Implementation of a DC-Net Based Anonymous Messaging System

Camilo Gómez N.
NIC Chile Research Labs
University of Chile
camilo@niclabs.cl

Abstract

A DC-Net allows parties to communicate anonymously in a point-to-point network. Even though DC-Nets where proposed by Chaum (CACM 1985) more than thirty years ago, only lately the protocol has gained more attention because of both efficiency improvements in point to point communication, and several attacks discovered in onion routing, the most popular anonymous protocol. This work describes an implementation of a practical variant of DC-Nets (proposed by Franck, Hevia, and van Der Graaf 2016) which includes a useful API. Preliminary experiments indicate that the protocol is better suited for non-interactive applications such as an anonymous bulletin board. We indeed implement such a mobile app and discuss possible extentions.

1 Introducción

Mantener el anonimato navegando en Internet se ha vuelto una demanda significativamente más popular en los último años. Usuarios de todas las latitudes reclaman que se respete su derecho a la privacidad[1]; posiblemente como consecuencia de las revelaciones realizadas tanto por el grupo *Wikileaks* como por el ex-empleado de la *CIA* Edward Snowden. Estas revelaciones ilustraron el masivo monitoreo por parte de organismos de inteligencia[2] a la actividad diaria de millones de personas que hacen uso de la red.

Bajo ese panorama, se ha hecho necesario contar con herramientas que aseguren el anonimato en la red. Sin embargo, garantizar anonimato ante adversarios sofisticados requiere dos condiciones: (1) uso y testeo masivo por parte de un significativo número de usuarios, y (2) garantías matemáticas (criptográficas) de la seguridad del protocolo que se está utilizando.

El software más popular y utilizado para permanecer anónimo mientras se navega en Internet es *TOR* "The Onion Router"[3]. *Onion-routing* [GRS96], el protocolo implementado por TOR, ha sufrido recientes cuestionamientos, tales como si es o no realista suponer que el adversario no es capaz de monitorear todos los canales de comunicación – uno de los supuestos más cruciales de TOR. La pregunta es sin duda razonable dada la extensa capacidad de monitoreo que poseen organismos de inteligencia para con todos los usuarios de Internet[4]. Aun así, en la práctica *TOR* sigue siendo la herramienta más popular para navegar de manera anónima en Internet, siendo incluso recomendada por el mismo Snowden[5] para evitar las intromisiones de la *NSA* y otras organismos similares. Dado que concentrar nuestra privacidad en una sola herramienta puede ser riesgoso, se hace necesario evaluar la factibilidad de protocolos alternativos que toleran adversarios globales (capaces de monitorear todos los canales de comunicación) tales como las DC-nets. Este trabajo busca aportar en este estudio de alternativas.

2 Anonimato via DC-Nets

2.1 *Dining Cryptographers Problem*

El problema de la Cena de Criptógrafos (*Dining Cryptographers Problem*) fue propuesto por David

[1] http://www.livescience.com/37398-right-to-privacy.html

[2] http://www.huffingtonpost.com/news/nsa-spying/

[3] https://www.torproject.org/

[4] http://apps.washingtonpost.com/g/page/world/nsa-slideshow-on-the-tor-problem/499/

[5] http://www.inc.com/larry-kim/5-online-privacy-tips-from-edward-snowden.html

Chaum en el año 1985 [Cha85, Cha88] para ejemplificar un protocolo de envío de mensajes de manera anónima entre un grupo de participantes.

El problema es el siguiente: tres criptógrafos están cenando en su restaurante favorito. Al terminar la cena, el mesero se acerca a su mesa y les informa que el *maître* indica que su cuenta ya está pagada. Los criptógrafos, sorprendidos, quieren resolver si la cuenta fue pagada por alguno de ellos, o si simplemente fue pagada por algún agente externo (la *NSA*, por ejemplo). Obviamente, como buenos criptógrafos, desean que su protocolo no revele la identidad del criptógrafo pagador (si existiera) pero al mismo tiempo permita dilucidar si uno de ellos pagó o fue un agente externo. Luego de pensarlo, el problema es resuelto con el siguiente protocolo:

