran la confidencialidad e integridad de los datos transmitidos utilizando la red pública.

La tecnología de túneles ("tunneling") es un modo de transferir datos en la que se encapsula un tipo de paquetes de datos dentro del paquete de datos de algún protocolo, no necesariamente diferente al del paquete original. Al llegar al destino, el paquete original es desempaquetado volviendo así a su estado original.

Como se usan redes públicas, en general Internet, es necesario prestar debida atención a las cuestiones de seguridad, que se aborda a través de esquemas de autenticación y encriptación.

Las técnicas de autenticación son esenciales en las VPNs, ya que aseguran a los participantes de la misma que están intercambiando información con el usuario o dispositivo correcto. La autenticación en VPNs es conceptualmente parecido al logueo en un sistema como nombre de usuario y contraseña, pero con mayores necesidades en la validación de identidades. La autenticación es llevada a cabo generalmente al inicio de una sesión, y luego aleatoriamente durante el curso de la misma, para asegurar que no haya algún tercer participante que se haya entrometido en la conversación. La autenticación también puede ser usada para asegurar la integridad de los datos. Los datos son procesados con un algoritmo de hashing para derivar un valor incluido en el mensaje como checksum. Cualquier desviación en el checksum indica que los datos fueron corruptos en la transmisión o interceptados y modificados en el camino.
Ejemplos de sistemas de autenticación son: Challenge Handshake Authentication Protocol (CHAP) y RSA.

Todas las VPNs tienen también algún tipo de tecnología de encriptación, que esencialmente empaqueta los datos en un paquete seguro. La encriptación es considerada tan esencial como la autenticación, ya que protege los datos transportados de poder ser vistos y entendidos en el viaje de un extremo a otro de la conexión.

Retomando el tema de los túneles, una computadora conectada a una red mediante un túnel tiene al menos *dos* direcciones IP. Una es la dirección mediante la cual tiene acceso al ISP (Internet Service Provider), la otra es una dirección de red, la cual se incluye dentro de los paquetes que viajan por el túnel, usualmente llamada dirección virtual (virtual address).

Cuando se establece un túnel entre dos PCs , estas utilizan su dirección virtual para comunicarse, y es el protocolo de "tunneling" el que se encarga de realizar el transporte para esta comunicación. Estos protocolos de "tunneling" generalmente encapsulan la comunicación virtual dentro de una comunicación real.

2.2.1 Protocolos

A continuación veremos en detalle algunos de los protocolos más conocidos y utilizados para las tecnologías de tunneling.

IP in IP [12]

Este protocolo de túnel ha estado presente en Linux desde hace mucho tiempo. Su objetivo es conectar dos LAN's que normalmente no serían capaces de comunicarse, a través de direcciones IP locales y utilizando las direcciones de los routers para el mundo.

No permite realizar broadcast ni es portable para IPv6.

Su técnica de encapsulación es relativamente simple. Se agrega una header IP antes del header original y entre ellos se agregan headers de información específica de la configuración del túnel. Las direcciones IP del header que se le agrega, tienen el origen y el destino de los "endpoints" del túnel.

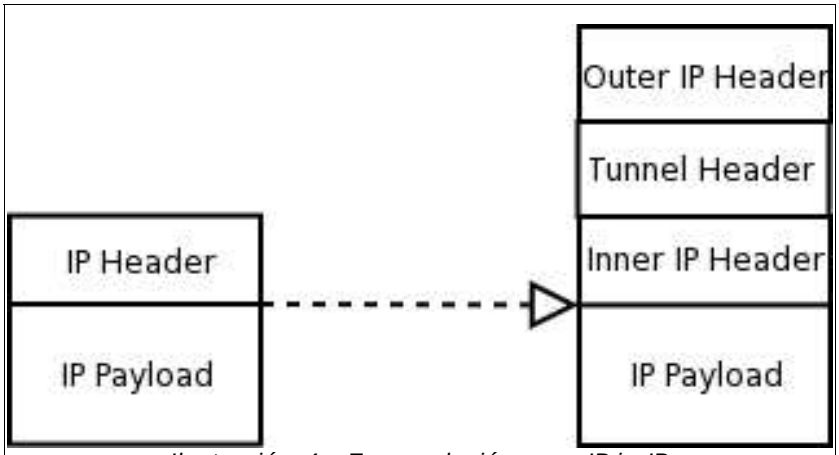


Ilustración 4.- Encapsulación para IP in IP

GRE (Generic Routing Encapsulation) [13]

GRE fue originalmente desarrollado por Cisco y permite realizar algunas cosas más que con IP in IP. Por ejemplo, se puede transportar tráfico multicast e IPv6.

Es un protocolo que provee mecanismos para la encapsulación de paquetes, sin un protocolo de transporte. Cuando un sistema tiene un

paquete que necesita ser encapsulado y ruteado, este paquete es primero encapsulado por GRE que posiblemente incluirá el ruteo. El paquete resultante de GRE puede ser luego encapsulado en otro protocolo y forwardeado.

Tanto GRE como IP in IP están implementadas a nivel de kernel. Las implementaciones que siguen son a nivel de usuario.

PPTP (Point to Point Tunneling Protocol) [14]

PPTP es un protocolo desarrollado por Microsoft, U.S. Robotics, Ascend Communications, 3Com/Primary Access, ECI Telematics conocidas colectivamente como PPTP Forum, para implementar redes privadas virtuales o VPN.

El protocolo de túnel punto a punto (PPTP) simula un túnel para (o encapsula) el tráfico en paquetes IP. Permite a los usuarios ejecutar de forma remota aplicaciones que dependen de determinados protocolos de red. Esta tecnología de red admite redes privadas virtuales (VPN) multiprotocolo, permitiendo así a los usuarios remotos el acceso seguro a redes empresariales a través de Internet u otras redes.

PPTP no especifica ningún cambio con respecto al protocolo PPP, pero si describe una nueva vía de comunicación, se define una arquitectura cliente-servidor en donde el servidor PPTP Network Server (PNS) es plausible de correr en cualquier plataforma, mientras que el cliente PPTP Access Concentrator (PAC) opera en una plataforma con acceso discado.

El paquete PPTP está compuesto por un header de envío, un header IP, un header GREv2 y el paquete de carga.

- El header IP contiene información relativa al paquete IP, como ser, direcciones de origen y destino, longitud del datagrama enviado, etc.
- El header GREv2 contiene información sobre el tipo de paquete encapsulado y datos específicos de PPTP concernientes a la conexión entre el cliente y servidor.
- El paquete de carga es el paquete encapsulado, que, en el caso de PPP, el datagrama es el original de la sesión PPP que viaja del cliente al servidor.

Para la autenticación, PPTP tiene tres opciones de uso: CHAP, MS-CHAP, PAP. Para la encriptación, PPTP utiliza el sistema RC4 de RSA, con una clave de sesión de 40 bits.

L2TP (Layer Two Tunneling Protocol) [15]

Es un protocolo de túnel de Internet normalizado. A diferencia del protocolo de túnel punto a punto (PPTP), L2TP no requiere conectividad IP entre la estación de trabajo cliente y el servidor. L2TP sólo requiere que el túnel proporcione conectividad punto a punto orientada a paquetes.

Este protocolo se puede utilizar en medios como ATM, Frame Relay y X.25, de manera que lo hace transparente tanto para los usuarios como para las aplicaciones.

L2TP ofrece la misma funcionalidad que PPTP, permite a los clientes establecer túneles entre las redes que intervienen. A través de "tunnelización" con L2TP, un Proveedor de Internet (ISP), puede crear un túnel virtual de manera de conectar usuarios remotos con LANs o WANs.

El escenario típico L2TP, cuyo objetivo es tunnelizar marcos PPP entre el sistema remoto o cliente LAC y un LNS ubicado en una LAN local, es el que se muestra en la siguiente figura:

LAC = L2TP Access Concentrator
LNS = L2TP Network Server

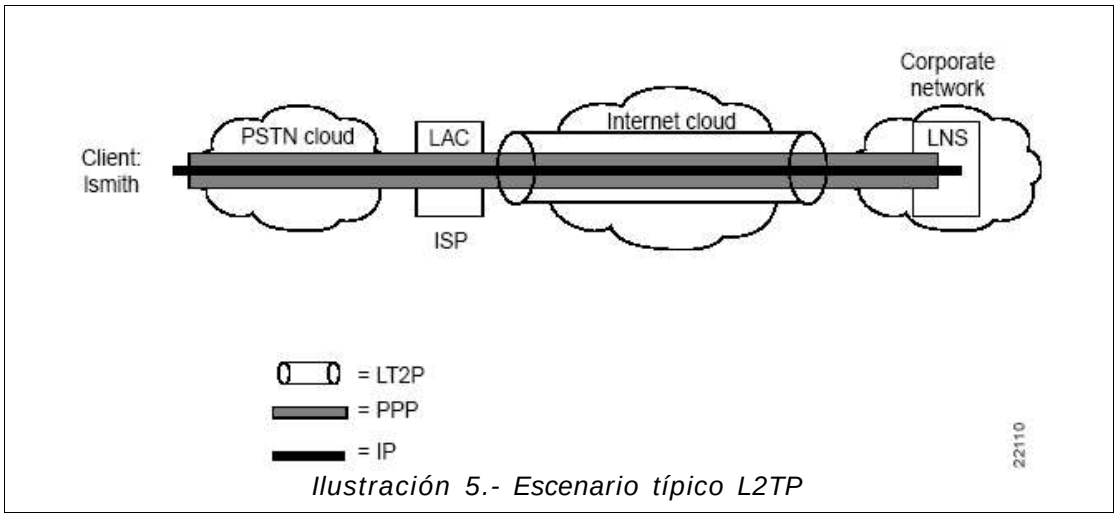


Ilustración 5.- Escenario típico L2TP

Un L2TP Access Concentrator (LAC) es un nodo que actúa como un extremo de un túnel L2TP y es el par de un LNS. Un LAC se sitúa entre un LNS y un sistema remoto y manda paquetes entre ambos. Los paquetes entre el LAC y el LNS son enviados a través del túnel L2TP y los paquetes entre el LAC y el sistema remoto es local o es una conexión PPP. Un LAC localizado en el ISP intercambia paquetes PPP con los usuarios remotos, y se comunica a través de mensajes de request/response con el LNS para establecer un túnel.

Un L2TP Network Server (LNS) actúa como el otro extremo de la conexión L2TP y es el otro par del LAC. El LNS es la terminación lógica de una sesión PPP que está siendo puesta en un túnel desde el sistema remoto por el LAC.

L2TP utiliza dos tipos de mensajes: de control y de datos.

- Los mensajes de control son usados para el establecimiento, el mantenimiento y el borrado de los túneles, y las llamadas.
- Los mensajes de datos encapsulan los marcos PPP y son enviados a través del túnel.

La siguiente figura muestra la relación entre los marcos PPP y los mensajes de control a través de los canales de control y datos de L2TP.

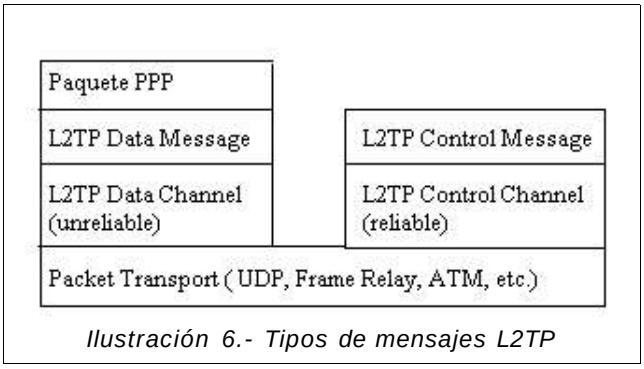


Ilustración 6.- Tipos de mensajes L2TP

L2TP utiliza el puerto 1701. El paquete entero de L2TP, incluyendo la parte de datos y el encabezado, viaja en un datagrama UDP. El que inicia un túnel L2TP toma un puerto UDP de origen que esté disponible, pudiendo ser o no el 1701 y envía a la dirección de destino sobre el puerto 1701. Este extremo toma un puerto libre, que puede ser o no el 1701, y envía la respuesta a la dirección de origen, sobre el mismo puerto iniciador. Luego de establecida la conexión, los puertos quedan estáticos por el resto de la vida del túnel.

Para la autenticación en el protocolo L2TP, tanto el LAC como el LNS comparten un secreto único. Cada extremo usa este mismo secreto al actuar como autenticado o autenticador.

Para asegurar los paquetes del protocolo L2TP, se requiere que el protocolo de transporte tenga la posibilidad de brindar servicios de encriptación, autenticación e integridad en su totalidad. Como tal, L2TP sólo se preocupa por la confidencialidad, autenticidad e integridad de los paquetes L2TP entre los puntos extremos del túnel, no entre los extremos físicos de la conexión.

IPSec (Secure IP over the Internet) [16]

IPSec es una extensión al protocolo IP que proporciona seguridad a IP y a los protocolos de capas superiores. Fue desarrollado para el nuevo estándar IPv6 y después fue portado a IPv4.

IPSec emplea dos protocolos diferentes – AH (cabecera de autenticación) y ESP (carga de seguridad encapsulada) - para asegurar la autenticación, integridad y confidencialidad de la comunicación.
Puede proteger el datagrama IP completo o sólo los protocolos de capas superiores. Estos modos se denominan, respectivamente, modo túnel y modo transporte.

- En modo túnel el datagrama IP se encapsula completamente dentro de un nuevo datagrama IP que emplea el protocolo IPSec.

- En modo transporte IPSec sólo maneja la carga del datagrama IP, insertándose la cabecera IPSec entre la cabecera IP y la cabecera del protocolo de capas superiores.

Para proteger la integridad de los datagramas IP, el protocolo IPSec emplea códigos de autenticación de mensaje basados en resúmenes (HMAC - Hash Message Authentication Codes).

Para proteger la confidencialidad de lo datagramas IP, el protocolo IPSec emplea algoritmos estándares en el cifrado simétrico. El estándar IPSec exige la implementación de NULL y DES. En la actualidad se suelen emplear algoritmos más fuertes: 3DES, AES y Blowfish.

Para que los participantes de una comunicación puedan encapsular y desencapsular los paquetes IPSec, se necesitan mecanismos para almacenar las claves secretas, algoritmos y direcciones IP involucradas en la comunicación. Todos estos parámetros se almacenan en asociaciones de seguridad (SA – Security Associations). Las asociaciones de seguridad, a su vez, se almacenan en bases de datos de asociaciones de seguridad (SAD - Security Asocciation Databases).

2.2.2 Otras soluciones Open/Free

Luego de ver algunos de los protocolos para túneles, se describen a continuación algunas soluciones Open/Free disponibles para establecer túneles.

Este estudio tiene como objetivo analizar cómo es llevado a la práctica el concepto de “tunnelizacion” por parte de estas soluciones.

VTun [17]

Vtun es una forma sencilla de crear un túnel virtual sobre redes TCP/IP con compresión y encriptación de tráfico.

Soporta túneles IP, PPPy Ethernet entre otros. VTun es un sistema cliente/servidor. La maquina servidor escucha conexiones de clientes VTun en un puerto específico. El cliente inicializa la creación del túnel conectándose al puerto del servidor.

Es un software con licencia GPL, los sistemas operativos que soporta son Linux, Solaris, FreeBSD, OpenBSD, NetBSD, y otros clones BSD.
Su desarrollo se estancó en el 2003.

Open VPN [18]

OpenVPN es una tecnología de "tunneling" relativamente nueva y con excelente performance.
Esta herramienta está disponible como un demonio robusto y fácilmente configurable para VPNs que puede ser usado para conectar de manera segura una o más redes privadas usando túneles encriptados sobre Internet.
Está basado en SSL (Secure Sockets Layer), un protocolo comúnmente usado para asegurar las transacciones sobre Internet.

OpenVPN trabaja en dos modos, por un lado usa el driver TUN para paso de tráfico IP o usa el driver TAP para el paso de tráfico Ethernet.

Una interfaz tun (mediante el driver TUN) es un adaptador de red virtual que aparece como una conexión punto a punto para el sistema operativo, pero este adaptador en vez de enviar los bits, los pone en espacio de usuario. Cualquier programa de usuario puede abrir ese adaptador como un archivo y leer y escribir paquetes desde y en él.
Una interfaz tap (mediante el driver TAP) tiene una producción similar, sólo que emula ethernet en vez de punto a punto.

Paquetes IP desde la interfaz son encriptados y encapsulados en una conexión y enviados a un host remoto sobre Internet. El host remoto, desencripta, autentica y desencapsula el paquete IP, para luego ponerlo en su interfaz. Utiliza la librería de OpenSSL [19] para encriptación, autenticación y certificación de los túneles.

Es un software libre, que trabaja en espacio de usuario y es multiplataforma, con distribuciones para Linux, Solaris, OpenBSD, FreeBSD, NetBSD, Mac OS X y Windows.

También es importante destacar que existe muy buena documentación en lo que respecta a su configuración y uso. Lamentablemente no es fácil encontrar documentación sobre su implementación.

3 – ANÁLISIS

En este proyecto lo que se busca es llegar a una solución a través de la integración de herramientas ya existentes, tanto para la mensajería instantánea como para la comunicación IP.
Utilizando la información del estudio realizado en el capítulo anterior, el primer paso de la solución es analizar las distintas opciones y elegir cuál de ellas se utilizarán.
Luego la solución se basa en determinar cuál es la mejor manera de adaptar e integrar las soluciones elegidas para alcanzar el objetivo.

3.1 Análisis de Componentes

Basado en el estudio del capítulo anterior, se realiza un análisis comparativo de las distintas opciones con el objetivo de elegir los componentes que se van a utilizar para la solución.
Esta sección es separada en dos, primero analizamos las distintas opciones de mensajería instantánea y luego de túneles.

3.1.1 Análisis Comparativo de Mensajería Instantánea.

Los parámetros elegidos para el análisis son:

Tipo: Este parámetro toma dos valores, uni- protocolo, multi- protocolo, y se refiere a sí la aplicación permite múltiples protocolos de mensajería o solamente uno.
Este parámetro es importante porque cuanto más amplia sea nuestra aplicación podemos lograr mejores resultados, al poder probar la aplicación para distintos protocolos y luego compararlas.

Plataformas: Enumera los sistemas operativos en los cuales puede ser utilizada la solución. Este parámetro está directamente relacionado con nuestro objetivo de construir una herramienta que pueda ser utilizada tanto en Windows como en Linux.

Licencia de Software: Describe el estado del código con el cual se construyó la aplicación. Propietario para las aplicaciones en las cuales el código es propiedad del creador y no está disponible, GPL para las licencias de software libre.

Documentación: Este parámetro se refiere a los trabajos escritos disponibles que refieren a la aplicación. De éstos trabajos los tipos que nos interesan refieren a documentación de uso de la aplicación y

principalmente documentación sobre la construcción de la herramienta. Este parámetro es relativo a este trabajo y la calificación se basa en estudio previo.

Las características buscadas en este caso son que el cliente sea multi-protocolo, utilizable tanto en Linux como en Windows, con licencia de software GPL y con buena documentación.

El resultado de este análisis, es el que se muestra a continuación:

Cliente	Tipo	Plataformas	Licencia de Software	Documentación
AIM	uni- protocolo	Windows/Linux	Propietario	BUENA
GABBER	uni- protocolo	Linux	GPL	MUY BUENA
GAIM	multi- protocolo	Windows/Linux	GPL	MUY BUENA
ICQ	uni- protocolo	Windows	Propietario	REGULAR
COPETE	multi- protocolo	Linux	GPL	BUENA
MSN Messenger	uni- protocolo	Windows	Propietario	REGULAR
YAHOO	uni- protocolo	Windows/Linux	Propietario	REGULAR

Tabla 1.- Análisis comparativo de Clientes IM.

Luego de realizada esta comparación, se define que el cliente que tiene las características que buscamos es el Gaim [11].

Este cliente, además de permitirnos utilizar su librería libgaim para la comunicación mediante mensajería instantánea, está en desarrollo, actualizándose varias veces al año, lo que nos hace pensar que se mantendrá en funcionamiento acoplándose a los cambios de los protocolos que implementa.

3.1.2 Análisis Comparativo de Túneles

Los parámetros elegidos para el análisis similares al estudio anterior.

Las características que nos interesan en este caso es que las plataformas incluyan tanto Linux como Windows y que la documentación sea relativamente buena.
El resultado de este análisis, es el que se muestra a continuación:

Nombre	Plataformas	Licencia de Software	Documentación
IP in IP	Linux	GPL	BUENA
GRE	Linux	GPL	BUENA
PPTP	Windows/Linux	Propietario	REGULAR
L2TP	Windows/Linux	Propietario	REGULAR
IPSec	Windows/Linux	GPL	BUENA
VTun	Linux	GPL	MUY BUENA
OpenVPN	Windows/Linux	GPL	MUY BUENA

Tabla 2.- Análisis Comparativo de Túneles

De este análisis se concluye que el componente que más se adecua a las características deseadas es el OpenVPN[18].

Así como el Gaim, esta herramienta también está en desarrollo. Tiene muy buena documentación sobre su uso, pero no así sobre su implementación.

3.2 Solución Propuesta

La solución propuesta es entonces integrar Gaim con OpenVPN. La integración se basa en construir las funciones de mensajería instantánea utilizando el Gaim, de forma que la comunicación que requiera el OpenVPN para su implementación del túnel, las realice a través de ellas.

Cuando una aplicación utiliza el túnel para su comunicación con otra máquina, y envía un paquete, el OpenVPN obtendrá este paquete y hará los cambios necesarios (extra headers, encriptación) para luego enviarlos a la otra máquina. El cambio que se realiza a una operación normal del OpenVPN, es que el envío no es a través de una conexión real, sino es hecho a través de la mensajería. Así luego de los cambios, estos paquetes son enviados por el Gaim al usuario que hayamos elegido para realizar este túnel, e inversamente, cuando esta aplicación, detecta a través del Gaim la recepción de un nuevo mensaje notifica al OpenVPN para indicarle que ha recibido un nuevo paquete.

Está es una ilustración sobre como estos componentes se relacionan y como se realiza el transporte de los datos:

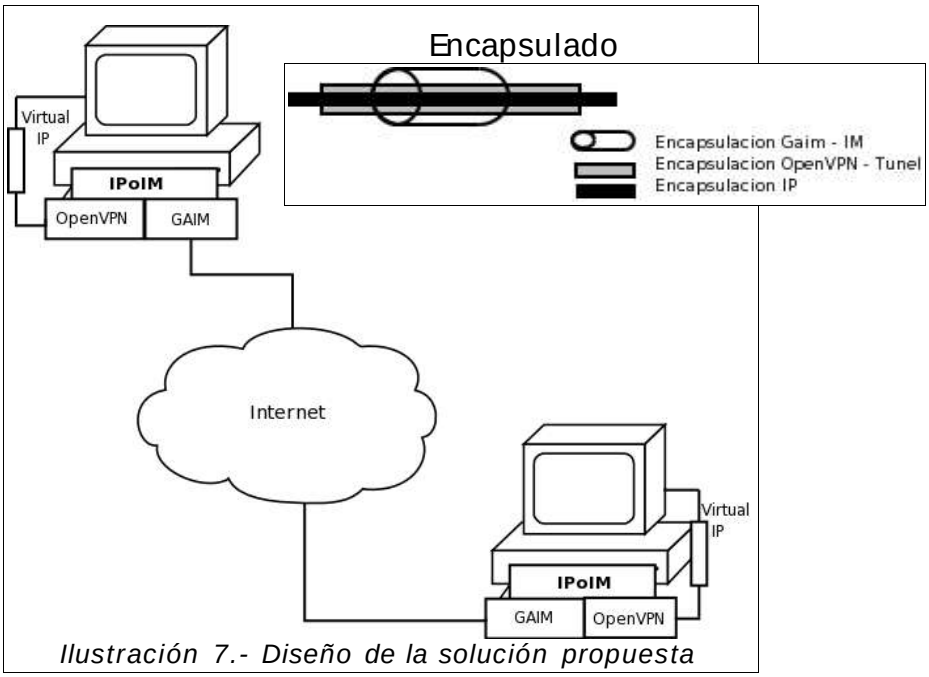


Ilustración 7.- Diseño de la solución propuesta

En conclusión la solución propuesta se basa en

Adaptar e integrar el OpenVPN y el Gaim para que la tunelización utilice la mensajería instantánea para el transporte.

El OpenVPN será el encargado de la configuración y administración del túnel. El Gaim de administrar las conexiones al servidor de IM, el envío y la recepción de IMs, las distintas cuentas creadas y los protocolos que utilizan.

Nuestro aporte será realizar la integración de estos dos componentes:

- Implementar funciones dependientes del Gaim que permitan encapsular las funcionalidades básicas de mensajería (inicio de sesión, envío y recepción de mensajes).

- Adaptar el OpenVPN mediante modificaciones, para lograr que se comunique a través de las funciones anteriores.

3.2.1 Gaim

Para poder utilizar el Gaim en la solución se necesitan tener funciones que permitan:

1.- *Iniciar Sesión:*
A partir de una cuenta de usuario iniciar la sesión de mensajería que corresponda.
En primer lugar se debe poder obtener las cuentas registradas en la aplicación Gaim. Luego utilizando esa información, enviar al servidor de mensajería un pedido de inicio de sesión y luego realizar todos los pasos requeridos por el protocolo para completar esta operación.

Información requerida para esta función:
- Cuenta con la cual se requiere el inicio de sesión.

Información obtenida de esta función:
- Inicio de sesión exitoso.

2.- *Enviar Mensajes IMs:*
Esta función deberá ser capaz de enviar un mensaje a través de la mensajería. Esta operación deberá ser independiente del usuario logueado, es decir esta función recibirá suficiente información del usuario que envía y del usuario al que se dirige este mensaje como para poder realizar los pasos requeridos para completarla.

Información requerida para esta función:
- Cuenta de usuario con sesión iniciada, que envía este mensaje.
- Usuario destinatario del mensaje.
- Mensaje de texto a enviar.

Información obtenida de esta función:

- Envío de mensaje exitoso.

3.- *Recepcionar Mensajes IMs:*
Esta función deberá detectar cuando el usuario con sesión iniciada ha recibido un mensaje. Esta función no será invocada por la aplicación, sino que deberá activarse cuando el usuario recibe un nuevo mensaje. También deberá ser capaz de enviar una notificación de recepción a la aplicación para que esta sea capaz de procesarlo.

