UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Implementación de un Modelo de Comunicación Anónima.

TRABAJO DE TITULACIÓN

AUTOR: Leiva Paladinez, Milton Iván

DIRECTOR: Torres Tandazo, Rommel Vicente PhD.

LOJA – ECUADOR

2016

Septiembre, 2016

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

PhD**.** Rommel Vicente Torres Tandazo.

DIRECTOR DEL PROYECTO DE TITULACIÓN

De mi consideración:

El presente trabajo de titulación: “Implementación de un Modelo de Comunicación Anónima” desarrollado por Milton Iván Leiva Paladinez, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, enero de 2016

f).....
PhD. Rommel Vicente Torres
CI. 1102885330

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

“Yo, Milton Iván Leiva Paladinez, declaro ser autor (a) del presente trabajo de titulación: “Implementación de un Modelo de Comunicación Anónima”, de la Titulación de Sistemas Informáticos y Computación, siendo el PhD. Rommel Vicente Torres Tandazo director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice:
“Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad”.

f……...……..
Autor: Milton Iván Leiva Paladinez
C.I. 1105660920

DEDICATORIA

Dedico esta tesis a la memoria de mi madre María Isabel Paladines Acaro y a mi abuelo Miguel Antonio Leiva que desde el cielo me cuidan. A mi padre Iván Estenio Leiva que en el pasado me ayudó a formarme como persona, a mi abuela Elisamira Caraguay que me instruyó cuando era un infante, a mi familia Leiva y a mi familia Paladines. Finalmente de manera especial dedico esta tesis a mis hermanos Jorge Daniel Leiva Paladinez y Carlos Alberto Leiva Caraguay que son mi motivo para seguir adelante.

A mis queridos compañeros y amigos que con ellos he vivido y he aprendido a través de sus consejos y experiencias compartidas y son parte de mi formación profesional.

AGRADECIMIENTO

Agradezco a Dios por darme vida y salud y por cuidar a seres queridos en el cielo, a la Virgen del Cisne "La Churonita" por derramar sus bendiciones; a mis familiares, amigos y personas conocidas, a mis docentes por compartir sus enseñanzas y experiencias conmigo, a mis compañeros por ser parte de mi formación, viviendo alegrías y tristezas y superando barreras y obstáculos.

A la Universidad Técnica Particular de Loja, que me acogió para brindarme la oportunidad de cumplir mis sueños y metas siguiendo su filosofía que es "Buscar la verdad y formar al hombre, a través de la ciencia para que sirva a la sociedad".

Al PhD. Rommel Vicente Torres Tandazo, por guiarme con sus sabios consejos durante el desarrollo del presente trabajo ya que con su paciencia y perseverancia me ha motivado a seguir adelante, brindándome su confianza y enseñándome que se puede lograr a cumplir grandes objetivos.

.

A todas y cada una de las personas que me apoyaron ya sean compañeros, amigos o docentes de mi ex colegio "La Dolorosa" o amigos de mi hermano, ya que con sus palabras de aliento y anécdotas, aportaron a mi formación tanto personal como profesional y me motivaron a seguir adelante.

Milton Iván Leiva Paladinez

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES.

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto de fin de titulación se centra en la investigación de la comunicación anónima, cuyo objetivo es ocultar los identificadores de red (direcciones IP). Para ello se analizan las diferentes técnicas y protocolos anónimos de intercambio de información que existen actualmente. Posteriormente se utiliza el algoritmo de enrutamiento a través de capas como base para diseñar un prototipo comunicación anónima. Finalmente, se evalúa la latencia que genera el prototipo al momento de intercambiar información.

PALABRAS CLAVES: Anonimato, Nodos.

ABSTRACT

This project focuses on the investigation of anonymous communication, which aims to hide network IDs (IP addresses). Different techniques and protocolos for anonymous information exchange are analyzed. Subsequently, the layers routing algorithm is used for designing an anonymous communication prototype. Finally, the latency generated by the prototype when exchanging information is evaluated..

KEYWORDS: Anonymity**,** Network nodes**,**

INTRODUCCIÓN

Actualmente la privacidad es fundamental al momento de navegar por la Web, los protocolos de seguridad existentes sólo se encargan de ocultar los datos que fluyen por la red, pero no oculta la identidad de destinatarios y remitentes.

La seguridad de la información digital es primordial al momento que viaja por la red, especialmente si dicha información pertenece a instituciones financieras o gubernamentales, los mismos que invierten grandes cantidades de dinero para proteger sus datos.

Existen muchas formas de proteger la información como puede ser a través de usuarios y contraseñas, permisos de acceso entre otras, pero éstas no garantizan que realmente estén protegidas de aquellos atacantes informáticos que utilizan sus habilidades para obtener información de los equipos computacionales, ya que primeramente tratará de conocer su identificador en la red (dirección IP) de un computador utilizando cualquier software y vigilando el tráfico en la red para capturar una víctima para posteriormente vulnerar su seguridad.

En este trabajo, se analiza la posibilidad de proteger la identidad de los usuarios en la red a través de una comunicación anónima que garantice la privacidad y seguridad al momento que se intercambia información.

El presente trabajo de fin de titulación está organizado de la siguiente manera:

Capítulo I: en este apartado se realiza la descripción del actual problema en la comunicación a través de las redes, y el espionaje de información en los equipos computacionales.

Capítulo II: contiene el marco teórico de la comunicación anónima, además de las diferentes técnicas que actualmente existen. Aquí se estudia sus componentes y funcionamiento, además, se estudia las aplicaciones que utilizan técnicas de comunicación anónima.

Capítulo III: en este apartado se describe dos técnicas que se considera idóneas para poder utilizar una de ellas y desarrollar un prototipo de comunicación anónima en la red, se brinda una explicación del funcionamiento del prototipo.

Capítulo IV: contiene la descripción de los requerimientos del prototipo además de su desarrollo, en donde se aplicó la metodología ágil Scrum.

Capítulo V: en este apartado se realiza la validación y se discute los resultados sobre los tiempos que genera el prototipo

GLOSARIO

Anonimato: Ocultación de la dirección IP con el objetivo de proteger la comunicación entre el emisor y receptor utilizando diferentes direcciones.

Broadcast: Es una forma de transmisión de la información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea.

Colisión: Interferencia entre tramas el cual no permite el flujo de información.

Dirección IP: Es un número que identifica, de manera lógica y jerárquica a la interfaz de red de un dispositivo (computadoras, Tablet, Smartphone).

Dominio de colisión: Es un segmento físico de una red de computadores donde es posible que las tramas puedan "colisionar" (interferir) con otros.

Homónimia: Ocultación de la dirección IP utilizando una misma dirección IP igual para cada equipo.

Nodos: Son los diferentes equipos que se encuentran interconectados (computadoras, switchs, routers, servidores) para que los paquetes o mensajes fluyan a través de la red.

Túneles: Es una técnica que consiste en encapsular un protocolo de red con la finalidad de transportar dicho protocolo a través de la red, creando una red privada virtual.

Tráfico: Detección de flujo de información o paquetes que se transmite por la red utilizando un protocolo. El tráfico puede ser gestionado utilizando un sniffer como Wireshark.

.

CAPÍTULO I

PROBLEMÁTICA

1.1 Descripción del problema.

La importancia de la comunicación anónima surge desde las revelaciones sobre el espionaje de la Agencia Nacional de Seguridad de EE.UU, donde Edward Snowden un exempleado de la CIA (Central de Inteligencia Americana) asegura que existía un programa de vigilancia de las comunicaciones tanto en los dispositivos móviles como en equipos computacionales.

Otro caso es el Julian Assange fundador de wikileaks donde es acusado de filtrar información privada del gobierno de los Estados Unidos.

En la actualidad existen un gran número de servicios web ya sean buscadores, tiendas en línea, servicios en la nube entre otros, que garantizan la seguridad de su información pero no garantizan la privacidad a los usuarios, por lo tanto, la mayor parte de los actuales protocolos de seguridad sólo se encargan de proteger los paquetes que viajan por la red, pero no se responsabilizan de ocultar el identificador de red (dirección IP) cuando existe una comunicación entre el emisor y el receptor.

La privacidad es fundamental en la comunicación entre clientes y servidores o en una comunicación de punto a punto (peer to peer), es decir, la comunicación que existe entre clientes o entre servidores, debido a que los equipos computacionales que intercambian información pueden pertenecer a instituciones gubernamentales, financieras o incluso a los usuarios, en donde es indispensable protegerlos de los ciberatacantes.

Las empresas buscan preservar su información debido a que se ha convertido en un recurso valioso, por eso algunas consideran que invertir en seguridad de la información es primordial.

A su vez, las personas que navegan por internet tienen derecho a la privacidad y a la seguridad cuando comparten su información ya sea en redes sociales, mediante chat, videoconferencias, entre otras.

Las grandes corporaciones financieras que gestionan grandes cantidades de información valiosa, tienen la responsabilidad de protegerla de terceras personas que buscan utilizarlas para su propio beneficio.

En internet se puede encontrar una gran cantidad de recursos (manuales, videos, blogs) que muestran cómo vulnerar un sistema o robar información. Los ciberatacantes utilizan estos recursos para aplicarlos en sitios que no se encuentran bien protegidos, dejándolos inutilizables para los usuarios.

Los ciberatacantes no sólo utilizan técnicas como SQL Injection, Cross-site scripting o JavaScript Injection para vulnerar las aplicaciones web, sino que también utilizan técnicas como Spoofing que consiste en suplantar el identificar de red e intervenir en la comunicación de dos equipos para extraer información.

En una sociedad de la información no sólo basta con crear cultura sobre protección de los datos, sino que también se debe estudiar las nuevas formas de cómo extraen

información y aportar con una solución a dicho problema. Es por eso que la implementación de un modelo de comunicación anónima es una forma de brindar seguridad y privacidad a los usuarios, permitiéndoles navegar en la red sin que tenga que preocuparse de ser vigilados o que su información sea mal utilizada.

Con la implementación de comunicación anónima en la red se pretende generar confianza al momento de utilizar un servicio o una aplicación, protegiendo a los usuarios de posibles ataques.

Al realizar la implementación de comunicación anónima en la red, se brindará seguridad a los identificadores de red los cuales pueden ser detectados con un analizador de protocolos.

1.2 Objetivos

Objetivos generales

- Implementar una forma de comunicación que oculte la identificación tanto del emisor como del receptor en el intercambio de información.

Objetivos específicos

- Analizar los diferentes tipos de comunicación anónima que existen actualmente.
- Implementar un algoritmo de comunicación anónima y diseñar un prototipo para comprender el funcionamiento de comunicación anónima.
- Determinar como el prototipo afecta la latencia en el intercambio de información.

CAPÍTULO II

MARCO TEÓRICO

James F. Kurose y Keith W. Ross afirman que la protección de la información que fluye por la red se realiza mediante técnicas criptografías como MD5, SHA, DSA, entre otras. (James F. Kurose y Keith W. Ross, 2010).

El problema de la encriptación es que existe la posibilidad de la intervención de un tercer equipo el cual no permitiría una seguridad adecuada de la información.

Para resolver este inconveniente David Chaum en 1981 propone realizar una combinación entre varios equipos para formar la Mixnet (combinación entre computadoras), la cual es una forma de comunicación anónima y es la base para que muchos autores realicen sus prototipos (Chaum, 1981).

El estudio de la comunicación anónima está tomando mucha relevancia, incluso existen técnicas y protocolos para brindar privacidad y por ende la seguridad a los usuarios, pero existen desventajas como son la disminución del ancho de banda, mayor latencia y la degradación de la usabilidad.

2.2 Técnicas y protocolos para la comunicación anónima.

La comunicación anónima en la red puede definirse como la ocultación de los identificadores de red (direcciones IP) de los equipos computacionales que actúan como emisores o de receptores con el fin de protegerlos de aquellos que quieren interceptar su comunicación para obtener su información.

Según los autores Jian Ren y Jie Wub basándose en las investigaciones de Chaum, existen diferentes técnicas para mantenerse en el anonimato, por ejemplo, Mixnet, Onion Router, Crowd entre otras.

Existen 4 categorías para realizar la comunicación anónima: esquemas basados en routing, basados en peer to peer, esquemas basados en Mixnet y esquema alternativos (Ren y Wub. 2010).

2.2.1 Comunicación basada en routing.

La comunicación anónima basada en routing consiste en elegir el mejor camino para retransmitir los paquetes que fluyen en la red, generalmente se escoge un grupo de nodos para sean los encargados de transmitir la información desde un emisor a un receptor (Ren y Wub. 2010). Este esquema la comunicación anónima se logra debido a que el emisor se agrupa con otros nodos para formar una ruta por donde se transmitirá el mensaje para que llegue al receptor.

En esta categoría se encuentra la técnica de comunicación anónima **Crowds.**

Crowds.

Desarrollado por Reiter y Rubin, su objetivo es proporcionar una forma de privacidad ocultando la identidad de la máquina cuando ésta tenga acceso a la web. (Danezi, 2004).

En la Fig. 1 se puede observar como el nodo 1 y el nodo 5 que pertenecen a un grupo denominado Crowd; los mismos que solicitan a un web server realizar "blending in crowd", que consiste en formar una ruta utilizando el nodo 6 y el nodo 4 o cualquiera de los nodos para retransmitir el paquete para que llegue a su destinatario que es el nodo 2.

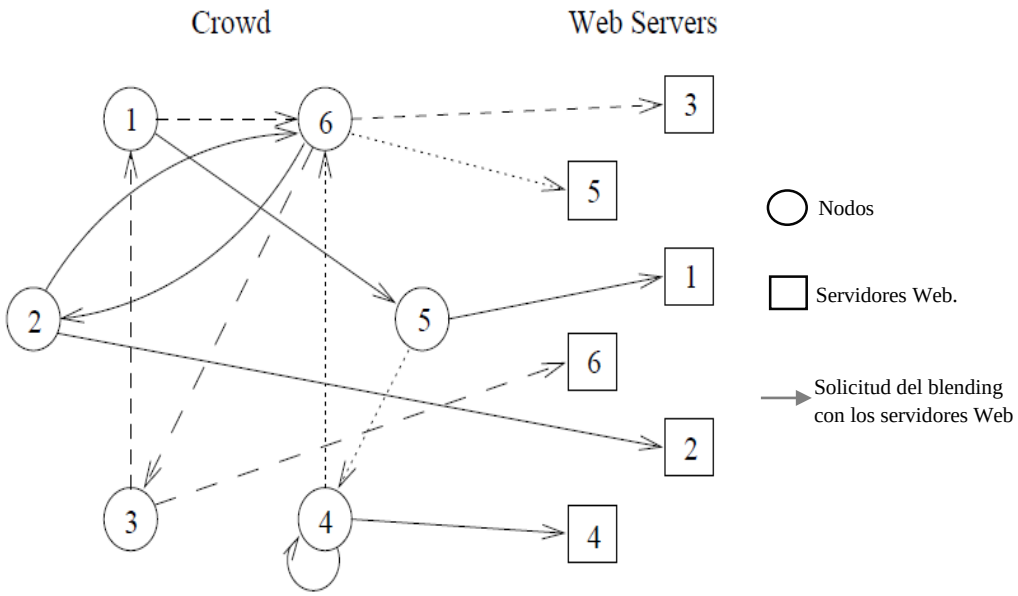

Fig. 1 Funcionamiento de Crowds.

Fuente: Ren y Wub. (2010) Survey on anonymous communications in computer networks. *Computer Communications*, *33*(4), 420-431.

a) Componentes de Crowds.

Crowds está conformada por los siguientes componentes:

- **Web Server:** Este componente es el encargado de seleccionar el grupo de nodos por donde se retransmitirá el mensaje.
- **Crowds:** Son el conjunto de nodos en la red, que serán los encargados de realizar la comunicación anónima.

b) Funcionamiento de Crowds.

Al enviar un mensaje se realiza un proceso conocido como **jondo,** en donde un **Web Server** seleccionará a un grupo de nodos para formar el **Crowds** los cuales serán los responsables de que el mensaje llegue al destinatario sin revelar la identidad de quién lo envió. (Reiter y Rubin, 1999).

c) Fortalezas y Debilidades.

- **Fortalezas**

Una de las principales fortalezas de **Crowds** es su capacidad de proteger a los usuarios de los espías de tráfico de red a través de sus grupos, siendo un grupo encargado de proteger a emisores y receptores.

- **Debilidades**

Un inconveniente que presenta está técnica de comunicación anónima es la administración de una gran cantidad de nodos que formarán los **Crowds,** entre mayor sea el número, mayor será la latencia de paquetes.

2.2.2 Comunicación anónima basada en peer to peer.

Una comunicación anónima peer to peer (punto a punto), consiste en realizar una comunicación directa entre las computadoras de los clientes o entre servidores, que al momento de intercambiar su información, sus identidades serán ocultadas utilizando túneles y el protocolo PNAT o realizando combinaciones y retrasmitiendo el mensaje a través de los túneles.

Tarzan y Morphmix son los protocolos que pertenecen a la comunicación anónima peer to peer.

Tarzan.