(1) Cada uno de los criptógrafos lanza una moneda al aire, de forma secreta. Luego de esto, le muestra la moneda al compañero que está inmediatamente a su derecha. (2) Cada criptógrafo informa al resto si las dos monedas que puede ver (la propia y la de su compañero a la izquierda) son iguales o no, pero con una condición: si el criptógrafo viendo las monedas no pagó la cuenta, simplemente dice la verdad. Si no, si no fue el (o ella) quien hizo el pago, el criptógrafo revela lo contrario a lo que ve (si eran iguales dice que eran distintas y viceversa). (3) Si un número impar de criptógrafos revela que las monedas eran distintas, implica que fue uno de ellos quien pagó la cuenta, de lo contrario significa que la *NSA* fue la que realizó el pago.

En el artículo original se demuestra que este protocolo (posteriormente rebautizado como *DC-Net*). brinda anonimato incondicional: aún si un atacante tuviera todo el poder de cómputo posible, le es imposible poder identificar al emisor del mensaje. Este anonimato se mantiene tanto para todos los participantes del protocolo como para cualquier observador externo.

Con lo anterior, Chaum brinda un protocolo que hace posible transmitir mensajes públicos de manera anónima. Chaum también propuso algunas variaciones, tales como protocolos que permiten enviar mensajes de largo arbitrario.

2.2 DC-Net Generalizada

Existe una generalización práctica para utilizar el protocolo DC-Net con mensajes de largo mayor a 1 bit. En vez de manifestar un bit (pagó o no la cuenta), cada participante envía un mensaje (cadena de bits) arbitrario. Además se reemplaza el lanzamiento de una moneda vista entre dos participantes con una llave aleatoria compartida entre ambos participantes. Por último, en vez de reemplazar el mensaje enviado dependiendo de que valor tomó la moneda (el "O Exclusivo" o `XOR` entre el mensaje y el valor de la moneda), el mensaje a enviar y las llaves compartidas que posee el participante son sumadas módulo un valor fijo suficientemente grande.

En la práctica, primero es necesario establecer el *anonymity-set*, esto es, quiénes serán los participantes del protocolo, los responsables de enviar y aportar con sus mensajes para asegurar el anonimato del potencial emisor. Luego, se utiliza una ronda de compartimiento de llaves, donde cada par de participantes comparten una llave secreta (este proceso podría realizarse utilizando protocolo *Diffie-Hellman*). Una vez que todos poseen llaves compartidas entre sí, cada uno de los participantes enviará uno de los dos siguientes mensajes al resto: un valor K igual a la suma de las llaves que posee compartidas, ó un valor $K + M$ igual a la suma de las llaves que posee compartidas más un mensaje M que quiere enviar de manera anónima. Con esto, cada participante envía un valor $m = K$ si es que no quiere transmitir nada y solo quieren aportar en el anonimato del resto, ó $m = K + M$ si es que sí desea enviar un mensaje M. Finalmente, cada uno de los participantes suma los valores que recibió, lo que hará que las llaves compartidas se cancelen y sólo quede visible el mensaje que compartió uno de los participantes. De esta manera, el emisor se mantiene anónimo tanto para el resto de los participantes como para cualquier observador externo que monitoree todos los valores intercambiados en el protocolo.

2.2.1 Colisión de Mensajes

Lo anterior funciona siempre y cuando a lo más un participante envíe un mensaje. A priori, no es claro que los participantes puedan ponerse de acuerdo respecto a quién de los participantes enviará el mensaje, sin violar el anonimato del protocolo. Por ello, una colisión de mensajes – esto es, el envío de un mensaje por parte de dos o más participantes – es normal en la ejecución de una DC-Net. Cuando una colisión ocurre, al realizar la suma los valores enviados por todos los participantes, las llaves se cancelan resultando la suma de los dos (o más) mensajes enviados. Dado que este valor es típicamente un mensaje ininteligible, una colisión implica un fracaso en la transmisión de los mensajes involucrados. Esto es un problema conocido. Varias implementaciones y diseños propuestos han buscado hacerse cargo y resolverlo para poder permitir el envío de varios mensajes en una misma iteración.