Información requerida para esta función:
- Ninguna

Información obtenida de esta función:
- Notificación de nuevo mensaje.
- Cuenta para la cual se recibió el mensaje.
- Usuario que lo envía.
- Mensaje de texto recibido.

A través de la implementación de estas funciones, se construye una API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) (ver Implementación, Pág: 56) que permite la comunicación entre cualquier aplicación y el Gaim. Teniendo claras las especificaciones necesarias para utilizar esta API, cualquier aplicación podría utilizar estas tres funciones referentes a la mensajería instantánea.
En este caso, las necesitamos para poder hacer el envío y la recepción de paquetes de red del túnel a través de ella.

Estas tres funciones completan las operaciones que se consideran necesarias para utilizar los protocolos de mensajería instantánea.

3.2.2 OpenVPN

Éste es el otro componente de software que se utilizará. Lo que queremos obtener de él, es que no modifique su funcionamiento, a excepción del envío de los paquetes de red del túnel que serán ahora enviados a través de la mensajería.

Entonces necesitamos los siguientes cambios

- La aplicación deberá incorporar en su configuración, la cuenta del usuario de mensajería instantánea que utilizará. También se deberá incluir en la configuración el usuario de mensajería instantánea con el cual yo quiero establecer mi túnel, el cual deberá ser un usuario con el cual se pueda establecer una comunicación, un usuario perteneciente a la lista de contactos.

- Incluir al inicio de la aplicación el “inicio de sesión” de la mensajería instantánea. Este momento se da cuando la aplicación

inicializa sus parámetros y antes del comienzo de la comunicación que utilizará. Es ahí donde la modificación hará que la aplicación espere hasta que se haya completado la operación de inicio de sesión, para luego poder utilizarla para la comunicación.

- La aplicación no deberá establecer una conexión con la otra maquina con la cual quiere establecer el túnel.

- Se deberá reemplazar la función de *envío de paquetes* para que la aplicación utilice la función de *envío de IM* antes propuesta.

- Se deberá incorporar a la aplicación la función de *recepción de IM* antes propuesta en sustitución de la *recepción de los paquetes.*

A través de la realización de estos cambios, esta aplicación podrá ser capaz de llevar a cabo nuestra solución.

3.2.3 IPoIM

Surge IpoIM como integración de las herramientas Gaim y OpenVPN y como solución propuesta al objetivo del proyecto.
Como resultado de la integración, se muestra a continuación la secuencia de acciones necesarias para completar los requerimientos:

Inicialización de IPoIM

Inicializar OpenVPN
/* OpenVPN utiliza su configuración para inicializar los parámetros del túnel.*/

Usuario Mensajería = Configuracion.UsuarioMensajeria
/* Desde la configuración se obtiene el usuario configurado en la maquina local. Este usuario es el que se utiliza para la sesión de mensajería */

Iniciar Sesión(Usuario Mensajería)
/* Se inicia sesión con este usuario a través de la funcionalidad implementada en la sección Gaim. */

Se establecen dos esperas activas:

1.- Llegada de paquete de red por el túnel.	**2.- Llegada de un paquete por la mensajería instantánea.**
1.1.- túnel -> openvpn 1.1.1.- Leido desde el túnel por OpenVPN 1.1.2.- Procesado a efectos de incluirle extra headers y encriptación y seguridad.	**2.1.- gaim -> ipoim** 2.1.1.- Recepcionado por el gaim 2.1.2.- Detectado por el ipoim a través de la funcionalidad: Recepcionar mensajes de IMs (MensajeIM)
1.2.- openvpn -> ipoim 1.2.1.- Obtener usuario receptor del paquete 1.2.2.- Invocar la función: Enviar Mensajes IMs(Mensaje, Receptor).	**2.2.- ipoim -> openvpn** 2.2.1.- Procesado por el OpenVPN a efectos de corroborar la correctitud y seguridad del mismo.
1.3.- ipoim -> gaim 1.3.1.- Enviado a través de la mensajería instantánea mediante el Gaim.	**2.3.- openvpn -> túnel** 2.3.1.- Escrito por el OpenVPN en el túnel.

túnel : comunicación entre la aplicación y el túnel.
openvpn : funcionalidades implementadas por el OpenVPN.
gaim : funcionalidades implementadas por el Gaim.
ipoim : funcionalidades agregadas por el IP over IM.

En conclusión los componentes OpenVPN y Gaim trabajarán en conjunto a través de la implementación agregada por IPoIM. La solución es construir una sola aplicación que incluya las soluciones de los dos componentes y lo necesario para adaptarlos e integrarlos.

3.2.4 Codificación

Los paquetes que son detectados por el OpenVPN para luego ser enviados son paquetes binarios. A efectos de la implementación es un espacio de memoria que contiene el paquete IP. Por otro lado la mensajería instantánea maneja su comunicación en base a mensajes de texto.
Es por eso que en esta instancia nos encontramos con el tema de codificar los binarios para pasarlos a mensajes de texto al enviar. Así también se recibe este texto decodificarlo para retornarlo a su estado original.
Es por este motivo, que se decidió utilizar un algoritmo de codificación (conocido) que intervenga en el pasaje de binario a texto y viceversa.

Base 64

El algoritmo que se decide utilizar es *Base64*. Base64 utiliza un subconjunto de 65 (64 + 1, un carácter relleno “especial”) caracteres del US-ASCII, permitiendo 6 bits para cada carácter.
A modo de ejemplo, el carácter 'm' tiene un valor Base64 de 38 (ver Tabla 3.- Alfabeto Base64), y cuando esta representado en forma binaria, es 100110.

Codificación
Cuando se codifica una secuencia, se convierte en su valor US-ASCII (En US-ASCII a cada carácter se le asigna un valor).

Por ejemplo:
El carácter "m" tiene el valor decimal de 109
El carácter "n" tiene el valor decimal de 110
El carácter "e" tiene el valor decimal de 101
de este modo 'mne' (una secuencia 3 8-bit-bytes) es
109 110 101
en forma decimal.

Cuando convertimos esta secuencia a binario nos queda lo siguiente:
01101101 01101110 01100101

Estos 3 8-bit-bytes se concatenan (se juntan) para hacer una cadena de 24 bits:
011011010110111001100101

Esta cadena de 24 bits se divide en 4 secciones de 6 bits:
011011 010110 111001 100101

Estos 4 valores binarios son convertidos al sistema decimal obteniendo lo siguiente:
27 22 57 37

Ahora cada carácter del conjunto de Base64 tiene un valor decimal (ver Tabla 3.- Alfabeto Base64), entonces cambiaremos estos valores decimales por los equivalentes en Base64:
27 = b, 22 = W, 57 = 5, 37 = l

Debajo está la tabla del conjunto de caracteres Base64 con sus valores decimales:

Valor	Codificación	Valor	Codificación	Valor	Codificación	Valor	Codificación
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6

8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Tabla 3.- Alfabeto Base64

Decodificación

Para descifrar una secuencia codificada en Base64 se debe hacer lo mismo pero al revés:

1) Convertir cada carácter al valor decimal de la tabla Base64.
2) Convertir este valor decimal en binario.
3) Reunir los 6 bit binarios de cada carácter en una gran cadena.
4) Separar esta secuencia en grupos de 8 bit (empezando de derecha a izquierda).
5) Convertir cada valor binario 8-bit en un numero decimal.
6) Convertir este valor decimal en su equivalente US-ASCII.

Esta codificación es la que se usará para transformar los paquetes que son binarios, en mensajes de texto que pueden ser enviados por la mensajería instantánea.

Por lo tanto lo que viajará a través de la mensajería serán los paquetes del OpenVPN codificados en Base64.

3.3 Arquitectura de la Solución

En la siguiente figura se plantea la arquitectura del sistema:

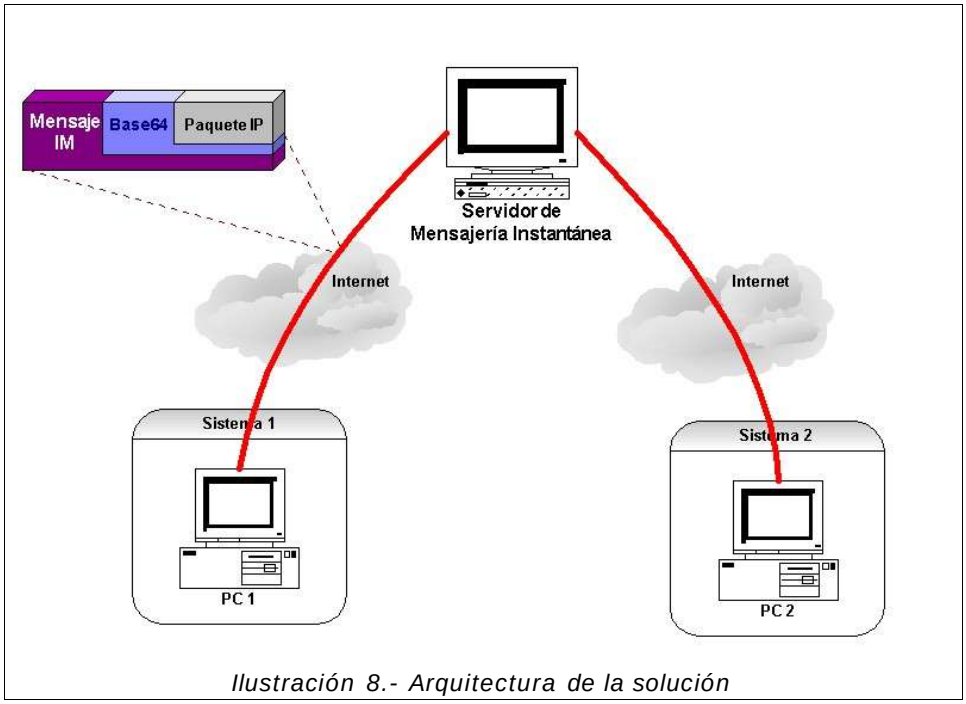


Ilustración 8.- Arquitectura de la solución

Tenemos dos sistemas equivalentes (Sistema 1 y Sistema 2), los cuales utilizan IPoIM para comunicarse.
Para lograr esta comunicación se incorpora un nuevo sistema a la arquitectura: el Servidor de Mensajería Instantánea. A través de este servidor pasaran todos los paquetes de esta comunicación. Estos paquetes son encapsulados en base64 y luego enviados como mensaje de texto a través de la mensajería.

Durante el inicio de sesión de la mensajería instantánea existen otros servidores participantes de esta arquitectura, pero como su aparición ocurre solamente al inicio de la aplicación no se incluyeron, a efectos de clarificar el esquema.

3.4 Resumen

En resumen lo se plantea como solución está basado en tres trabajos principales e independientes.

- Primero la construcción de una API para la mensajería instantánea, basada en la implementación del Gaim, que nos permitirá luego utilizar cualquier protocolo de mensajería indistintamente.

- Luego la adecuación del OpenVPN a nuestra solución. Esto es, utilizar lo ya implementado por el OpenVPN y modificarlo para cumplir nuestro propósito.

- Agregar a los mensajes entre el OpenVPN y la API del Gaim una codificación para poder encapsular los paquetes dentro de la mensajería.

El objetivo de estas implementaciones es construir un prototipo funcional. Este prototipo servirá para realizar pruebas, y obtener resultados sobre la mensajería instantánea como transporte de datos.

4 – IMPLEMENTACIÓN

Dentro de este capítulo no se abarcan las implementaciones desde un punto de vista técnico (los detalles técnicos se especifican en el Apéndice A: Implementación). Simplemente se detalla a grandes rasgos las modificaciones e implementaciones que se realizaron.

4.1 Plan de Implementación

Los pasos para la implementación se basaron en construir prototipos por funcionalidad que nos llevaran a atacar los problemas de cada decisión tomada, para luego ensamblar los conocimientos obtenidos durante el desarrollo, en la construcción de una solución mediante prototipos incrementales.

Como se pudo ver en el capítulo anterior, la solución planteada se basa en la utilización de dos componentes de software completamente independientes. Esto permite poder realizar distintos enfoques a lo largo de la implementación de la solución.

El plan de implementación se basa en 3 etapas:

- **Prototipo 1.-** Tráfico IP sobre interfaces virtuales.

En primera instancia, se intentó conocer a fondo las soluciones a nivel de implementación para las VPNs. Estas implementaciones traen un concepto de conexión virtual que permite a las aplicaciones utilizar una conexión punto a punto con otra maquina. Esta conexión es simulada por las aplicaciones VPNs para ser transparente al resto del sistema. Es importante para saber cómo será el funcionamiento del prototipo que se quiere construir, conocer en profundidad como es implementada esta simulación y mediante que herramientas o procedimientos se la puede manipular. De esta inquietud surge el primer prototipo construido.

- **Prototipo 2.-** Mensajería Instantánea

Luego también se necesita conocer el funcionamiento del componente elegido para la mensajería instantánea, Gaim. En pos de construir una API para luego incorporarla a la solución final, es útil construir un prototipo que utilice estas funcionalidades.

- **Prototipo Incremental.-** IP over IM.

Finalmente, cuando se tienen resueltos los dos prototipos anteriores se puede comenzar a construir el prototipo final. Esta etapa se realizará en forma incremental, para mitigar los riesgos, al ir incorporando nuevas funcionalidades. Que sea incremental significa que este prototipo ira creciendo hasta convertirse en la solución final.

Las motivaciones de este plan es lo que a continuación se detalla.

Prototipo 1 – Motivación: Aprendizaje.

Para el caso de las aplicaciones de túneles antes vistas, en las cuales se incluye el OpenVPN, utilizan un componente de software externo para resolver el acceso de las aplicaciones al túnel. Para ello utilizan "Universal TUN/TAP device driver" [20].
Este software provee recepción y transmisión de paquetes en espacio de usuario. Puede ser visto como una interfaz simple punto a punto o ethernet, en donde, en vez de enviar/recibir paquetes hacia/desde el medio físico, envía/recibe hacia/desde programas.

TUN: Interfaz de Red Virtual Punto a Punto.
TAP: Interfaz de Red Virtual Ethernet.

TUN trabaja con frames IP, mientras que TAP con frames ethernet.
Tiene distribuciones para los sistemas operativos, Linux kernels 2.2.x, 2.4.x, 2.6.x y Windows (NT).

Este componente de software es también parte de la solución por estar incluido dentro del OpenVPN y es por eso que se construyó un prototipo para comprender su funcionamiento e investigar como se utiliza desde las aplicaciones construidas.

Prototipo 2 – Motivación: Riesgos

El componente de software a utilizar para la mensajería instantánea, Gaim, es una aplicación con interfaz gráfica. En nuestra solución no utilizamos interfaz gráfica, ya que se basa en la modificación del OpenVPN y éste es una aplicación de consola, mas precisamente un demonio. Es por eso que una de las motivaciones principales para construir este prototipo es lograr utilizar las funcionalidades del Gaim, sin incluir su interfaz gráfica.
Es también importante poder conocer cómo y cuan modular es su implementación antes de comenzar a incorporarla en nuestra solución. Se necesita un estudio de su implementación, revisando su código fuente y la documentación disponible.

Prototipo Incremental – Motivación: Construir solución final.

Luego de tener un claro panorama de las herramientas utilizadas, se puede comenzar a construir la solución final.
Esta implementación se basa en:

1. Estudiar la implementación del OpenVPN: Se necesita un estudio de los distintos módulos del OpenVPN, y de cómo interactúan. Para realizar este estudio, se deben analizar las documentaciones disponibles, revisar el código fuente, y realizar pruebas con la aplicación.

2. Incorporar a esta implementación los cambios necesarios: En esta parte se determinan cuales serían las modificaciones necesarias para lograr nuestra solución. Esto es un resultado del estudio realizado anteriormente y del conocimiento obtenido a través de la construcción del prototipo 1.

3. Incorporar el estudio de la implementación del Gaim: Utilizar los conocimientos obtenidos en la construcción del prototipo 2, y los dos puntos anteriores para implementar la integración de los dos componentes, Gaim y OpenVPN.

4.2 Metodología

Al ser este un proyecto de reutilización de componentes externos, la documentación con respecto a los detalles de implementación se desarrolla como una guia para el lector, es decir, el detalle del proceso seguido con cada uno de las soluciones a efectos de determinar su funcionamiento, e identificar los puntos importantes para este proyecto.
Por lo tanto, a continuación se detalla el proceso para la implementación.

4.3 Detalles de Implementación

En esta sección se incluyeron los detalles de las implementaciones de los tres prototipos antes descriptos. El tercer prototipo irá incluyendo las funcionalidades hasta convertirse en la implementación de la solución final.

4.3.1 Prototipo 1.- Tráfico IP sobre interfaces virtuales.

El objetivo de este prototipo es conocer como funciona internamente la aplicación que vamos a utilizar, saber que solución utiliza para la

simulación de una conexión.
En primer lugar se realizó una estudio del software TUN/TAP que es el que utiliza el OpenVPN para crear una interfaz virtual. La interfaz virtual es una conexión de red, que es vista solamente desde el sistema operativo, pero no tiene una correspondencia física (tarjeta de red, módem, etc) en el sistema.
Este estudio contiene la construcción de una aplicación que sea ejecutable tanto para Linux como para Windows, y que sea capaz de utilizar esta conexión virtual, para el envío y la recepción de tráfico IP.

Esto se resume en dos funciones:

- Detectar y obtener los paquetes IP escritos por aplicaciones que utilizan la conexión virtual para comunicarse.

- Escribir paquetes IP en la conexión virtual.

La primera etapa de este estudio involucra la instalación del componente de software, los pasos para su instalación son parte del contenido del Apéndice B: Instalación.

Luego de instalado este software queda en cada sistema operativo como cualquier otra interfaz de red ethernet o ppp.

La segunda etapa consiste en implementar un módulo capaz de detectar los paquetes IP escritos por las aplicaciones en la interfaz virtual.
Para la detección y el análisis de paquetes de red, se utiliza una librería conocida PCAP [21], que tiene implementaciones tanto para Windows, como para Linux, y con la cual basamos esta implementación.
Se realizaron 4 aplicaciones, separándolas por plataforma (Windows y Linux) y luego por funcionalidad (enviar, recibir).

Obtener paquetes IP:

Primero se abre la interfaz virtual mediante la función

```
pcap_t *pcap_open_live(const char *device, int snaplen,
                       int promisc, int to_ms, char *errbuf)
```

la cual permite a través del nombre de la interfaz `device`, abrirla y detectar los paquetes que recibe.

Luego se establece un filtro a la referida detección, para que los paquetes obtenidos sean solamente los IP. Esto se realiza mediante la invocación a las siguientes funciones.

```
int pcap_lookupnet(const char *device,
                   bpf_u_int32 *netp,
                   bpf_u_int32 *maskp,
                   char *errbuf)
```

```
int pcap_compile(pcap_t *p,
                 struct bpf_program *fp,
                 char *str,
                 int optimize,
                 bpf_u_int32 netmask)


int pcap_setfilter(pcap_t *p,
                   struct bpf_program *fp)
```

Luego la aplicación establece una espera a la llegada de paquetes. Puede realizarse almacenando los paquetes en un archivo directamente mediante las funciones:

```
int pcap_loop(pcap_t *p,
              int cnt,
              pcap_handler callback,
              u_char *user)

void pcap_dump(u_char *user,
               struct pcap_pkthdr *h,
               u_char *sp)
```

o utilizando la función:

```
int pcap_next_ex(pcap_t *p, struct pcap_pkthdr **pkt_header,
                 const u_char **pkt_data)
```

con la cual se obtienen por un lado el header del paquete, `pkt_header` y por otro lados los datos, `pkt_data`.

Enviar paquetes IP:

Para escribir paquetes IP en la interfaz virtual se utilizan dos metodologías distintas según el sistema operativo, ya que la librería PCAP ofrece funciones de envío de paquetes para el caso de Windows, pero no para el caso de Linux.

En el caso de Windows se utilizo la función:

```
int pcap_sendpacket (pcap_t *p,
                     u_char *buf,
                     int size)
```

En el caso de Linux se utilizaron raw sockets, mediante las funciones implementadas para sockets.

En los dos casos, estos paquetes IP son previamente leídos desde un archivo, que puede ser generado con la implementación anterior (*Obtener paquetes IP*).

Resultados

Estas funcionalidades nos permiten conocer el nivel de complejidad en la utilización de conexiones virtuales mediante el software TUN/TAP.

De este trabajo podemos concluir que la utilización del software de TUN/TAP para las conexiones virtuales, no conlleva una complejidad mayor que la implementación de una comunicación con otro tipo de dispositivo de conexión.

4.3.2 Prototipo 2.- Mensajería Instantánea

El segundo prototipo a construir se refiere a la mensajería instantánea. El objetivo de este prototipo es desarrollar una aplicación en consola capaz de comunicar mensajes de texto mediante mensajería instantánea.

Para lograrlo, se debe remover todo lo referido a interfaz gráfica y contar con las siguientes tres funciones:

- Iniciar Sesión
- Enviar Mensajes IM
- Recibir Mensajes IM

Estas funcionalidades serán implementadas en base a la implementación del componente de software: Gaim.

Una de las primeras dificultades encontradas al comenzar el trabajo con este componente, es la documentación. La documentación que existe y está disponible, es muy detallista en “que realiza” cada una de las funciones que se implementan en los módulos del código fuente, pero carece de información de “como se utilizan e integran” estas funciones.

Implementación de Funcionalidades mediante Gaim

Previo al comienzo de esta implementación, se realizó un estudio que diera un panorama general de la implementación del Gaim. Este estudio tiene como objetivo identificar cuales son los módulos, estructuras y funciones que serán necesarias.

Análisis de implementación.

En el diseño de programa del Gaim cada funcionalidad está en un módulo diferente. Para verlo más claramente, a continuación se detallan los archivos que componen la API del Gaim:

```
file  account.h         Account API.
file  accountopt.h      Account Options API.
file  away.h            Away API.
file  blist.h           Buddy List API.
file  buddyicon.h       Buddy Icon API.
file  cmds.h            Commands API.
file  connection.h      Connection API.
file  conversation.h    Conversation API.
file  debug.h           Debug API.
file  eventloop.h       Gaim Event Loop API.
file  ft.h              File Transfer API.
file  imgstore.h        IM Image Store API.
file  internal.h        Internal definitions and includes.
file  log.h             Logging API.
file  network.h         Network API.
file  notify.h          Notification API.
file  plugin.h          Plugin API.
file  pluginpref.h      Plugin Preferences API.
file  pounce.h          Buddy Pounce API.
file  prefs.h           Prefs API.
file  privacy.h         Privacy API.
file  proxy.h           Proxy API.
file  prpl.h            Protocol Plugin functions.
file  request.h         Request API.
file  roomlist.h        Room List API.
file  server.h          Server API.
file  signals.h         Signal API.
file  sound.h           Sound API.
file  sslconn.h         SSL API.
```

Luego de analizar estos archivos y también el archivo que contiene el `main()` del Gaim, se llego a la conclusión de que las APIs que realizan las funciones básicas de mensajería instantánea y serán suficientes para la implementación de este prototipo son:

- Account API:

  Contiene los datos de las cuentas registradas, nombre de usuario, protocolo de mensajería que utiliza, contraseña, entre otras.

  Comienza el proceso de inicio de sesión.

- Connection API:

  Maneja las conexiones asociadas a las cuentas. No es usado directamente, sino que cada `Account` utiliza una `Connection` para establecer y mantener la conexión.

- Conversation API:

  Directamente relacionado con las conversaciones entre dos o más usuarios.

  Contiene las funciones para enviar y recibir mensajes instantáneos.

- Gaim Eventloop API:

  Maneja la continuidad de los eventos dentro del Gaim, define que evento se debe disparar después de otro.

- Buddy List API:

  Implementa las funciones asociadas a la lista de contactos.

- Plugin API:

  Maneja además de los plug-ins para la interfaz gráfica, los distintos protocolos de mensajería que se usan en la aplicación. Los protocolos son también diseñados como plugins de la aplicación, para hacerla extensible.

  A través de ella podemos cargar la lista de los protocolos de mensajería que están disponibles para utilizar.

El resto de los archivos del Gaim, son utilizados indirectamente.

Eventos

Cada API tiene una estructura que contiene las funciones que son invocadas por los eventos. Cada una de estas estructuras, tiene definidos eventos asociados únicos. Estas funciones pueden ser reescritas por cualquier aplicación, para así capturarlos.
Por ejemplo, en el caso de la API de Account tenemos

```
typedef struct _GaimAccountUiOps GaimAccountUiOps;
struct _GaimAccountUiOps{
      void (*notify_added)(GaimAccount *account,
                           const char *remote_user,
                           const char *id,
                           const char *alias,
                           const char *message);
};
```

Con la cual se puede capturar el evento de `notify_added`. Este evento se refiere a cuando el usuario es agregado por otro usuario a su lista de contactos.

Una aplicación que quisiera reescribir el resultado de la invocación a este evento, debería utilizar la función

```
gaim_accounts_set_ui_ops(GaimAccountUiOps *ops)
```

Muchos de estos eventos son utilizados por el Gaim para invocaciones a

funciones que despliegan ventanas de su interfaz gráfica. Por esto, en la implementación del prototipo se deben anular estas invocaciones.

Siguiendo con el ejemplo del account antes visto, esto se podrá lograr a través de la llamada:

```
gaim_accounts_set_ui_ops(NULL)
```

Señales

Las API's del Gaim implementan la emisión de señales. Estas señales son identificadas por su nombre y cada API tiene una cantidad limitada de señales que emite.
Cualquier aplicación puede registrarse como receptora de estas señales. Registrarse como receptora, significa la invocación de una función implementada a la llegada de la señal.

Para inicializar el registro de las señales en un componente, se debe invocar al inicio `gaim_component_init()` por cada componente.
Continuando con el ejemplo de account.

```
gaim_account_init()
```

y una de las señales que emite es

```
account-connecting  Cuando la cuenta está en proceso de
                    inicio de sesión.
```

Para asociar a la emisión de una señal una función implementada externamente, se hace a través de la función:

```
gulong gaim_signal_connect(void * instance,
                           const char* signal,
                           void *handle,
                           GaimCallback func,
                           void *data
                           )
func: la función a invocar.
```

Este manejo de señales es utilizado en el prototipo implementado para detectar cuando se finalizó con el inicio de sesión y para indicarnos cuando llegan nuevos mensajes IM.