Emil Sit y Josh Cates son los creadores de este protocolo, encargado de proveer el anonimato entre clientes y servidores sin que éstos intervengan, utiliza los túneles y el protocolo PNAT. (Sit y Josh, 2001).

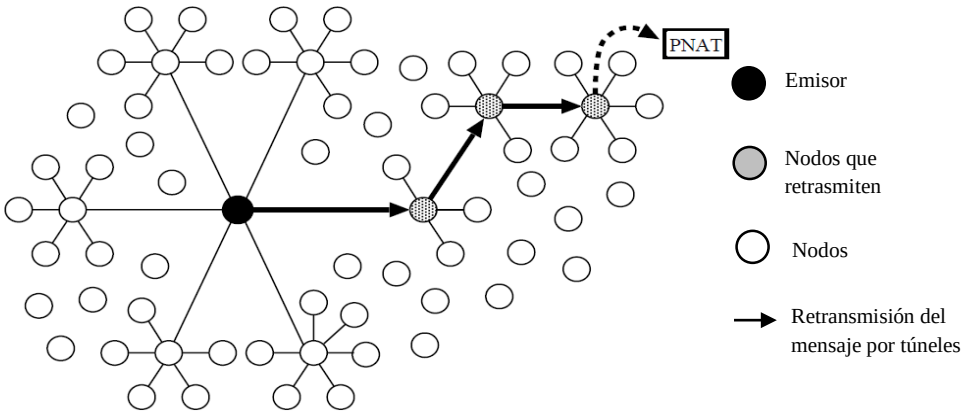

Fig. 2 Protocolo de Tarzan.

Fuente: Freedman, M. J., & Morris, R. (2002). Tarzan: A peer-to-peer anonymizing network layer. In Proceedings of the 9th ACM conference on Computer and communications security (pp. 193-206). ACM.

a) Componentes de Tarzan.

En la Fig. 2 se puede observar que el protocolo Tarzan, está constituido por los siguientes componentes:

- **Túneles:** Son los encargados de encapsular los protocolos con otro protocolo para que posteriormente éste sea transportado a través de la red junto con los paquetes a los diferentes nodos que, en condiciones normales, no lo aceptaría.
- **PNAT:** (Traductor de direcciones y puertos). Protocolo encargado de traducir direcciones de red (de privadas a públicas), y también de traducir los puertos que son las interfaces o medios para comunicarse con un programa a través de una red.
- **Nodos:** Es el conjunto de equipos computacionales, switches, router y todo equipo que conforman una topología de red, los cuales son responsables de que el mensaje sea enviado al destinatario

b) Funcionamiento de Tarzan.

Cuando se envía un mensaje, los nodos utilizan los túneles para retransmitir los mensajes, una vez que llega el mensaje al último nodo, éste utiliza el protocolo PNAT, para traducir la dirección y puerto, es decir que una dirección IP y puerto privado que sirve identificar equipos o dispositivos dentro de una red doméstica, cambia a una pública que es una dirección que sirve para asignar a cualquier equipo o dispositivo conectado de forma directa a Internet.

c) Fortalezas y Debilidades.

- **Fortalezas.**

Entre las principales fortalezas tenemos una comunicación privada utilizando túneles, y la garantía de ocultar la dirección gracias al protocolo PNAT.

- **Debilidades.**

Los túneles utilizan el protocolo UDP, el cual no garantiza la entrega de los paquetes, por lo tanto es una desventaja al momento de retransmitir los paquetes.

Morphmix.

MorphMix es una red combinada (mix network) peer to peer, en los cuales no se distinguen los clientes. En la Fig. 3 se puede observar el sistema que comprende un conjunto de nodos, en donde cada uno de ellos puede establecer un circuito (combinación) utilizando otros nodos para acceder a un servidor de forma anónima.

Basándonos en la Fig. 3 se puede apreciar como los nodos “a”, “b” y “c” se combinan para acceder a “s” que es un servidor.

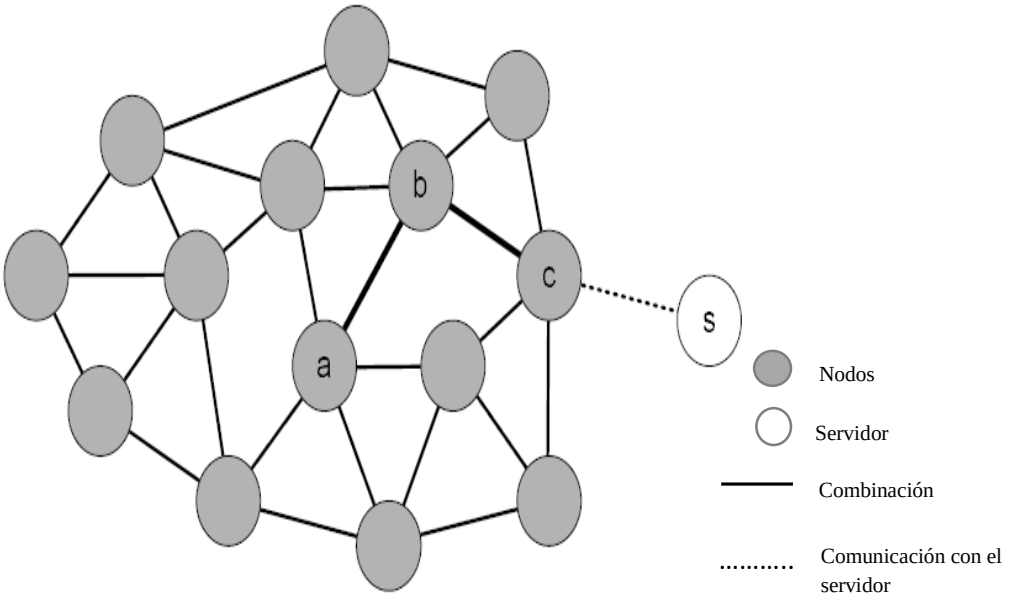

Fig. 3 Idea básica de Morphmix.

Fuente: Rennhard, M. (2004). MorphMix–A Peer-to-Peer-based System for Anonymous Internet Access (Doctoral dissertation, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH).

a) Componentes de Morphmix.

Según Rennhard, el protocolo Morphmix consta de los siguientes componentes (RENNHARD, 2004):

- El **protocolo de configuración del túnel** en donde se utilizará los túneles para el anonimato que permitirá realizar circuitos (combinaciones) para realizar una comunicación con los servidores de forma anónima.
- El **mecanismo de detección de colisión.** Las colisiones se pueden dar por el aumento de número de nodos, los cuales trasmiten información y por ende existe la posibilidad de que dos o más de ellos transmitan a la vez. Para mitigar esa falencia, existe un mecanismo para detectar circuitos que contienen varios nodos operando y verificar que se encuentren en óptimas condiciones.
- El **mecanismo de descubrimiento de puntos** para asegurarse de que los nodos que se van integrando a la topología de red, se agrupen con otros nodos que se encuentran disponibles y detecten nodos corruptos.

b) Funcionamiento de Morphmix.

Primero se identifica los nodos que forman parte de la red que ocultará los mensajes, en la Fig. 4 se puede apreciar el intercambio de llaves públicas y privadas con el fin de evitar nodos corruptos se incorporaren al sistema.

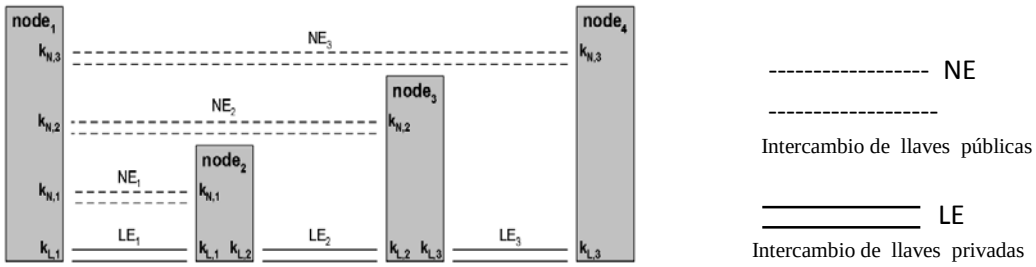

Fig. 4 Intercambio de llaves públicas.

Fuente: Rennhard, M. (2004). MorphMix–A Peer-to-Peer-based System for Anonymous Internet Access (Doctoral dissertation, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH).

Para conectarse de forma anónima se inicializa la comunicación a través de túneles en donde cada nodo elige el túnel para comunicarse.

Una vez comunicado con cada uno de los nodos, éstos son responsables de elegir un camino para retransmitir los paquetes.

c) Fortalezas y Debilidades

- **Fortalezas.**

Entre las fortalezas está la garantía de detección de colisión para que no existan cuellos de botella y la gestión de los diferentes nodos que existen en la red.

- **Debilidades.**

Utiliza el protocolo UDP el cual no garantiza la entrega de los paquetes, por lo tanto es una desventaja al momento de retransmitir los paquetes.

2.2.3 Comunicación anónima basada en Mixnet

La comunicación anónima basada en esquemas Mixnet consiste en crear una comunicación entre servidores proxys (equipo intermedio que representa a un equipo cliente en el sistema de comunicación anónima), para poder retransmitir los mensajes los cuales se encuentran encriptados y a medida que vaya transmitiéndose los mensajes por la red cada nodo lo va desencriptando utilizando las llaves públicas y privadas.

Entre las principales técnicas de comunicación anónima basada en Mixnet tenemos: **Onion Routing, Web Mixes, Mixminion.**

Enrutamiento a través de capas.

Desarrollado por Reed, Syrveson y Goldschlag, su principal función es empaquetar el mensaje a través de capas (mensaje encriptado con llave pública) por cada router que circule. Para descubrir el destinatario al cual se tiene que enviar el mensaje, cada nodo

debe utilizar la llave pública y privada para poder leer el mensaje original. (Syrveson, Goldschlag. 2004)

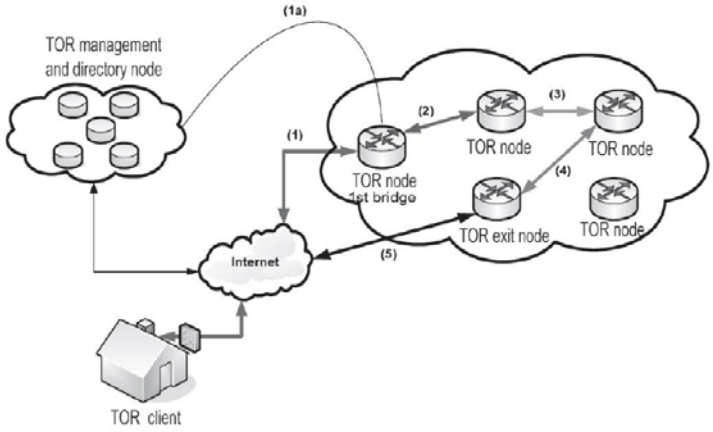

Fig. 5 Funcionamiento de Tor utilizando el algoritmo a través de capas

Fuente: Reed, M. G., Syverson, P. F., & Goldschlag, D. M. (1998). Anonymous connections and onion routing. Selected Areas in Communications, IEEE Journal on, 16(4), 482-494.

a) Componentes del enrutamiento a través de capas.

Según Reed, Syrveson y Goldschlag y como se puede observar en la Fig. 5**,** ésta técnica está conformada por los siguientes componentes:

- Un **primer Nodo** (1st bridge), el cual es el encargado de administrar y seleccionar los nodos mediante un algoritmo, estos serán los encargados de ocultar el mensaje.
- Los **nodos intermedios** (TOR node) los cuales son los responsables de garantizar de que exista comunicación anónima, generalmente son el conjunto de routers que pertenecen a la topología de red. Son los responsables de retransmitir el mensaje.

Un **nodo final** (exit node), el cual será el encargado de enviar el mensaje al destinatario.

b) Funcionamiento del enrutamiento a través de capas.

Se establece una conexión con el primer nodo, donde se encarga de encriptar la dirección y los datos utilizando los algoritmos de encriptación (Diffie-Hellman, Funciones Hash, etc).

Al realizar el proceso de encriptación, se elige los nodos que participarán en el encaminamiento para enviar los paquetes

Los nodos que se eligieron, serán los encargados de desencriptar o quitar la capa utilizando sus llaves públicas y privadas para finalmente enviar el mensaje al destinatario. Estas llaves son intercambiadas vía TLS que es una seguridad de la capa

de transporte brindando autenticación y privacidad de la información mediante el uso de criptografía.

c) Fortalezas y Debilidades

- **Fortalezas.**

Entre las principales tenemos la garantía de ocultar los mensajes a través de algoritmos de encriptación, y una gestión de los diferentes nodos los cuales serán los encargados de hacer fluir la información.

- **Debilidades.**

Mayor tiempo de llegada de los paquetes a su destino debido al procesamiento que se realiza tanto para encriptar como para ocultar la dirección IP.

Web Mixes.

Basándose en las investigaciones de Berthold, Federrath, Köpsell, esta técnica de comunicación anónima genera una gran cantidad de tráfico utilizando "dummy messages" que son considerados como mensajes falsos; los cuales ocultan el verdadero mensaje desde el emisor hacia el receptor. (Berthold, Federrath y Köpsell, 2001)

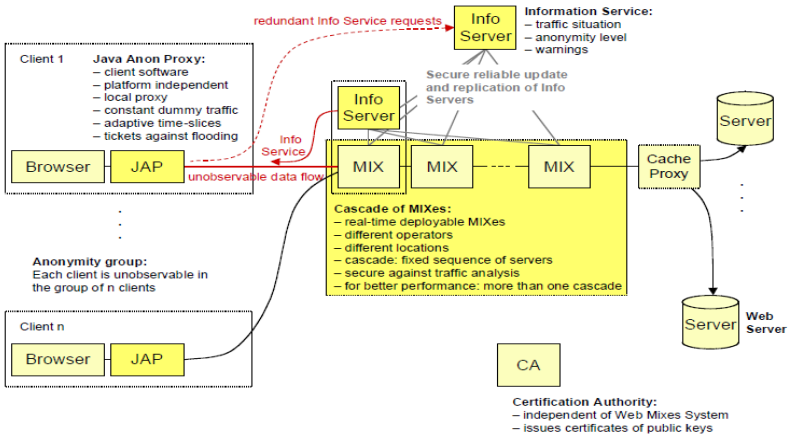

Fig. 6 Web Mixes y sus componentes.

Fuente: Berthold, O., Federrath, H., & Köpsell, S. (2001, January). Web MIXes: A system for anonymous and unobservable Internet access. In Designing Privacy Enhancing Technologies (pp. 115-129). Springer Berlin Heidelberg.

a) Componentes de Web Mixes.

Como se puede apreciar en la Fig. 6, esta técnica de comunicación anónima contempla tres partes lógicas:

- **Java Anon Proxy (JAP).-** Es una aplicación desarrollado en el lenguaje de programación Java, el cual se comunica con los **MIXES** para solicitarles que oculten el mensaje, permitiendo así acceder a la web de forma anónima.
- **MIXES.-** Son los conjuntos de nodos (router, switches, servidores) que al recibir la solicitud de la aplicación JAP, éstos se encargan de ocultar el mensaje, brindado así la comunicación anónima.
- **Servidores.-** Son los encargados de guardar información en un cache proxy que sirve para conservar el contenido solicitado por el usuario, para posteriormente utilizarla y generar "dummy messages" (mensajes falsos que ocultan al verdadero).

b) Funcionamiento de Web Mixes.

Se utiliza el programa JAP (Java Anon Proxy) que es encargado de solicitar y registrar al usuario en el grupo que conforman los Mixes.

Una vez realizada la solicitud para enviar un mensaje, los **Mixes** utilizan la información almacenada en la **cache proxy,** y realizan el proceso de generación de "**dummy messages**", que consiste en crear una gran cantidad de mensajes con dicha información generando un gran tráfico, con la finalidad de que los atacantes no detecten el verdadero mensaje cuando se retransmita por la red.

c) Fortalezas y Debilidades

- **Fortalezas.**

Una de las fortalezas de Web Mixes es que permite la cantidad de flujo de información el cual no permite a los atacantes identificar el mensaje.

- **Debilidades**

La degradación de la usabilidad, debido al gran proceso que se realiza. Además de una mayor latencia por el tráfico que se crea al momento de enviar un mensaje.
Existe la posibilidad de realizar un filtro de los mensajes y así el atacante puede descubrir el mensaje.

Mixminion

Pese a no ser una técnica para ocultar direcciones IP, Mixminion es analizado en este apartado debido a que es necesario estudiar las técnicas, procesos y algoritmos de ocultación de información que usa.

Este protocolo de comunicación pertenece al grupo de protocolos anónimos denominados "remailer", sus predecesores son Mixmaster y Cypherpunk, los cuales son los encargados de enviar los emails sin revelar sus remitentes.