2.2.2 Participantes Maliciosos

Otro aspecto importante de la seguridad de los protocolos basados en DC-Net es su tolerancia a los participantes maliciosos. Tales participantes envían men-

sajes erróneos – que no se ajustan al protocolo acordado – o omiten enviarlos cuando debieran. Si es que no se evita este comportamiento, podría resultar en un protocolo erróneo, que nunca concluya con un mensaje transmitido, o bien que entregue mensajes equivocados. Afortunadamente, la mayor parte de los protocolos resuelven este problema integrando primitivas criptográficas al envío de los mensajes.

3 Variante Práctica de DC-Net

Este trabajo se basa en una variante práctica del protocolo DC-Net propuesta por J. van de Graaf y Christian Franck [FvdG14], que hace uso de primitivas criptográficas (*Pedersen commitments* y *zero-knowledge proofs* principalmente) para sortear los problemas de colisión de mensajes y participantes maliciosos.

El diseño que se explicará a continuación es un trabajo en curso realizado en conjunto con los autores antes mencionados, y se omitirán los detalles correspondientes a demostraciones de seguridad, debido a que lo importante a destacar son los desafíos que implican realizar la implementación del protocolo.

3.1 Primitivas Criptográficas

3.1.1 *Pedersen Commitments* [Ped91]

Sea G_q un grupo de orden q, en donde el problema del logaritmo discreto se crea difícil de resolver. Sean g, h generadores de G_q elegidos aleatoriamente. Sea $s \in Z_q$ un secreto que no se desea comprometer. Además sea $r \in Z_q$ elegido aleatoriamente. Se le llama un *Pedersen commitment* sobre s al valor:

$$c := g^s h^r$$

Los *Pedersen commitments* brindan dos propiedades importantes: ocultamiento perfecto (*unconditionally hiding*) y vinculación computacional (*computationally binding*). Esto quiere decir que, si una persona calcula un *commitment* sobre un cierto valor s, (1) la persona se compromete a dicho valor (pero sin revelarlo), ya que, por un lado es imposible para un tercero conocer s a partir de c, y (2) es computacionalmente difícil demostrar que dentro de c existe un valor distinto a s.

3.1.2 *Zero-Knowledge Proofs*

Una *zero-knowledge proof* permite a una persona poder demostrar el conocimiento de cierto valor α que cumple una propiedad (en Inglés, *statement*), sin revelar el valor de α (el testigo) en esa demostración. Se pueden construir demostraciones para diferentes tipos de propiedades (*statements*), como por ejemplo: el conocimiento de un logaritmo discreto; la igualdad de diferentes logaritmos con distintas bases, además de combinaciones con operadores lógicos, entre otros. Además existen maneras para poder demostrar propiedades genéricas sobre logaritmos discretos [CS97].

3.2 Compartición de Llaves

El protocolo comienza con cada participante P_i intercambiando dos valores k_{ij}, r_{ij} con cada otro participante P_j dentro del *anonymity-set*. Además se establece que $k_{ij} = -k_{ji}$ y $k_{ii} = 0$ (ídem para r). Finalmente cada participante P_i calculará el valor $K_i = \sum_{j=1}^{n} k_{ij}$, donde n es el número total de participantes (ídem para calcular r_i).

3.3 Envío de *Commitments* y *ZKP*

3.3.1 Llaves Compartidas

Cada P_i calculará $c_{K_i} = g^{k_i} h^{r_{K_i}}$ (*commitment* sobre K_i). Luego de esto deberá enviar al resto de los participantes c_{K_i} junto con una PoK[6] indicando que conoce los valores (K_i, r_{K_i}) dentro de c_{K_i}.