Cliente de consola IM con Gaim

Para desarrollar este cliente de mensajería instantánea se utilizaron las APIs de Gaim que analizamos anteriormente.

Inicialización

La primer tarea a realizar es la inicialización. Incluye la carga de los directorios de trabajo, en donde el Gaim va a encontrar implementaciones de los protocolos, archivos de configuración, etc. y esto se hace mediante la función:

Plugin API:

```
void gaim_plugins_set_search_paths(size_t count,
                                   char ** paths)

count: cantidad de directorios a establecer.
paths: array de chars de largo "count" con los
directorios a establecer.
```

Establece los directorios de búsqueda de plugins.

Otra de las tareas de la inicialización es iniciar las llamadas a eventos y señales de cada una de las API's del Gaim.
El Gaim utiliza el "Main loop and Events" contenido en la librería GTK+ [22] (mediante la función `void gtk_main(void)`), para quedar en un loop infinito que es solamente interceptado por eventos. Luego de invocada la función `gtk_main()`, cualquier operación que se requiera hacer en el Gaim es mediante eventos o señales.

Para implementar las tres funciones objetivo de este prototipo (inicio de sesión, envío y recepción de mensajes), se debe primero invocar a la función que inicia el proceso de "inicio de sesión" para la cuenta elegida. Luego de iniciado este proceso, se deja que el Gaim a través de sus eventos informe a la aplicación sobre lo que sucede con el proceso de este inicio de sesión, o con cualquier otro evento que suceda.

Por lo tanto, la inicialización es lo primero a implementar. A esta inicialización se le pueden agregar también funciones que reciban eventos y señales asociados.

Inicio de sesión

Para el caso del inicio de sesión, usamos el Account API. Esta API implementa todo lo que respecta a usuarios con sus protocolos, y el inicio de sesión IM con el servidor que corresponda.
La estructura de una `GaimAccount` cuenta con la información necesaria para lograr esta conexión

```
typedef struct _GaimAccount  GaimAccount;
```

Representa cada cuenta configurada en el Gaim

```
struct _GaimAccount {
     char *username;
     char *alias;
     char *password;
     char *user_info;
     char *buddy_icon;
     gboolean remember_pass;
     char *protocol_id;
     GaimConnection *gc;
     GHashTable *settings;
     GHashTable *ui_settings;
     GaimProxyInfo *proxy_info;
     GSList *permit;
     GSList *deny;
     int perm_deny;
     GaimLog *system_log;
     void *ui_data;
};
```

Los que nos interesan son:

username: nombre de usuario
password: contraseña
protocol_id: identificador del protocolo
Por ej: prlp-msn
gc: conexión usada por la cuenta

Las cuentas son guardadas por el Gaim dentro de un archivo xml "accounts.xml" (Ver Apéndice B: Instalación), para levantar la estructura de cuentas se debe utilizar la función:

Account API:

```
gboolean gaim_accounts_load ()
```

Carga las cuentas configuradas.

Una vez obtenida la cuenta, con la información necesaria en su estructura, podemos llamar a la función:

Account API:

```
GaimConnection* gaim_acount_connect(GaimAccount *account)
account: cuenta a conectar.
```

Conecta la cuenta.

Esta función primero crea la conexión (`GaimConnection`), la cual será usada durante la sesión de IM. Con la información de la cuenta, que incluye información también sobre el protocolo, comienza con el procedimiento de "iniciar sesión" de la mensajería.

Luego de invocar a esta función, la respuesta y la información del progreso de la conexión es manejada mediante señales.

Protocolos

La implementación de los protocolos en Gaim es extensible. El Gaim define una interfase y se debe implementar esa interfase para utilizar un protocolo a través del Gaim. Luego se compilan como una librería, en el caso de Linux extensiones la y so, y en el caso de Windows dll.

Recibir IMs

La solución encontrada para detectar la recepción de mensajes que llegan para la cuenta conectada, es registrando la señal `received-im-msg` emitida por Conversation API.

Para poder registrarse a esta señal, primero se necesita obtener el "handle" de Conversation mediante la función:

```
void* gaim_conversations_get_handle(void)
```

Para registrarse luego a través de la función:

```
gulong gaim_signal_connect(void * instance,
                           const  char* signal,
                           void *handle,
                           GaimCallback func,
                           void *data)
```

```
instance: la instancia a conectar la señal.
signal: el nombre de la señal a conectar.
handle: el handle del que recibe la señal.
func: la función a invocar.
data: los datos a enviar a la función.
```

Desde la aplicación la llamada sería de la forma:

```
gaim_signal_connect(gaim_conversations_get_handle(),
                    "received-im-msg",
                    g_main_context_default(),
                    GAIM_CALLBACK
                    (gaim_IPoIM_received_im_msg),
                    NULL);
```

en donde `gaim_IPoIM_received_im_msg`, es la función implementada dentro del prototipo. Esta función es la que realiza la recepción de los

mensajes IM y los imprime en pantalla:

```
void gaim_IPoIM_received_im_msg(GaimAccount *account,
                                char *sender,
                                char *buffer,
                                int flags,
                                void *data);
```

```
account: cuenta que recibe el mensaje.
sender: usuario que envía el mensaje.
buffer: mensaje recibido.
flags: banderas.
data: parámetros extra.
```

Los parámetros de la función son definidos por la emisión de la señal, a excepción del parámetro data, en donde se pueden pasar parámetros extra, definidos por el implementador a la función.
El parámetro `buffer` que contiene el mensaje recibido, es luego lo que se imprime en pantalla, para cumplir con la funcionalidad de recibir IMs.

Envío de IMs

Para lograr que lo escrito en consola pueda enviarse como mensaje IM a algún usuario, se tuvo que incorporar los threads a la implementación. Esto se debe a que el Gaim, luego de iniciado su loop principal, como vimos anteriormente sólo responde a eventos. En el caso de trabajar con una ventana de interfaz gráfica se disparan los eventos provocados en la misma, por ejemplo el click en un botón.
En la implementación de este prototipo, al no tener interfaz gráfica y para poder permanecer esperando en consola por la escritura desde el teclado para el envío, se resolvió que el uso de los threads era la solución mas apropiada. Para ello se utilizaron dos threads, un lector y un escritor. Dentro del lector se implementa la lectura desde consola, mas el envío de mensajes. Dentro del escritor el loop principal, incluyendo la recepción de mensajes IM.

Para enviar los mensajes se debe seguir los siguientes pasos:

1. Crear una nueva conversación con el usuario a quien se le envía el mensaje mediante la función:

   Conversation API:

   ```
   GaimConversation* gaim_conversation_new
                        (GaimConversationType type,
                         GaimAccount * account,
                         const char * name)
   ```

```
type : tipo de conversación enumerado
(GAIM_CONV_UNKNOWN, GAIM_CONV_IM, GAIM_CONV_CHAT,
GAIM_CONV_MISC).
account: La cuenta que envía el mensaje.
name: nombre de la conversación.
```

Crea una nueva conversación

2. Enviar el mensaje a través de la función:

Conversation API:

```
void gaim_conv_im_send(GaimConvIm * im,
                       const char * message)

im: conversación a la que se le enviara el mensaje.
message: mensaje a enviar.
```

Envía un mensaje mediante la conversación de IM.

Para utilizar esta función se pasa como primer parámetro GAIM_CONV_IM(GaimConversation), a partir de una conversación creada con la función anterior.

Módulo Principal

Para utilizar la función `gtk_main()` de la librería GTK que implementa el loop principal que guia la implementación del Gaim, se debe invocar la función que inicializa esta funcionalidad de la librería,

```
void gtk_init(int *argc, char ***argv);
```

Inicializa todo lo necesario para operar con las herramientas gtk.

Luego de esto se realiza la *inicialización* detallada anteriormente.

Para obtener una cuenta previamente configurada mediante el nombre de usuario y protocolo se utiliza la función:

Account API:

```
GaimAccount * gaim_accounts_find (const char *name,
                                  const char *protocol)

name: nombre de usuario en la cuenta.
protocol: identificador del protocolo (ej.: prlp-jabber)
```

Retorna la cuenta especificada para ese nombre y protocolo.

Luego de obtener la cuenta, se utiliza el proceso detallado en *Inicio de sesión*.

Se registra la función asociada a la señal de "`received-im-msg`", detallada en *Recibir IMs*.

En resumen, para incializar una conexión utilizando lo implementado por el gaim se debe en primer lugar:

1. Incializar componentes, directorios de búsqueda y configuración.

2. Elegir una cuenta a utilizar, e invocar la función `gaim_account_connect(g)`.

3. Asociar una función a la emision de la señal, `received-im-msg`.

De esta manera damos por inicializados los componentes y podemos comenzar la aplicación.

Para comenzarla se crean e inician los dos threads necesarios indicados en *Enviar Mensajes IMs*.

Para el manejo de los threads se utilizaron funciones de la libreria Glib [23]. Los threads se crearon mediante:

```
GThread* g_thread_create  (GThreadFunc func,
                           gpointer data,
                           gboolean joinable,
                           GError **error)

func : la función que ejecuta el nuevo thread.
data : el argumento pasado a thread.
joinable : si es un join thread,
error : retorna el error cometido.
```

Crea un nuevo thread.

Se crean para el modulo principal entonces, dos threads:

- El primero espera por la llegada de mensajes IM, y recibe las señales y eventos emitidos por la implementación del Gaim.

- El segundo espera por la escritura de texto en la consola. Por cada escritura detectada, ésta se envía como mensaje IM a un usuario definido.

Resultados

La construcción de este prototipo fue fundamental para comprender la implementación del Gaim. Esta implementación resulto ser más compleja de lo esperado, ya que su implementación esta muy ligada entre si. Si bien la implementación tiene una encapsulación adecuada, la interdependencia entre los módulos fue la que hizo difícil su comprensión y utilización.
La construcción de este prototipo, independientemente, facilitará mucho el trabajo del prototipo final, porque en esta etapa ya se atacaron todas las dificultades de la solución elegida para mensajería instantánea, Gaim.

4.3.3 Prototipo Incremental - IP over IM.

Este prototipo es implementado de forma incremental, es decir, se le irán agregando las funcionalidades y se lo irá modificando para llegar al objetivo final.
El principal componente a investigar en este punto de la implementación es el OpenVPN. Lamentablemente no se encuentra disponible ninguna documentación sobre su implementación, aunque sí un foro de discusión de desarrolladores del componente.

Análisis de implementación de OpenVPN.

Para poder utilizar lo implementado en el componente OpenVPN se realiza un estudio previo de sus archivos y módulos.
Luego de analizarlos, se concluye que los que se consideran importantes para la construcción de este prototipo son:

```
files buffer.c buffer.h   Buffer usado para los paquetes
files forward.c forward.h Transmisión de paquetes
files init.c init.h       Inicialización de variables
files openvpn.c openvpn.h Main principal
files options.c options.h Opciones de configuración
```

A través del conocimiento de estos archivos que contienen los datos necesarios para esta implementación, se podrá luego comenzar con las modificaciones necesarias.

buffer:
Dentro de este archivo se encuentran las estructuras de datos y funciones utilizadas para manejar los datos que contienen los paquetes.

forward:
Las implementaciones para la lectura/escritura sobre la conexión virtual, así como lectura/escritura sobre la conexión real, son llevadas a cabo a través de este módulo.

init:
Aquí se implementan las inicializaciones sobre los parámetros y estructuras de datos.

openvpn:
Este es el módulo principal y además implementa las estructuras de datos que contienen toda la información necesaria para administrar la aplicación.

options:

Contiene las estructuras de datos y las funciones que manejan todos los parámetros que recibe la aplicación.

Detalles generales sobre la implementación del OpenVPN

Lo primero que realiza el OpenVPN es inicializar su estructura principal, `struct context`, la cual contiene toda la información sobre las opciones proporcionadas a la aplicación y los datos y estados de las conexiones, entre otros.

Luego realiza la creación y configuración de la interfaz virtual que utilizara durante la ejecución de la aplicación:

En Linux esto se realiza mediante software, se invoca a través de la función `ifconfig`.

En Windows debe estar previamente creada, y utiliza la primera que se encuentre disponible (no utilizada por otra aplicación).

Para la comunicación tanto con la conexión virtual como con la real, la aplicación utiliza `sockets` para todas las operaciones, abrir, escribir y leer.

En su módulo principal, el OpenVPN implementa una iteración infinita, en donde se espera por la recepción de señales. Estas señales llegan pre-filtradas a la iteración principal, conteniendo sólo las cuatro que son de interés para la aplicación.

Estas señales se detallan a continuación:

1. La conexión real TCP/UDP ha recibido datos: Esta señal ocurre cuando llega un paquete desde otra máquina, con la cual tengo establecido un túnel. Al recibir esta señal, la aplicación lee los datos recibidos desde la conexión, verifica su integridad y remueve del paquete todos lo extra headers incluidos al tunelizar (desempaquete). Luego realiza un pedido de escritura en la conexión virtual con el resto de los datos.

2. La conexión real TCP/UDP lista para recibir escritura: Ocurre cuando la aplicación tiene datos listos para ser enviados al otro punto del túnel. La aplicación entonces construye un nuevo paquete a partir de, los datos mas extra headers incluidos al tunelizar. Luego escribe estos datos sobre la conexión.

3. La conexión virtual lista para recibir escritura: Esta señal es recibida luego del punto 1, los datos que llegaron en este

punto, son entonces escritos en la conexión virtual.

4. La conexión virtual ha recibido datos: Esta señal se recibe cuando alguna aplicación está enviando datos hacia el túnel. El sistema entonces toma estos datos para luego invocar el punto 2.

Para ilustrar mejor estas operaciones está la siguiente imagen.

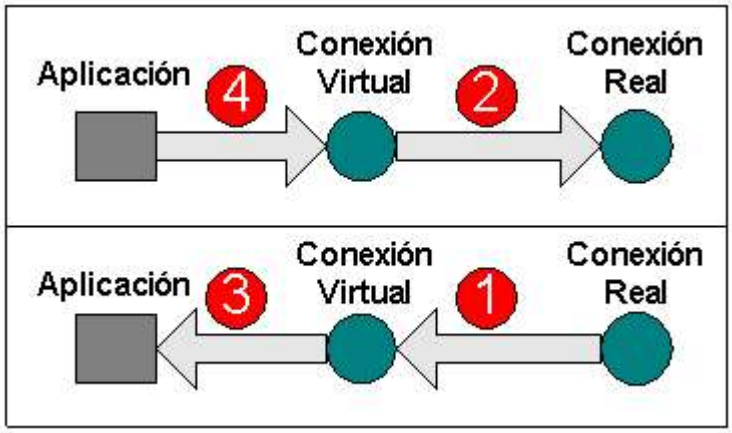


Ilustración 9.- Señales del OpenVPN y su relación con la aplicación y las conexiones.

Todas estas señales son procesadas por la función

files forward.h forward.c

```
void process_io (struct context *c)
```

Procesa las señales recibidas.

Esta función luego invoca otras 4 (o 6) funciones según la señal recibida:

files forward.h forward.c

```
void process_outgoing_link(struct context *c);
```

Escribe en la conexión real, punto 2 de la figura.

```
void process_outgoing_tun(struct context *c);
```

Escribe en la conexión virtual, punto 3 de la figura.

```
void read_incoming_link (struct context *c);
void process_incoming_link (struct context *c);
```

Lee y procesa lo recibido desde la conexión real, punto 1 de la figura.

```
void read_incoming_tun (struct context *c);
void process_incoming_tun (struct context *c);
```

Lee y procesa lo recibido desde la conexión virtual, punto 4 de la figura.

Además de estas cuatro señales, el OpenVPN se apoya en una estructura `context`. Esta estructura contiene los datos recibidos y a enviar durante la ejecución de la aplicación, y mediante ella es que la aplicación es administrada. Esta estructura también contiene los `sockets` creados para cada una de las conexiones.

Estos puntos resumen, a nuestro criterio, lo más importante que se debe conocer del OpenVPN para poder comenzar a construir el prototipo funcional.

IPoIM (IP over IM)

Este prototipo, como vimos anteriormente, será una adaptación del OpenVPN a los requerimientos. Para construir nuestro prototipo se harán modificaciones al OpenVPN y se le incluirán las funcionalidades de la mensajería.

Luego del análisis anterior sobre la implementación del OpenVPN, se concluye que las modificaciones a realizar son fundamentalmente en las señales. Para nuestra solución las señales 3 y 4 deben permanecer con el mismo comportamiento y son las 1 y 2 las que se deben reemplazar por las operaciones de recepción y envío de mensajería respectivamente.

El punto 2 (envío de paquetes) debe ser reemplazado por la función

Envío de Mensaje IM

El punto 1 (recepción de paquetes) debe ser reemplazado por la función

Recepción de Mensaje IM

Para poder implantar estos cambios en el código del OpenVPN, se

intentó modificar lo mínimo posible los códigos originales e incluir código extra mediante la inclusión de nuevos archivos fuentes al proyecto.

Primero se detallaran las implementaciones independientes, fuera de la implementación del OpenVPN, y luego las modificaciones que se realizaron sobre el código original.

Implementaciones

Las implementaciones independientes, representan los módulos que se implementaron no como parte de los archivos del OpenVPN sino como archivos independientes.

Lo implementado será luego utilizado para la implantación de las modificaciones al código original del OpenVPN.

API Gaim_IPoIM (files gaim_ipoim.h gaim_ipoim.c)

El primer paso fue implementar un modulo que comunique los códigos fuentes del Gaim con los del OpenVPN. Para esto se hicieron implementaciones de las tres funciones, que como vimos en el capítulo de solución completan las tres operaciones básicas de la mensajería instantánea, inicio de sesión, envió y recepción de IMs.

```
void init_gaim(char *account,
               char *protocol,
               struct context *c)
```

`account: nombre de la cuenta con la cual se inicia la sesión.`
`protocol: nombre del protocolo usado por la cuenta.`
`c:  contexto con el cual opera el OpenVPN.`

`Inicializa e inicia el proceso de inicio de sesión.`

Esta función primero inicializa los directorios y las estructuras de datos para poder utilizar las funcionalidades del Gaim.
Luego utiliza la cuenta y el protocolo para inicializar el proceso de inicio de sesión.
Finalmente registra a la función `gaim_IPoIM_received_im_msg` a la señal de recepción de mensajes IM `received-im-msg` de `Conversation API`, y utiliza el contexto `c` para que éste sea pasado como parámetro al invocar esta función de recepción. Este contexto es necesario, ya que luego será utilizado al recibir mensajes y procesarlos como paquetes.

```
void gaim_IPoIM_send_im_msg(char *account,
                            char *protocol,
                            char *to,
                            char *msg)
```

account: nombre de la cuenta que tiene previamente completo el inicio de sesión y será utilizado como quien envía del mensaje.
protocol: nombre del protocolo usado por la cuenta.
to: nombre de usuario al cual se le envía el mensaje.
msg: mensaje de texto a ser enviado mediante mensajería instantánea.

Envía un mensaje.

Esta función se encarga de enviar el mensaje mediante el Gaim. Para lograrlo primero ubica la conexión (GaimConnection) mediante la ubicación de la cuenta (GaimAccount) que tiene establecida la sesión, esto se hace utilizando los parámetros account y protocol. Luego crea una nueva conversación (GaimConversation) mediante el parámetro to y account. Con esta conversación y el mensaje invoca a la función del Gaim: gaim_conv_im_send; para finalmente enviar el mensaje.

```
void gaim_IPoIM_received_im_msg(GaimAccount *account,
                                char *sender,
                                char *buffer,
                                int flags,
                                void *data)
```

GaimAccount: la cuenta que recibió el mensaje.
sender: usuario que envió el mensaje.
buffer: texto recibido.
flags: banderas.
data: en este parámetro vienen los datos que se indican cuando se hace el registro al evento, en este caso el data contiene una estructura del tipo (struct context *) que se le indicó en la función init_gaim.

Recibe los mensajes IM.

Esta función es invocada por el Gaim cada vez que se recepciona un mensaje. Lo que implementa es solamente una invocación a una función nueva implementada como parte del modulo OpenVPN, process_new_im (ver *Recepción de paquetes*, más adelante en esta sección). Esta será la encargada de procesar los mensajes recibidos, y se detalla más adelante. A esta función se le pasa el mensaje recibido y el data struct context recibido como parámetro.

API Encoding (files encoding.h encoding.c)

Encoding es básicamente una API que permite que la solución sea extensible con respecto a la codificación que se utilice. Como vimos en el capítulo 3, sección *Codificación* , se necesita una codificación para transferir los paquetes que los transforme de binarios a texto y viceversa. En nuestro caso elegimos utilizar el Base64, pero mediante esta API se podría cambiar el algoritmo a utilizar para realizar esta transformación sin modificar el resto del código.

```
int ALGORITMO_BASE64 = 1;
```

Esta constante define los códigos de los algoritmos disponibles para utilizar. Si se quisiera agregar un nuevos algoritmo, se debería incluir el fuente en donde este implementado, y agregar la constante que lo identifique (Por ej,: `int MI_ALGORITMO = 2`).

```
int encode(int type,
           const void *data,
           int size,
           char **str);
```

```
type: es el código de algoritmo que se quiere utilizar
para la codificación.
data: los datos a codificar.
size: largo de los datos a codificar, largo de data.
str: variable utilizada para retornar el resultado de la
codificación.
```

Codifica binario a texto.

La función lee el parámetro type y dependiendo de su valor invoca a la función de codificación correspondiente al algoritmo. Retorna el éxito o fracaso resultante.

```
int decode(int type,
           const char *str,
           void *data);
```

```
type: es el código de algoritmo que se quiere utilizar
para la de codificación.
str: el texto a decodificar.
data: variable utilizada para retornar el resultado de la
decodificación.
```

Decodifica texto en binario.

La función lee el parámetro `type` y dependiendo de su valor invoca a la función de decodificación correspondiente al algoritmo. Retorna el éxito o fracaso resultante.

Modificaciones al código

Parámetros de configuración

Según las especificaciones de las funciones que vimos en la API del Gaim, API Gaim_IPoIM, para poder utilizarlas se necesita saber el nombre de la cuenta y el protocolo con el cual se iniciará la sesión, y también el nombre de usuario del otro punto de la conexión.
Para estos se modificó la estructura de opciones, agregando los parámetros necesarios:

file options.h

```
struct options

char *account_ipoim;
```
Nombre de la cuenta que inicia sesión IM

```
char *protocol_ipoim;
```
Nombre del protocolo asociado a la cuenta

```
char *account_toipoim;
```
Nombre de usuario con el cual se comunicará el túnel

y en la función:

file: options.c

```
static int
add_option (struct options *options,
         int i,
         char *p[],
         const char *file,
         int line,
         const int level,
         const int msglevel,
         const unsigned int permission_mask,
         unsigned int *option_types_found,
         struct env_set *es)
```

Se encarga de validar y cargar los opciones de configuración.

Luego de introducir estas modificaciones, las opciones de configuración

del ipoim pueden ser manejados como cualquier otro parámetro de configuración del OpenVPN mediante el prefijo:

```
--ipoim account protocol accountto
```

Iteración Principal

Otra de las modificaciones importantes se realizó sobre la iteración principal. Previo al ingreso de la aplicación a la iteración principal se agregó la función que inicializa e inicia la sesión de mensajería implementada en la Gaim API

file: gaim_ipoim.h

```
void init_gaim(char *account, char *protocol, struct context *c).
```

A esta función se le pasan las opciones agregadas a la configuración `account` y `protocol`, más el `context` el cual contiene el contexto dado para la instancia corriente del OpenVPN.

Para que el sistema espere hasta que se haya completado el proceso de inicio de sesión en la mensajería, se agregó:

file: openvpn.h

```
int connected;
```

Esta variable es inicializada en 0. Al inicializar el Gaim la aplicación registra el evento:

Connection API
```
void (*connected)(GaimConnection *gc);
```

para así en la función invocada por este evento, asignar el valor 1 s esta variable. De esta manera le indicamos a la aplicación que el proceso de inicio de sesión ha finalizado exitosamente.

Es de esta manera, que la iteración principal no comienza hasta que la variable `connected` no tenga su valor en 1.

Luego, también dentro de la iteración principal, se removieron las atenciones a las señales 1 y 2 (recepción y envío de paquetes IP, ver Ilustración 9.- Señales del OpenVPN y su relación con la aplicación y las conexiones.).

Envío de Paquetes

Como vimos anteriormente en el análisis del OpenVPN, el proceso de las señales se realiza en el archivo `forward` a través de la función `void process_io (struct context *c)`, la cual a su vez invoca a funciones correspondientes a cada señal.

Para implementar ipoim se debió modificar la función que se corresponde con el punto 2,

file: forward.c

```
void process_outgoing_link (struct context *c)
```

a efectos de reemplazar el envío mediante `sockets` y sustituirlo por el envío mediante la función antes implementada en la API del Gaim_IPoIM:

file: gaim_ipoim.h

```
void gaim_IPoIM_send_im_msg(char *account,
                            char *protocol,
                            char *to,
                            char *msg)
```

Para poder invocar esta función, los parámetros `account`, `protocol` y `to` los obtenemos de la estructura `struct options` incluida en `struct context`, la cual es recibida como parámetro de la función.

El parámetro `msg` lo obtenemos del atributo `to_link` de la estructura `struct context`. Este `to_link` es del tipo `struct buffer`. Este buffer contiene el mensaje a ser enviado a través del túnel.
Para que este paquete pueda ser procesado luego por la aplicación en el otro punto del túnel, se debe agregar al envío la dirección IP virtual origen.

Por lo tanto el mensaje es enviado a través de la mensajería de la forma:

```
from_addr: dirección IP local del túnel
to_link: paquete original
len_data: largo del paquete
```

Estos tres datos se copian a un espacio de memoria y se codifican a mensaje de texto utilizando la función antes implementada en la API del Encoding

```
int encode(int type,
           const void *data,
           int size,
           char **str);
```

```
en donde  type = ALGORITMO_BASE64
          data = from_addr + to_link + len_data
          size = len_data.
```

El resultado obtenido en `str` es entonces el cuarto parámetro que completa la llamada a la función `gaim_IPoIM_send_im_msg`.

Recepción de Paquetes.