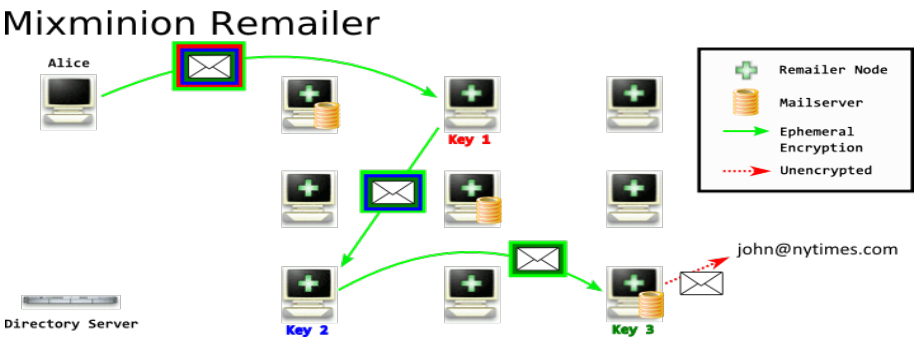

Fig. 7 Mixminion.

Fuente: Ritter, T Remailers We've Got. (2013) Recuperado de https://crypto.is/blog/remailers_weve_got

a) Componentes de Mixminion.

Como se puede observar en la Fig. 7, esta técnica consta de un **conjunto de nodos** (Remailer node) los cuales son servidores encargados de encriptar los mensajes.

- Un **servidor principal** (Mail Server)**,** encargado de enviar un mensaje al destinatario.
- El protocolo **SMTP,** el cual es el medio por donde se enviará los mensajes.

b) Funcionamiento Mixminion.

Como todos los servidores de correo éste utiliza el protocolo SMTP para enviar un mensaje, pero con la diferencia de que con las llaves públicas y privadas modifica las cabeceras de los bloques de información del mensaje para no revelar al remitente. Como ejemplo podemos ver en la Fig. 7**,** La remitente Alice es remplazado por Jhon, brindando así el anonimato.

c) Fortalezas y Debilidades

- **Fortalezas.**

Una fortaleza que se puede encontrar en este tipo de comunicación anónima es la encriptación de información que realizan los servidores.

- **Debilidades.**

Una desventaja que presenta es que sólo sirve para servidores de correo electrónico.

2.2.4 Comunicación anónima basada en esquemas alternativos.

Existen otras formas de comunicación anónima que se basaron en técnicas o protocolos que ya fueron diseñados con anterioridad, pero que fueron mejorados para mitigar el problema de latencia, o la disminución del ancho de banda.

Hordes.

Según Brian Neil Levine y Clay Shields, este protocolo de comunicación anónima, también está compuesto por un grupo de servidores proxys al igual que Crowds, con la diferencia que se utilizan las llaves públicas y se trasmite vía multicast. (Levine, Shields, 2002).

a) Componentes de Hordes.

- **Iniciador:** es un nodo que se encarga de pedir al grupo de nodos que se agrupen para formar una ruta y ocultar el mensaje que se envía desde un emisor a un receptor y así ofrecer el anonimato.
- **Hordes:** conjunto de nodos en la red por donde circulará el mensaje y a la vez ocultará el identificador de red tanto de emisores como de receptores. Al igual que Crowds el emisor se asocia con otros nodos para formar una ruta y enviar un mensaje al receptor.
- **Nonce:** es un identificador que sirve para incorporar a los nodos al sistema y así evitar nodos falsos que pueden ser ciberatacantes.

b) Funcionamiento de Hordes.

Existe un iniciador que envía una solicitud al servidor para unirse al grupo de Hordes.

Una vez que el servidor recibe la solicitud, éste responde con una firma al iniciador, donde se envía un *nonce* (identificador) para que el nodo se pueda incorporar al sistema.

Los nodos que pertenecen a Hordes actualizan constantemente el *Nonce* para evitar infiltración de posibles atacantes.

Finalmente, se envía cada mensaje a través de multicast y UDP en la capa de transportes. Se envía también las llaves públicas y privadas, para que posteriormente el mensaje sea desencriptado utilizando las llaves conocidas por el destinatario.

c) Fortalezas y Debilidades.

- **Fortalezas.**

Entre sus fortalezas está la identificación de nodos corruptos, es decir aquellos nodos que no pertenecen a la red, por ejemplo equipos de atacantes informáticos.

- **Debilidades.**

Utiliza el protocolo de transporte UDP, el cual no garantiza la entrega de paquetes.

P5 (Protocolo de privacidad personal punto a punto).

Es una comunicación peer to peer donde se construye un árbol binario con los nodos que conforman la red para enviar un mensaje a través de broadcast en donde la raíz o el padre es un emisor y uno de los hijos es el receptor dependiendo de la jerarquía.

a) Componentes de P5

- P5, consta solo de un **conjunto de nodos,** los cuales formarán un árbol en donde será transmitido el mensaje.

b) Funcionamiento

El procedimiento para obtener el anonimato con P5 se indica en la Fig. 8, se crea un árbol binario con los nodos que conforman la red, el paquete se envía a través del canal hasta que llegue al respectivo nodo receptor. Los nodos que conforman el árbol retransmiten el mensaje a través de broadcast.

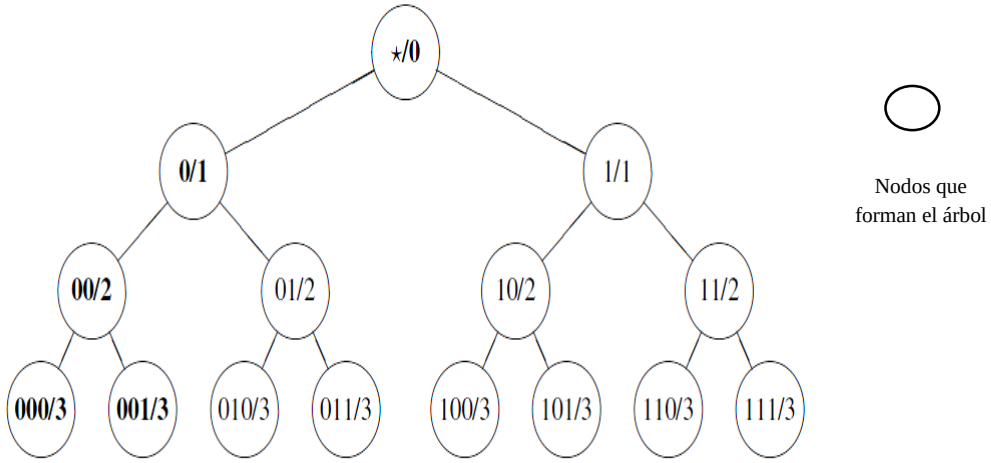

Fig. 8 Envío de un mensaje utilizando P5.

Fuente: Srinivasan, R. S. B. B. A. (2002). P5: A Protocol for Scalable Anonymous Communication. IEEE Security and Privacy**.**

c) Fortalezas y Debilidades.

- **Fortalezas.**

Envío de paquetes a través de broadcast que permitirá disminuir el tiempo de retardo en donde los mensajes llegarán de una manera más rápida.

- **Debilidades.**

Una de las debilidades que presenta esta técnica es la de formar una jerarquía, debido a que algunos esquemas de red no permiten realizarla, por ser topologías muy complejas.

2.3. Comparación entre las técnicas de comunicación anónima.

En la tabla 1 se puede observar las diferentes técnicas y protocolos en donde se ha tomado como atributos la latencia, capa en la que se implementa, la confiabilidad y la usabilidad, para realizar la comparación entre las mismas y conocer sus características.

Tabla 1 Comparación entre las técnicas y protocolos

Comparación entre técnicas y protocolos	Latencia cualitativa	Capa de implementación	Confiabilidad en la llegada del paquete	Usabilidad
CROWDS	Media	Aplicación	Media	Media
TARZAN	Media	Aplicación	Baja	Baja
Morphmix	Media	Aplicación	Media	Baja
Onion Routing	Media	Aplicación	Alta	Alta
Web Mixes	Alta	Aplicación	Baja	Baja
Hordes	Baja	Transporte	Media	Media
P5	Baja	Transporte	Media	Media

Fuente: Autor.

Como se puede apreciar en la Tabla 1, algunas de las técnicas presentan una alta latencia debido al tiempo de procesamiento de la técnica, lo que conlleva a que exista un retardo en la llegada del paquete. Otras técnicas infieren en la usabilidad debido a su forma de entregar el mensaje.

CAPÍTULO III

ANÁLISIS PARA EL PROTOTIPO DE COMUNICACIÓN

Para implementar un prototipo y comprender el funcionamiento interno del mismo, se ha seleccionado dos técnicas de anonimato y se ha realizado el análisis de las mismas describiendo sus pseudocódigos. Además en este apartado se hace una introducción del prototipo a implementar.

3.1 Algoritmos de comunicación anónima.

Conociendo las diferentes técnicas y protocolos que existen actualmente y utilizando la tabla de comparación (Tabla 1) del Capítulo 1, se determinó que Crowds y Onion Routing son los algoritmos más idóneos en el contexto de este trabajo de titulación, razón por la cual, serán analizados para determinar y a posterior poder implementar un prototipo que brinde comunicación anónima en la red.

Los pseudocódigos de los algoritmos que se ha seleccionado son los siguientes:

3.1.1 Algoritmo de enrutamiento a través de capas.

La Fig. 9 muestra los componentes del algoritmo de Onion Routing; en donde nos basaremos para describir el pseudocódigo.

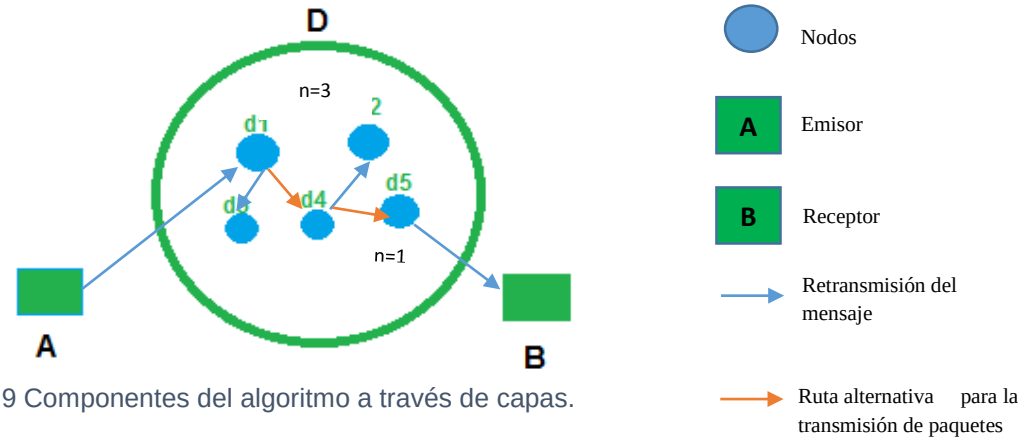

Fig. 9 Componentes del algoritmo a través de capas.

Fuente: Autor.

1. Siendo d1, d2, d3, d4, d5 los nodos en la red (D), y A emisor y B el receptor.
2. Emisor A debe enviar mensaje a un nodo principal del conjunto de D que será elegido aleatoriamente.
3. Nodo principal que pertenece a D, es decir d1 inicializará aleatoriamente un número, el mismo que indicará el número de nodos que debe atravesar el mensaje.
4. Mientras el mensaje circule por los nodos en la red, decrementar el número de nodos que se seleccionaron.
5. Si n=1 es el último nodo será el encargado de enviar al receptor.

3.1.2 Algoritmo de Crowds.

Este algoritmo, como se puede ver en la Fig. 10, funciona de la siguiente manera:

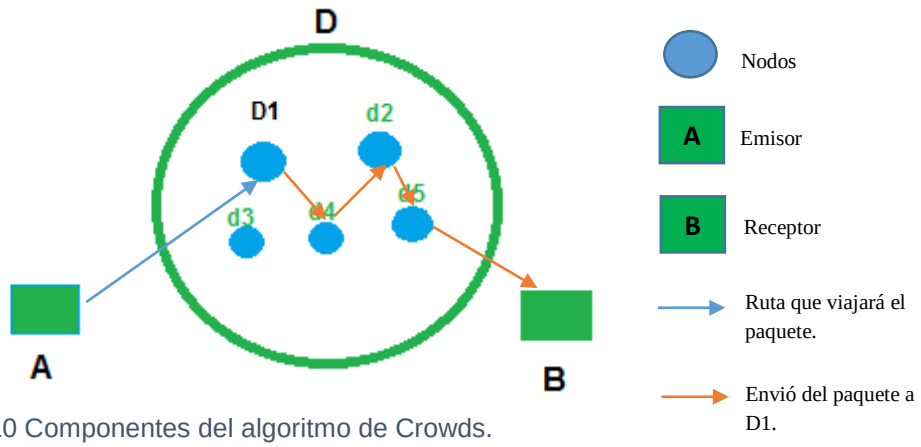

Fig. 10 Componentes del algoritmo de Crowds.

Fuente: Autor.

1. Siendo D1 administrador de los nodos.
2. D1 escoge aleatoriamente un número de nodos (D).
3. D1 forma la ruta para retransmitir el paquete utilizando los nodos que conforman la red.
4. Finalmente se envía confirmación de mensaje entregado a D1, utilizando la ruta.

3.2 Comparación y selección del algoritmo a implementar.

Al realizar la comparación de los algoritmos de enrutamiento a través de capas y Crowds, se puede apreciar una diferencia que existe entre los dos, y es la forma de cómo retransmiten el mensaje. En el primer algoritmo cada nodo es responsable de enviar el mensaje y en el segundo un grupo de nodos es responsable de retransmitir el mensaje.

No se usará Crowds para desarrollar el prototipo debido a que existe una desventaja cuando uno de los nodos se encuentre inactivo, el cual no permitirá enviar el mensaje ya que los nodos son una parte fundamental porque forman la ruta por donde retransmitirán el mensaje tal como se puede ver en la Fig. 10.

En cambio, si utilizamos el algoritmo de enrutamiento a través de capas (Onion Routing), se tendría una garantía de que el mensaje llega al destinatario debido a que cada nodo cuenta con una ruta alternativa como se puede apreciar en la Fig. 9 para retransmitir el mensaje cuando uno de los nodos se encuentre inactivo.

Por lo tanto para el desarrollo de la aplicación se utilizará el algoritmo de enrutamiento a través de capas debido a que garantiza la entrega del mensaje.

3.3 Explicación del prototipo de comunicación anónima.

Para comprender de manera general el funcionamiento de una comunicación anónima en la red, se ha desarrollado un prototipo que utiliza el pseudocódigo de onion routing, sin considerar la encriptación ni las llaves públicas y privadas. En la Fig.11 se puede apreciar el funcionamiento esperado del prototipo.

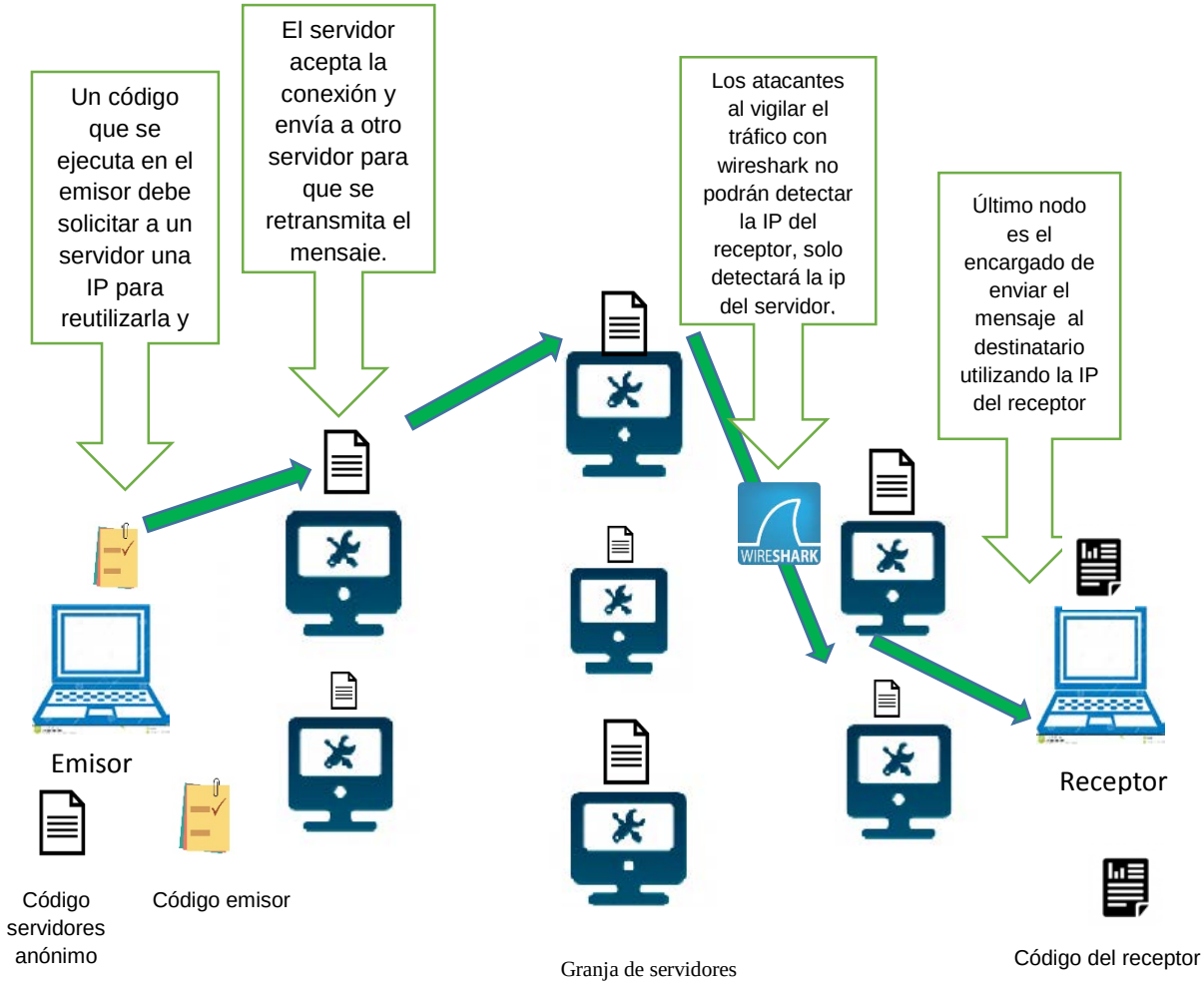

Fig. 11 Funcionamiento del prototipo de comunicación anónima

Fuente: Autor.