Al recibir los *commitment* y las PoK de cada participante P_i, se deben verificar cada una de las PoK, además debe comprobarse que $\prod_{i=1}^{n} c_{K_i} = 1$ (propiedad que se tiene debido a que las llaves deben cancelarse unas con otras). De no tenerse esta última propiedad (lo que significa que al menos un participante envió valores erróneos), existen maneras que permiten encontrar al participante malicioso en cuestión. En la actual implementación se priorizó encontrar participantes maliciosos en etapas subsecuentes del protocolo, debido a que un participante malicioso en esta etapa no puede comprometer el anonimato que brinda el mensaje, sino simplemente retrasar el desarrollo del protocolo.

3.3.2 Valores Individuales

Cada participante P_i debe establecer tres valores distintos (dependiendo si enviará o no un mensaje):

- m_i: mensaje plano que enviará el participante. Si no enviará ningún mensaje, $m_i = 0$.
- pad_i: cadena aleatoria de bits que se utiliza para reducir la posibilidad de enviar dos mensajes iguales (más adelante en el protocolo se explicará su uso). Si no enviará ningún mensaje, $pad_i = 0$.
- b_i: bit estableciendo si se enviará o no un mensaje, $b_i = 1$ si se enviará un mensaje, 0 si no.

[6] *Proof of Knowledge* es una *Zero-Knowledge Proof* donde el participando demuestra el conocimiento de un cierto valor dentro de un *statement*, sin revelar dicho valor.

Luego de establecer estos valores, cada P_i formará el mensaje M_i, que resulta ser una concatenación de los 3 valores anteriores (la concatenación de valores 0^n es para prevenir que se produzcan *overflow* al sumarlos posteriormente):

$$M_i = 0^n \mid\mid m_i \mid\mid 0^n \mid\mid pad_i \mid\mid 0^n \mid\mid b_i$$

Posteriormente cada P_i deberá escoger 3 valores aleatorios: r_{m_i}, r_{pad_i} y r_{b_i}, para luego calcular *commitments* sobre los 3 valores anteriores: c_{m_i}, c_{pad_i} y c_{b_i} (utilizando como valores aleatorios los escogidos previamente). Con estos *commitments* cada P_i deberá realizar una demostración de que el mensaje M_i está bien formateado, esto es que: "el último bit es 1 o el último bit es 0 al igual que todo el mensaje" $(b_i = 1 \vee (b_i = 0 \wedge m_i = 0))$. Se envía al resto de los participantes los 3 *commitments* calculados junto con la demostración anterior.

Posteriormente se verifican las demostraciones enviadas por cada uno de los participantes.

3.3.3 Mensaje formateado

El próximo paso es enviar una *PoK* sobre el mensaje M_i. El punto importante a destacar aquí es que no se envía un *commitment* asociado a M_i, sino que éste se calcula en base a los 3 *commitments* enviados anteriormente c_{m_i}, c_{pad_i} y c_{b_i}. Por ello cada P_i envía la *PoK* correspondiente, y es verificada por el resto de los participantes calculando el *commitment* correspondiente.

3.3.4 Mensaje de Salida

Finalmente cada P_i formará el mensaje de salida $O_i = M_i + K_i$. Al igual que antes, debe además crear una *PoK* sobre O_i en el *commitment* c_{O_i} (que también se puede calcular como función de los *commitments* anteriormente enviados). Se enviará el mensaje O_i junto con la *PoK* correspondiente y será revisada por el resto de los participantes.

3.4 Envío de Mensaje de Salida

Todos los participantes enviaron su respectivo mensaje de salida O_i, los cuáles deben ser sumados entre todos formando el valor $C^1 = (M^*, t) = \sum_{i=1}^{n} O_i$ (M^* es la suma de los valores m_i y pad_i de todos los participantes, mientras que t corresponde a la suma de cada uno de los valores b_i). Teniendo estos valores, se pueden producir tres casos:

1. $t = 0$: Ningún participante quiso enviar un mensaje, por lo que el protocolo finaliza.

2. $t = 1$: Solamente un participante envió un mensaje, por lo que el valor M^* corresponde a ese mensaje, el cual fue recibido por todos los participantes y se mantiene el anonimato de su emisor. El protocolo finaliza.