Recepción de Paquetes – Threads

Para poder recepcionar los paquetes que llegan a través de la mensajería instantánea, nos encontramos con el mismo problema de paralelismo, atacado en el prototipo de mensajería.
En el caso del prototipo estos threads manejaban el paralelismo entre el envío y la recepción de IMs, pero en este caso, los threads manejarían por un lado la recepción de IMs y el proceso de los mismos, y por el otro lado la recepción y el envío de paquetes a la conexión virtual, más el envío de paquetes a la conexión real.

Para incluir estos cambios se modificó la función que contiene la Iteración principal,

`file: openvpn.c`.

```
static void tunnel_point_to_point (struct context *c)
```

Iteración principal

A esta función se le agregan dos threads:

```
GThread *g_tunnel_point_to_point_send;
GThread *g_tunnel_point_to_point_receive;
```

Dentro del thread `g_tunnel_point_to_point_send` se corre el proceso que implementa la iteración principal del OpenVPN.
`void tunnel_point_to_point_send (void *cvoid)`

Dentro del thread `g_tunnel_point_to_point_receive`, es implementada la espera por la recepción de mensajes IM, de la misma manera que se hizo en el prototipo del Gaim.

```
void tunnel_point_to_point_receive ()
```

Para el correcto funcionamiento de la aplicación con estos dos threads, se incluyó también la mutuo exclusión de las operaciones sobre la `struct context`, ya que es utilizada tanto al enviar como al recibir

paquetes. Por lo tanto se agrego la variable

```
static GMutex *mutex;
```

Recepción de Paquetes – mensajería instantánea.

Para la llegada de nuevos mensajes de mensajería instantánea, se utiliza la función implementada en la API Gaim_IPoIM:

file: gaim_ipoim.h

```
void gaim_IPoIM_received_im_msg(GaimAccount *account,
                                char *sender,
                                char *buffer,
                                int flags,
                                void *data)
```

(registrada previamente mediante la señal `received-im-msg` al `Conversation API` en la función `init_gaim`).

Esta función, a su vez invoca a la función:

file: openvpn.c

```
void process_new_im(char *msg,
                    void *data)

msg: mensaje recibido.
data: contexto (struct context).
```

Esta función procesa el paquete recibido como mensaje por el Gaim. Para ello modifica el `struct context` que se obtiene mediante el parámetro `data`.
Cuando el mensaje es recepcionado, primero se realiza su decodificación, mediante la función del API Encoding

```
int decode(int type,
           const char *str,
           void *data)

en donde  type = ALGORITMO_BASE64
          str = msg
```

Una vez obtenido el resultado de la decodificación en el parámetro `data`, se reconstruyen desde memoria, los tres valores contenidos:

```
from_addr: direccion IP de origen.
packet: paquete recibido.
```

```
largo: largo del paquete.
```

Con estos tres datos se modifica el contexto y se invoca a la función,

file: forward.c
```
void process_outgoing_tun (struct context *c)
```

Escribe en la conexión virtual, punto 3 de la Ilustración 9.- Señales del OpenVPN y su relación con la aplicación y las conexiones.

Y así esta función hace que se transmita a la conexión virtual lo recibido.

En conclusión las modificaciones al OpenVPN fueron:

- Inclusión de nuevos parámetros de configuración (cuenta, protocolo y usuario receptor)
- Comunicación TCP o UDP, reemplazada por comunicación IM.
- Inclusión de threads para el manejo de la recepción de IM en paralelo con el resto de la aplicación.

Resumen

En resumen la implementación estuvo basada en la construcción de prototipos, a través del análisis de las implementaciones de los distintos componentes.

Del Gaim se utilizaron las APIs para la construcción de las 3 funciones que cumplen con las tareas de mensajería.
Al OpenVPN se le modificó el código fuente para adaptarlo a la nueva función.

Estos trabajos construyeron el sistema tanto para Linux como para Windows.

Para Windows además se tuvo que implementar el servicio brindado por el IPoIM como una extensión de la aplicación, es decir como una dll (dinamic loadable library), además del ejecutable.
Esta modificación se debe a que los protocolos de mensajería instantánea del Gaim para Windows, son implementados de esta manera y solo pueden ser utilizados accediendo a ellos mediante una extensión y no una aplicación ejecutable.

Por lo tanto para Windows se agrego el fuente:

file: win_openvpn.c

que contiene las funciones necesarias para completar la implementación para este sistema operativo.

4.4 Plan preliminar de pruebas

El plan de pruebas a realizar sobre esta nueva aplicación se realiza en dos etapas: testing y pruebas.

Es importante la realización de la etapa de testing ya que al estar modificando código implementado externo, el riesgo de cometer errores de implementación puede ser mayor.
La etapa de pruebas de esfuerzo será importante a efectos de evaluar el prototipo construido.

4.4.1 Testing

Plan

La etapa de testing se basa en utilizar el túnel para identificar errores u omisiones en la implementación. Esto se puede testear utilizando el túnel por ejemplo para compartir información o utilizar servicios desde uno de los puntos.

Para cumplir esta etapa se decidió que la aplicación debería:

- Establecer el túnel utilizando algún protocolo de mensajería instantánea.
- Compartir archivos mediante este túnel.
- Iniciar algún servicio en una de las maquina y utilizar este servicio desde la otra.

Si la aplicación es capaz de realizar exitosamente estos tres puntos, podemos utilizarla para el resto de las pruebas.

Testeo

El testing se realizó en primera instancia utilizando el protocolo Jabber. Esta elección se debe a que el protocolo Jabber tiene implementado una herramienta que simula ser un servidor de mensajería Jabber, que puede ser instalado en cualquier máquina y utilizado localmente sin necesitar conexión a Internet.
Con este primer testing se pudieron detectar alguno de los errores de implementación.

Luego, en una segunda instancia se utilizó el protocolo MSN. Para realizar estos test se contó con dos máquinas con el mismo sistema operativo, y utilizando cuentas MSN se estableció el túnel.
El túnel permaneció establecido para poder realizar los próximos dos test: compartir archivos y compartir servicios.
La compartición de archivos se realizó mediante pscp (p secure copy) y el servicio que se utilizó fue el ssh.

Para asegurarnos el correcto funcionamiento del túnel se utilizó la herramienta Ethereal [24] (o tethereal), la cual está disponible en distribuciones para Linux y para Windows.
Esta herramienta permite visualizar los paquetes, detallando los datos que contienen. La siguiente figura muestra un ejemplo de lo que se puede visualizar:

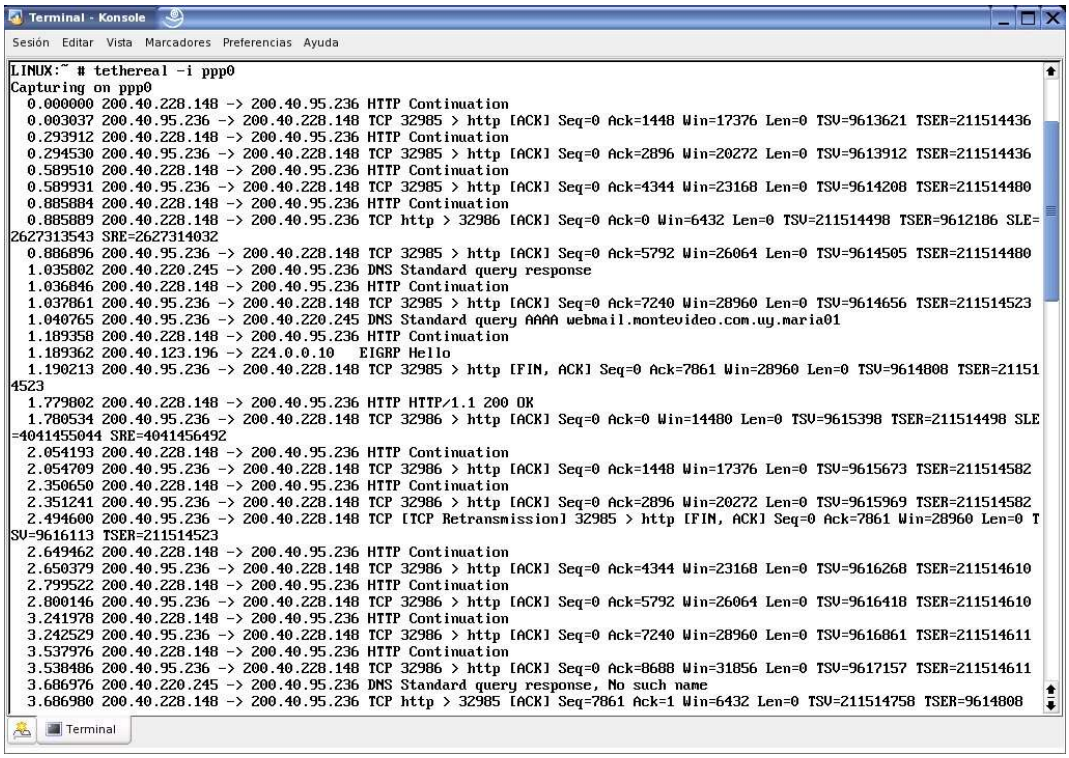


Ilustración 10.- Tethereal

Este ejemplo corresponde a una captura de datos en una conexión ppp, en donde se realiza una navegación web.
Se puede ver en la ilustración que esta herramienta permite visualizar uno a unos los paquetes capturados obteniendo datos de cada paquete. En la ilustración se pueden ver IPs origen y destino, protocolo, entre otros.

4.4.2 Pruebas y Resultados Esperados

Estas pruebas son las que validarán el prototipo implementado y con las cuales obtendremos resultados sobre el comportamiento de la mensajería instantánea como trasporte de datos.

Al comienzo de este proyecto, el primer desafío había sido comprobar su viabilidad. Luego de responder afirmativamente a este desafío, quedaba por conocer su comportamiento. Los resultados de estas pruebas son fundamentales a efectos de conocer cual es el comportamiento, y la estabilidad del prototipo implementado.

Con respecto a los resultados sobre la mensajería como transporte, uno de los puntos más débiles que tiene es que no es independiente. Depende de los servidores de mensajería instantánea, y ésto puede modificar sensiblemente los resultados de las pruebas y la evaluación final.
Los tiempos de respuesta podrán estar atados al nivel de saturación de estos servidores, o a la cantidad de información extra que incluyan los protocolos de mensajerías para el traslado de los paquetes.

El resultado de estas pruebas se basan en medir la performance del túnel construido. Para ello se utilizarán herramientas disponibles que realicen mediciones sobre la comunicación. Estas herramientas deberán dar como resultado parámetros conocidos y utilizados generalmente para este tipo de mediciones.

Lo que se espera es obtener una comunicación estable; es decir que pueda permanecer establecida sin cortes durante el tiempo que se requiera (más allá de su dependencia con la estabilidad del servicio de mensajería instantánea) y que sus tiempos de respuesta se mantengan relativamente constantes en el tiempo.

5 – PRUEBAS

La motivación de estas pruebas son el validar y obtener resultados sobre el comportamiento de la mensajería instantánea como trasporte de datos, mediante el prototipo implementado. Este capitulo entonces, refiere al proceso, la metodología y los resultados obtenidos en las pruebas.

En primer lugar se enumeran y definen algunos de los términos generalmente utilizados en la medición de calidad de un servicio de red. Luego se realiza un análisis sobre las herramientas disponibles para estas mediciones. En este caso se busca apoyarse en especificaciones o estudios previos, que definan conjuntos de pruebas y resulten completos para el objetivo de este capitulo.

Por último, se detalla la arquitectura utilizada, las distintas pruebas ejecutadas y los resultados obtenidos.

5.1 Términos para Mediciones de Rendimiento de red

En esta primera sección se busca conocer y determinar cuales son los parámetros mas utilizados al momento de medir la calidad en un servicio de red.
Estas definiciones servirán al momento de evaluar y concluir sobre los resultados, y también serán de utilidad para comprender y elegir las herramientas a utilizar en las pruebas. [39]

Ancho de banda (bandwidth): La capacidad de un enlace de transportar información por unidad de tiempo.

Rendimiento (throughput): Representa la capacidad de un enlace de transportar información útil que puede transmitirse por unidad de tiempo.

Pérdida de Paquetes (packet loss): Es la tasa de pérdida de paquetes. Representa el porcentaje de paquetes transmitidos que se descartan en la red.

Retraso (Delay): El delay o latencia se define como el intervalo de tiempo en el envío de datos entre una computadora y otra, e incluye el tiempo que le toma al software construir el mensaje, así como el tiempo de transferencia de bits entre una computadora y otra.

Jitter: Refiere a la variación en el retraso (delay) de los paquetes. Se trata de la variación de latencia producida por la congestión o distinto tiempo de tráfico de paquetes.

RTT (round trip time): El RTT es el tiempo que le lleva a un paquete llegar a destino y volver.

Todos estos parámetros son importantes para caracterizar el rendimiento de una red. Comúnmente se utilizan el ancho de banda y el delay. Según el análisis de estos parámetros, si una aplicación manda una gran cantidad de mensajes pequeños, su rendimiento será impactado por el delay y si la aplicación manda mensajes grandes, su rendimiento será impactado por el ancho de banda. En general, las aplicaciones se desempeñan mejor cuando el delay es bajo y el ancho de banda es grande.

Estas definiciones serán utilizadas en las próximas secciones, y algunas de ellas serán utilizadas en los resultados presentados.

5.2 Análisis de Herramientas

Este análisis comenzó con un estudio de especificaciones existentes sobre herramientas disponibles para las pruebas. El objetivo fue basar las pruebas en alguna especificación estándar que respalde los resultados obtenidos.

Según el resultado de este análisis, la especificación elegida para utilizar de base en las pruebas es:

RFC 2398 - Some Testing Tools for TCP Implementors.
Network Working Group
S. Parker, C. Schmechel - Sun Microsystems, Inc.
August 1998.
http://www.rfc- archive.org/getrfc.php?rfc=2398

Este memo enumera herramientas disponibles para la evaluación de algunos de los términos antes definidos, además de contener también herramientas para la visualización de los resultados de los mismos.
En este estudio se las detalla una a una, a efectos de tener un panorama general de su disponibilidad, su actualización (ya que el memo data del año 98), y su distribución para las distintas plataformas.
El objetivo es elegir entre ellas las que mas se adecuen a nuestros objetivos y luego utilizarlas para las pruebas.

Las herramientas y el resultado de la investigación se describe a continuación.

DBS – Distributed Benchmark System [25]

DBS es una herramienta que permite coordinar múltiples transferencias de datos para obtener el resultado del comportamiento TCP. Permite medir la performance de TCP en distintos ambientes.
La ultima distribución disponible de esta herramienta data de Junio del 98 solamente para sistemas Linux.

DummyNet – A simple approach to the evaluation of network protocols [26]

Dummynet es una herramienta flexible originalmente diseñada para testear protocolos de red. La herramienta trabaja limitando ancho de banda, y simulando demoras, pérdida de paquetes y efectos multidireccionales. La ultima versión de DummyNet es del 2000 con distribución solo para sistemas BSD.

NetPerf [27]

Netperf es un patrón de pruebas que puede ser usado para medir varios aspectos de la performance de una red. Se focaliza en la transferencia de datos a granel para la medida de performance y en las mediciones de tiempos para el request/response. Los ambientes analizados a través del NetPerf incluyen, TCP y UDP, via BSD Sockets, para ambos IPv4 e IPv6. Incluye también mediciones de utilización de CPU.
Esta herramienta está en desarrollo, su ultima versión fue lanzada en el 2005 y contiene distribuciones tanto para Linux como para Windows.

NIST Net - A Linux- based Network Emulation Tool [28]

NIST Net es un paquete de emulación de red. Permite a una PC instalarse como un router para emular una amplia variedad de condiciones de red. NIST Net es implementada como una extensión del modulo kernel del sistema operativo Linux y su interfaz de aplicación es basada en el sistema XWindow. Esta interfaz permite al usuario seleccionar y monitorear tráfico específico que pasa a través del simulado router y aplicar "efectos" a los paquetes IP.
Su ultima distribución es del 2005.

Orchestra - A Probing and Fault Injection Environment for Testing Protocol Implementations [29]

Esta herramienta es una librería que provee al usuario la habilidad de construir una capa sobre un protocolo para construir pruebas que provoquen fallas. Construido para la evaluación y validación de la tolerancia a fallas y características de sincronización de protocolos.
No se encontró distribución disponible.

Packet Shell [30]

Es un conjunto de extensiones TCL (Tool Command Language) que permiten el desarrollo de tests de nivel de paquetes, utilizando el lenguaje de scripts TCL. Esta herramienta crea comandos TCL que

permiten crear, modificar, enviar y recibir paquetes sobre la red. Las operaciones para cada protocolo están limitados por una librería linkeable y dinámica llamada librería del protocolo. Las librerías actuales son: IP, IPv6, extensiones de IPv6, ICMP, ICMPv6, entre otras.
Tiene distribución solamente para Solaris.

Tcpanaly [31]

Es una herramienta para analizar automáticamente el comportamiento de las implementaciones TCP, inspeccionando su actividad. Lo hace a través de los archivos en formato pcap que pueden ser obtenidos mediante tcpdump [37], ethereal [24], etc.
No se encontró ninguna distribución disponible de esta herramienta.

TCPTrace [32]

Esta es una herramienta de análisis de archivos TCP. Lee de un archivo de salida en formato pcap. Para cada conexión, mantiene el tiempo transcurrido, bytes/segmentos enviados y recibidos, retransmisiones, round trip times, ventanas de advertencias, throughput, entre otros.
Su ultima versión es del 2004 y tiene distribuciones tanto para Linux como para Windows.

Tracelook [33]

Es un programa TCL/Tk para ver gráficamente el contenido de los archivos en formato pcap. Cuando se gráfica una conexión, el usuario puede seleccionar las variables a ser graficadas.

TReno (Traceroute RENO) [34]

TReno es una herramienta de testeo diseñada para testear performance, throughput. Utiliza la técnica de enviar paquetes UDP con TTL baja, los hosts y los routers a lo largo de la trayectoria al destino final enviarán detrás los mensajes excedidos de TTL, que tienen características similares a los paquetes del TCP ACK, y eso es utilizado para las mediciones.
Su ultima versión fue lanzada en el 2002 y tiene distribuciones para Linux pero no para Windows.

TTcp (Test TCP) [35]

Herramienta de medición para performance en los protocolos TCP y UDP. Ejecutada desde linea de comando y basada en sockets, obtiene como resultado bytes y llamadas por segundo entre dos sistemas.
Tiene distribuciones tanto para Linux como para Windows y su ultima versión fue lanzada en el 2004.

Xplot [36]

Es una herramienta para graficar los resultados de análisis de trafico TCP mediante comando propios escritos en un archivo de texto.

Análisis Comparativo de Herramientas

El análisis comparativo de las herramientas incluidas en el mencionado RFC se realizó con el objetivo de elegir las que más se adecuen y que utilizaremos para las pruebas. Dentro de este análisis comparativo no se incluyeron aquellas que sólo implementan la visualización de los resultados.

Como el prototipo implementado es tanto utilizable en Windows como en Linux, daremos preferencia a las herramientas de test que tengan distribuciones para los dos sistemas operativos, y así facilitarnos la posibilidad de realizar las mismas pruebas en los dos sistemas.

	Ultima versión	Tipo test			Plataforma		
		Performance	Correctitud	Stress	Windows	Linux	Otros
DBS	Jun 98	X		X		X	
DummyNet	Jun 00	X	X				X
NetPerf	Oct 05	X			X	X	
NIST Net	Jul 05	X	X			X	
Orchestra	N/A	X	X				X
Packet Shell	Feb 96	X	X				X
Tcpanaly	N/A	X	X				
TCPTrace	Jun 04	X	X		X	X	X
Treno	Oct 02	X				X	X
TTcp	May 05	X			X	X	

Tabla 4.- Análisis Comparativo de Herramientas de Pruebas

Luego de este análisis concluimos elegir las herramientas: NetPerf, TCPTrace y TTcp , para las mediciones.

Uso de las Herramientas

NetPerf

Esta herramienta esta basada en una arquitectura cliente/servidor, en donde el servidor corre la aplicación `netserver` y el cliente la `netperf` con los parámetros definidos para el test.

El uso del NetPerf se basa en la construcción de tests a partir de los parámetros definidos por la aplicación. Por defecto la herramienta brinda algunos tests que resultan adecuados para las pruebas que queremos realizar, estos test se separan en dos grupos; los que miden la trasferencia bruta de datos, y los que miden tiempos en el request/response.

TCPTrace

El TCPTrace obtiene los resultados a partir de un archivo pcap (archivo que contiene trafico de red capturado). Este archivo debe ser obtenido mediante otra herramienta, como por ej.: ethereal[24] o tcpdump [37]
Esta aplicación realiza el análisis de estos archivos, y obtiene resultados de througput, rtt, packet loss, entre otros, permitiendo también graficar los resultados mediante XPlot[36].

TTcp

El TTcp tiene también arquitectura cliente/servidor, a los que le llama “trasmitter” (ttcp -t) y “receiver” (ttcp -r). Esta aplicación envía desde el “transmitter” al “receiver” una cantidad definida de bytes y obtiene los resultados. Los resultados son los bytes por segundos, y la cantidad de llamadas por segundo.

5.3 Arquitectura General de Pruebas y Herramientas de Test

5.3.1 Arquitectura General de Pruebas

La arquitectura de las pruebas se diseñó teniendo en cuenta los objetivos del presente proyecto; el obtener medidas o indicadores de la performance de la mensajería instantánea como transporte de datos.
Se decidió utilizar las PCs con la aplicación que establece el túnel y realiza la comunicación con la mensajería instantánea (IPoIM) dedicadas a esta tarea, e incluir otras PCs para correr los tests.

Se busco una arquitectura simple que cumpliera con los requisitos, determinándose la siguiente:

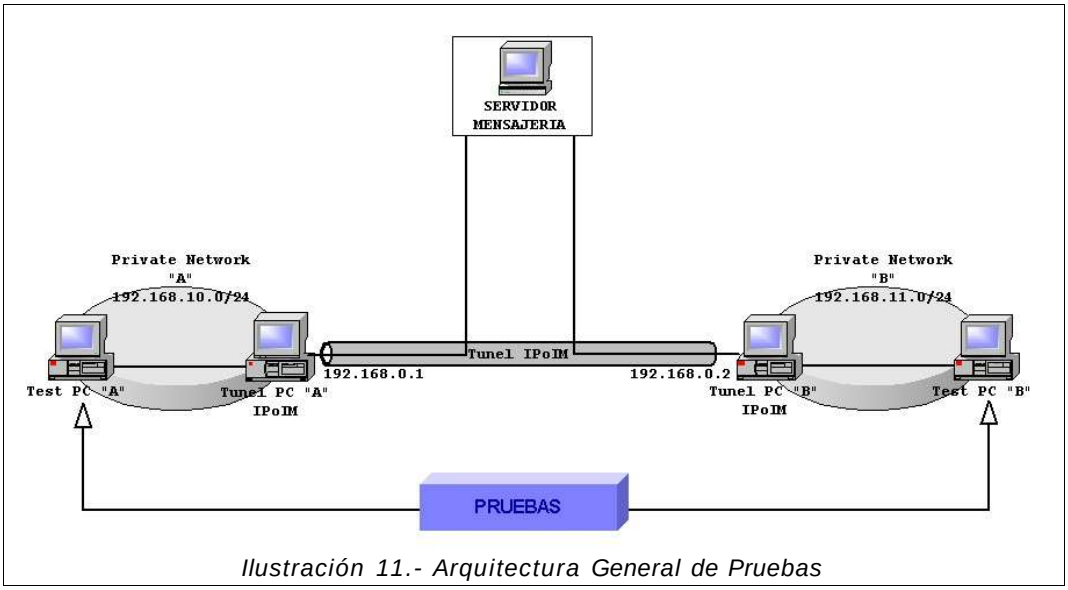


Ilustración 11.- Arquitectura General de Pruebas

Tenemos dos redes privadas, cada una con dos máquinas, una “Test PC X” y una “Túnel PC X”. Genéricamente en cada red privada “X” `192.168.x.0/24` se definieron las IP, `192.168.x.2` para “Test PC X” y `192.168.x.1` para “Túnel PC X”.

Los “Test PC X” serán utilizadas para correr los test seleccionados en la sección anterior.
Los “Túnel PC X” serán utilizados para comunicar las redes privadas a través de IPoIM.

Por lo tanto, el flujo de datos de los test tendrán como origen y/o destino:

Red Privada "A" ***192.168.10.0/24***	***Red Privada "B"*** ***192.168.11.0/24***
Test PC "A"	***Test PC "B"***
192.168.10.2	192.168.11.2

Se estableció el túnel "IP over IM" sobre el servidor de mensajería, entre las máquinas "Túnel PC X" de cada red, con las direcciones IP virtuales 192.168.0.1 y 192.168.0.2.

Por lo tanto el flujo de datos de una red privada a la otra tendrán como origen y/o destino:

Red Privada "A" ***192.168.10.0/24***	***Red Privada "B"*** ***192.168.11.0/24***
Túnel PC "A"	***Túnel PC "B"***
192.168.0.1	192.168.0.2

Y por último, los caminos completos que recorren los paquetes en los test serán de la forma:

Red Privada "A" ***192.168.10.0/24***			***Red Privada "B"*** ***192.168.11.0/24***		
Test PC "A"	***Túnel PC "A"***		***Túnel PC "B"***		***Test PC "B"***
192.168.10.2	192.168.10.1	192.168.0.1	192.168.0.2	192.168.11.1	192.168.11.2

5.3.2 Herramientas de Test

Pruebas con la herramienta NetPerf

Como vimos anteriormente los tests del NetPerf se pueden separar en dos grupos; los que miden transferencia de datos, y los que miden tiempos para request/response.

Los test elegidos de transferencia de datos son:

TCP_STREAM: Este test obtiene datos de performance (throughput) transfiriendo cierta cantidad de datos desde el sistema corriendo el netperf al sistema corriendo el netserver (el tiempo que lleva el establecer la conexión no es incluido en el cálculo del throughput).

TCP_MAERTS (STREAM al revés): Este test es similar al anterior con la diferencia que los datos fluyen desde el netserver al netperf.

TCP_SENDFILE: Realiza lo mismo que el TCP_STREAM, con excepción de que obtiene los datos a enviar de un archivo definido por el usuario de la aplicación.