Como se puede apreciar en la Fig.11 el sistema de comunicación anónima, está conformado por una granja de servidores que ejecutan un código específico. El emisor utiliza la granja de servidores para enviar mensajes al receptor.

El emisor tiene una interface (código) para intercambiar mensajes con la granja de servidores. El anonimato se logra dentro la granja de servidores al cambiar la dirección IP del emisor y receptor en el mensaje mientras éste transita por la granja. Cada servidor que recibe el mensaje, cambia la dirección de origen con su propia dirección y selecciona aleatoriamente la dirección de destino de una pila de direcciones (direcciones de los equipos de la granja de servidores). El número de servidores, *m*, que debe pasar el mensaje es generado aleatoriamente. Cada vez que el mensaje para por un servidor el valor de *m* se decrementa en 1. Cuando *m* = 1 el servidor correspondiente entrega el mensaje al receptor. El receptor envía un mensaje de confirmación al emisor siguiendo el mismo proceso. En el capítulo IV se habla con más detalle el funcionamiento del prototipo.

CAPÍTULO IV

REQUERIMIENTOS, DESARROLLO Y FUNCIONAMIENTO DEL PROTOTIPO DE COMUNICACIÓN ANÓNIMA.

4.1 Metodología de desarrollo ágil Scrum.

Para diseñar el prototipo se ha seleccionado como Metodología de desarrollo a SCRUM, debido a que permite desarrollar el prototipo por fases; también permite implementar mejoras en cada fase.

Scrum es una forma de desarrollar los proyectos de software de una manera ágil y como se puede ver en la Fig. 12 está conformada por los siguientes componentes:

Fig. 12 Metodología Scrum

Fuente: Implementando Scrum con Team Foundation **2012 Recuperado de:** https://roanboc.wordpress.com/2013/12/23/implementando-scrum-con-team-foundation-2012/

Product Back-log. En esta fase se describe los requerimientos del sistema por parte del dueño del producto (Product Owner).

Planificación de Sprint. En esta fase se determina el orden que va a ser desarrollado los requerimientos de un sistema, además del tiempo que conlleva desarrollar previo un análisis.

Sprint Back-Log. Una persona denominada Scrum Master es el encargado de colocar en una tabla los requerimientos y las especificaciones que debe cumplir el sistema, según el orden que se dio en la planificación. Para ello existe una reunión con los programadores.

Desarrollo del Sprint. Los programadores comienzan a desarrollar la aplicación. Al momento que terminan, hace la entrega del sprint, que es un prototipo con sus respectivas funcionalidades según los requerimientos del cliente.

Revisión y estado del proyecto. El usuario comprueba de que la aplicación cumpla con lo requerido.

4.2 Requerimientos de la aplicación

Primero se elabora el listado de requerimientos (**Product Bakclog)** el cual consiste en describir los requerimientos de la aplicación como se indica en la Tabla 2 los cuales son los siguientes:

Tabla 2 Requerimientos del prototipo.

No	Requerimiento	Prioridad
1	Un servidor debe asignar una dirección IP para que un cliente la reutilice y pueda enviar un mensaje.	Media
2	Un servidor debe gestionar los nodos guardando todas las direcciones de los nodos que están disponibles además los números de nodos.	Alta
3	Todos los servidores de comunicación anónima (nodos) deben reportar la dirección IPv4 con la que se encuentran identificados en la red.	Media
4	El primer servidor (nodo) que forman parte de la comunicación debe inicializar el número de nodos que atravesará el mensaje.	Alta
5	El último servidor debe ser el encargado de enviar el mensaje al receptor en caso de no ser el último debe elegir a otro nodo para retransmitir el mensaje.	Alta
6	Cuando el mensaje atraviesa por cada servidor de comunicación anónima, debe haber un decremento del número de nodos a partir de la inicialización. El anonimato se verificará con el analizador de protocolos.	Media

Fuente: Autor

4.3 Alcance del prototipo.

1. El prototipo será implementado en un dominio de broadcast. Para que el sistema de comunicación anónima propuesto funcione en Internet o para cualquier red los enrutadores deberán tener implementado el código del prototipo dentro de su sistema operativo. La inclusión del código esta fuera del alcance del presente trabajo de fin de titulación.
2. Se ha definido que el valor de *m* (número de servidores que debe atravesar el mensaje dentro de la granja) sea igual al número total de servidores de la granja (j) menos 1. $m = j - 1$, ya que se ha considerado el tiempo de transmisión y la posible congestión que se puede generar en la granja de servidores.
3. Un servidor de la granja de servidores es el responsable de notificar y gestionar los servidores que entran y dejan el sistema. Lleva un registro de la direcciones

y del estado de cada uno. Dentro de sus funciones está compartir esta información con el resto de servidores. El gestor de servidores es un posible punto de fallo del prototipo, ya que si este equipo es vulnerado o queda fuera de servicio, la información de la granja de servidores y el servicio de comunicación anónima dejarían de funcionar. Como trabajo adicional se propone una mejora al gestor la cual involucra redundancia y encriptación para el gestor de servidores.

4.4 Desarrollo de la aplicación

Para desarrollar cada uno de los requerimientos se divide en **sprint back-log** el cual consiste en analizar la manera de cómo se va a desarrollar la aplicación para cumplir con los requerimientos para lo cual tenemos los siguientes **sprints.**

4.4.1 Sprint 1: Desarrollo del código servidor que gestiona los nodos.

Este código abarcará los requerimientos 2, 3 y 4 de la tabla 2, y estará compuesto por tres componentes:

- El primer componente es un socket servidor el cual espera el mensaje que va a retransmitir enviándole a un socket cliente.
- El segundo componente es un socket cliente el cual se comunica con otro nodo el cual es un socket servidor que espera un mensaje.
- El tercer componente es un socket servidor el cual se le denominará “Asignador” cuya función consiste en dar una dirección aleatoria para que se utilizada por el cliente para enviar su mensaje de forma anónima. El servidor socket y cliente socket trabajarán con el puerto 9400 y el Asignador con el puerto 8800.

Este sprint comprende la Fase 1 y la Fase 4 del esquema de trabajo, el código se encuentra desde el anexo 3 hasta el anexo 6 y su forma de funcionamiento se lo puede ver en la Fig. 13

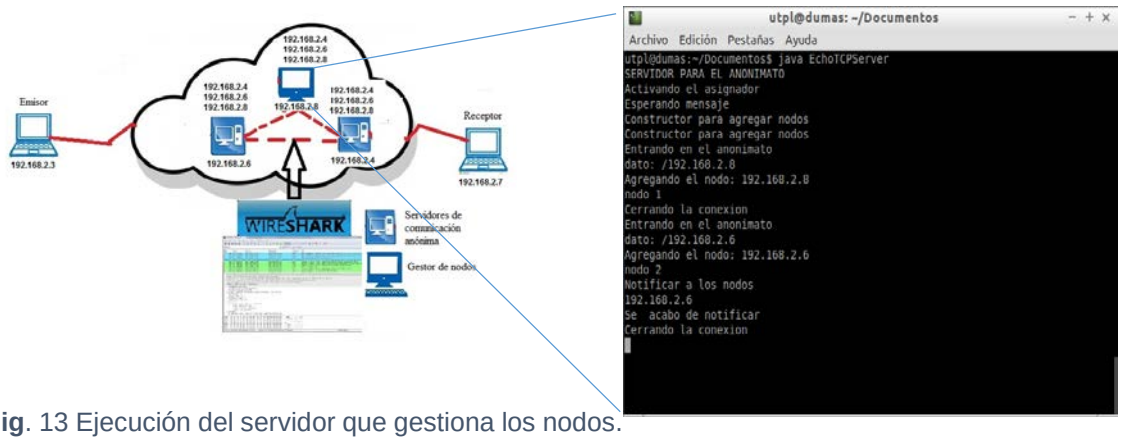

Fig. 13 Ejecución del servidor que gestiona los nodos.

Fuente: Autor.

4.4.2 Sprint 2: Desarrollo del cliente que solicita la comunicación anónima.

Se abarcará el requerimiento 1 de la tabla se desarrollará de la siguiente manera:

- El cliente primero se comunicará mediante un socket cliente con el socket servidor del gestor de nodos para que le asigne una dirección IP.
- Una vez que tenga la dirección IP de uno de los nodos, mediante una interfaz gráfica solicitará el mensaje a enviar y la dirección del receptor.
- En caso de haber problemas con la comunicación del nodo, este reportará al servidor que gestiona los nodos para que lo elimine.

Este sprint comprende la Fase 3 del esquema de trabajo, el código se lo puede apreciar en el anexo 1 y 2, y su forma de funcionamiento se lo puede ver en la Fig. 14.

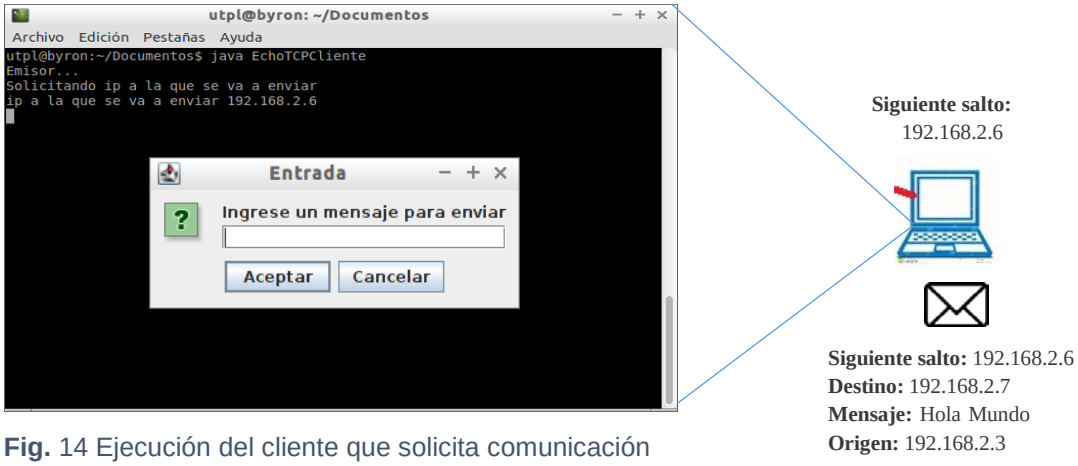

Fig. 14 Ejecución del cliente que solicita comunicación

Fuente: Autor.

4.4.3 Sprint 3: Desarrollo de los servidores que pertenecen a la comunicación anónima.

Se abarcará los requerimientos 5 y 6 de la tabla 2, el código de los servidores se encuentra desde el anexo 7 hasta el anexo 10, la del receptor se encuentra en el anexo 11 y 12, y el funcionamiento será es el siguiente:

- Receptarán el mensaje que el cliente les envié.
- Receptarán un ArrayList con las IPs de los nodos que pertenecen a la comunicación anónima.
- Generarán un número aleatorio dependiendo del tamaño del ArrayList (N) para inicializar **m** que es el número de nodos para retransmitir el mensaje en donde m <=N.
- En caso de ser el último nodo es decir m=1 se enviará al receptor, caso contrario ir decrementando **m.**

Este sprint comprende la Fase 2, la Fase 5 y la Fase 6 del esquema de trabajo, la forma de funcionamiento de los servidores de comunicación anónima se lo puede apreciar en la Fig. 15, y la del Receptor en la Fig. 16.

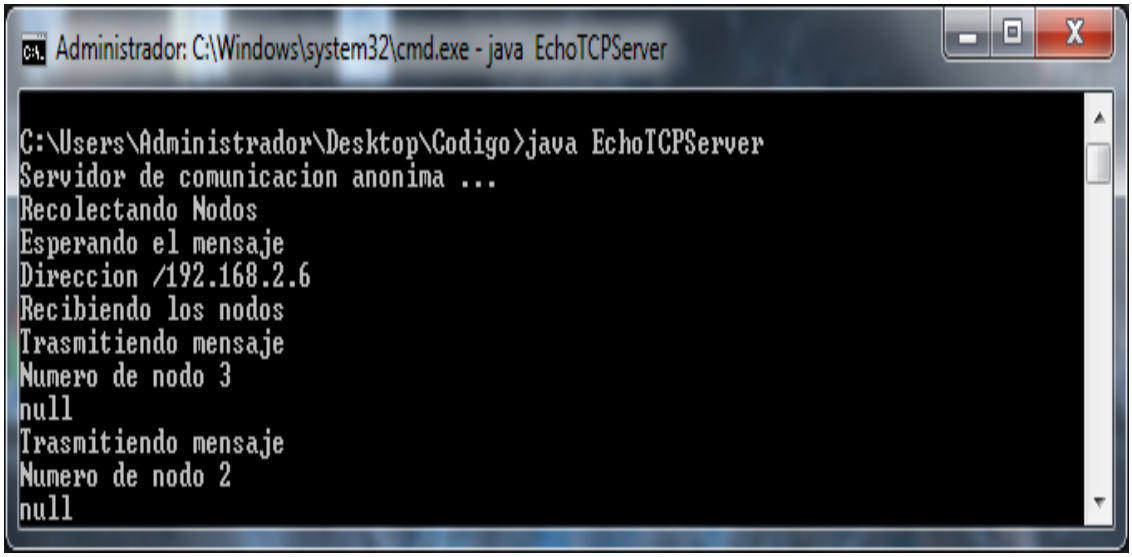

Fig. 15 Ejecución de los servidores.

Fuente: Autor.

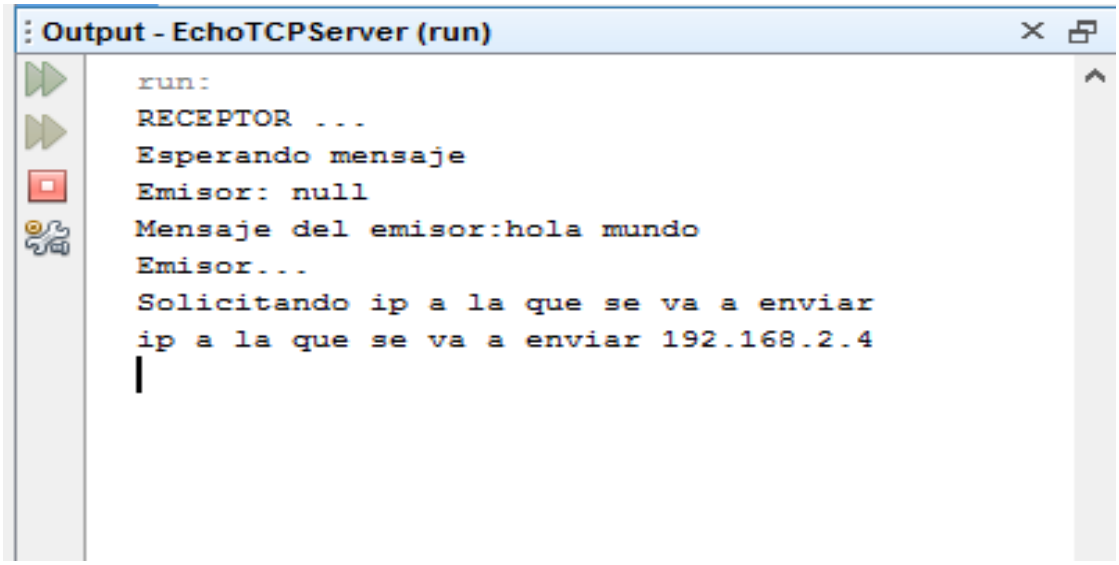

Fig. 16 Llegada del mensaje al Receptor

Fuente: Autor.

4.4.4 Esquema de trabajo

Utilizando la Fig. 17, se indicará los procedimientos que debe cumplir el prototipo de comunicación anónima; la descripción de cada uno de ellos se lo realizará por fases.