3. $t > 1$: Dos o más participantes enviaron un mensaje, por lo que se ha producido una colisión de mensajes (M^* consiste en la suma de varios mensajes m_i). Para resolver esto se correrá un protocolo de resolución de colisiones basado en *superposed-receiving* [Wai89] explicado a continuación.

3.5 Resolución de Colisiones

Para resolver la colisión, la idea principal es correr nuevamente el protocolo pero solamente un subconjunto de los participantes que tienen su mensaje involucrado en la colisión enviarán mensajes. Para ello se calcula el valor $\bar{M} = M^*/t$. Si el mensaje $M_i < \bar{M}$, entonces se reenviará en la ronda $2k$ (donde k es el número de la ronda donde se produce la colisión). En caso contrario se reenviará en la ronda $2k + 1$.

Una parte importante de *superposed-receiving* es que la ronda $2k + 1$ puede ser calculada como función de los mensajes enviados en las rondas k y $2k$ de la siguiente manera: $C^{2k+1} = C^k - C^{2k}$. Por ello las rondas $2k + 1$ se le llaman rondas virtuales, y las rondas $2k$ se llaman rondas reales. De esta manera se disminuye la cantidad de rondas reales a realizarse para la resolución de las colisiones.

De esta manera, en las rondas subsecuentes la cantidad de mensajes involucrados irá disminuyendo, haciendo que posibles colisiones que se produzcan (que se resuelven recursivamente de la misma manera) sean menores, implicando que después de t rondas reales se habrán enviados los t mensajes involucrados en la primera colisión. Importante a destacar es que en las rondas siguientes deben enviarse *PoK* indicando que el mensaje que se envía en dicha ronda es el mismo que estuvo involucrado en alguna colisión anterior (o se envía un mensaje vacío).

Con esto, al enviarse los t mensajes involucrados en la primera colisión, el protocolo finaliza y todos los participantes pueden ver que su mensaje fue enviado y se mantiene su anonimato frente tanto a los otros participantes como a observadores externos que vean el desarrollo del protocolo de manera pasiva. Como fue dicho anteriormente, el diseño de este protocolo es un trabajo en curso, por lo que mayores detalles de demostraciones de seguridad se están actualmente investigando y preparando una publicación más específica.

4 Detalles de Implementación

El objetivo principal de este trabajo viene siendo la implementación del protocolo explicado anterior-

mente como API, de manera tal que pueda ser utilizado en variadas aplicaciones que necesiten contar con un protocolo de mensajería anónima entre varios participantes. Algunos detalles y desafíos que implican la implementación de un protocolo de DC-NET son explicados a continuación.

4.1 Tecnologías Involucradas

La implementación se está desarrollando en *Java* principalmente por el hecho que una primera prueba de concepto será utilizar la API implementada en una aplicación móvil *Android* (basado en *Java*), por lo que su integración sería más simple y rápida. No existe ninguna razón de fondo (*performance* o alcance del lenguaje) por lo que se escogió dicho lenguaje de programación, por lo que los mismos resultados se obtendrían si es que se implementara en otro lenguaje más común en otras implementaciones de DC-NET, como lo es *C++*.

Para la conexión entre los participantes se utiliza *ZeroMQ*[7], que, en palabras de sus autores, "son *sockets* en esteroides". Funciona para abstraer la implementación de la capa de transporte de la aplicación y despreocuparse de reconexiones o manejar pérdidas de datos. Con *ZeroMQ* se simplifica la implementación de *broadcasting* o conexiones punto a punto entre los distintos participantes (ambos tipos de conexiones necesarias para el protocolo).

4.2 Nodo Directorio y Nodos Participantes

Un desafío importante a considerar en la implementación (y que el diseño del protocolo no se preocupa) es como descubrir la locación del resto de los participantes que formarán parte del *anonymity-set*. Para ello se tomó la decisión de contar, además de los nodos participantes, con un nodo Directorio. Este nodo funcionará como punto de entrada al *anonymity-set* y será el responsable de informar la dirección IP de cada uno de los nodos participantes. Además de esto, el nodo Directorio tiene como responsabilidad establecer los parámetros necesarios para correr el protocolo (número de participantes, valores públicos para realizar los *commitments*, entre otros), por lo que también se vuelve un punto de control dentro del protocolo. Un aspecto importante es que todo lo que relaciona el protocolo con mantener anonimato (envío y verificaciones de demostraciones de seguridad) no pasa por el nodo Directorio, por lo que su incorporación no altera en nada el requisito de seguridad buscado (emisores de mensajes anónimos).