Los test elegidos de request/response son:

TCP_RR: El test TCP_RR (TCP Request/Response) puede ser pensado como un `ping`, sincrónico, una transacción por vez. En este test los tiempos de establecer la conexión TCP, no se incluyen en los resultados.

TCP_CC: Este test simplemente mide que tan rápido el par de sistemas pueden abrir y cerrar conexiones entre ellos.

TCP_CRR: Es una mezcla de los tests anteriores, el cual mide la performance de iniciar la conexión, intercambiar un simple request/response y cerrar la conexión.

Cada uno de los test se realizaron 3 veces durante el tiempo recomendado, un mínimo de 60 segundos cada uno.
Los resultados obtenidos son el throughput y el tiempo de respuesta en las operaciones de open/request/response/close.

Pruebas con la herramienta TCPTrace

Para las pruebas con la herramienta TCPTrace se realizaron capturas de diferentes operaciones sobre la conexión, para luego obtener gráficos y resultados sobre el throughput y sobre el RTT.
En el detalle de cada test se especifica qué se captura exactamente para cada una de las pruebas.

Pruebas con la herramienta TTcp

Las pruebas con la herramienta TTcp se realizaron en tres iteraciones de la misma, promediando el resultado, y modificando en cada test los parámetros:
- l : largo en bytes de los bufers leídos o escritos en la comunicación.
- n : numero de bufers escritos en la comunicación.

Los resultados obtenidos de este test nos dan la velocidad en Kilobits por segundo para cada transmisión de datos.

5.4 Pruebas y Resultados

Las pruebas se realizaron en ambientes diferentes, estos ambientes se lograron modificando el servidor de mensajería y el área de red por donde se transportan los datos.

5.4.1 Prueba 1 - MSN / Internet

Esta prueba se realiza utilizando MSN Messenger como servidor de mensajería, por lo tanto el túnel se establece sobre Internet y los datos se transportan mediante esta vía. La conexión a Internet es un ADSL de 512/128 Kbps, compartido por las dos redes privadas.

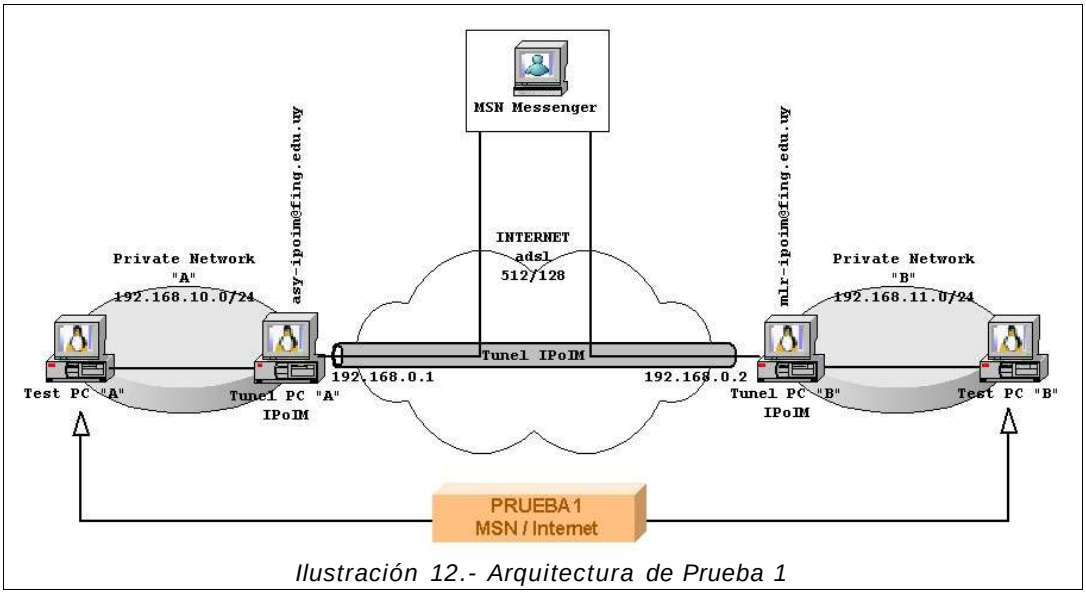


Ilustración 12.- Arquitectura de Prueba 1

Prueba de Comunicación

Se realiza primero una prueba de comunicación entre las máquinas TestPC "A" y TestPC "B" mediante el comando `ping`.

```
# ping 192.168.10.2 -c 60
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=62 time=597 ms
....
<removido para simplificar la visualización>
....
64 bytes from 192.168.10.2: icmp_seq=60 ttl=62 time=605 ms

--- 192.168.10.2 ping statistics ---
60 packets transmitted, 60 received, 0% packet loss, time 59192ms
rtt min/avg/max/mdev = 583.884/593.983/614.978/6.398 ms
```

NetPerf - Resultados

Los resultados se muestran como un promedio de los resultados en cada iteración.

	# test	Test	Iteración 1	Iteración 2	Iteración 3	Promedio	
Transferencia de datos	1	TCP_STREAM	26.19	22.98	24.19	**24.45**	**Kbits/s**
	2	TCP_SENDFILE	21.51	22.37	21.63	**21.72**	**Kbits/s**
Tiempos de Request/Response	3	TCP_RR	1.48	1.57	1.57	**1.54**	**trans/s**
	4	TCP_CC	0.65	0.67	0.67	**0.66**	**trans/s**
	5	TCP_RRC	0.42	0.42	0.42	**0.42**	**trans/s**

Tabla 5.- Resumen de Test con NetPerf

Detalles

En estos detalles se especifican las lineas de comando ejecutadas y los resultados obtenidos para cada test.

Test 1 – NetPerf – STREAM

```
# netperf -t TCP_STREAM -H 192.168.10.2 -f k -l 60
```

Resultados:

```
Throughput iteración 1 = 26.19 10³ bits/seg
Throughput iteración 2 = 22.98 10³ bits/seg
Throughput iteración 3 = 24.19 10³ bits/seg
```

Throughput Promedio = 24.45 10³ bits/seg

Test 2 – NetPerf – SENDFILE

El archivo utilizado: netperf-2.4.1.tar.gz, tamaño = 1,4 MB

```
# netperf -t TCP_SENDFILE -F /tmp/netperf-2.4.1.tar.gz -H
192.168.10.2 -f k -l 60
```

Resultado:

```
Throughput iteración 1 = 21.51 10³ bits/seg
Throughput iteración 2 = 22.37 10³ bits/seg
Throughput iteración 3 = 21.63 10³ bits/seg
```

Throughput Promedio = 21.72 10³ bits/seg

Test 3 – NetPerf – Request/Response

```
# netperf -t TCP_RR -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de RR/segundo iteración 1 = 1.48 t/s
Transacción de RR/segundo iteración 2 = 1.57 t/s
Transacción de RR/segundo iteración 3 = 1.57 t/s
```

Transacción de RR/segundo promedio = 1.54 t/s

Test 4 – NetPerf – Open/Close

```
# netperf -t TCP_CC -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de CC/segundo iteración 1 = 0.65 t/s
Transacción de CC/segundo iteración 2 = 0.67 t/s
Transacción de CC/segundo iteración 3 = 0.67 t/s
```

Transacción de CC/segundo promedio = 0.66 t/s

Test 5 – NetPerf – Request/Response Open/Close

```
# netperf -t TCP_RRC -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de RRC/segundo iteración 1 = 0.42 t/s
Transacción de RRC/segundo iteración 2 = 0.42 t/s
Transacción de RRC/segundo iteración 3 = 0.42 t/s
```

Transacción de RRC/segundo promedio = 0.42 t/s

TCPTrace - Resultados

Para estas pruebas se captura el tráfico durante una sesión de `ssh` en donde una de las máquinas realiza el logueo al servicio `ssh`, luego se utiliza el comando `pscp` para transferir un archivo de tamaño: 237.797 bytes; y luego se realizan dos `find` para listar la estructura de archivos de la maquina remota.

Los resultados de throughput resultantes de esas operaciones, se muestran en la siguiente figura:

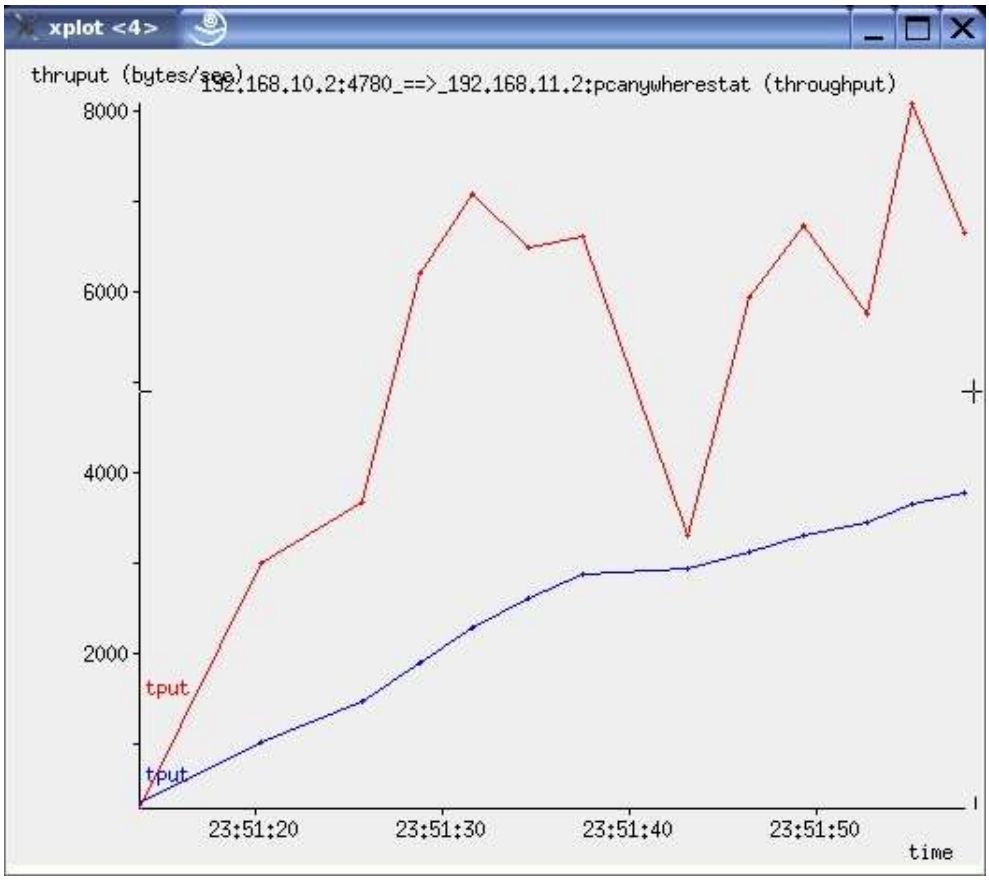


Ilustración 13.- Gráfica throughput obtenido mediante Tcptrace.

En donde la linea inferior del gráfico representa el throughput promedio en cada punto del tiempo de la conexión y la linea superior el throughput generado por las 20 últimas muestras.
Los valores de throughput mostrados en la gráfica están representados en bytes por segundo, pero si lo expresamos en 10^3 bits/segundo como para el NetPerf, y consideramos el punto medio del throughput promedio tenemos:

`3000 bytes/seg = 24000 bits/seg = 24 10³ bits/seg.`

Este resultado es consistente con lo medido con la herramienta anterior, NetPerf.

Otro resultado interesante que podemos obtener con esta herramienta, son las muestras de rtt (round trip time) en milisegundos, sobre el tiempo de conexión.

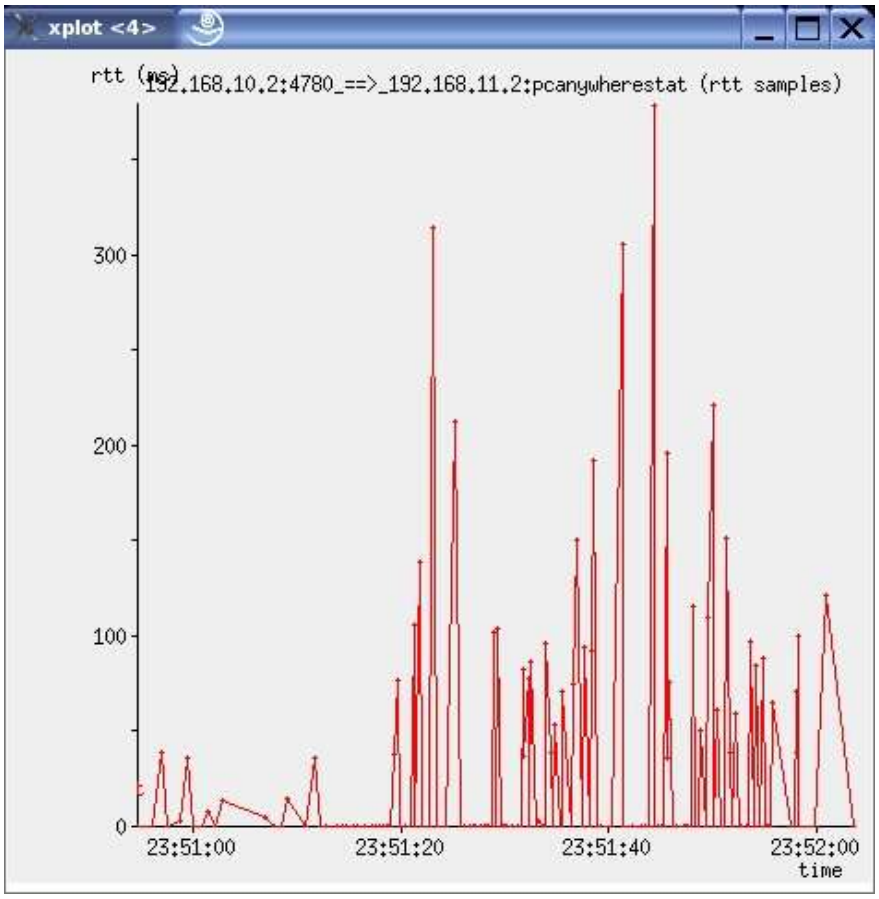


Ilustración 14.- Muestras RTT

También esta herramienta permite sacar otros datos interesantes del tráfico capturado,

TCP connection:
host c: 192.168.10.2:4780
host d: 192.168.11.2:pcanywherestat
filename: filetransfer.ssh

	c->d:	d->c:	
total packets:	269	180	total de paquetes vistos
ack pkts sent:	268	180	total de paquetes ack vistos
unique bytes sent:	241806	2173	bytes únicos enviados (sin retransm)
actual data pkts:	253	23	cantidad de paquetes con datos
actual data bytes:	241806	2317	bytes vistos
rexmt data pkts:	0	3	retransmisiones
rexmt data bytes:	0	144	retransmisiones en bytes
outoforder pkts:	0	0	paquetes vistos llegar en desorden
ttl stream length:	241806 bytes	2173 bytes	Theorical Stream Lenght
missed data:	0 bytes	0 bytes	diferencia entre unique y ttl
data xmit time:	64.545 secs	64.575 secs	tiempo total de transmisiones

Tabla 6.- Resultados TCPTrace

TTcp - Resultados

Para las pruebas con la herramienta TTcp utilizamos distintos tamaños y cantidad de buffers. Estas pruebas fueron realizadas en un día y hora diferentes de las anteriores, con el objetivo de verificar si se mantenía la performance y los resultados previos.

Un informe de los resultados obtenidos con esta herramienta, es el que se muestra continuación:

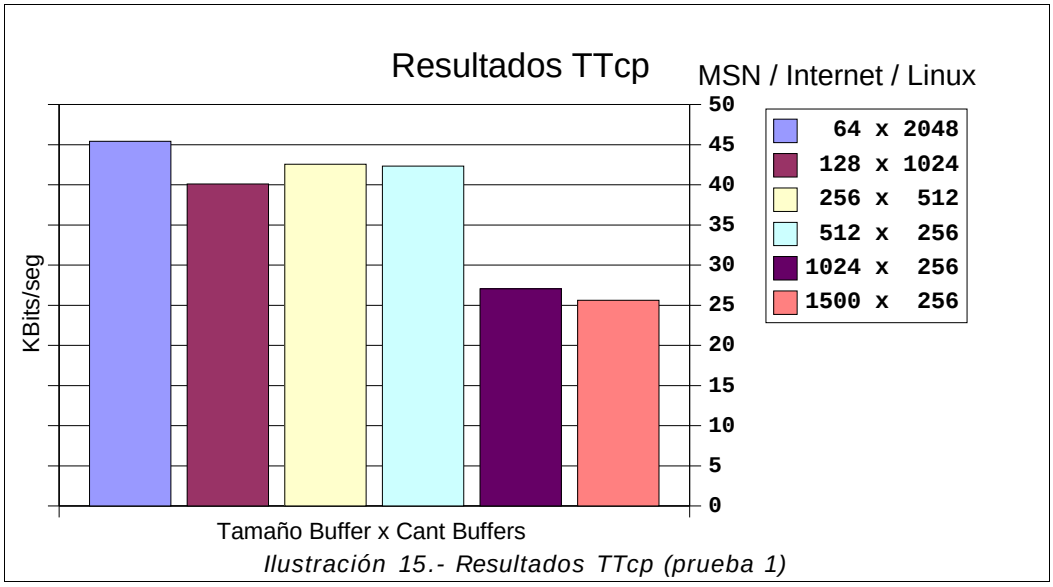


Ilustración 15.- Resultados TTcp (prueba 1)

Detalles

	Kbytes/seg				Kbits/seg
Tam.Buf x Cant Buff = Total Bytes	iteración 1	iteración 2	iteración 3	Promedio	
64 x 2048 = 131072 bytes = 1.02 Mbits	6.55	6.08	4.41	5.68	45.44
128 x 1024 = 131072 bytes = 1.02 Mbits	3.79	6.19	5.06	5.01	40.11
256 x 512 = 131072 bytes = 1.02 Mbits	5.96	6.27	3.73	5.32	42.56
512 x 256 = 131072 bytes = 1.02 Mbits	4.85	6.47	4.56	5.29	42.35
1024 x 256 = 262144 bytes = 2.05 Mbits	3.49	2.96	3.7	3.38	27.07
1500 x 256 = 384000 bytes = 3.00 Mbits	2.88	3.54	3.19	3.2	25.63

En este último test para el caso de buffers de 1500 bytes, se obtuvieron resultados consistentes con los vistos en las herramientas anteriores, 25.63 Kbits.
El resto de los resultados no tienen el comportamiento esperado. Al contrario del comportamiento ethernet, cuanto mas pequeños los buffers que se transmiten, mejor es el resultado en Kbits/seg.
De todas maneras, se debería hacer un análisis más profundo para poder concluir sobre este comportamiento.

Completando estas pruebas, se capturaron los paquetes transmitidos durante el último test de TTcp (1500 x 256), para analizar con la herramienta TCPTrace y así obtener su throughput .

El resultado de este análisis se resume en el siguiente gráfico:

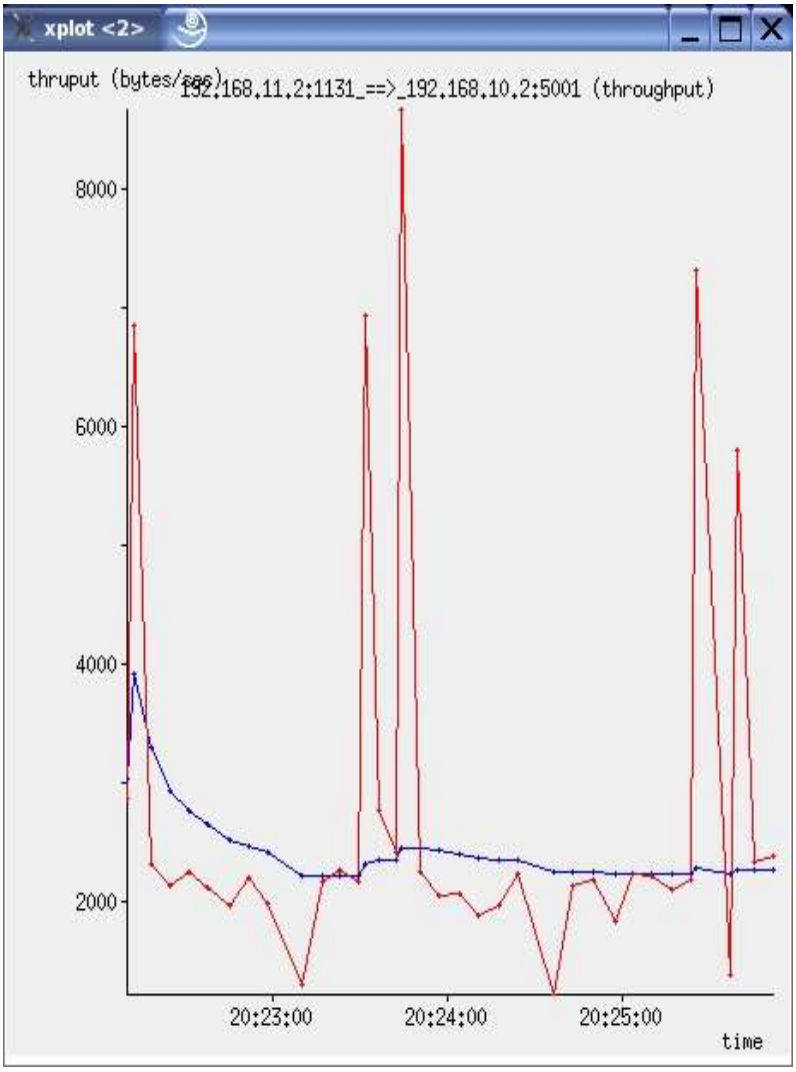


Ilustración 16.- Gráfica throughput obtenida mediante el TcpTrace de la captura de un test Ttcp

Utilizando la misma metodología que en el gráfico anterior sobre el throughput (el punto medio del throughput promedio, en este caso la linea que representa el promedio es la más estable), se obtiene en este caso un valor promedio de 2500 bytes/seg, lo que nos da un resultado de: 20 Kbits/seg.
Este resultado es también consistente con las pruebas anteriores.

Y los demás resultados que nos proporciona el TCPTrace:

TCP connection:
host c: 192.168.10.2:1131
host d: 192.168.11.2:5001
filename: capturaTTCP_1500_256

	c->d:	d->c:	
total packets:	691	690	total de paquetes vistos
ack pkts sent:	690	690	total de paquetes ack vistos
unique bytes sent:	384000	0	bytes únicos enviados (sin retransm)
actual data pkts:	687	0	cantidad de paquetes con datos
actual data bytes:	520500	0	bytes vistos
rexmt data pkts:	181	1	retransmisiones
rexmt data bytes:	136500	1	retransmisiones en bytes
outoforder pkts:	0	0	paquetes vistos llegar en desorden
ttl stream length:	384000	0	Theorical Stream Lenght
missed data:	0 bytes	0 bytes	diferencia entre unique y ttl
data xmit time:	227.439 secs	0.000 secs	tiempo total de transmisiones

Tabla 7.- Otros resultados TCPTrace

Hasta el momento estas herramientas nos dan un panorama general de como se comporta y que rendimiento tiene la conectividad implementada con el prototipo en este ambiente.
Con respecto al throughput, se resume que el promedio es de alrededor de 22 Kb/seg.

Con respecto a los resultados obtenidos de RTT, se comparan con los obtenidos con la herramienta `ping` a efectos de conocer su comportamiento.
Los resultados del `ping` fueron:

```
rtt min/avg/max/mdev = 583.884/593/614.978/6.398ms
```

Si se observa el gráfico: Ilustración 14.- Muestras RTT, se detecta que existe una diferencia entre los valores obtenidos con una herramienta y con la otra.
En una primera instancia, este comportamiento podría haber sido causado por una saturación en la comunicación, ya que la operación solicitada en el caso de la ilustración, era de “listado de estructura de archivos”.

Por último, los resultados del TTcp nos muestran la variación en Kbits/seg según la cantidad de bytes en el buffer para el test. Los resultados de la captura y análisis comprueban que los valores se mantienen para distintos días y horas, para el caso de 1500 bytes de buffer.

Algunas Variaciones

A la arquitectura antes propuesta, se le realizan algunas variaciones sobre los sistemas operativos de las máquinas. Estas pruebas corroboran la interoperabilidad del prototipo y también dan nuevos resultados con las herramientas de test.

La nueva arquitectura queda entonces:

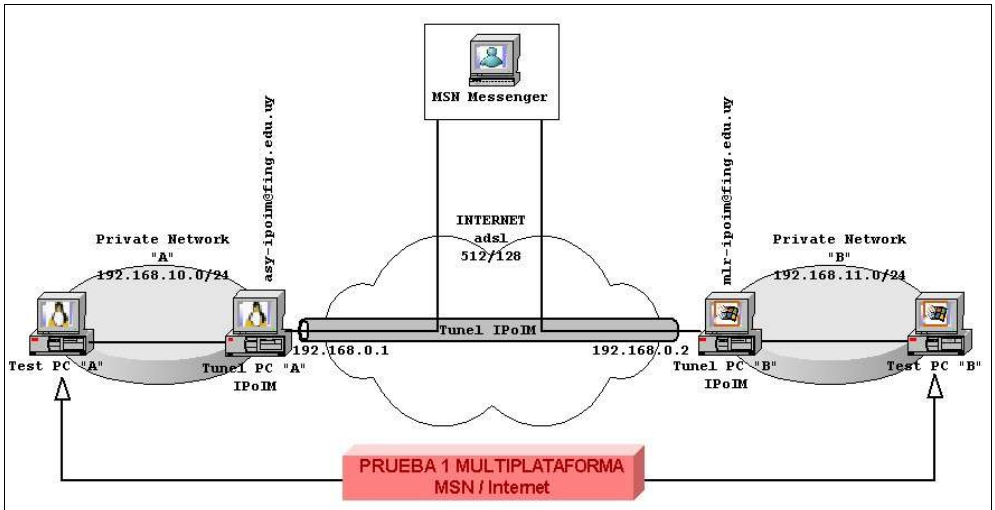


Ilustración 17.- MSN / Internet / Linux y Windows

Las PCs que están en la red privada “A” 192.168.10.0/24 utilizan sistema operativo Linux y las que están en la “B” 192.168.11.0/24, el sistema operativo Windows. El túnel, por lo tanto, se establece entre una maquina en Linux y otra en Windows, así como los tests.