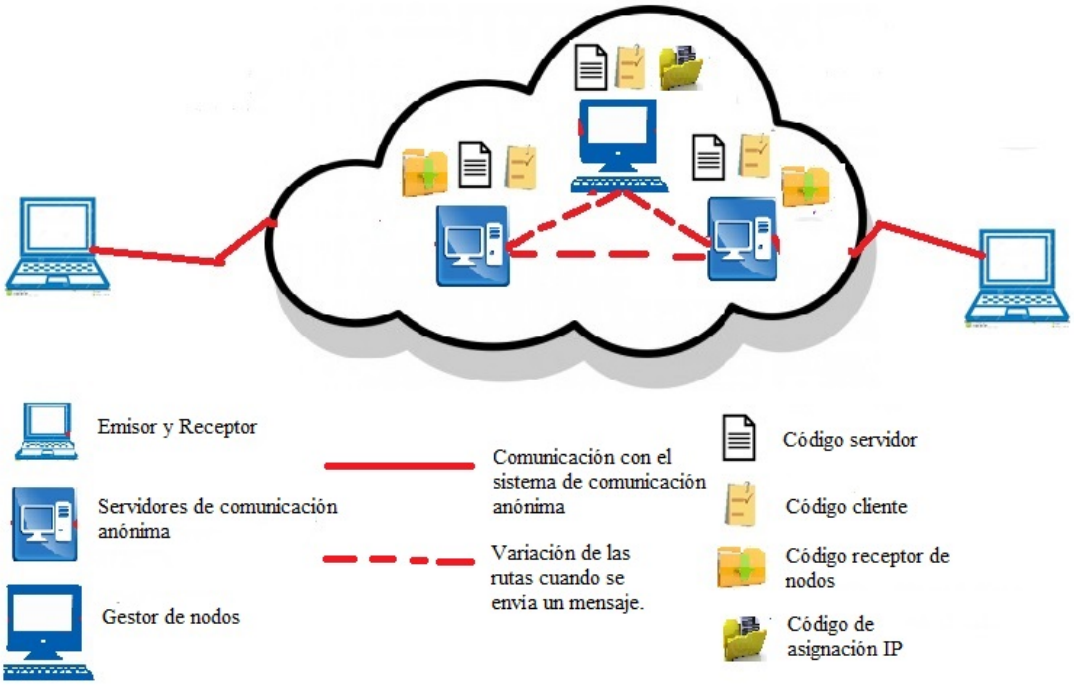

Fig. 17 Esquema de funcionamiento de la aplicación.

Fuente: Autor.

Fase 1 (t1). Gestión de los servidores de comunicación anónima.

En la gestión de los servidores, cuando el gestor de servidores de comunicación anónima se levanta; éste debe guarda su propia dirección IP y esperar las direcciones de todos los servidores que pertenecen al servicio de comunicación anónima.

Fase 2 (t2). Servidores de comunicación anónima.

En la segunda fase, los servidores de comunicación anónima son los responsables de notificar su dirección IP al gestor de nodos, y a su vez esperan la lista direcciones que están disponibles, utilizando el código receptor de nodos (servidores de comunicación anónima).

Fase 3 (t3). Petición de un cliente al sistema de comunicación anónima.

Un cliente a través de una aplicación el cual contiene un código específico, solicita al sistema una dirección IP de uno de los servidores de comunicación anónima, para que éste sea el intermediario para enviar su mensaje junto con la dirección del receptor y espera confirmación de la llegada del mensaje.

Fase 4 (t4). Asignación de la dirección IP.

El gestor de nodos (servidores de comunicación anónima) utilizando su registro de direcciones IP que almaceno en su memoria, el cual es un arrayList que se encuentra

en su código, seleccionará aleatoriamente una de ellas para enviarle al cliente, para que la reutilice y pueda enviar su mensaje.

Fase 5 (t5) Retransmisión de los mensajes.

Cuando el cliente utilice la dirección de un servidor, éste debe retransmitir el mensaje inicializando **m,** que es un número aleatorio de saltos que debe hacer el mensaje antes de que llegue al destinatario en donde **m ≤ N** siendo N el número total de servidores que están disponibles.

Fase 6 (t6) Llegada al receptor.

Cuando el mensaje se retransmite a través de los servidores, m se va decrementando y cuando m = 1 en un servidor, éste será el responsable de enviar al receptor.

4.5 Funcionamiento del prototipo para la comunicación anónima.

En esta sección se describirá de una manera más minuciosa la manera de cómo funciona el prototipo y para verificar que exista la comunicación anónima, se utilizará un software que analiza los protocolos de red (wireshark), donde se observará direcciones IP diferentes, lo que resulta difícil identificar la dirección del emisor o del receptor, cuando se envié un mensaje. A continuación se detallará cada uno de los procedimientos que el prototipo debe cumplir los cuales tenemos:

4.5.1 Notificación y Recepción de direcciones IP.

Basándose en la Fig. 18 se puede apreciar la notificación y la recepción de direcciones de cada uno de los equipos que pertenecen a la comunicación anónima.

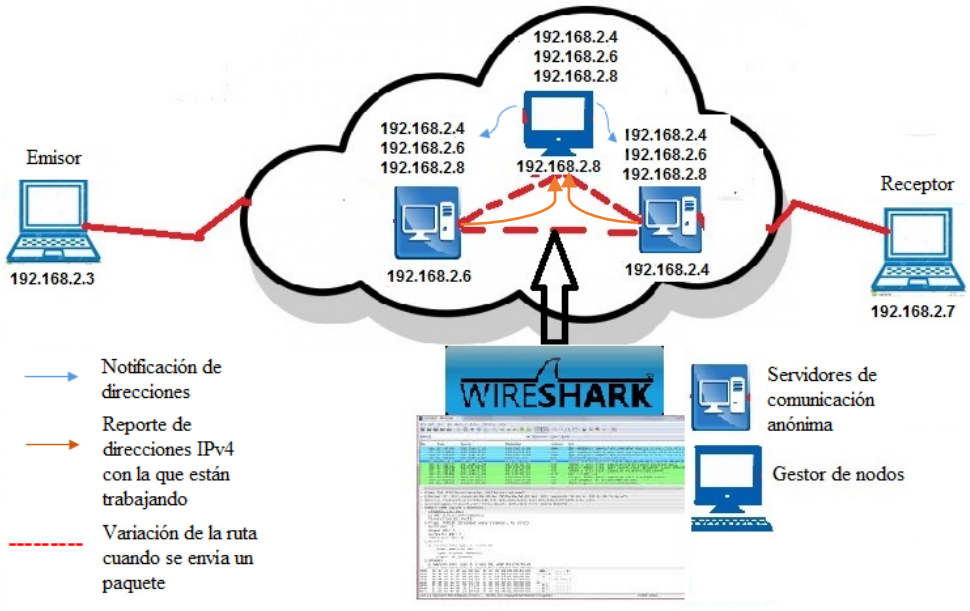

Fig. 18 Escenario de funcionamiento de la comunicación anónima.

Fuente: Autor.

4.5.2 Solicitud de la comunicación anónima

Cuando un cliente solicita una comunicación anónima, éste debe comunicarse con un gestor de servidores el cual contiene un registro de direcciones incluyendo su propia dirección como se puede apreciar en la Fig. 19 en donde se elegirá aleatoriamente una de ellas para posteriormente ser reutilizada.

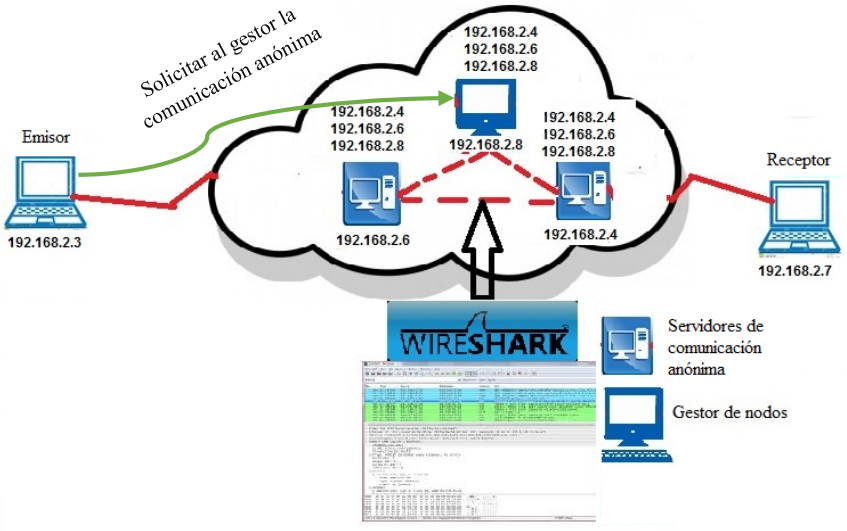

Fig. 19 Solicitud de una comunicación anónima.

Fuente: Autor

4.5.3 Asignación de dirección.

Como se puede apreciar en la Fig. 20 una vez que el cliente tenga una dirección asignada por el gestor de nodos, éste formará un mensaje junto con la dirección destino para finalmente utilizar la dirección asignada y retransmitir el mensaje.

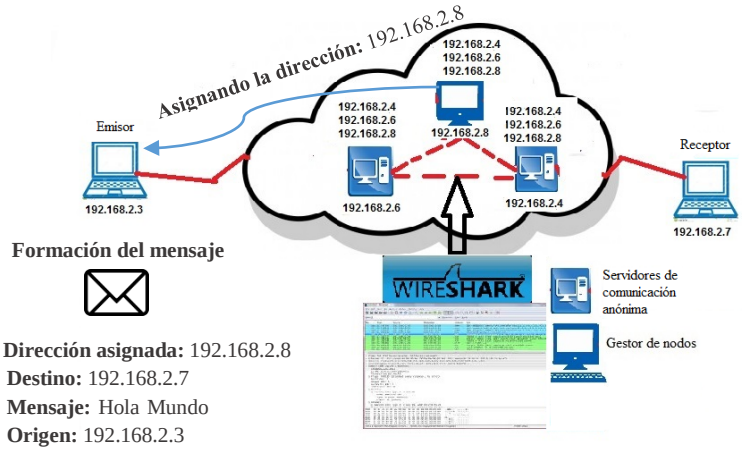

Fig. 20 Asignación de una dirección y formación de un mensaje.

Fuente: Autor.

4.5.4 Retransmisión de mensajes.

Cuando se comienza a retransmitir el mensaje, un servidor de comunicación anónima comienza la inicialización de número de saltos de manera aleatoria, el cual indica el número de equipos que debe atravesar el mensaje antes de ser entregado. En la Fig. 21 se puede apreciar que el servidor con la dirección 192.168.2.8 inicializa el número de saltos en 2 (m=2).

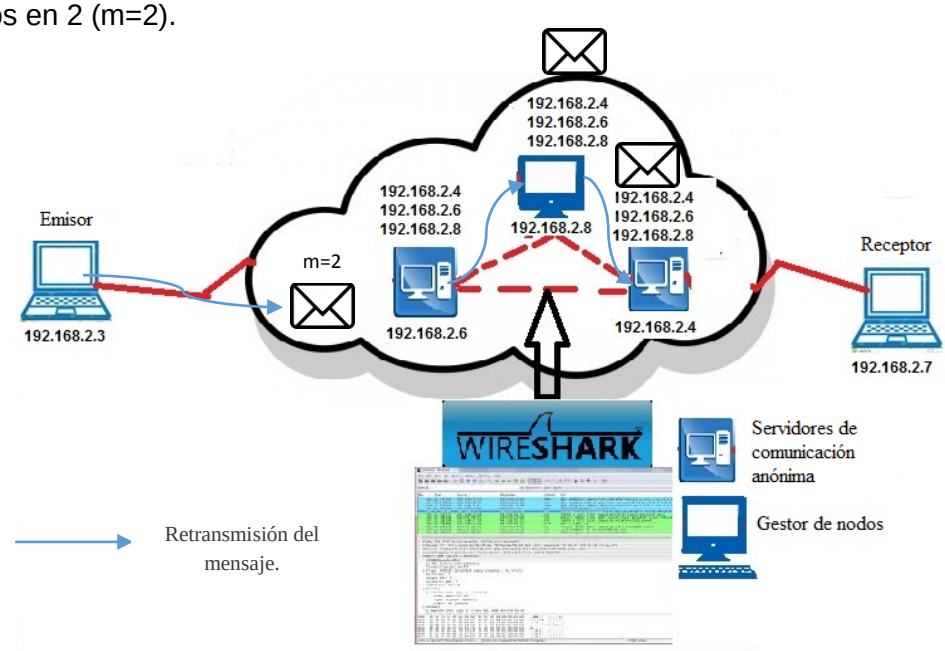

Fig. 21 Retransmisión de mensajes.

Fuente: Autor.

4.5.5 Detección de direcciones aleatorias.

Utilizando un sniffer como Wireshark, cuando se retransmita el mensaje, se detectará una variación de direcciones que pertenecen a los servidores de comunicación anónima. El objetivo es que a través de la variación de direcciones IP y la detección de tráfico, no se pueda identificar la dirección IP del emisor ni la del receptor. En la Fig. 22 se puede observar las direcciones de los servidores de comunicación anónima que utilizan el protocolo TCP para retransmitir el mensaje sin que se detecte la dirección del receptor que es la 192.168.2.7.

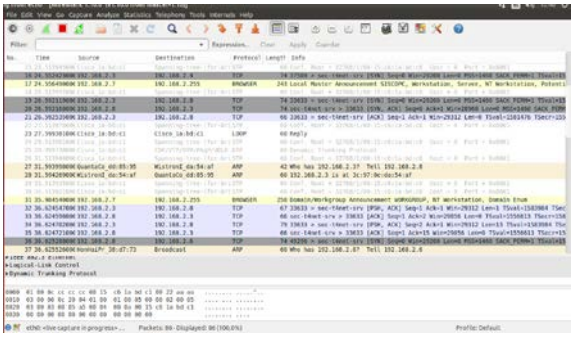

Fig. 22 Detección de las direcciones de los servidores utilizando Wireshark

Fuente: Autor

4.5.6 Envió del mensaje al receptor.

Cada vez que se retransmite el mensaje, el número de saltos se va decrementando y cuando el número llegue a uno es decir m=1, el servidor responsable será el encargado de enviar el mensaje al receptor como se ve en la Fig. 23**.**

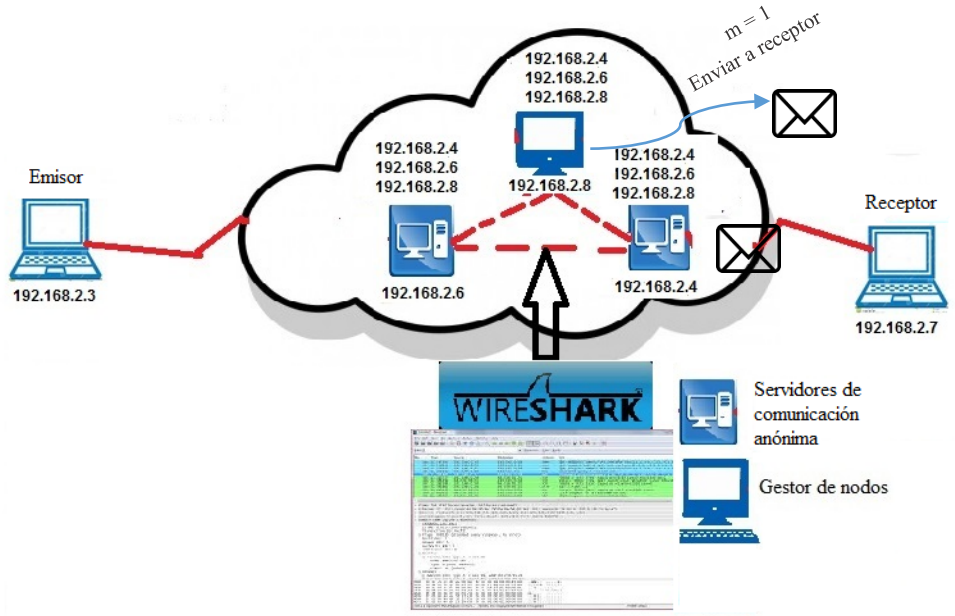

Fig. 23 Llegada del mensaje al receptor.

Fuente: Autor

4.5.7 Confirmación de llegada de mensaje.

Tal como se ve en la Fig. 24 cuando llegue el mensaje al receptor, éste debe confirmar al emisor utilizando nuevamente el sistema de comunicación anónima.

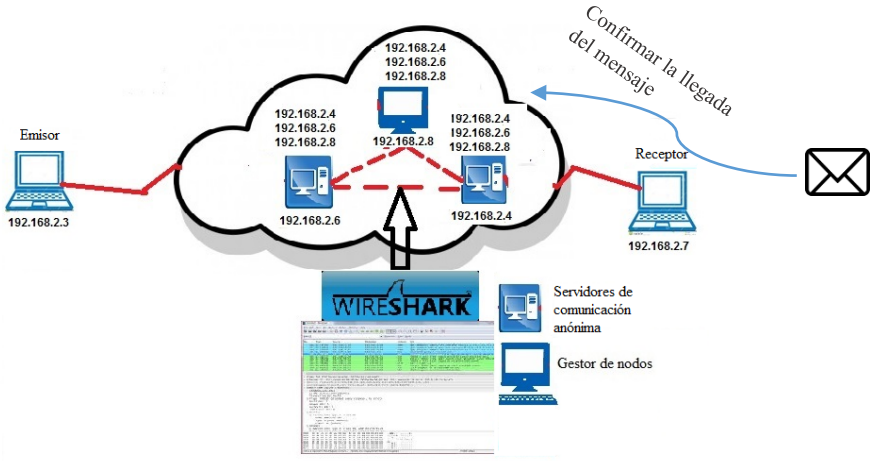

Fig. 24 Confirmación de mensaje utilizando el sistema de comunicación anónima.