TTcp - Resultados

Para las pruebas con la herramienta TTcp utilizamos distintos tamaños y cantidad de buffers (los mismos utilizados en las pruebas anteriores).

Los resultados obtenidos se presentan mediante este gráfico:

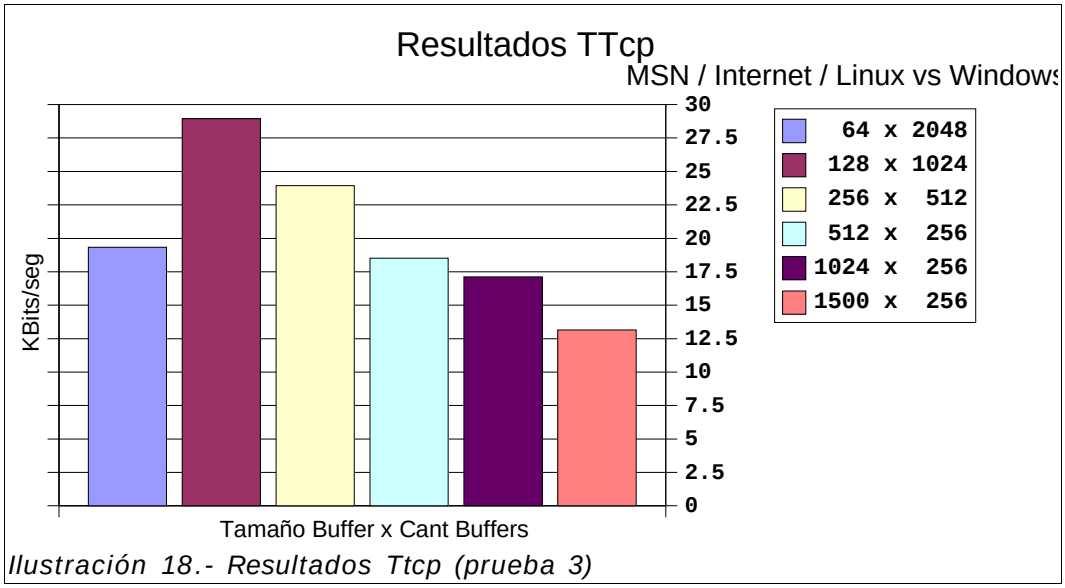


Ilustración 18.- Resultados Ttcp (prueba 3)

Detalles

	Kbytes/seg				Kbits/seg
Tam.Buf x Cant Buff = Total Bytes	iteración 1	iteración 2	iteración 3	Promedio	
64 x 2048 = 131072 bytes = 1.02 Mbits	2.78	2.43	2.04	2.42	19.33
128 x 1024 = 131072 bytes = 1.02 Mbits	3.52	3.92	3.41	3.62	28.93
256 x 512 = 131072 bytes = 1.02 Mbits	3.88	2.71	2.39	2.99	23.95
512 x 256 = 131072 bytes = 1.02 Mbits	2.52	2.17	2.25	2.31	18.51
1024 x 256 = 262144 bytes = 2.05 Mbits	2.19	2.16	2.07	2.14	17.12
1500 x 256 = 384000 bytes = 3.00 Mbits	1.67	1.61	1.65	1.64	13.15

Tabla 8.- Detalles TTcp

Comparativo de las pruebas para distintas plataformas.

Las pruebas con máquinas en Windows fueron hechas el mismo día y a casi la misma hora que las pruebas con las máquinas en Linux para los test.

	Linux vs Linux	***Linux vs Windows***
TTcp (64 x 2048) Kbits/seg	45.44	19.33
TTcp (128 x 1024) Kbits/seg	40.11	28.93
TTcp (256 x 512) Kbits/seg	42.56	23.95
TTcp (512 x 256) Kbits/seg	42.35	18.51
TTcp (1024 x 256) Kbits/seg	27.07	17.12
TTcp (1500 x 256) Kbits/seg	25.63	13.15

	Linux vs Linux	***Linux vs Windows***
Ping RTT min/avg/max	583 / 593 / 614	560 / 570 / 647

A priori, se podría concluir que el sistema operativo Linux tuvo una mejor performance que Windows. Igualmente estas conclusiones serían apresuradas, ya que se desconoce el motivo de esta diferencia y se deberían realizar pruebas mas exhaustivas para corroborar este comportamiento.

5.4.2 Prueba 2 - Jabber / LAN

Esta prueba se realiza utilizando Jabber como protocolo de mensajería, y utilizando un servidor de mensajería para Jabber implementado para redes locales, Jive Messenger [38].
Tanto las Pcs de los test como las del túnel utilizan el sistema operativo Linux durante esta prueba.
Para esta arquitectura se hicieron pruebas con la herramienta Netperf.

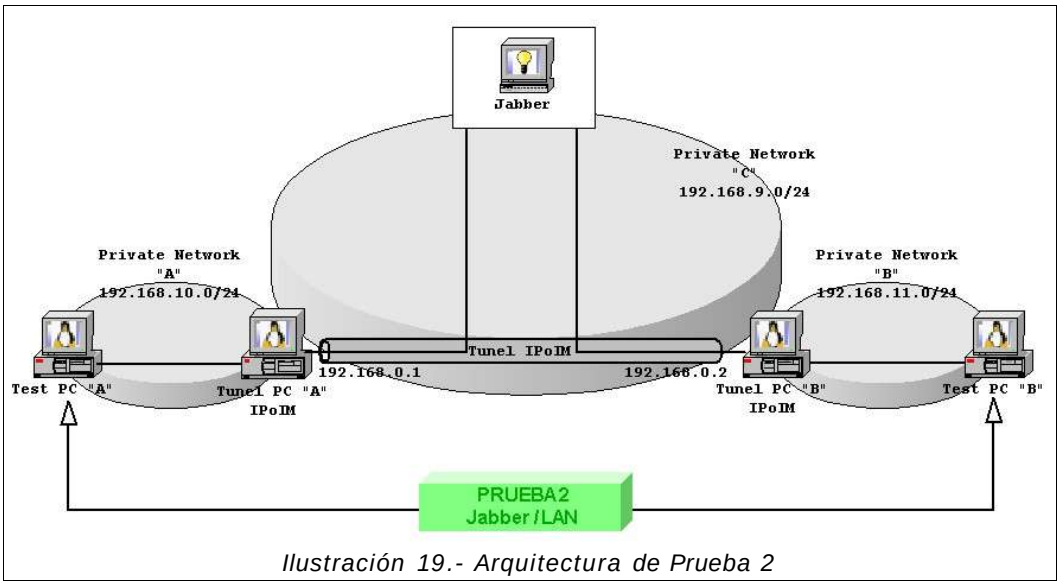


Ilustración 19.- Arquitectura de Prueba 2

NetPerf - Resultados

Los resultados se muestran como un promedio de los resultados en cada iteración.

	# test	Test	Iteración 1	Iteración 2	Iteración 3	Promedio	
Transferencia de datos	1	TCP_STREAM	142.05	131.95	140.58	**138.19**	**Kbits/s**
	2	TCP_MAERTS	264.44	266.98	267.54	**266.32**	**Kbits/s**
	3	TCP_SENDFILE	137.53	137.13	137.45	**137.37**	**Kbits/s**
Tiempos de Request/Response	4	TCP_RR	37.06	38.38	37.43	**37.62**	**trans/s**
	5	TCP_CC	13.05	13.02	12.32	**12.8**	**trans/s**
	6	TCP_RRC	4.68	4.7	4.65	**4.68**	**trans/s**

Tabla 9.- Resultados NetPerf (prueba 2)

Detalles

En estos detalles se especifican las líneas de comando ejecutadas y los resultados obtenidos para cada test.

Test 1 – NetPerf – STREAM

```
# netperf -t TCP_STREAM -H 192.168.10.2 -f k -l 60
```

Resultados:

```
Throughput iteración 1 = 142.05 10³ bits/seg
Throughput iteración 2 = 131.95 10³ bits/seg
Throughput iteración 3 = 140.58 10³ bits/seg
```

Throughput Promedio = 138.19 10³ bits/seg

Test 2 – NetPerf – MAERTS

```
# netperf -t TCP_MAERTS -H 192.168.10.2 -f k -l 60
```

Resultados:

```
Throughput iteración 1 = 264.44 10³ bits/seg
Throughput iteración 2 = 266.98 10³ bits/seg
Throughput iteración 3 = 267.54 10³ bits/seg
```

Throughput Promedio = 266.32 10³ bits/seg

Test 3 – NetPerf – SENDFILE

El archivo utilizado: netperf-2.4.1.tar.gz, tamaño = 1,4 MB

```
# netperf -t TCP_SENDFILE -F /tmp/netperf-2.4.1.tar.gz -H
192.168.10.2 -f k -l 60
```

Resultado:

```
Throughput iteración 1 = 137.53 10³ bits/seg
Throughput iteración 2 = 137.13 10³ bits/seg
Throughput iteración 3 = 137.45 10³ bits/seg
```

Throughput Promedio = 137.37 10^3 bits/seg

Test 4 – NetPerf – Request/Response

```
# netperf -t TCP_RR -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de RR/segundo iteración 1 = 37.06 t/s
Transacción de RR/segundo iteración 2 = 38.38 t/s
Transacción de RR/segundo iteración 3 = 37.43 t/s
```

Transacción de RR/segundo promedio = 37.62 t/s

Test 5 – NetPerf – Open/Close

```
# netperf -t TCP_CC -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de CC/segundo iteración 1 = 13.05 t/s
Transacción de CC/segundo iteración 2 = 13.02 t/s
Transacción de CC/segundo iteración 3 = 12.32 t/s
```

Transacción de CC/segundo promedio = 12.80 t/s

Test 6 – NetPerf – Request/Response Open/Close

```
# netperf -t TCP_RRC -H 192.168.10.2 -f k -l 60
```

Resultado:
```
Transacción de RRC/segundo iteración 1 = 4.68 t/s
Transacción de RRC/segundo iteración 2 = 4.70 t/s
Transacción de RRC/segundo iteración 3 = 4.65 t/s
```

Transacción de RRC/segundo promedio = 4.68 t/s

Comparativo de las pruebas en distintas áreas de red

Como resumen se muestra la diferencia entre la transmisión de datos sobre una red local y sobre Internet, los resultados se muestran como un promedio de los resultados de los tests.

	Red Local	***Internet***
Throughput #1 (Kbits/seg)	138.19	24.45
Throughput #2 (Kbits/seg)	137.37	21.72

	Red Local	***Internet***
Transacciones #1 (#trans/seg)	37.62	1.54
Transacciones #2 (#trans/seg)	12.8	0.66
Transacciones #3 (#trans/seg)	4.65	0.42

Estos resultados parecen lógicos, sabiendo la gran diferencia en velocidad de transmisión entre una red de área local y una red de área ancha.

5.4.3 Otros Resultados Experimentales

También se realizaron otras pruebas para la validación y seguimiento de los paquetes durante la comunicación. Para esto se utilizaron las herramientas `ping` y ethereal, a efectos de capturar y analizar en detalle, paquete a paquete.
Estas capturas se ubicaron en una de las máquinas partícipes en el túnel.

El comando `ping` se realizó desde la maquina Test PC "B" con la IP 192.168.11.2 hacia su par Test PC "A" con la IP 192.168.10.2

El resultado del `ping` en Test PC "B" es:

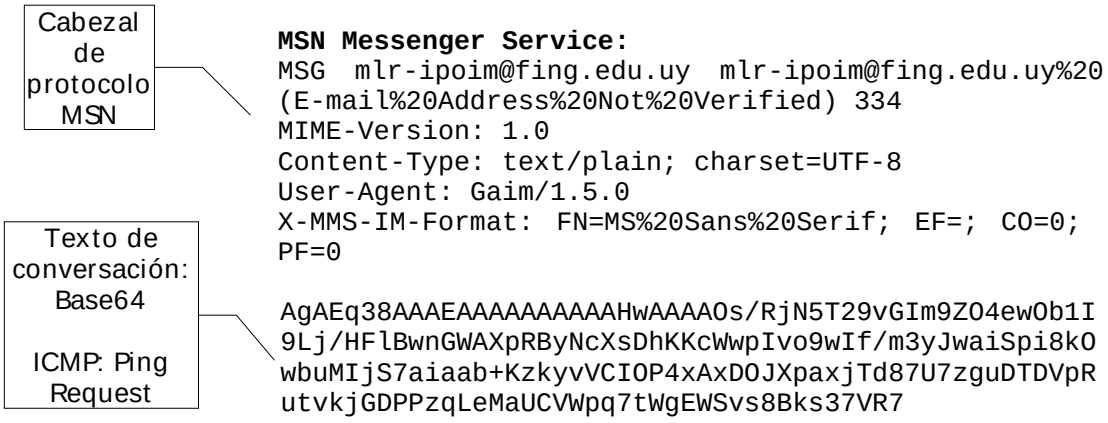
```
# ping 192.168.10.2 -c 60
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=62 time=597 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=62 time=588 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=62 time=583 ms
....
<removido para simplificar la visualización>
....
64 bytes from 192.168.10.2: icmp_seq=58 ttl=62 time=596 ms
64 bytes from 192.168.10.2: icmp_seq=59 ttl=62 time=593 ms
64 bytes from 192.168.10.2: icmp_seq=60 ttl=62 time=605 ms

--- 192.168.10.2 ping statistics ---
60 packets transmitted, 60 received, 0% packet loss, time 59192ms
rtt min/avg/max/mdev = 583.884/593.983/614.978/6.398 ms
```

Luego en la maquina Tunel PC "A", se realizaron 2 capturas:

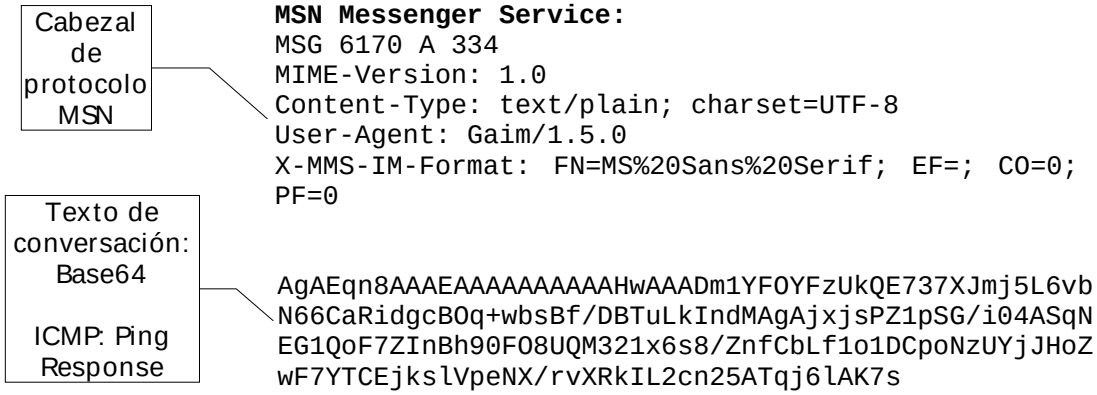
1. La comunicación hacia/desde Internet (mensajería instantánea).
2. La comunicación hacia/desde la red privada "A".

En la captura 1 se detecta la conversación con el servidor de mensajería. Se determina que en el detalle de la captura, se cumple la secuencia de mensajes:

1.1.- Recepción de IM (PING REQUEST)



1.2.- Envío de IM (PING RESPONSE)



1.3.- ACK de envío de IM

```
MSN Messenger Service:
ACK 6170
```

En esta captura, esta secuencia de mensajes (1.1, 1.2, 1.3) se repite por cada `ping`.

En la captura 2, la cual obtiene los paquetes de maquina Tunel PC "A" hacia/desde Test PC "A", se detectan los mensajes del protocolo ICMP del `ping`.

TestPC "B" -> TestPC "A" 192.168.11.2 -> 192.168.10.2	ICMP	Echo (ping) request
TestPC "A" -> TestPC "B" 192.168.10.2 -> 192.168.11.2	ICMP	Echo (ping) reply

Este análisis muestra como se corresponden los paquetes de mensajería instantánea con los de `ping` a nivel de red.

Es decir, por cada request enviado por "Test PC B" se captura:

1. Recepción de mensaje IM en "Tunel PC B" a través de la conexión a Internet.
2. Un mensaje de ICMP request desde Tunel PC "B" a Test PC "B"
3. Un mensaje de ICMP response desde Test PC "B" a Tunel PC "B"
4. Envío de mensaje IM de Tunel PC "B" a través de la conexión a Internet.

lo que completa la secuencia de mensajes para el protocolo.

5.5 Conclusiones

Las pruebas realizadas nos dieron una evaluación del prototipo implementado. Los resultados obtenidos indican que el prototipo permite tomar mediciones sobre la mensajería como transporte de datos, dando distintos resultados según los ambientes de prueba.

Durante las pruebas realizadas se detectaron algunos errores de implementación. La mayoría de estos errores fueron "debugueados" y corregidos, pero igualmente con baja frecuencia se siguen dando algunos errores que no se pudieron rastrear.
Estos errores fueron de dos tipos:

- `Segmentation fault`
- `SSL no podía desencriptar los paquetes que recibía.`

Estos casos no pudieron ser corregidos ya que los errores no correspondían a un patrón, es decir que realizando la misma prueba y bajo condiciones similares, no se volvían a dar.

Según nuestro criterio, son satisfactorios los resultados de performance, ya que en su mayoría fueron consistentes. Esta consistencia nos permite obtener mediciones a gran escala, como para tener un panorama general del comportamiento y el resultado del prototipo implementado.

En particular, los resultados obtenidos con el comando `ping`, respecto al RTT hacen pensar que el transporte de datos tiende a ser estable. Se realizaron varias mediciones con este comando, en distintos días y horas, y los tiempos de respuesta siempre rondaron en los 550ms y nunca superaron los 700ms, bajo condiciones normales (sin saturación) con servidor de mensajería MSN, y utilizando diferentes conexiones a Internet. Igualmente estas pruebas no son suficientes para afirmar "estabilidad", pero si dan buenos augurios.

También los resultados obtenidos de throughput con distintas herramientas fueron satisfactorios, y para ambientes similares, consistentes.
El resultado de `22 Kbits/seg` obtenido en las primeras pruebas, es comparable con la velocidad que se puede esperar de un módem. Es importante tener en cuenta, que este valor es relativo al protocolo MSN y con una conexión compartida a Internet mediante un ADSL de 512/128 Kbps.

Se espera que estos resultados cambien sustancialmente para otros protocolos o bajo otras condiciones. Esto podría deberse a protocolos con menor/mayor overhead o a protocolos con menor/mayor cantidad de usuarios y mejores/peores servidores de mensajería.

Pruebas interesantes se podrían hacer, utilizando otros protocolos de

mensajería instantánea confrontándolos con los resultados de MSN. También se podría probar utilizar algún protocolo de mensajería que no utilice “Servidor de Mensajería” para una conversación IM. Utilizando estos protocolos el traslado de los paquetes sería directo entre los usuarios participantes de la conversación.
Seguramente, en este caso los resultados serán sensiblemente mejores, ya que los paquetes recorrerían un camino directo hacia la máquina destino.

Como conclusión final, consideramos que las pruebas realizadas y los resultados obtenidos, son suficientes para los objetivos de este proyecto. Igualmente queda una amplia variedad de pruebas posibles de realizar, ya que existe un gran numero de variables que podrían modificar los resultados (arquitectura, protocolo de mensajería, etc). También quedaron algunos “efectos” interesantes para analizar, como por ej. el comportamiento de las pruebas con distintos tamaños de buffers con la herramienta TTcp, y las diferencias encontradas en los valores de RTT.

6 – CONCLUSIONES Y TRABAJO FUTURO

Este trabajo muestra empíricamente que es posible utilizar los servicios de mensajería instantánea para transportar cualquier tipo de información.

Esta demostración se logró construyendo un prototipo funcional. Para construir dicho prototipo se buscaron soluciones open/free adecuadas, tanto para el transporte de datos como para la mensajería. Las soluciones elegidas, Gaim y OpenVPN, presentan una gran ventaja a nuestro trabajo, ya que son herramientas con suficiente tiempo en el mercado como para garantizar su madurez y estabilidad.

Las pruebas que se realizaron con el prototipo construido resultaron satisfactorias en los resultados obtenidos, y confirmaron la viabilidad del transporte de datos sobre la mensajería instantánea.

Aportes

A pesar de construir en este trabajo solamente un prototipo funcional, él poder lograr enviar y recibir mensajes IP a través de una VPN mediante la mensajería instantánea, tiene consecuencias inmediatas:

- Podría brindar una funcionalidad adicional a los clientes de mensajería instantánea, contando con la adición de la seguridad y confiabilidad que ofrecen las VPNs.

- Podría solucionar problemas de configuración de seguridad en usuarios externos o móviles de redes privadas. No requiere la necesidad de abrir puertos en firewalls, ni armar configuraciones especiales, simplemente requiere poder comunicarse mediante la mensajería. Las pruebas sobre la conectividad son directas, a través de una conversación IM.

Los resultados obtenidos en las pruebas son solamente una pequeña muestra del comportamiento de la mensajería para transporte de datos. Igualmente consideramos que podrían ser de utilidad para investigaciones sobre el tema de mensajería instantánea, tanto en lo que respecta a seguridad en los servicios de mensajería como seguridad en las empresas que utilizan el servicio.

Balance

Este trabajo tuvo una importante dedicación a la investigación y estudio de las tecnologías relacionadas. Los estudios sobre herramientas y tecnologías incluyen no sólo información de los mismos, sino también conceptos que son específicos en el área, los cuales debieron ser estudiados también.
También fue extensa la dedicación a la comprensión y adaptación de las herramientas open/free elegidas. Estas herramientas, no solamente no contenían documentación adjunta suficiente, sino que su implementación no estaba lo suficientemente encapsulada como para facilitar la comprensión a través de la lectura de sus códigos fuentes. Se trabajó sobre estas soluciones para comprender su arquitectura, su implementación, y su funcionamiento. La escasa documentación sobre estas soluciones a nivel de implementación, afectó negativamente este trabajo. La dedicación sobre este punto superó las expectativas, haciendo retrasar el cronograma planificado inicialmente.

Al contrario con las pruebas, al utilizar especificaciones estándares y herramientas recomendadas, la dedicación resultó menor a la esperada, obteniendo más de los resultados esperados.

Trabajos Futuros

- *Mejoras del prototipo*

  El prototipo fue construido con el objetivo de ser funcional. Es por esta razón que para convertirlo en un producto se deben tener en cuenta algunas de las omisiones realizadas durante su implementación:

  - Restringir los envíos y recepciones solamente a los usuarios del túnel. En las implementaciones sobre la mensajería, no se verifica en ningún momento que el usuario que envía un mensaje IM sea el usuario con el cual se estableció el túnel.

  - Avanzar en la adaptación del Gaim, a efectos de aislar los recursos utilizados del mismo. Por ejemplo, se podrían remover todas las implementaciones para las interfaces gráficas.

  - Ídem para OpenVPN. El ejemplo en este caso podría ser remover todas las implementaciones sobre conexiones TCP o UDP reales.

  - Corregir los errores no sistemáticos detectados en las

pruebas.

- En la versión de plataforma Linux, el prototipo obliga a tener el servidor X levantado, y además tener permisos para desplegar cosas en él. Para mejorarlo se deberían eliminar estas dependencias, y permitir que funcione como demonio o desde consola de texto.

- Corregir la finalización de la aplicación, para que ésta pueda ser terminada mediante comandos.

- Implementar algún sistema de reintento ante caídas en la conexión con la mensajería instantánea.

- La versión en la plataforma Windows, obligó al prototipo a ser una dll para poder utilizar los protocolos de mensajería instantánea. Para mejorarlo se podrían modificar los protocolos o utilizarlos mediante otro mecanismo.

- *Dependencias de Soluciones Open/Free*

  Si bien este trabajo involucraba la utilización de soluciones open/free para los conceptos IP e IM, también se podría haber realizado sin utilizarlos.
  Durante la investigación y el estudio de estas soluciones, se llego a tener un conocimiento profundo de ellas, y es por eso que con los conocimientos ahora adquiridos se podría llegar a construir un prototipo implementando sus funcionalidades sin necesidad de depender de las soluciones anteriores.
  Esta implementación si bien se vería beneficiada por la independencia podría verse afectada por los costos adicionales de mantenimiento.

- *Otras implementaciones*

  Al finalizar con la implementación de este prototipo, surgió la idea de implementar funcionalidades similares a las de “socket”. La idea es implementar las funcionalidades ofrecidas por los sockets, open/read/write/close y mediante estas operaciones utilizar la mensajería instantánea como transporte.

APÉNDICE A: IMPLEMENTACIÓN

En este apéndice se describen los detalles técnicos de las implementaciones realizadas para el desarrollo del prototipo. Esta implementación esta basada en el análisis y la solución propuesta en el capítulo 3.
Estas implementaciones pueden también contribuir a la comprensión de algunos aspectos de la arquitectura y el diseño de las soluciones open/free utilizadas.

Implementación

A los efectos de lograr la mayor independencia posible de los códigos base utilizados, mejorando la portabilidad entre versiones, se buscó realizar mínimas modificaciones en los códigos de los productos.
Se utilizaron los paquetes para agregar funcionalidad, y se modificaron los códigos fuentes que crean el ejecutable del OpenVPN, convirtiéndolo en el ejecutable principal del IPoIM.
Asimismo, se comprobó que los servicios de IM existentes no transportan mensajes en formatos binarios (i.e., un paquete de red *raw*), debiendo éstos ser convertidos a texto. A este proceso lo denominamos codificación, y consiste en la conversión de mensajes binarios arbitrarios en texto transportable por los protocolos de IM y viceversa.

El detalle de la implementación del prototipo contiene tres partes:

Mensajería Instantánea: En donde se implementaron funcionalidades para relacionar al OpenVPN con la mensajería instantánea mediante el Gaim.

Codificación: Interfaz mínima para la utilización de distintas rutinas de codificación.

Adaptaciones a los fuentes del OpenVPN: Para adaptar la implementación del OpenVPN a los objetivos del proyecto, se modificaron algunos de los archivos que contienen el código fuente del OpenVPN.

Las implementaciones de mensajería instantánea y codificación se realizaron en archivos independientes de los fuentes del OpenVPN y Gaim.

Mensajería Instantánea

Implementación de las funcionalidades necesarias para utilizar la mensajería instantánea, utilizando las implementaciones del Gaim.

GAIM – IPoIM

La configuraciones de cuentas de usuario para la mensajería se hacen mediante la aplicación gaim, o editando el archivo "accounts.xml" e ingresando los datos necesarios (ver Apéndice B: Configuración).

Files: gaim_ipoim.h gaim_ipoim.c

Inicializar gaim e iniciar proceso de inicio de sesión.

```
void init_gaim(char *account,
               char *protocol,
               struct context *c
               );
```

Objetivo:
Inicializar los directorios y demás estructuras de datos de gaim. Iniciar proceso de inicio de sesión.

Parámetros:
account = usuario que inicia sesión
protocol = protocolo de ese usuario
struct context *c = contexto utilizado por el openvpn para su funcionamiento.

IMPLEMENTACIÓN:
- Iniciar el Gaim. La primer parte de esta función es una copia de lo implementado en el fuente: main.c (lineas desde 867 a 914)
- Buscar la cuenta configurada en el gaim para los parámetros account y protocol mediante la función
  GaimAccount *g=gaim_accounts_find(account, protocol)
- Iniciar el proceso de "inicio de sesión" mediante la función:
  gaim_account_connect(g)
- Mediante la invocación a estas funciones:

```
void *conv_handle = gaim_conversations_get_handle();
gaim_signal_connect(
          conv_handle, "received-im-msg",
          g_main_context_default(),
          GAIM_CALLBACK(gaim_IPoIM_received_im_msg),
          (void *)c
      );
```

se registra a la función gaim_IPoIM_received_im_msg para ser invocada cada vez que se recibe un mensaje y es pasado el contexto c como parámetro.

Recibir los mensajes IM

```
void gaim_IPoIM_received_im_msg(GaimAccount *account,
                                char *sender,
                                char *buffer,
                                int flags,
                                void *data
                                );
```

Objetivo:
Recibir mensajes IM mediante gaim.

Parámetros:
- account = cuenta receptora del mensaje
- sender = usuario que envía el mensaje
- buffer = mensaje IM
- flags = banderas del mensaje
- data = otros parámetros

IMPLEMENTACIÓN:
- Remover los caracteres para la visualización en HTML con la función
  `char *msg = gaim_markup_strip_html (buffer);`
- Invocar a la función del openvpn que procesa los paquetes recibidos
  `process_new_im(msg, data);`
  (ver Modificaciones a openvpn.c)

Enviar los mensajes IM

```
void gaim_IPoIM_send_im_msg(char *account,
                            char *protocol,
                            char *to,
                            char *msg
                            );
```

Objetivo:
Enviar mensajes IM mediante gaim.

Parámetros:
- account = usuario que envía el mensaje
- protocol = el protocolo utilizado por esa cuenta
- to = usuario destinatario del mensaje
- msg = mensaje que se envia

IMPLEMENTACIÓN:

- Buscar la cuenta para el envío del mensaje, invocando la función:
  GaimAccount *g=gaim_accounts_find(account, protocol)
- Crear una nueva conversación utilizando la cuenta y el usuario destinatario del mensaje mediante la función:
  GaimConversation *c = gaim_conversation_new (GAIM_CONV_IM, g, to);
- Utilizando la conversación antes creada, enviar el mensaje mediante la función:
  gaim_conv_im_send(GAIM_CONV_IM(c), msg);

Otras Implementaciones

También se implementaron las siguientes funciones:

```
static GaimCoreUiOps *
IpoIM_core_get_ui_ops(void);

GaimEventLoopUiOps *
IpoIM_eventloop_get_ui_ops(void);

static GaimConnectionUiOps *
gaim_IPoIM_connection_get_ui_ops(void);
```

en donde cada una reescribe las funciones a ser invocadas por el disparo de los eventos. Cada una de esas estructuras tiene eventos asociados.

La funciones reescritas:

```
void gaim_IPoIM_conectado(GaimConnection *gc);
void gaim_IPoIM_desconectado(GaimConnection *gc);
```

definidas para GaimConnectionUiOps informan al openvpn del estado de conexión a la mensajería instantánea. Ésta información es enviada al openvpn para iniciar/detener el transporte de los datos.

Codificación

Implementa la interfaz mínima para utilizar distintos tipos de algoritmos para la codificación de binario a texto.

API Encoding

Solamente se implementó el algoritmo base 64 en este prototipo.

Files: encoding.h encoding.c

Algoritmos

```
int ALGORITMO_BASE64 = 1;
```

Descripción:
Son definidos como variables, identificados por el numero asociado.

Codificación

```
int encode(int type,
           const void *data,
           int size,
           char **str);
```

Objetivo:
Codificar binarios a texto, implementada para pasar los paquetes IP binarios a mensajes de texto.

Parámetros:

type = numero asociado al algoritmo a utilizar
data = datos a codificar
size = tamaño de los datos
str = parámetro de retorno, texto codificado

Retorno:

largo del texto codificado.

IMPLEMENTACIÓN:

- Condicionar el parámetro type obteniendo el algoritmo elegido
- Invocar la función que implementa el algoritmo para codificar.
- En el caso de base64 se utiliza la función:
  base64_encode(data, size, str)
  implementada en la distribución del openvpn,
  files:base64.h base64.c

Decodificación

```
int decode(int type,
           const char *str,
           void *data)
```

Objetivo:
Decodificar binarios a texto, implementada para pasar los mensajes de texto a paquetes IP binarios.

Parámetros:

type = numero asociado al algoritmo a utilizar
str = datos codificados
data = parámetro de retorno, datos decodificados

Retorno:

largo de los datos decodificados.

IMPLEMENTACIÓN:

- Condicionar el parámetro type obteniendo el algoritmo elegido
- Invocar la función que implementa el algoritmo para decodificar
- En el caso de base64 se utiliza la función:
  base64_decode(str, data)
  implementada en la distribución del openVPN,
  files:base64.h base64.c

Adaptaciones de los fuentes del OpenVPN

Las siguientes implementaciones, son cambios en los archivos fuentes originales del OpenVPN.

Options

Implementa todo lo que respecta a parámetros de configuración del openvpn.

Files: options.h options.c

Estructura de Opciones

```
struct options
{
.... (demás opciones del openvpn)
char *account_ipoim;
char *protocol_ipoim;
char *account_toipoim;
```

Objetivo:

Agregar a la estructura de opciones los paraemtros:
account_ipoim: cuenta de usuario que inicia sesión de mensajería instantánea (p.e.: mlr-ipoim@fing.edu.uy).
protocol_ipoim: identificador de protocolo en gaim (p.e.: prlp-msn para MSN Messenger).
account_toipoim: usuario con el cual se realiza la comunicación.

Mensaje de Uso

```
static const char usage_message[] =
.... (demás opciones del openvpn)
     "IPoIM config options:\n"
     "--ipoim account protocol accountto: Utilizar cuenta
'account' para el protocolo 'protocol' y comunicarse con
'accountto'.\n"
```

Objetivo:
Modificar el mensaje que explica el uso de la herramienta, para incluirle las nuevas opciones de configuración.

Función de lectura de opciones de OpenVPN.

```
static int
add_option (struct options *options,
         int i,
         char *p[],
         const char *file,
         int line,
         const int level,
         const int msglevel,
         const unsigned int permission_mask,
         unsigned int *option_types_found,
         struct env_set *es)
```

IMPLEMENTACIÓN:

- Se usa el mismo mecanismo con el cual se asignan otras opciones:

```
if ((streq (p[0], "ipoim"))&& p[1]&& p[2]&& p[3])
{
  options->account_ipoim = p[1];
  options->protocol_ipoim = p[2];
  options->account_toipoim = p[3];
}
```

Forward

Este módulo implementa las comunicaciones realizadas por OpenVPN.

Files: forward.h forward.c

Envío de Paquetes

```
void process_outgoing_link (struct context *c)
```

Objetivo:
Enviar los paquetes como un mensaje IM.
Esta función es invocada por cada paquete listo a ser enviado a la otra punta del túnel. Los datos a enviar son parte del contexto.

IMPLEMENTACIÓN:

- Remover invocación a la función:

```
size = link_socket_write (c->c2.link_socket,
                          &c->c2.to_link,
                          to_addr);
```

función implementada en el archivo socket.h, linea 708.
Esta función recibe como parámetros, el socket, el buffer a enviar y la dirección IP del receptor.
La cabecera de esta función es:

```
static inline int
link_socket_write (struct link_socket *sock,
                   struct buffer *buf,
                   struct sockaddr_in *to)
```

- Obtener dirección IP local del túnel, información que está en el contexto.

```
struct link_socket_info *lsi =
          get_link_socket_info (c);
struct sockaddr_in from_addr = lsi->lsa->local;
```

- Obtener largo de la dirección y de los datos a enviar:

```
int len_dir = sizeof((struct sockaddr_in) (from_addr));
int len_data = BLEN(&c->c2.to_link);
```

- Asignar memoria para almacenar los datos a enviar, utilizando los largos antes obtenidos.

```
void *pqt = malloc(len_dir + len_data + sizeof(int));
```

- Copiar al espacio de memoria asignada los datos a enviar.

```
mempcpy(
        mempcpy(
                mempcpy(pqt, &from_addr, len_dir),
                &len_data, sizeof(int)),
        BPTR(&c->c2.to_link), len_data
        );
```

- Codificar los datos, y obtener un mensaje de texto.

```
char ** msg = malloc(sizeof(char));
size = encode(1, pqt,
              len_dir + len_data + sizeof(int),
              msg);
```

- Enviar el mensaje mediante la función implementada para la mensajería:

```
gaim_IPoIM_send_im_msg(c->options.account_ipoim,
                       c->options.protocol_ipoim,
                       c->options.account_toipoim,
                       *msg);
```

- El resto de la implementación de esta función, no se modifica.

OpenVPN

Estas modificaciones son implementadas en el módulo principal del openvpn, para adecuar su funcionamiento a los requerimientos.

Files: openvpn.h openvpn.c

Control sobre la mensajería instantánea

```
int connected;
```

Descripción:
Variable que indica si el usuario esta "logueado" al sistema de mensajería instantánea, si ya completó el proceso de inicio de sesión.(1 = TRUE, 0 = FALSE)

```
void set_connected(int i);
```

Objetivo:
Establecer el valor de la variable connected

Parámetros:
i = valor a establecer en la variable

IMPLEMENTACIÓN:
- Asignar el valor i a la variable connected.

Implementación de los Threads

```
static GMutex *mutex = NULL;
```

Descripción:
Mutuo excluir el contexto en que trabaja el openvpn en el paralelismo de la recepción de mensajes IM y el resto de las funciones.

Thread 1

```
void tunnel_point_to_point_send (void *cvoid)
```

Objetivo:
Implementar el modulo principal.

Parámetros:
cvoid = parametros del thread, en este caso el contexto.

IMPLEMENTACIÓN:
- Basado en una copia de la función que implementa el modulo principal del openvpn:
  (openvpn.c, static void tunnel_point_to_point (struct context *c))
- Las modificaciones realizadas a esa copia son:
  - Verificar la variable connected, antes de comenzar el proceso.
  - Procesar sólo en los casos en que el estado sea:
    SOCKET_WRITE: Se deben escribir datos en el túnel.
    TUN_READ: Se deben leer datos del túnel.
  - La llamada a la función que procesa los estados:
    void process_io (struct context *c) del archivo forward.c linea 1392.
    es mutuo excluida mediante las funciones:
    g_mutex_lock (mutex), se bloquean los cambios sobre el contexto antes de procesar alguno de los eventos.

    g_mutex_unlock (mutex), se desbloquean los cambios sobre el contexto, después de procesar los eventos.

Thread 2

```
void tunnel_point_to_point_receive ();
```

Objetivo:
Implementar la espera de eventos del gaim.

Parámetros:

IMPLEMENTACIÓN:

- Mediante la utilización de la función:
  gtk_main();
  se permite la ejecución de los eventos de gaim, los cuales incluyen el proceso de inicio de sesión, y la recepción de mensajes IM.

Mensajería Instantánea

```
void process_new_im(char *msg,
                    void *data)
```

Objetivo:
Por cada mensaje recibido mediante el gaim, esta función adecua el contenido del mensaje y modifica el contexto utilizado por el openvpn. Estas modificaciones hacen que el openvpn procese el nuevo mensaje recibido como un nuevo paquete recibido.

Parámetros:
msg = mensaje de texto recibido.
data = datos a adjuntar al mensaje. En este caso es el contexto del openvpn.

IMPLEMENTACIÓN:

- Castear el contenido de data, en una variable de tipo (struct context *) c.

- Bloquear la variable mutex ya que se va a modificar el contexto.
  g_mutex_lock (mutex);

- Decodificar el mensaje de texto (base64), y almacenar el resultado en una nueva variable.

  ```
  void *pqt = malloc(strlen(msg));
  int len = decode(1, msg, pqt);
  ```

  donde decode es la función implementada en el módulo encoding.

- Modificar el contexto con el contenido de pqt.

  - De los primeros bytes de pqt obtener, la dirección IP del túnel del que envía el mensaje.
    Con esa información se modifica el contexto:

    ```
    int len_dir=sizeof((struct sockaddr_in)(c->c2.from));
    memcpy(&(c->c2.from), pqt, len_dir);
    ```

- El resto de los datos contienen el paquete IP y su largo. Modificar entonces el contexto y asignar esos datos a las variables utilizadas por el openvpn para la recepción de mensajes.

```
pqt += len_dir;
int largo;
memcpy(&largo, pqt, sizeof(int));
pqt += sizeof(int);

c->c2.buf.len = largo;
memcpy(BPTR(&c->c2.buf), pqt, largo);
```

- Invocar la función

```
/*
 * Input:  c->c2.buf
 * Output: c->c2.to_tun
 */

void process_incoming_link (struct context *c)
implementado en el archivo forward.c, linea 659.
```

Esta función procesa (verifica el contenido del buf) el contexto y asigna en la variable to_tun del contexto el paquete a escribir en el túnel.

- Invocar la función:

```
/*
 * Input: c->c2.to_tun
 */

void process_outgoing_tun (struct context *c)
implementado en el archivo forward.c, linea 1083.
```

Esta función escribe en el túnel el contenido de la variable to_tun del contexto.

- Desbloquear la variable mutex

```
g_mutex_unlock (mutex);
```

Ejecución Principal

```
static void tunnel_point_to_point (struct context *c)
```

Objetivo:
Esta función era originalmente utilizada por el openvpn para su ejecución principal. Fue sustituida por

```
void tunnel_point_to_point_send (void *cvoid)
```

Esta función ahora realiza la inicialización del gaim, y luego la creación de los threads.

Parámetros:
c = contexto del openvpn.

IMPLEMENTACIÓN:

- Los dos threads a se utilizados:
```
GThread *g_tunnel_point_to_point_send;
GThread *g_tunnel_point_to_point_receive;
```

- Inicializar el gaim, mediante la función implementada
```
init_gaim(c->options.account_ipoim,
          c->options.protocol_ipoim,
          c);
```
a la cual se le pasan como parámetros los datos del usuario de mensajería configurados en *options*, y el contexto.

- Crear la variable mutex:

```
mutex = g_mutex_new ();
```

, y crear los threads:

```
g_tunnel_point_to_point_send =
    g_thread_create((void *)tunnel_point_to_point_send,
                    (void *)c,
                     TRUE,
                     NULL);
```
el que se encargará de la ejecución principal del openvpn, a la cual se le pasa como parámetro el contexto c.

```
g_tunnel_point_to_point_receive =
  g_thread_create((void*)tunnel_point_to_point_receive,
                  NULL,
                  TRUE,
                  NULL);
```
el que se encargará de la ejecución de los eventos gaim.

- Esperar por la finalización de los threads
```
g_thread_join(g_tunnel_point_to_point_send);
g_thread_join(g_tunnel_point_to_point_receive);
```
y liberarar la memoria utilizada para la variable mutex.
```
g_mutex_free(mutex);
```

Modificaciones al main

```
#ifdef WIN32
 int openvpn_main(HINSTANCE hint, int argc, char *argv[])
#else
 int main (int argc, char *argv[])
#endif
```

Objetivo:
Para la ejecución en windows de este prototipo, se agrega el cabezal de main.

```
gtk_init (&argc, &argv);
```

Objetivo:
Agregar al inicio de la ejecución, la inicialización del biblioteca gtk.

```
#ifdef WIN32
        wgaim_init(hint);
#endif
```

Objetivo:
Agregar para windows la inicialización del gaim. La implementación de la función
void wgaim_init(HINSTANCE hint) esta en el archivo win32/win32dep.c, linea 415.

```
connected = 0;
```

Objetivo:
Inicializar la variable connected para bloquear el inicio de la ejecución hasta que no se complete el proceso de inicio de sesión.

```
#ifdef WIN32
        wgaim_cleanup();
#endif
```

Objetivo:
Agregar la finalización del gaim para windows. La implementación de la función
void wgaim_cleanup(void) esta en el archivo win32/win32dep.c, linea 500.

APÉNDICE B: INSTALACIÓN

La instalación está separada en secciones. Primero se detallan las instalaciones (compilación y configuración) de las soluciones open/free utilizadas. Luego se detalla la instalación del prototipo implementado.

Las plataformas en las cuales fueron testeados estas instalaciones son:
- SuSE Linux 9.1 y 9.2, Kernel 2.6.4
- Microsoft Windows 2000

Gaim v1.5.0 - Instant Messaging client

La distribución está disponible en: http://gaim.sourceforge.net/
La versión utilizada para este trabajo: 1.5.0 (*File: gaim-1.5.0.tar.gz*)

Requerimientos de Instalación:

Compiladores:
Linux: gcc (se utilizó versión 3.3.3)
Windows: mingw para msys (se utilizó versión 1.0).
Bibliotecas:
GTK+ 2.0 http://www.gtk.org/
Otras bibliotecas específicas:
Para el soporte de audio:
- libao: http://freshmeat.net/projects/libao
- libaudiofiles: http://www.68k.org/~michael/audiofile

Para correcciones ortográficas:
- libgtkspell: http://gtkspell.sf.net/

Para utilizar MSN Messenger:
- Soporte SSL, es necesaria una de las dos librerías:
  - GnuTLS http://www.gnu.org/software/gnutls
  - NSS y NSPR de el proyecto Mozilla: http://www.mozilla.org

Archivos de Configuración:
prefs.xml: archivo de configuración general.
accounts.xml: información sobre las cuentas.
blist.xml: lista de contactos.

Instalación:

1. 'cd' al directorio que contiene el código fuente.
2. Configurar el paquete de acuerdo al sistema
```
# ./configure
```
3. Compilar los fuentes.

```
# make
```

4. Instalar los programas y demás archivos de datos.

```
# make install
```

Configuración:

Se detallan solamente las configuraciones utilizadas para este proyecto.

Registrar nueva cuenta: Se pueden utilizar dos métodos, el método gráfico y el método manual.
Método gráfico: Accounts -> Add
Método manual: editar el archivo accounts.xml

Linux: `/home/`*`usuario`*`/.gaim/`

Windows: `C:\Documents and Settings\`*`usuario`*`\Datos de programa\.gaim\`

y agregar bajo el tag: `<accounts version='1.0'>` los datos:

```
<account>
  <protocol>Nombre del Protocolo</protocol>
  <name>Nombre de Usuario</name>
  <password>Contraseña</password>
  <alias>Texto visible</alias>
  <settings>
      Configuraciones dependientes de cada protocolo.
  </settings>
</account>
```

Dónde los posibles *Nombre de Protocolo* son especificados en el archivo de configuración prefs.xml (por ej.: prpl-msn)

Ejemplo:

Cuenta de MSN Messenger utilizada para las pruebas.

```
<account>
  <protocol>prpl-msn</protocol>
  <name>mlr-ipoim@fing.edu.uy</name>
  <password>111111</password>
  <alias>Maria</alias>
  <settings>
    <setting name='check-mail' type='bool'>0</setting>
    <setting name='server'
type='string'>messenger.hotmail.com</setting>
    <setting name='http_method' type='bool'>0</setting>
    <setting name='port' type='int'>1863</setting>
  </settings>
  <settings ui='gtk-gaim'>
    <setting name='auto-login' type='bool'>0</setting>
  </settings>
</account>
```

Agregar un contacto: Esta función solamente se puede hacer

mediante la interfaz gráfica. Se realiza conectado al servicio de mensajería y requiere respuesta del contacto agregado.
Luego de estar conectado, en la ventana de los contactos, Buddies -> Add Buddy. En la ventana emergente se deben ingresar los datos del contacto a agregar, donde es solamente obligatorio el nombre de usuario.

OpenVPN v2.0 - A Secure tunneling daemon

La distribución está disponible en: http://openvpn.net/
La versión utilizada para este trabajo: 2.0 (*File: openvpn-2.0.tar.gz*)

Requerimientos de Instalación:

Compiladores:
Linux: gcc (se utilizó versión 3.3.3)
Windows: mingw para msys (se utilizó versión 1.0).
Requerimientos:
- Permisos de Administrador.
- TUN y/o TAP driver para permitir programas de espacio de usuario controlar una interfaz virtual punto a punto IP o Ethernet. Para linux este driver está incluido en los kernels 2.4+, para windows vienen drivers dentro de la distribución del OpenVPN para Win-NT.

Bibliotecas: (opcionales pero recomendadas)
Encriptación:
- OpenSSL (0.9.5 o mayor): http://www.openssl.org/

Compresión:
- LZO: http://www.oberhumer.com/opensource/lzo/

Threads:
- PThreads

Otras bibliotecas:
Autoconf 2.50 o mayor + Automake 1.5 o mayor:
http://www.gnu.org/software/software.html
Dmalloc library: http://dmalloc.com/

Instalación:

1. 'cd' al directorio que contiene el código fuente.
2. Configurar el paquete de acuerdo al sistema
   ```
   # ./configure
   ```
3. Compilar los fuentes.
   ```
   # make
   ```
4. Instalar los programas y demás archivos de datos.
   ```
   # make install
   ```

Configuración:

La configuración que se explica a continuación no intenta ser completa ni detallada. Existen una gran variedad de manuales y explicativos disponibles sobre la configuración de la aplicación OpenVPN. Simplemente aquí se exponen las configuraciones que se usaron durante el proyecto, las cuales resultaron completas para lograr el

objetivo.

Esta configuración esta basada en los datos obtenidos de:

OpenVPN Static Key Mini- HOWTO
http://openvpn.net/static.htm

La configuración de claves estáticas es de las más simples, y es ideal para VPN's punto a punto y para tareas de testing.
Ventajas:

- Configuración Simple
- Evitar mantener X509 PKI (Public Key Infraestructure)

Desventajas:

- Escalabilidad limitada. Un servidor y un cliente.
- Falta de *perfect forward secrecy* (clave compromiso en total desconexión de sesiones previas)
- La clave secreta debe existir en forma de texto plano en cada punto de la VPN.
- La clave secreta debe ser intercambiada utilizando algún canal seguro preexistente.

Ejemplo Simple

Este ejemplo demuestra como configurar una conexión simple punto a punto mediante OpenVPN. Se creará un túnel VPN, el cual tendrá en una punta el servidor con la dirección IP 192.168.0.1 y en la otra punta al cliente con la dirección IP 192.168.0.2. La comunicación entre cliente y servidor será encriptada sobre el puerto UDP 1194 (puerto por defecto del OpenVPN).

Generación de Clave

Generar una clave estática:

```
# openvpn --genkey --secret static.key
```

donde los parámetros:

genkey = Genera una clave randómica a ser utilizada como un secreto compartido. Para ser usado con la opción `--secret`.
secret file = Escribe la clave al archivo `file`.

Este comando una vez ejecutado, generará el archivo `static.key` en el directorio corriente. Este archivo deberá ser copiado a los dos puntos del túnel, cliente y servidor, mediante algún canal seguro.

Archivo de Configuración de Servidor

Se genera el archivo a utilizar desde el servidor para el uso del OpenVPN. Este archivo debe contener las siguientes lineas:

```
dev tun
ifconfig 192.168.0.1 192.168.0.2
secret static.key
```

donde los parámetros:

dev tunX|tapX = interfaz tun o tap.

ifconfig l rn =

TUN: configurar la interfaz punto a punto con la dirección IP l como local y rn como remota. l y rn deben ser intercambiadas en la otra punta del túnel. l y rn deben ser direcciones privadas fuera de las subredes usadas por cada punta.
TAP: configurar la interfaz con la dirección IP l como local y rn como mascara de subred.

Archivo de Configuración de Cliente

Se genera el archivo a utilizar desde el servidor para el uso del OpenVPN. Este archivo debe contener las siguientes lineas:

```
remote myremote.mydomain 1194
dev tun
ifconfig 192.168.0.2 192.168.0.1
secret static.key
```

donde los parámetros:

remote host [port] = Nombre o dirección IP del host remoto.

Configuración de Firewall

Se debe asegurar que:

- El puerto UDP 1194 está abierto en el servidor.
- La interfaz virtual TUN usada por el OpenVPN no está bloqueada ni en el cliente, ni en el servidor (en linux probablemente se llame tun0, en Windows algo como Local Area Connection *n*)

Testear la configuración de la VPN

Ejecutar la aplicación openvpn con sus respectivos archivos de configuración en el cliente y servidor, modificando *myremote.mydomain* (parámetro remote en configuración de cliente) para el nombre de dominio o dirección IP publica del servidor:

```
# openvpn configfile
```

Para verificar si la correctitud de la VPN, se podría utilizar el comando ping desde el servidor al cliente:

```
Server 192.168.0.1
```

```
# ping 192.168.0.2
```

IPoIM – Internet Protocol over Instant Messaging
(*File: ipoim.tar.gz*)

Requerimientos de Instalación:

Al ser este prototipo una integración de dos herramientas, los requerimientos se desprenden de ellas.

Distribución:

El archivo de distribución *ipoim.tar.gz* contiene los archivos:

- dogaimchanges.pl : Programa Perl para el armado del Makefile y la compilación de los fuentes del gaim en Linux con las modificaciones.
  Resultado : Makefile.gaimchanges

- dogaimchanges32.pl : Programa Perl para el armado del Makefile y la compilación de los fuentes del gaim en Microsoft Windows con las modificaciones.
  Resultado : Makefile.gaimchanges

- domakefile.pl: Programa Perl para el armado del Makefile del prototipo ipoim para Linux. A través de este makefile se compilan los códigos fuentes del openvpn adaptados y las nuevas implementaciónes.
  Resultado: Makefile.ipoim

- encoding.c encoding.h : archivos de implementación.