Fuente: Autor

CAPÍTULO V

VALIDACIÓN Y DISCUSIÓN DE RESULTADOS.

5.1 Validación.

El prototipo se implementó en un solo dominio de broadcast, simulando una red. Dentro de esta red se colocó equipos para la granja de servidores, también se colocó un equipo emisor y un equipo receptor, como se puede apreciar en la Fig. 11, del capítulo III. Para hacer funcionar el prototipo se debe seguir los pasos que se encuentran definidos en el Anexo 13

Es importante mencionar que todos los equipos tienen un código específico para generar el anonimato. Los equipos en la granja de servidores ejecutan un analizador de protocolos el cual nos permite verificar si existe el anonimato en donde no se debe detectar la dirección IP del emisor.

Debido a que el prototipo imprime cierta carga computacional, hemos definido la latencia como indicador más importante y que será usado dentro de las pruebas de validación del prototipo. Queremos determinar cuánto afecta en tiempo la implementación de nuestros algoritmos de anonimato definido en el prototipo.

La obtención de la latencia se la hizo a través de la información suministrada por la herramienta Wireshark, un analizador de protocolos que permite hacer mediciones del tiempo sobre el tiempo de ida y vuelta de un paquete.

Se realizaron pruebas para cuando los paquetes deben atravesar 1, 2 y 3 servidores, se midió los tiempos con cada escenario. También obtuvimos los tiempos para cuando un mensaje no utiliza el prototipo. En otras palabras no existe anonimato. Finalmente se comparó estos cuatro escenarios y se determinó cuantitativamente como afecta el prototipo a la latencia. En la Tabla 3 se puede ver un resumen de los datos para la implementación y pruebas para cada uno de los escenarios descritos.

Tabla 3 Parámetros que se considera para realizar las pruebas.

ITEM	**Valor**
Cantidad de servidores en la granja	3
Cantidad de saltos que debe atravesar el paquete	1, 2 y 3
Cantidad de mensajes por prueba	10 mensajes
Tamaño del paquete	16 bytes
Dominio de broadcast	1

Fuente: Autor.

5.2 Discusión de resultados.

Para la discusión de resultados se consideró medir los tiempos; para ello se analizará el tiempo cuando no se utiliza el sistema de comunicación y cuando el sistema de comunicación anónima emplea 1, 2 y 3 saltos para la transmisión de los mensajes.

5.2.1 Latencia promedio sin prototipo

Los mensajes llegarán del emisor al receptor sin utilizar el sistema de comunicación anónima. Los resultados de la latencia promedio se pueden observar en la Tabla 4 y están representados gráficamente en la Fig. 25.

Tabla 4 Resultados obtenidos sin el prototipo

Latencia sin prototipo	
Paquete	**RTT ms**
1	3,2
2	4,5
3	2,4
4	3,67
5	1,15
6	2,34
7	3,45
8	2,04
9	1,93
10	4,16
Promedio	2,884

Fuente: Autor.

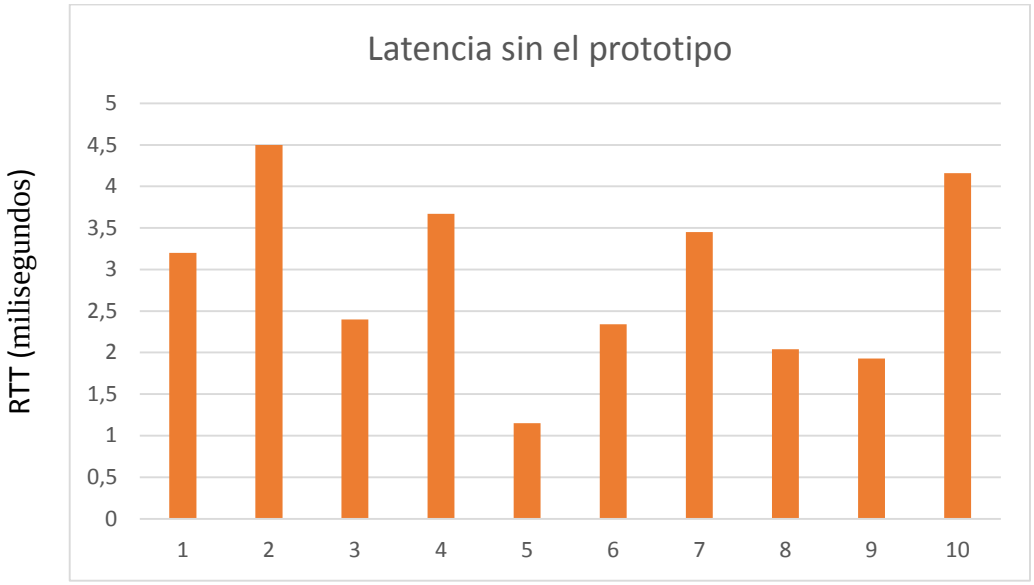

Fig. 25 Latencia sin prototipo.

Fuente: Autor.

5.2.2 Latencia promedio con 1 salto usando el prototipo

En este caso el prototipo está configurado para que el mensaje sólo realice un salto entre el emisor y el receptor. Los resultados del tiempo promedio se pueden observar en la Tabla 5 y están representados gráficamente en la Fig. 26**.**

Tabla 5 Resultados obtenidos con 1 salto usando el prototipo.

Latencia utilizando un salto	
Paquete	**RTT s**
1	10,82
2	15,03
3	12,06
4	14,23
5	11,15
6	12,34
7	10
8	12,04
9	10,93
10	14,16
Promedio	12,276

Fuente: Autor.

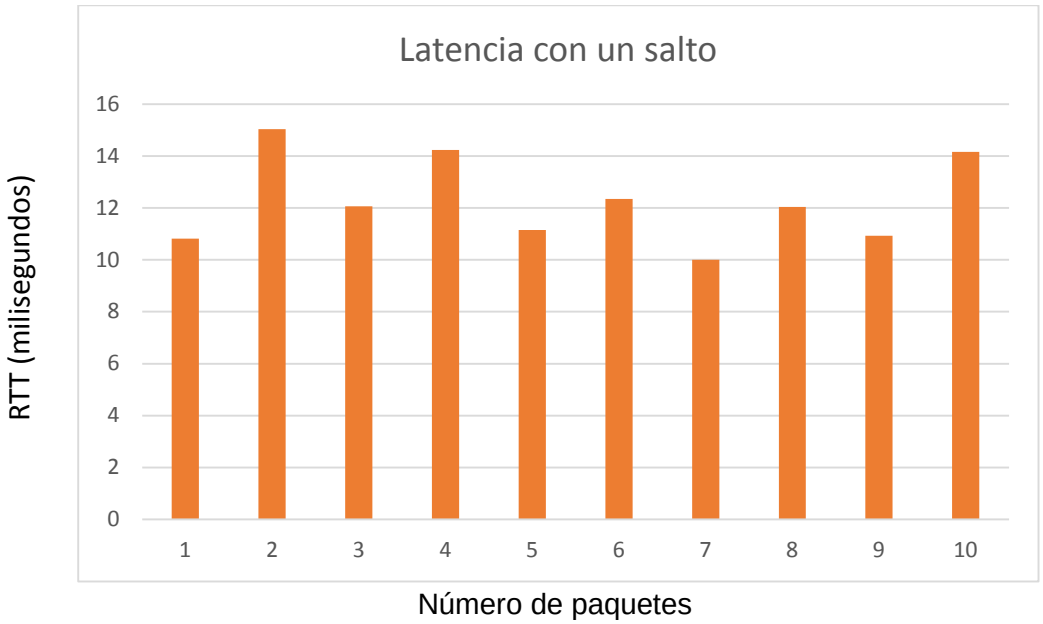

Fig. 26 Latencia con 1 salto usando el prototipo.
Fuente: Autor.

5.2.3 Latencia promedio con 2 saltos usando el prototipo

En este caso el prototipo está configurado para que el mensaje sólo realizan dos saltos entre el emisor y el receptor. Los resultados de la latencia promedio se pueden observar en la Tabla 6 y están representados gráficamente en la Fig. 27**.**

Tabla 6 Resultados obtenidos con 2 saltos usando el prototipo.

Latencia con dos saltos	
Paquete	**RTT s**
1	13,07
2	10,12
3	10,03
4	9,9
5	8,18
6	13,01
7	13,93
8	14,05
9	15,06
10	20,04
Promedio	12,739

Fuente: Autor.

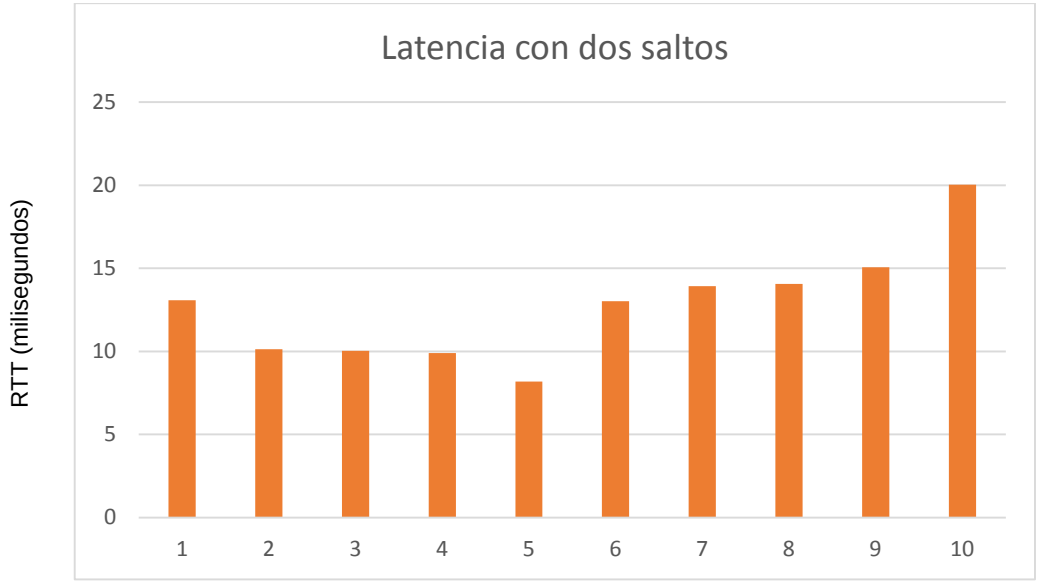

Fig. 27 Latencia con 2 saltos usando el prototipo.

Fuente: Autor.

5.2.4 Latencia promedio con 3 saltos usando el prototipo

En este caso el prototipo está configurado para que el mensaje solo realice tres saltos entre el emisor y el recepto. Los resultados de la latencia promedio se pueden observar en la Tabla 7 y están representados gráficamente en la Fig. 28.

Tabla 7 Resultados obtenidos con 3 saltos usando el prototipo.

Latencia con tres saltos	
Paquete	**RTT s**
1	19,37
2	10,33
3	11,46
4	12,12
5	8,14
6	13,15
7	14,56
8	15,09
9	16,14
10	13,04
Promedio	13,34

Fuente: Autor.

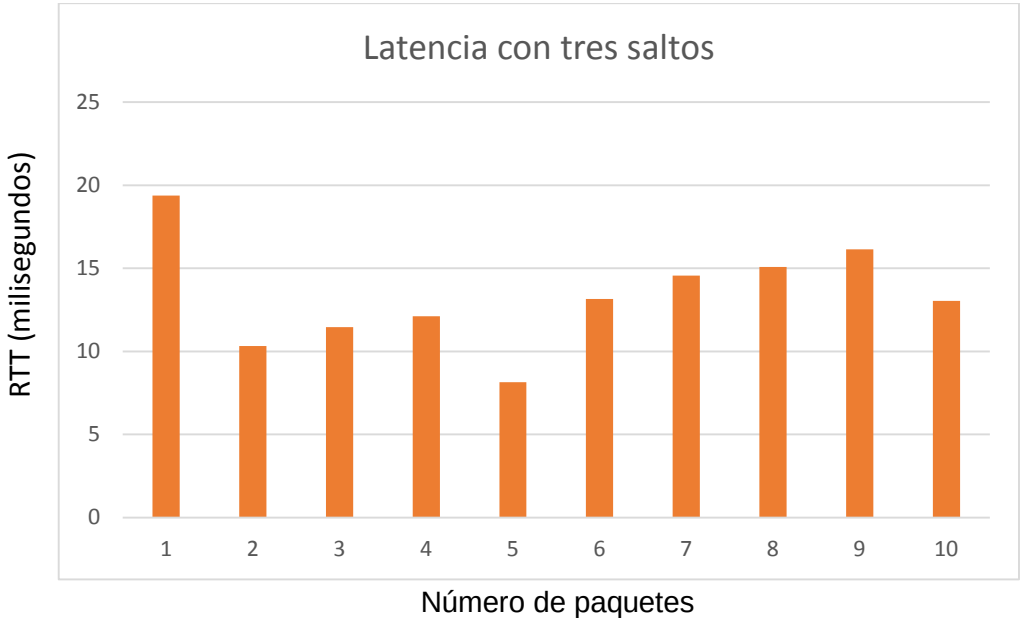

***Fig**. 28 Latencia con 3 saltos usando el prototipo*

Fuente: Autor.

Como se puede ver en la Tabla 8 y en la Fig. 29 al comparar el promedio de cada uno de los resultados, obtenemos la siguiente información**.**

Tabla 8 Tabla final con los resultados obtenidos según los saltos.

Cantidad de saltos	Promedios
0	2,89
1	12,28
2	12,74
3	13,34

Fuente: Autor.

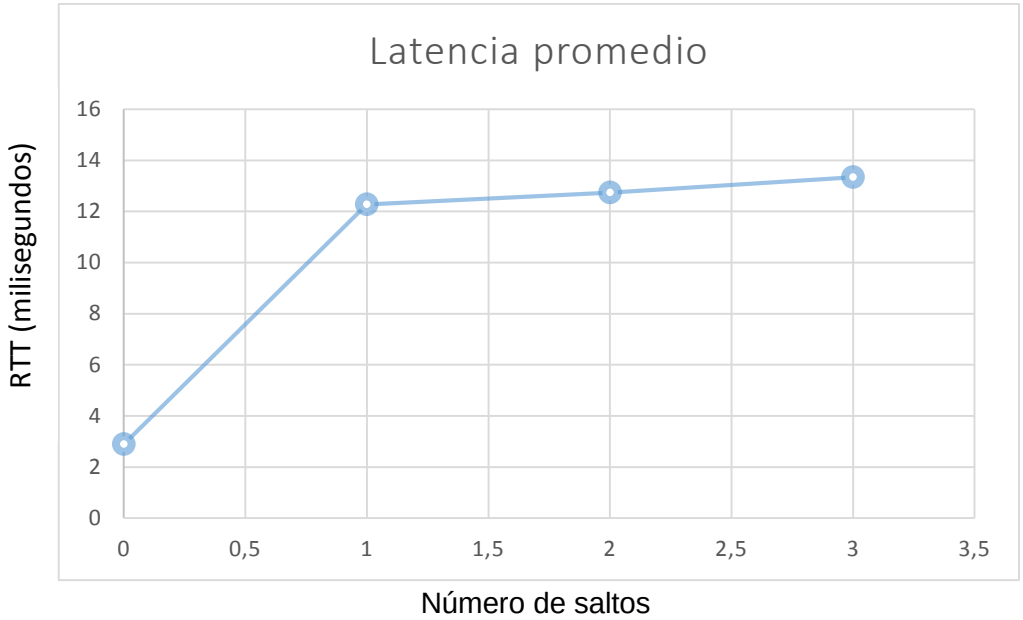

Fig. 29 Latencia con diferentes números de saltos.

Fuente: Autor.

Como se puede apreciar en la Tabla 8 y en la Fig. 29**,** que entre mayor sea el número de servidores, mayor es el tiempo en que se retrasmite; por lo tanto existe una proporcionalidad directa entre el número y el tiempo. Por esta razón es importante que el número de servidores (nodos) que va atravesar el mensaje sea un valor lo suficientemente alto para permitir el anonimato, y suficientemente pequeño para que el impacto en la latencia sea el menor posible.