- gaim- 1.5.0.tar.gz : distribución de Gaim utilizada.

- gaim_ipoim.c gaim_ipoim.h: archivos de implementación.

- INSTALL : Script principal que realiza los cambios necesarios para armar prototipo en Linux.

- INSTALL.32 : Script principal que realiza los cambios necesarios para armar prototipo en Microsoft Windows.

- ipoim.gaim.diff: Archivo que contiene las diferencias entre la implementación original del *gaim* y las implementaciones de este prototipo.

- ipoim.openvpn.diff: Archivo que contiene las diferencias entre la implementación original del *openvpn* y las implementaciones de en este prototipo.

- main_gaim.c: archivos de implementación.

- makefile.ipoim.w32: Makefile para compilar el prototipo en Microsoft Windows.
- openvpn-2.0.tar.gz: distribución de OpenVPN utilizada.
- README: archivo explicativo de la instalación del prototipo.
- win_openvpn.c: archivo de implementación (sólo Microsoft Windows).

Instalación Linux:

1. Descomprimir distribución, compilar e instalar Gaim (ver Instalación gaim-1.5.0) en el directorio corriente.
2. Descomprimir distribución, compilar e instalar OpenVPN (ver Instalación openvpn-2.0) en el directorio corriente.
3. Instalar el prototipo mediante:
   ```
   # ./INSTALL
   ```
   Este instalador genera en el directorio openvpn-2.0 el ejecutable del prototipo:
   ```
   # ./ipoim
   ```
4. Si no se utilizan las distribuciones en el directorio corriente, se debe editar los archivos:
   ```
   INSTALL
         openvpn_2_0 = '~openvpn-2.0'
         gaim_1_5_0  = '~gaim-1.5.0'

   domakefile.pl
         openvpn_srcdir = '~openvpn-2.0'
         gaim_srcdir    = '~gaim-1.5.0/src' *relativo al
         openvpn.

   dogaimchanges.pl
         gaim_srcdir = '~gaim-1.5.0/src'
   ```

Instalación Windows:

Para compilar el prototipo se necesita:

MinGW - Minimalist GNU for Windows
http://www.mingw.org/
con las distribuciones de binutils, gcc-g++, w32api, gdb, make.

Instalación OpenVPN – Windows

- Extraer, compilar e instalar:

LZO, OpenSSL (ver Instalación OpenVPN)
Perl - http://www.cpan.org/

- Editar en el archivo makefile.w32 las entradas
  OPENSSL
  LZO (y si la versión es >=2, cambiar también -llzo por -llzo2).
  DMALLOC (si se utiliza)

- Ejecutar
```
$cd ~openvpn
$make -f makefile.w32
```

Instalación Gaim – Windows

- Instalar GTK+2.0 runtime y agregarlo al PATH.
  http://www.gtk.org/

- Instalar Perl 5.8 para Windows en `c:\Perl`.

- Crear al mismo nivel que el directorio del gaim el directorio `win32-dev`

- Extraer al directorio `win32-dev`
  GTK+ Dev 2.6.9
  http://prdownloads.sourceforge.net/gaim/gtk-dev-2.6.9-rev-a.tar.gz

  Perl 5.8
  http://gaim.sourceforge.net/win32/perl582.tar.gz

  Tcl 8.4.5
  http://gaim.sourceforge.net/win32/tcl-8.4.5.tar.gz

  GtkSpell / Aspell
  http://ftp.gnu.org/gnu/aspell/w32/aspell-dev-0-50-3-3.zip
  http://gaim.sourceforge.net/win32/gtkspell-2.0.6.tar.gz

  Mozilla NSS
  ftp://ftp.mozilla.org/pub/mozilla.org/security/nss/releases/NSS_3_9_RTM/WIN954.0_OPT.OBJ/nss-3.9.zip
  ftp://ftp.mozilla.org/pub/mozilla.org/nspr/releases/v4.4.1/WIN954.0_OPT.OBJ/nspr-4.4.1.zip

  SILC Toolkit
  http://gaim.sourceforge.net/win32/silc-toolkit-0.9.12.tar.gz

- Modificar el archivo

`~gaim-1.5.0/plugins/perl/common/Makefile.mingw`
cambiar
`PERL := /cygdrive/c/perl/bin/perl` por
`PERL := /c/perl/bin/perl`

- Ejecutar
  `$ cd ~/gaim`
  `$ make -f Makefile.mingw install`

- Se creará el directorio "win32-install-dir". En ese directorio quedan luego el ejecutable y las dlls asociadas.

Instalación IPoIM – Windows

- Editar el archivo INSTALL.w32 y ajustar las entradas
  `openvpn_2_0 = '~openvpn-2.0'`
  `gaim_1_5_0 = '~gaim-1.5.0'`

- En el archivo makefile.ipoim.w32 ajustar las entradas
  OPENSSL
  LZO
  GAIM_SRC_DIR
  DMALLOC (si se utiliza)

- Ejecutar
  `$ .\INSTALL.w32`
  Este instalador genera en el directorio del openvpn-2.0 el ejecutable del prototipo:
  `$ .\ipoim`
- Si no estuviera instalado el software de TUN/TAP, hacerlo desde:
  Panel de Control ->
  Agregar o Quitar Hardware ->
  Agregar Dispositivo Nuevo (desde lista) ->
  Adaptadores de Red.
  utilizando los drivers que están en la distribución de openvpn,
  `\tap-win32\i386\OemWin2k.inf`
  `\tap-win32\i386\tap.cat`

- Asegurarse que todas las dlls usadas tanto por Gaim como por OpenVPN estén en directorios del PATH.

- Para utilizar el protocolo X de mensajería instantánea:
  - Editar el `Makefile.mingw` de ese protocolo en `~/gaim/src/protocols/x/`

  - Modificar en
    `##`
    `## LIBRARIES`

```
##
LIBS = cambiar -lgaim por -lipoim
```

- Recompilar el protocolo
  ```
  $ make -f Makefile.mingw
  ```

- Copiar la dll generada al directorio donde se encuentran los protocolos del gaim.
  `C:\Documents and Settings\`*`Usuario`*`\Datos de programa\.gaim\plugins`

Configuración:

Para la configuración de este prototipo se deben conocer:
- la configuración de Gaim para registrar una nueva cuenta y agregar un contacto.
- la configuración de OpenVPN para establecer un túnel.

1. Registrar en el Gaim la cuenta de usuario a utilizar para el túnel. Por ej.:

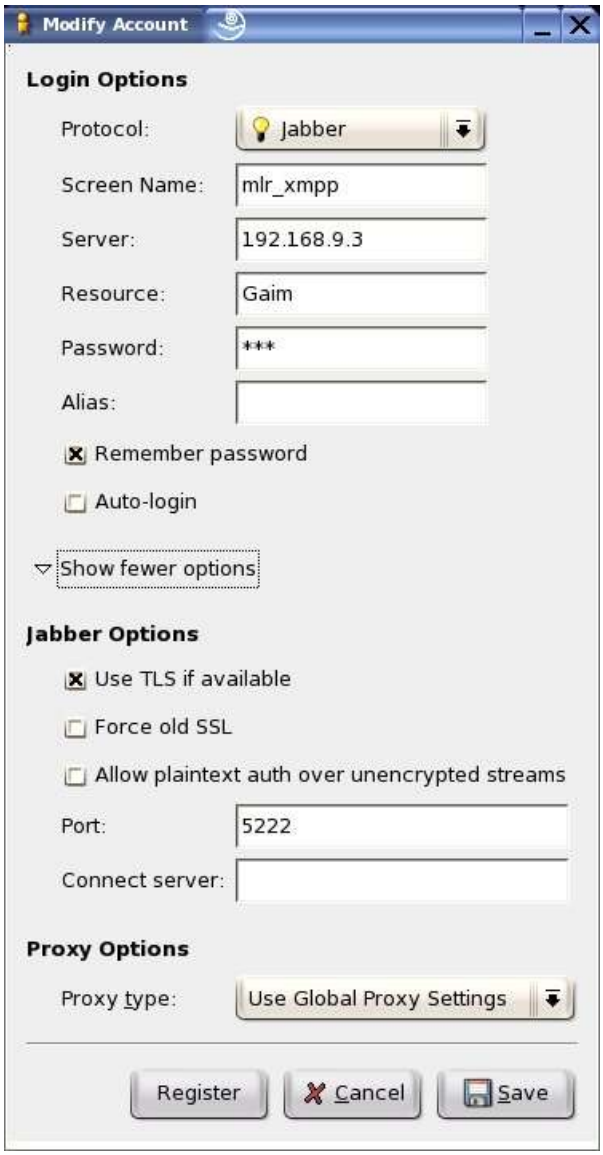


Ilustración 20.- Ejemplo de cuenta en Gaim

Esto mismo se puede realizar directamente en el archivo de cuentas, el mismo ejemplo quedaría:

```
<account>
  <protocol>prpl-jabber</protocol>
  <name>mlr_xmpp@192.168.9.3/Gaim</name>
  <password>111</password>
  <settings>
   <setting name='use_tls' type='bool'>1</setting>
   <setting name='old_ssl' type='bool'>0</setting>
   <setting name='auth_plain_in_clear' type='bool'>0</setting>
   <setting name='port' type='int'>5222</setting>
  </settings>
  <settings ui='gtk-gaim'>
   <setting name='auto-login' type='bool'>0</setting>
  </settings>
 </account>
```

La configuración de la cuenta debe tener clickeado “Remember

Password" o para el archivo incluirla en el tag <password>.

2. Si el usuario de la otra punta del túnel no está en la lista de contactos del usuario local, agregarlo.

3. Configurar el túnel mediante los parámetros del OpenVPN (ver Configuración de OpenVPN).

4. Agregar dentro de los parámetros del OpenVPN.

```
--ipoim account protocol accountto
```

donde:
account = usuario local registrado en paso 2.
protocolo = protocolo utilizado para la cuenta (los nombres de los protocolos están en el archivo prefs.xml de la configuración del gaim.)
accountto = usuario perteneciente a la lista de contactos, otra punta del túnel.

Por ej.

```
--ipoim mlr_xmpp@192.168.9.3/Gaim prpl-jabber asy_xmpp@linux
```

Glosario

ACK
ACKnowledgement, acuse de recibo. Es un mensaje que se envía para confirmar que un mensaje o un conjunto de mensajes han llegado.

ADSL
Asymmetric Digital Subscriber Line, tecnología de acceso a Internet de banda ancha.

AES
Advanced Encryption Standard, es un esquema de cifrado por bloque adoptado como un estándar de encriptación por el gobierno de los Estados Unidos.

API
Application Program Interface. Conjunto de convenciones de llamadas, que definen como un servicio es invocado a través de un paquete de software.

ASCII
American Standard Code for Information Interchange, código americano estándar para el intercambio de información.

ATM
Asyncronous Transfer Mode, tecnología de telecomunicaciones desarrollada para hacer frente a la gran demanda de capacidad de transmisión para servicios y aplicaciones.

Blowfish
Codificador de bloques simétricos, diseñado en 1993 e incluido en un gran número de suites de codificadores y productos de cifrado.

broadcast
Sistema de reparto de paquetes donde una copia del paquete dado es enviado a todos los hosts de la red.

BSD
Berkeley Software Distribution. Término usado cuando se describen diferentes versiones del software UNIX de Berkeley.

Buddy List
Ver Lista de Contactos

CHAP
Challenge Handshake Authenticacition Protocol, protocolo de autenticación por desafío mutuo. CHAP es usado periódicamente para verificar la identidad de los participantes de una comunicación. Definido en el RFC 1994: PPP Challenge Handshake Authentication Protocol (CHAP).

Cisco Systems
Empresa multinacional ubicada en San Francisco, California (EEUU), dedicada principalmente a la fabricación, venta y mantenimiento de equipos de telecomunicaciones.

DES

Data Encryption Standard, es un algoritmo de cifrado. Desde hace algunos años, el algoritmo ha sido sustituido por AES.

Ethernet

Tecnología de redes, define las características de cableados y señalización de nivel físico y los formatos de trama del nivel de enlace de datos.

Frame Relay

Servicio conmutación de paquetes de longitudes variables, para transportar datos sobre áreas extensas.

GCC

Gnu Compiler Collection, compilador que soporta diferentes lenguajes entre ellos C y C++.

GPL

General Public License es una licencia creada por la Free Software Fundation y principalmente orientada a los términos de distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre.

GRE

Generic Routing Encapsulation es un protocolo de "tunneling" diseñado para la encapsulación de paquetes de red. Definido en los RFCs 1701 y 1702.

GREv2

La versión PPTP del protocolo GRE, la cual añade extensiones específicas.

HTTP

HyperText Transfer Protocol es el protocolo usado en transacciones web.

ICMP

Internet Control Message Protocol, es el protocolo usado para manejar mensajes de error y control en la capa IP.

KDE

K Desktop Environment, es un entorno de escritorio gráfico e infraestructura de desarrollo para sistemas Unix y en particular Linux. Actualmente KDE es distribuido junto a distribuciones Linux.

HMAC

Keyed-Hashing for Message Authentication, es un tipo de código de autenticación de mensajes (Message Authentication Code – MAC) calculado usando criptografía en combinación con una clave secreta. Definido en el RFC 2104

IP

El Protocolo de Internet (Internet Protocol) es un protocolo usado para la comunicación de datos a través de una red de paquetes.

IPoIM

Internet Protocol over Instant Messaging.

IPSec

Internet Protocol Security, es una extensión al protocolo IP que añade cifrado fuerte para permitir servicios de autenticación y

cifrado y, de esta manera, asegurar las comunicaciones a través de dicho protocolo.

IPv4

Es la versión 4 del protocolo IP. Fue la primera versión del protocolo que se implementó extensamente, y forma la base de Internet.

IPv6

Es la versión 6 del protocolo IP. Destinado a sustituir al estándar IPv4, cuyo límite en el número de direcciones de red admisibles, está empezando a restringir el crecimiento de Internet y su uso.

IM

Instant Messaging, mensajería instantánea. Un tipo de comunicación que permite conversaciones entre dos o mas individuos mediante mensajes de texto en tiempo real.

ISP

Internet Service Provider, define al proveedor de Internet.

LAN

Local Area Network, define una red local de Pcs.

L2TP

Layer 2 Tunneling Protocol heredero aparente de los protocolos PPTP y L2F, creado para corregir las deficiencias de estos protocolos y establecerse como un estándar.

Lista de Contactos

Lista de usuarios de mensajería instantánea de un cliente. Los usuarios de esta lista son aquellos con los cuales el cliente se puede comunicar.

MS-CHAP

Es la versión de la empresa Microsoft del protocolo CHAP. Existe en dos versiones MS-CHAPv1 (definido en el RFC 2433) y MS-CHAPv2 (definido en el RFC 2759)

Norma X.25

Red de conmutación de paquetes basada en el protocolo HDLC proveniente de IBM. Establece mecanismos de direccionamiento entre usuarios, negociación de características de comunicación, técnicas de recuperación de errores.

ping

Un programa usado para testear alcance a un destino de red, mediante el envío de un ICMP echo request, y esperando por un ICMP echo response.

Plugin

Programa de software accesorio que extiende las capacidades de una aplicación existente.

PPP

Punto a Punto, protocolo a nivel de enlace que permite establecer una comunicación entre dos computadoras. Definido en el RFC 1661.

PPTP

Point to Point Tunneling Protocol, es un protocolo desarrollado por Microsoft, U.S. Robotics, Ascend Communications,

3Com/Primary Access, ECI Telematics conocidas colectivamente como PPTP Forum, para implementar redes privadas virtuales o VPN.

Puerto X

Un puerto de red es una interfaz para comunicarse. Los puertos de red suelen ser numerados y un cierto protocolo de transporte (como TCP o UDP) asigna alguno de esos números de puerto a la información que envía; la implementación del protocolo en el destino utilizará ese número para decidir a que programa entregar los datos recibidos.

RAS

Remote Access Server, refiere a cualquier combinación de hardware y software que permita el acceso remoto a herramientas o información.

RC4

Algoritmo de cifrado.

RFC

Request For Comments. Conjunto de notas técnicas y organizativas donde se describen los estándares o recomendaciones de Internet, comenzado en 1969.

RSA

Sistema criptográfico con clave pública, donde se hace pública la clave de cifrado y se oculta la clave de descrifrado.

Router

Enrutador o Encaminador, es un dispositivo hardware o software de interconexión de redes de computadoras, que opera a nivel de capa de red. Realiza el paso de los paquetes entre redes, basándose en la información de la capa de red.

SSH

Secure Shell, es el nombre de un protocolo y del programa que lo implementa, y se utiliza para acceder a máquinas remotas a través de una red

SSL

Secure Sockets Layer y Transport Layer Security (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras en Internet. SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía.

TCP

Transmission Control Protocol, es un protocolo de comunicación orientado a conexión y fiable del nivel de transporte. Es uno de los protocolos fundamentales en Internet. Definido en el RFC 793.

UDP

User Datagram Protocol, es un protocolo de comunicación de trasporte, basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera.

VPN

Virtual Private Network o Red Privada Virtual (RPV), es una tecnología de red que permite una extensión de la red local sobre una red pública o no controlada, como por ejemplo Internet.

WAN

Wide Area Network o Red de Área Amplia. Un ejemplo de este tipo de redes ser Internet.

XML

Extensible Markup Language, es una especificación que define una forma estándar de agregar marcas a un documento, es en definitiva un lenguaje para documentos que contienen información estructurada.

Bibliografía

Estudio del Open/Free (GNU/Linux) como plataforma de servicios de red en entornos empresariales.
Anexo: Servicios de Mensajería Instantánea y Presencia en la Empresa.
Daniel Caraballo, Mario Madera, Marcelo Odin.
Proyecto de Grado 2004, Instituto de Computación, Facultad de Ingeniería, Universidad de la República.

Reference Architecture for Multi- Protocol Instant Messaging Clients
Antonio D'souza, Kristina HildeBrand, Gilad Israeli
School of Computer Science – University of Waterloo
Diciembre 2004
http://www.cs.uwaterloo.ca/~kdhildeb/cs746/project.pdf [citado 05/03/2006]

Linux Advanced Routing & Traffic Control HOWTO.
Bert Hubert, Netherlabs BV.
Octubre 2003
http://www.ds9a.nl/lartc [citado 05/03/2006]

RFC 1208 – A Glossary of Networking Terms
O. Jacobsen, D. Lynch
Interop, Inc.
Marzo 1991
http://www.rfc- archive.org/getrfc.php?rfc=1208 [citado 05/03/2006]

Referencias

[1] Osterman Research Inc.
Firma investigadora especializada en el mercado de la mensajería.
http://www.ostermanresearch.com [citado 05/03/2006]

[2] I.D.C.
Estudio de mercado de la Mensajería Instantánea Empresarial
http://www.idc.com/getdoc.jsp?containerId=31809 [citado 05/03/2006]

[3] AOL Instant Messenger
http://www.aol.com/ [citado 05/03/2006]

[4] Oscar Protocol - *Open System for Communication in Realtime*
http://iserverd.khstu.ru/oscar/ [citado 05/03/2006]

[5] Gabber - *The GNOME Jabber Client*
http://gabber.sourceforge.net [citado 05/03/2006]

[6] ICQ
http://www.icq.com [citado 05/03/2006]
ICQ2Go!- *Your Web-based ICQ!* http://www.icq.com/icq2go [citado 05/03/2006]

[7] Yahoo! Messenger
http://www.yahoo.com [citado 05/03/2006]

[8] Kopete - *The KDE Instant Messenger*
http://kopete.kde.org [citado 05/03/2006]

[9] MSN Messenger Service 1.0 Protocol.
Instant Messaging and Presence Protocol, Internet Draft.
R. Movva, W. Lai. Microsoft.
Agosto 1999
http://www.hypothetic.org/docs/msn/ietf_draft.txt [citado 05/03/2006]
http://www.msn.com [citado 05/03/2006]

[10] Jabber Inc.
Enterprise Instant Messaging
http://www.jabber.com [citado 05/03/2006]

[11] Gaim - Instant Messaging client
http://gaim.sourceforge.net/ [citado 05/03/2006]

[12] IP in IP
RFC 1853 - IP in IP Tunneling
W. Simpson, Daydreamer
Octubre 1995
http://www.rfc-archive.org/getrfc.php?rfc=1853 [citado 05/03/2006]

[13] GRE - *Generic Routing Encapsulation*
RFC 1701 y 1702.
S. Hanks NetSmiths, Ltd.
T. Li, D. Farinacci, P. Traina cisco Systems
Octubre 1994
http://www.rfc-archive.org/getrfc.php?rfc=1701 [citado 05/03/2006]
http://www.rfc-archive.org/getrfc.php?rfc=1702 [citado 05/03/2006]

[14] PPTP - *Point-To-Point Tunneling Protocol*
Windows NT Server Product Documentation.
Chapter 11 - Point-To-Point Tunneling Protocol (PPTP).
http://www.microsoft.com/resources/documentation/windowsnt/4/server/proddocs/en-us/default.mspx [citado 05/03/2006]

[15] L2TP - *Layer Two Tunneling Protocol "L2TP"*
RFC 2661
W. Townsley, A. Valencia, cisco Systems
A. Rubens, Ascend Communications
G. Pall, G. Zorn, Microsoft Corporation
B. Palter, Redback Networks
Agosto 1999
http://www.rfc-archive.org/getrfc.php?rfc=2661 [citado 05/03/2006]

[16] IP Sec - *Secure IP over Internet*
The Linux Documentation Project
http://www.ipsec-howto.org/ [citado 05/03/2006]

[17] VTun
http://vtun.sourceforge.net/ [citado 05/03/2006]

[18] Open VPN - *The Open Source VPN solution.*
A Secure tunneling daemon
James Yonan
http://openvpn.sourceforge.net/ [citado 05/03/2006]

[19] OpenSSL - *Implementación Open Source de SSL (Secure Socket Layer) y TLS (Transport Layer Security)*
http://www.openssl.org [citado 05/03/2006]

[20] Virtual Point-to-Point(TUN) and Ethernet(TAP) devices
Universal TUN/TAP device driver.
Maxim Krasnyansky
1999-2000

http://vtun.sourceforge.net/tun/ [citado 05/03/2006]

[21] PCAP - *pcap - Packet Capture library.*
Van Jacobson, Craig Leres and Steven McCanne
Lawrence Berkeley National Laboratory, University of Califoria, Berkeley.
http://www.tcpdump.org/ [citado 05/03/2006]
WinPcap: The Windows Packet Capture Library :
http://www.winpcap.org/ [citado 05/03/2006]

[22] GTK+ - *The GIMP Toolkit*
Library for creating graphical user interfaces.
http://www.gtk.org [citado 05/03/2006]

[23] GLib - *General-purpose utility library.*
http://www.gnome.org [citado 05/03/2006]
http://developer.gnome.org/doc/API/glib/ [citado 05/03/2006]

[24] Ethereal – A Network Protocol Analyzer
Dump and analyze network traffic
http://www.ethereal.com [citado 05/03/2006]

[25] DBS- *Distributed Benchmark System*
Yukio Murayama, Suguru Yamaguchi.
Junio 1998.
http://www.ai3.net/products/dbs [citado 05/03/2006]

[26] DummyNet - *"Dummynet: a simple approach to the evaluation of network protocols".*
Luigi Rizzo
Dipartimento di Ingegneria dell'Informazione - Univ. di Pisa
ACM Computer Communication Review,
Vol. 27, N. 1, Enero 1997, pp. 31-41.
http://info.iet.unipi.it/~luigi/ip_dummynet/ [citado 05/03/2006]

[27] NetPerf
Benchmarking Methodology Working Group (BMWG).
IETF(The Internet Engineering Task Force).
http://www.netperf.org/netperf/NetperfPage.html [citado 05/03/2006]

[28] NIST Net - *A Linux-based Network Emulation Tool.*
Mark Carson, Darrin Santay
National Institute of Standards and Technology (NIST)
http://snad.ncsl.nist.gov/nistnet/ [citado 05/03/2006]

[29] Orchestra - *A Probing and Fault Injection Environment for Testing Protocol Implementations,*
Farnam Jahanian
Real-Time Computing Laboratory, Electrical Engineering and Computer Science Department, University of Michigan

http://www.eecs.umich.edu/RTCL/projects/orchestra/ [citado 05/03/2006]

[30] Packet Shell
Steve Parker, Chris Schmechel
Playground.Sun.COM, Sun Microsystems, Inc.
http://www.connectathon.org/talks96/psh.html [citado 05/03/2006]

[31] Tcpanaly - *Automated Packet Trace Analysis of TCP Implementations*
Vern Paxson
ACM SIGCOMM '97, Septiembre 1997, Cannes, France.
http://www.ssfnet.org/Papers/tcpanaly00.pdf [citado 05/03/2006]

[32] TCPTrace
Shawn Ostermann
Ohio University
http://www.tcptrace.org/ [citado 05/03/2006]

[33] Tracelook
Greg Minshall
Ipsilon Networks, Inc.
http://ita.ee.lbl.gov/html/contrib/tracelook.html [citado 05/03/2006]

[34] TReno
Matt Mathis, Jamshid Mahdavi
Pittsburgh Supercomputing Center (PSC), Carnegie Mellon University
http://www.psc.edu/networking/treno_info.html [citado 05/03/2006]

[35] TTcp
US Army Ballistics Research Lab (BRL)
http://renoir.csc.ncsu.edu/ttcp/ [citado 05/03/2006]

[36] XPlot
Tim Shepard
"TCP Packet Trace Analysis", MIT Laboratory for Computer Science MIT-LCS-TR-494.
http://www.xplot.org/ [citado 05/03/2006]

[37] tcpdump - *Dump traffic on a network.*
The Tcpdump Group
Lawrance Berkley National Laboratory, Network Reserch Group.
http://www.tcpdump.org/ [citado 05/03/2006]

[38] Jive Messenger
Instant messaging (IM) and chat server that implements the XMPP protocol.
http://www.jivesoftware.org/messenger [citado 05/03/2006]

[39] RFC 1242
Benchmarking Terminology for Network Interconnection Devices
S. Bradner
Harvard University
Julio 1991
http://www.rfc- archive.org/getrfc.php?rfc=1242 [citado 05/03/2006]