CONCLUSIONES

- Con el desarrollo del presente trabajo de fin de titulación se logró cumplir con los objetivos propuestos. Esto es, se ha implementado una forma de comunicación que oculta la identificación tanto del emisor y el receptor en el intercambio de información.
- Como resultado de la aplicación se pudo verificar que el identificador de red del emisor del mensaje no es detectado por un analizador de protocolos de red (Wireshark), brindando así el anonimato.
- El anonimato se logró ocultando la dirección IPv4 tanto del emisor y el receptor utilizando equipos intermediarios en donde un analizador de protocolos no pueda detectar la dirección IP de los equipos que intercambian información gracias a los equipos intermediarios.
- A pesar de no ser el objetivo principal del presente trabajo de fin de titulación se ha probado y determinado que el prototipo afecta directamente en la latencia, mientras más elementos tenga la granja de servidores y mientras más saltos sean utilizados en la transmisión del mensaje la latencia se incrementara matemáticamente. En las pruebas realizadas cada salto que se agrega imprime aproximadamente 0,6 ms adicionales a la latencia del paquete.
- En el prototipo, la cantidad de saltos mínimo requerido para asegurar el anonimato debe ser de por lo menos dos saltos.

TRABAJO FUTURO

El trabajo de fin de titulación deja abiertas algunas cuestiones que pueden ser abordados en futuros trabajos, por ejemplo:

1. Se debe probar el prototipo en más de un dominio de colisión. Esto involucra la inclusión del código de comunicación anónima en los enrutadores intermedios.
2. Se puede mejorar el impacto en la latencia mejorando el algoritmo utilizando heurísticas de algoritmia.
3. Se debe mejorar la encriptación de la carga útil de cada mensaje que pasa por el sistema de comunicación anónima de tal forma que se mejore aún más la seguridad del sistema.

BIBLIOGRAFÍA

Berthold, O., Federrath, H., & Köpsell, S. (2001, January). Web MIXes: A system for anonymous and unobservable Internet access. In *Designing Privacy Enhancing Technologies* (pp. 115-129). Springer Berlin Heidelberg.

Claessens, J., Preneel, B., & Vandewalle, J. (1999). Solutions for anonymous communication on the Internet. In *Security Technology, 1999. Proceedings. IEEE 33rd Annual 1999 International Carnahan Conference on* (pp. 298-303). IEEE.

Danezis, G. (2004). Designing and attacking anonymous communication systems. University of Cambridge, Computer Laboratory, Technical Report, (UCAM-CL-TR-594).

Diaz, C., Seys, S., Claessens, J., & Preneel, B. (2003, January). Towards measuring anonymity. In Privacy Enhancing Technologies (pp. 54-68). Springer Berlin Heidelberg.

Freedman, M. J., & Morris, R. (2002, November). Tarzan: A peer-to-peer anonymizing network layer. In Proceedings of the 9th ACM conference on Computer and communications security (pp. 193-206). ACM.

Levine, B. N., & Shields, C. (2002). Hordes: a multicast based protocol for anonymity. *Journal of Computer Security*, *10*(3), 213-240.

Kouro, K., Tamura, S., & Yanase, T. (2007, October). Anonymous network for product recycling. In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on (pp. 2308-2312). IEEE.

Mauw, S., Verschuren, J. H., & de Vink, E. P. (2004). A formalization of anonymity and onion routing. In *Computer Security–ESORICS 2004* (pp. 109-124). Springer Berlin Heidelberg.

Rennhard, M. (2004). MorphMix–A Peer-to-Peer-based System for Anonymous Internet Access (Doctoral dissertation, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH).

Ren, J., Li, Y., Jiang, T., & Li, T. (2012). Anonymous communication in overlay networks. *Security and Communication Networks*.

Ren, J., & Wub, J. (2010). Survey on anonymous communications in computer networks. Computer Communications, 33(4), 420-431.

Reed, M. G., Syverson, P. F., & Goldschlag, D. M. (1998). Anonymous connections and onion routing. Selected Areas in Communications, IEEE Journal on, 16(4), 482-494.

Reed, M. G., Syverson, P. F., & Goldschlag, D. M. (1996, December). Proxies for anonymous routing. In Computer Security Applications Conference, 1996., 12th Annual (pp. 95-104). IEEE.

Reiter, M. K., & Rubin, A. D. (1999). Anonymous web transactions with crowds. Communications of the ACM, 42(2), 32-48.

Ritter, T Remailers We've Got. (2013) Recuperado de https://crypto.is/blog/remailers_weve_got

Wright, M. K., Adler, M., Levine, B. N., & Shields, C. (2004). The predecessor attack: An analysis of a threat to anonymous communications systems. ACM Transactions on Information and System Security (TISSEC), 7(4), 489-522.

Venkitasubramaniam, P., He, T., & Tong, L. (2008). Anonymous networking amidst eavesdroppers. Information Theory, IEEE Transactions on, 54(6), 2770-2784.

Los 8 ataques cibernéticos más importantes del grupo 'Anonymous' 9 de Julio del 2015 Consultado de: **http://www.telemundo.com/entretenimiento/2015/07/09/los-8-ataques-ciberneticos-mas-importantes-del-grupo-anonymous.**

Cronología del 'caso Snowden', el joven que reveló el espionaje masivo de Estados Unidos. 7 de Julio del 2013 Consultado de: http://www.20minutos.es/noticia/1850380/0/caso-snowden/cronologia/espionaje-ee-uu/

Redes de Computadoras. Un enfoque descendente, 5ta Edición – James F. Kurose & Keith W. Ross

ANEXOS

ANEXO 1. Código para el emisor. (Socket Client).

```
public class EchoTCPCliente {

public static final int PORT= 9400;
public static final int Puerto=8800;
public static final String Gestor="192.168.2.4";
public static String SERVER_LOCATION;
private Socket clientSocket;
private Socket cliente;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
private int vuelta=1;
private boolean error=false;

public EchoTCPCliente() {
 System.out.println("Emisor...");
try{
  do{
  try {
   // Comunicación con el gestor de nodos(servidores de   //comunicación anónima).
  System.out.println("Solicitando ip a la que se va a enviar");
  clientSocket= new Socket(Gestor, Puerto);
  outToServer = new DataOutputStream(clientSocket.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

  outToServer.writeBytes("ip" +"\n");

  String ipAnonima=inFromServer.readLine();
  System.out.println("ip a la que se va a enviar "+ipAnonima);

  clientSocket.close();
 // Reutilización de la dirección de un servidor para ocultar el mensaje
  cliente = new Socket(ipAnonima, PORT);
  outToServer = new DataOutputStream(cliente.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(cliente.getInputStream()));

  String message= JOptionPane.showInputDialog(null, "Ingrese un mensaje para enviar");
  String ipreceptor= JOptionPane.showInputDialog(null, "Ingrese la direccion del receptor");
  outToServer.writeBytes(message+"\n");
  outToServer.writeBytes(ipreceptor+"\n");

  Enumeration<NetworkInterface> host =NetworkInterface.getNetworkInterfaces();
  NetworkInterface nic = host.nextElement();
  Enumeration<InetAddress> dir = nic.getInetAddresses();

  while(dir.hasMoreElements()){
   InetAddress direccion = dir.nextElement();
   if(direccion instanceof Inet4Address){
       outToServer.writeBytes(direccion.toString().substring(1,direccion.toString().length())+"\n");
   }

  }

  error=false;

  new EchoTCPServer().start();

  }catch(SocketException s){
     clientSocket= new Socket(Gestor, Puerto);
     outToServer = new DataOutputStream(clientSocket.getOutputStream());
     outToServer.writeBytes("val" +"\n");
     System.out.println("Hubo un error al comunicarse con el servidor");
     error=true;
  }
```

```
  }while(error);
}catch(UnknownHostException e){
  e.printStackTrace();
}
catch(IOException e){
  e.printStackTrace();
}
finally
{
  try
  {
   if(inFromServer!=null) inFromServer.close();
   if(outToServer!=null)  outToServer.close();
   if(clientSocket!=null) clientSocket.close();
   if(cliente!=null) clientSocket.close();

  }catch(IOException e)
   {
   e.printStackTrace();
   }

}

}
public static void main(String args[])
{
   new EchoTCPCliente();
}

}
```

ANEXO 2. Código para el emisor. (Socket Server)

```
public class EchoTCPServer extends Thread{

  public static final int PORT = 9400;
  private ServerSocket welcomeSocket;
  private Socket connectionSocket;
  private DataOutputStream outToClient;
  private BufferedReader inFromClient;

  public EchoTCPServer() {
    System.out.println("Esperando la confirmaciÃ³n ...");
  }

  public void run(){
    try{
      this.welcomeSocket = new ServerSocket(9400);
    // Espera de la confirmación de la llegada del mensaje.
       while(true)
       {
          connectionSocket= welcomeSocket.accept();
           this.inFromClient = new BufferedReader(new I
              nputStreamReader(this.connectionSocket.getInputStream()));
           this.outToClient = new          DataOutputStream(this.connectionSocket.getOutputStream());

          String str1 = this.inFromClient.readLine();
          String str2= this.inFromClient.readLine();
          System.out.println(str1);
          connectionSocket.close();
      }
    }
    catch (IOException localIOException1){
      localIOException1.printStackTrace();
    }

    finally {
      try{
        if (this.outToClient != null) {
          this.outToClient.close();
        }
        if (this.inFromClient != null) {
          this.inFromClient.close();
        }
        if (this.connectionSocket != null) {
          this.connectionSocket.close();
        }
        if (this.welcomeSocket != null) {
          this.welcomeSocket.close();
        }
      }catch (IOException localIOException3)
      {
        localIOException3.printStackTrace();
      }
    }
  }

  public static void main(String[] paramArrayOfString){
      EchoTCPServer localEchoTCPServer = new EchoTCPServer();
      localEchoTCPServer.start();
  }
}
```

ANEXO 3. Código para el gestor de nodos. (Socket Server)

```
public class EchoTCPServer  extends Thread {

  public static final int PORT = 9400;
  private ServerSocket welcomeSocket;
  private Socket connectionSocket;
  private DataOutputStream outToClient;
  private BufferedReader inFromClient;
  public static Random r= new Random();

  public int cantidad=0;
  public int numeroNodos;

  public EchoTCPServer(){

    System.out.println("SERVIDOR PARA EL ANONIMATO");
  }

  public void run(){
    try{
      this.welcomeSocket = new ServerSocket(9400);

        System.out.println("Esperando mensaje");
        String numero = JOptionPane.showInputDialog(null,"Ingrese el numero de nodos
        que estan disponibles");
        numeroNodos= Integer.parseInt(numero);
        Enumeration<NetworkInterface>  host =NetworkInterface.getNetworkInterfaces();
        NetworkInterface nic = host.nextElement();
        Enumeration<InetAddress> dir= nic.getInetAddresses();
        EchoTCPCliente nodos=new EchoTCPCliente();

     while (dir.hasMoreElements()){
       InetAddress direccion = dir.nextElement();
       if(direccion instanceof Inet4Address){
        nodos.agregarNodo(direccion.toString().substring(1,direccion.toString().length()));
        }
       }

       while(true)
       {
          connectionSocket= welcomeSocket.accept();

          System.out.println("Entrando en el anonimato");
          this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
          this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());

        String dato =this.inFromClient.readLine();
        System.out.println("dato: "+dato);

        if(dato.charAt(0)=='/'){
            String direccion=dato.substring(1,dato.length());
            nodos.agregarNodo(direccion);
            outToClient.writeBytes("Guardando direccion");
            cantidad++;
            System.out.println("nodo "+cantidad);
         }
        else{
           String str1 = dato;
           String ip=this.inFromClient.readLine();
           String emisor = this.inFromClient.readLine();
           String N= this.inFromClient.readLine();
           int m=0;
           if(N==null){
           do{
             m=r.nextInt(nodos.numeroNodos()+1);
```

```
            }while(m==0);
            }else{
             m= Integer.parseInt(N);
            }


            new EchoTCPCliente(str1,ip,emisor,m);
           }


         if(cantidad==numeroNodos){
               System.out.println("Notificar a los nodos");
               nodos.notificar();
               System.out.println("Se  acabo de notificar");
               cantidad++;
          }
             System.out.println("Cerrando la conexion");
             connectionSocket.close();
             inFromClient.close();
             outToClient.close();
    }
   }
   catch (IOException localIOException1)
   {
    localIOException1.printStackTrace();
   }

   finally
   {
    try
    {
     if (this.outToClient != null) {
      this.outToClient.close();
     }
     if (this.inFromClient != null) {
      this.inFromClient.close();
     }
     if (this.connectionSocket != null) {
      this.connectionSocket.close();
     }
     if (this.welcomeSocket != null) {
      this.welcomeSocket.close();
     }

    }
    catch (IOException localIOException3)
    {
     localIOException3.printStackTrace();
    }

   }
 }


  public static void main(String[] paramArrayOfString)
  {
     EchoTCPServer localEchoTCPServer = new EchoTCPServer();
     Asignador asignar = new Asignador();
     asignar.start();
     localEchoTCPServer.start();


  }
}
```

ANEXO 4. Código para el gestor de nodos. (Socket Client)

```
public class EchoTCPCliente{

public static ArrayList<String> nodos =new ArrayList<String>();
public static Random r = new Random();
public static int aleatorio;
public static final int PORT= 9400;
public static final int Puerto=20000;
public static String SERVER_LOCATION;

private static ObjectOutputStream outToServidor;
private static Socket clientSocket;
private static Socket clientes;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
private String mensaje;
private String ip;
private String emi;
private int Nnodos;
private int cont=0;

public EchoTCPCliente(){
   System.out.println("Constructor para agregar nodos");
}

public EchoTCPCliente(String str1, String dirip, String emisor, int n)
{

 mensaje=str1;
 ip=dirip;
 emi=emisor;
 Nnodos=n;
 System.out.println("Trasmitiendo mensaje");
try
{
  System.out.println("Numero de nodo "+Nnodos);
  if(n==1){
       SERVER_LOCATION=ip;
  }else{
      aleatorio = r.nextInt(2);
      SERVER_LOCATION = obtenerNodo(aleatorio);
  }

  clientSocket= new Socket(SERVER_LOCATION, PORT);
  outToServer = new DataOutputStream(clientSocket.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));


  outToServer.writeBytes(mensaje+"\n");
  outToServer.writeBytes(ip+"\n");
  outToServer.writeBytes(emi+"\n");
  outToServer.writeBytes(Integer.toString(Nnodos-1));

  clientSocket.close();
  outToServer.close();
  inFromServer.close();

}catch(UnknownHostException e)
{
  e.printStackTrace();
}
catch(IOException e)
{
  e.printStackTrace();
}
finally
```

```
{
  try
  {
   if(inFromServer!=null) inFromServer.close();
   if(outToServer!=null)  outToServer.close();
   if(clientSocket!=null) clientSocket.close();

  }catch(IOException e)
   {
   e.printStackTrace();
   }

}

}

public static void agregarNodo(String nodo){
  System.out.println("Agregando el nodo: "+nodo);
  nodos.add(nodo);
}

public static String obtenerNodo(int n){
   return nodos.get(n);
}

public static int numeroNodos(){
  return nodos.size();
}

public static void eliminar(int indice){
     nodos.remove(indice);
     notificar();
}

public static void notificar(){
int nNodos =numeroNodos();
for (int i=1;i<nNodos;i++){
    try
    {
    System.out.println(nodos.get(i));
    clientes=new Socket(nodos.get(i),Puerto);
    outToServidor = new ObjectOutputStream(clientes.getOutputStream());
    outToServidor.writeObject(nodos);
    outToServidor.flush();
    clientes.close();
    outToServidor.close();
    }catch(IOException e)
    {
     e.printStackTrace();
    }
}

}

}
```

ANEXO 5. Código para el gestor de nodos. (Socket Server)

```
public class EchoTCPServer extends Thread{

  public static final int PORT = 9400;
  private ServerSocket welcomeSocket;
  private Socket connectionSocket;
  private DataOutputStream outToClient;
  private BufferedReader inFromClient;
  public static Random r= new Random();

  public int cantidad=0;
  public int numeroNodos;

  public EchoTCPServer()
  {
    System.out.println("SERVIDOR PARA EL ANONIMATO");
  }

  public void run()
  {
    try
    {
      this.welcomeSocket = new ServerSocket(9400);

        System.out.println("Esperando mensaje");

         String numero = JOptionPane.showInputDialog(null,"Ingrese el numero de nodos que estan
disponibles");
         numeroNodos= Integer.parseInt(numero);
         Enumeration<NetworkInterface>  host =NetworkInterface.getNetworkInterfaces();
         NetworkInterface nic = host.nextElement();
         Enumeration<InetAddress> dir= nic.getInetAddresses();
         EchoTCPCliente nodos=new EchoTCPCliente();

     while (dir.hasMoreElements()){
       InetAddress direccion = dir.nextElement();
       if(direccion instanceof Inet4Address){
        nodos.agregarNodo(direccion.toString().substring(1,direccion.toString().length()));
        }
       }

       while(true){
          connectionSocket= welcomeSocket.accept();
          System.out.println("Entrando en el anonimato");
          this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
          this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());

         String dato =this.inFromClient.readLine();
         System.out.println("dato: "+dato);

         if(dato.charAt(0)=='/'){
              String direccion=dato.substring(1,dato.length());
              nodos.agregarNodo(direccion);
              outToClient.writeBytes("Guardando direccion");
              cantidad++;
              System.out.println("nodo "+cantidad);
          }
         else{
             String str1 = dato;
             String ip=this.inFromClient.readLine();
             String emisor = this.inFromClient.readLine();
             String N= this.inFromClient.readLine();
             int m=0;
             if(N==null){
             do{
```

```
            m=r.nextInt(nodos.numeroNodos()+1);
           }while(m==0);
           }else{
            m= Integer.parseInt(N);
           }


           new EchoTCPCliente(str1,ip,emisor,m);
          }


        if(cantidad==numeroNodos){
              System.out.println("Notificar a los nodos");
              nodos.notificar();
              System.out.println("Se  acabo de notificar");
              cantidad++;
         }
            System.out.println("Cerrando la conexion");
            connectionSocket.close();
            inFromClient.close();
            outToClient.close();
    }
   }
   catch (IOException localIOException1)
   {
    localIOException1.printStackTrace();
   }

   finally{
    try{
      if (this.outToClient != null) {
        this.outToClient.close();
      }
      if (this.inFromClient != null) {
        this.inFromClient.close();
      }
      if (this.connectionSocket != null) {
        this.connectionSocket.close();
      }
      if (this.welcomeSocket != null) {
        this.welcomeSocket.close();
      }

    }
    catch (IOException localIOException3)
    {
      localIOException3.printStackTrace();
    }

   }
 }


  public static void main(String[] paramArrayOfString)
  {
     EchoTCPServer localEchoTCPServer = new EchoTCPServer();
     Asignador asignar = new Asignador();
     asignar.start();
     localEchoTCPServer.start();


  }
}
```

ANEXO 6. Código para el gestor de nodos. (Socket Server Asignador)

```
 public class Asignador  extends Thread{
 public static final int PORT = 8800;
 private ServerSocket welcomeSocket;
 private Socket connectionSocket;
 private DataOutputStream outToClient;
 private BufferedReader inFromClient;
 public static Random r= new Random();
 public int aleatorio;

public Asignador(){
  System.out.println("Activando el asignador");
 }

public void run(){
   try
   {
     this.welcomeSocket = new ServerSocket(8800);
     EchoTCPCliente nodos=new EchoTCPCliente();

      while(true) {
        connectionSocket= welcomeSocket.accept();
        System.out.println("Cliente solicitando ip");

        this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
        this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());

        String dato = inFromClient.readLine();

        if(dato.equals("ip")){
        aleatorio = r.nextInt(nodos.numeroNodos());
        String ip= nodos.obtenerNodo(aleatorio);
        System.out.println ("Asignando la "+ dato+": "+ip);
        outToClient.writeBytes(ip);
        }else{
            System.out.println("Eliminar nodo de la posicion:"+aleatorio);
            nodos.eliminar(aleatorio);

        }


        connectionSocket.close();


     }
   }
   catch (IOException localIOException1)
   {
     localIOException1.printStackTrace();
   }

   finally
   {
     try
     {
       if (this.outToClient != null) {
         this.outToClient.close();
       }
       if (this.inFromClient != null) {
         this.inFromClient.close();
       }
       if (this.connectionSocket != null) {
         this.connectionSocket.close();
```

```
        }
        if (this.welcomeSocket != null) {
          this.welcomeSocket.close();
        }

      }catch (IOException localIOException3)
      {
        localIOException3.printStackTrace();
      }
    }
  }
}
```

ANEXO 7. Código para el servidor de comunicación anónima Ubuntu. (Socket Server Asignador).

```
public class EchoTCPServer extends Thread{
public static final int PORT = 9400;
private ServerSocket welcomeSocket;
private Socket connectionSocket;
private Socket clientSocket;
private DataOutputStream outToClient;
private BufferedReader inFromClient;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
public static Random r= new Random();

public EchoTCPServer() {
  System.out.println("Servidor de comunicacion anonima ...");
}

public void run(){
  try
  {    this.welcomeSocket = new ServerSocket(9400);
       System.out.println("Esperando el mensaje");
       clientSocket= new Socket("192.168.2.4", PORT);
       outToServer = new DataOutputStream(clientSocket.getOutputStream());

   Enumeration<NetworkInterface>  host =NetworkInterface.getNetworkInterfaces();
   NetworkInterface nic = host.nextElement();
   Enumeration<InetAddress> dir= nic.getInetAddresses();
   while (dir.hasMoreElements()){
         InetAddress direccion = dir.nextElement();
         if(direccion instanceof Inet4Address){
           System.out.println("Direccion "+direccion);
           outToServer.writeBytes(direccion.toString());
          }

    }
   outToServer.close();
   clientSocket.close();

     while(true){
       connectionSocket= welcomeSocket.accept();
       this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
       this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());
       this.inFromServer= new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
       String str1 = this.inFromClient.readLine();
       String ip = this.inFromClient.readLine();
       String emisor= this.inFromClient.readLine();
       String N=this.inFromClient.readLine();
       int m = 0;
       if(N==null){
         do{
         m=r.nextInt(ReceptorNodos.obtenerNumero()+1);
         }while(m==0);
       }else{
         m=Integer.parseInt(N);
       }
       new EchoTCPCliente(str1,ip,emisor,m);
       inFromClient.close();
       outToClient.close();
       connectionSocket.close();
       System.out.println(inFromServer.readLine());
       inFromServer.close();
    }
  }
```

```
    catch (IOException localIOException1)
    {
      localIOException1.printStackTrace();
    }

    finally
    {
      try
      {
        if (this.outToClient != null) {
          this.outToClient.close();
        }
        if (this.inFromClient != null) {
          this.inFromClient.close();
        }
        if (this.connectionSocket != null) {
          this.connectionSocket.close();
        }
        if (this.welcomeSocket != null) {
          this.welcomeSocket.close();
        }

      }
      catch (IOException localIOException3)
      {
        localIOException3.printStackTrace();
      }
    }
  }

  public static void main(String[] paramArrayOfString) throws InterruptedException
  {
     EchoTCPServer localEchoTCPServer = new EchoTCPServer();
     localEchoTCPServer.start();
     ReceptorNodos receptor = new ReceptorNodos();
     receptor.start();

  }
}
```

ANEXO 8. Código para el servidor de comunicación anónima Windows. (Socket Server Asignador).

```
 public class EchoTCPServer  extends Thread{
public static final int PORT = 9400;
private ServerSocket welcomeSocket;
private Socket connectionSocket;
private Socket clientSocket;
private DataOutputStream outToClient;
private BufferedReader inFromClient;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
public static Random r= new Random();

public EchoTCPServer(){
  System.out.println("Servidor de comunicacion anonima ...");
}

public void run(){
  try{

    this.welcomeSocket = new ServerSocket(9400);

   System.out.println("Esperando el mensaje");

   clientSocket= new Socket("192.168.2.4", PORT);
   outToServer = new DataOutputStream(clientSocket.getOutputStream());

    String direccion = InetAddress.getLocalHost().toString().substring(7,19);
          System.out.println("Direccion "+direccion);
          outToServer.writeBytes(direccion.toString());
          outToServer.close();
          clientSocket.close();

     while(true){
       connectionSocket= welcomeSocket.accept();
       this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
       this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());
       this.inFromServer= new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));

        String str1 = this.inFromClient.readLine();
        String ip = this.inFromClient.readLine();
        String emisor=this.inFromClient.readLine();
        String N= this.inFromClient.readLine();
        int m=0;
        if(N==null){
        do{
        m = r.nextInt(ReceptorNodos.obtenerNumero()+1);
        }while(m==0);
        }else{
         m =Integer.parseInt(N);
        }
        new EchoTCPCliente(str1,ip,emisor,m);
        connectionSocket.close();
        inFromClient.close();
        outToClient.close();

       System.out.println(inFromServer.readLine());
       inFromServer.close();
    }
  }
  catch (IOException localIOException1)
  {
    localIOException1.printStackTrace();
```

```
    }
    finally
    {
      try
      {
        if (this.outToClient != null) {
          this.outToClient.close();
        }
        if (this.inFromClient != null) {
          this.inFromClient.close();
        }
        if (this.connectionSocket != null) {
          this.connectionSocket.close();
        }
        if (this.welcomeSocket != null) {
          this.welcomeSocket.close();
        }

      }
      catch (IOException localIOException3)
      {
        localIOException3.printStackTrace();
      }
    }
  }

  public static void main(String[] paramArrayOfString)
  {
      EchoTCPServer localEchoTCPServer = new EchoTCPServer();
      localEchoTCPServer.start();
      ReceptorNodos receptor = new ReceptorNodos();
      receptor.start();
  }
}
```

ANEXO 9. Código para el servidor de comunicación anónima (Receptor de nodos). (Socket Client).

```
public class EchoTCPCliente{

public static ArrayList<String> nodos;
public static Random r = new Random();
public static int aleatorio;
public static final int PORT= 9400;
public static  String SERVER_LOCATION;

private Socket clientSocket;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
private String mensaje;
private String ip;
private String emi;
private int Nnodos;

public EchoTCPCliente(ArrayList n){
  System.out.println("Agregando nodos");
  nodos=n;
}
public EchoTCPCliente(String str1, String dirip,String emisor, int n)
{

 mensaje=str1;
 ip=dirip;
 Nnodos=n;
 System.out.println("Trasmitiendo mensaje");
try
{
  System.out.println("Numero de nodo "+Nnodos);
  if(n==1){
     SERVER_LOCATION = ip;
  }else{
     aleatorio = r.nextInt(nodos.size());
     SERVER_LOCATION = nodos.get(aleatorio);
  }

  clientSocket= new Socket(SERVER_LOCATION, PORT);
  outToServer = new DataOutputStream(clientSocket.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));


  outToServer.writeBytes(mensaje+"\n");
  outToServer.writeBytes(ip+"\n");
  outToServer.writeBytes(emi+"\n");
  outToServer.writeBytes(Integer.toString(Nnodos-1));


  clientSocket.close();
  outToServer.close();
  inFromServer.close();

}catch(UnknownHostException e)
{
  e.printStackTrace();
}
catch(IOException e)
{
  e.printStackTrace();
}
finally
{
  try
```

```
  {
   if(inFromServer!=null) inFromServer.close();
   if(outToServer!=null)  outToServer.close();
   if(clientSocket!=null) clientSocket.close();


  }catch(IOException e)
   {
   e.printStackTrace();
   }

}

}

}
```

ANEXO 10. Código para el servidor de comunicación anónima. (Socket Server).

```
public class ReceptorNodos extends Thread {
    public static final int Puerto=20000;
    private ServerSocket welcomeSocket;
    private Socket connectionSocket;
    private ObjectInputStream inFromClient;
    public static ArrayList<String> nodos;

    public ReceptorNodos(){
         System.out.println("Recolectando Nodos");
    }

    @Override
    public void run(){
        try {
            this.welcomeSocket = new ServerSocket(20000);

            while(true){
             connectionSocket= welcomeSocket.accept();
             System.out.println("Recibiendo los nodos");

             inFromClient= new ObjectInputStream(connectionSocket.getInputStream());
             nodos=(ArrayList)inFromClient.readObject();
             new EchoTCPCliente(nodos);
             connectionSocket.close();
            }

        } catch (IOException ex) {
            Logger.getLogger(ReceptorNodos.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(ReceptorNodos.class.getName()).log(Level.SEVERE, null, ex);
        }
        finally{
      try {
        if (this.inFromClient != null) {
          this.inFromClient.close();
        }
        if (this.connectionSocket != null) {
          this.connectionSocket.close();
        }
        if (this.welcomeSocket != null) {
          this.welcomeSocket.close();
        }

      }
      catch (IOException localIOException3)
      {
        localIOException3.printStackTrace();
      }
    }

    }

    public static int obtenerNumero(){
        return nodos.size();
      }

}
```

ANEXO 11. Código para el Receptor (Socket Server).

```
 public class EchoTCPServer extends Thread{
public static final int PORT = 9400;
private ServerSocket welcomeSocket;
private Socket connectionSocket;
private DataOutputStream outToClient;
private BufferedReader inFromClient;

public EchoTCPServer(){
  System.out.println("RECEPTOR ...");
}

public void run(){
  try{
    this.welcomeSocket = new ServerSocket(9400);
   System.out.println("Esperando mensaje");

     while(true){
       connectionSocket= welcomeSocket.accept();
       this.inFromClient = new BufferedReader(new
InputStreamReader(this.connectionSocket.getInputStream()));
       this.outToClient = new DataOutputStream(this.connectionSocket.getOutputStream());
       String mensaje = this.inFromClient.readLine();
       String emisor= this.inFromClient.readLine();
       String receptor=this.inFromClient.readLine();
       String N=this.inFromClient.readLine();
       System.out.println("Emisor: "+receptor);
       System.out.println("Mensaje del emisor:" +mensaje);
       new EchoTCPCliente(receptor);
       this.outToClient.writeBytes("Llego el mensaje");
       connectionSocket.close();

    }
  }
  catch (IOException localIOException1)
  {
    localIOException1.printStackTrace();
  }

  finally
  {
    try
    {
      if (this.outToClient != null) {
        this.outToClient.close();
      }
      if (this.inFromClient != null) {
        this.inFromClient.close();
      }
      if (this.connectionSocket != null) {
        this.connectionSocket.close();
      }
      if (this.welcomeSocket != null) {
        this.welcomeSocket.close();
      }

    }
    catch (IOException localIOException3)
    {
      localIOException3.printStackTrace();
    }
  }
}

public static void main(String[] paramArrayOfString) {
    EchoTCPServer localEchoTCPServer = new EchoTCPServer();
```

```
        localEchoTCPServer.start();
    }
}
```

ANEXO 12. Código para el Receptor (Socket Client).

```
public class EchoTCPCliente {

public static final int PORT= 9400;
public static final int Puerto=8800;
public static final String Gestor="192.168.2.4";
public static String SERVER_LOCATION;

private Socket clientSocket;
private Socket cliente;
private DataOutputStream outToServer;
private BufferedReader inFromServer;
private int vuelta=1;
private boolean error=false;

public EchoTCPCliente(String receptor)
{

 System.out.println("Emisor...");
try
{
  do{
  try {
  System.out.println("Solicitando ip a la que se va a enviar");
  clientSocket= new Socket(Gestor, Puerto);
  outToServer = new DataOutputStream(clientSocket.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

  outToServer.writeBytes("ip" +"\n");

  String ipAnonima=inFromServer.readLine();
  System.out.println("ip a la que se va a enviar "+ipAnonima);

  clientSocket.close();

  cliente = new Socket(ipAnonima, PORT);
  outToServer = new DataOutputStream(cliente.getOutputStream());
  inFromServer= new BufferedReader(new InputStreamReader(cliente.getInputStream()));

  String message= "Llego el mensaje";
  String ipreceptor= receptor;
  outToServer.writeBytes(message+"\n");
  outToServer.writeBytes(ipreceptor+"\n");

  error=false;
  }catch(SocketException s){
     clientSocket= new Socket(Gestor, Puerto);
     outToServer = new DataOutputStream(clientSocket.getOutputStream());
     outToServer.writeBytes("val" +"\n");
     System.out.println("Hubo un error al comunicarse con el servidor");
     error=true;
  }

  }while(error);
}catch(UnknownHostException e)
{
  e.printStackTrace();
}
catch(IOException e)
{
  e.printStackTrace();
}
finally
{
  try
  {
```

```
   if(inFromServer!=null) inFromServer.close();
   if(outToServer!=null)  outToServer.close();
   if(clientSocket!=null) clientSocket.close();
   if(cliente!=null) clientSocket.close();

  }catch(IOException e)
   {
   e.printStackTrace();
   }

}

}

}
```

ANEXO 13. Procedimientos y herramientas para el funcionamiento del prototipo.

Para hacer funcionar el prototipo se debe seguir el siguiente procedimiento:

El emisor y el receptor trabajan en el sistema operativo Windows 10 con el lenguaje de programación java versión 7, el emisor ejecutará el código especificado en el Anexo 1 y el Anexo 2. La dirección IP del emisor se configuró con 192.168.2.3; en cambio el receptor en el Anexo 11 y el Anexo 12, su dirección IP estática es 192.168.2.7.

La granja de servidores, incluyendo al equipo gestor de nodos, se implementaron en el sistema operativo Kubuntu versión 14 y el lenguaje de programación java versión 7.

El gestor de nodos o servidores ejecuta un código gestor especificado desde el Anexos 3 hasta el Anexo 6 con la dirección IP estática 192.168.2.4.

La granja de servidores ejecutarán los códigos especificados desde el Anexo 7 hasta el Anexo 10, las direcciones con las que trabajarán son la 192.168.2.6 y la 192.168.2.8.

Para compilar los códigos se puede utilizar cualquier IDE (Netbeans, entre otros), o modificar las variables de entorno java para ejecutarlos en el terminal de los sistemas operativos.

Los equipos deben estar conectados con un cable directo utilizando un switch para que exista la comunicación.

Se debe utilizar un analizador de protocolo (Por ejemplo, la herramienta Wireshark) para capturar los paquetes en tránsito y poder determinar la forma en la que se oculta la direcciones IP.