Esta tarea también se puede realizar entre los propios nodos participantes, sin la necesidad de incorporar un nodo Directorio. Si bien en esta investigación se priorizó la facilidad de implementar la variante utilizando el nodo adicional, correr alguna variante de *gossip protocol* para informar la identidad de participantes nuevos que vayan entrando a la sala podría solucionar el problema. Esta segunda variante además tiene la ventaja de no poseer un punto vulnerable (que sería el nodo Directorio), evitando ataques directos al nodo Directorio, retrasando (o incluso imposibilitando) la creación del *anonimity-set*. Reiterar que se implementó el nodo Directorio por simplicidad, pero se tienen en cuenta los posibles ataques que puede recibir el sistema, que no tienen incidencia en romper el anonimato que brinda el protocolo.

Actualmente el nodo Directorio inicializa estableciendo los parámetros públicos del protocolo y publica su dirección IP. Luego, cada nodo Participante que se quiera unir se conecta a la dirección pública del Directorio y espera que se complete la cuota de participantes establecida en un comienzo. Cuando se conectan n participantes al Directorio, éste informa la dirección IP de cada uno de los participantes a todo el resto, para que posteriormente inicien el protocolo solo enviándose mensajes entre ellos, finalizando así la labor del Directorio.

4.3 Primitivas Criptográficas

En la implementación actual, la gran mayoría de las primitivas criptográficas han sido implementadas desde cero, valiéndose principalmente de la biblioteca para manejar números grandes de *Java*, *BigInteger*[8]. Si bien esto no es una práctica recomendada (lo ideal es utilizar bibliotecas criptográficas ya probadas por la comunidad), no se ha descubierto ninguna por parte de los autores que se adecúe a las necesidades requeridas por el protocolo (*Pedersen Commitments* y *ZKP* asociadas). Este punto es algo importante a resolver ya que, como fue dicho, la implementación de primitivas criptográficas no es recomendado y es imperante utilizar implementaciones ya probadas y verificadas por la comunidad, como lo podría ser Charm-Crypto[9] o Scapi[10]. De todas maneras, la criptografía implementada se desarrolló utilizando interfaces, por lo que cuando se descubra una librería que cumpla los requerimientos criptográficos que se buscan, su implantación sea realizada de manera fácil.

4.4 API Resultante

La API que se tiene actualmente engloba los aspectos básicos del diseño del protocolo explicado anteriormente. Aun faltan ciertas demostraciones a imple-

[7] `http://zeromq.org/`

[8] `https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html`

[9] `http://charm-crypto.com/index.html`

[10] https://scapi.readthedocs.io/en/latest/

mentar (que aun no están listas en el diseño), pero la parte medular del protocolo está implementado y funcional, dejando como única gran labor al usuario final de la API establecer ciertos parámetros de seguridad (número de bits de los grupos necesarios para las demostraciones, por nombrar a alguno) además del mensaje a enviar por cada participante. Luego de establecer esos valores, el protocolo empieza a funcionar y corre perfectamente, revelando (a medida que van saliendo) los mensajes salientes del protocolo. Como es un trabajo en curso, queda como trabajo futuro la realización de pruebas de seguridad (atacar el sistema) para poder detectar *leaks* de información que derivarían en la pérdida de anonimato por parte de los emisores de mensajes.

4.5 Aplicación Móvil

Como prueba de concepto de la API implementada se creó una aplicación *Android* que ofrece la funcionalidad de establecer el envío de mensajes anónimos basado en el protocolo DC-NET explicado anteriormente.

Existen dos funcionalidades de la aplicación: ser un nodo Directorio o ser nodo Participante. Al ser Directorio, es necesario establecer parámetros básicos del protocolo (número de participantes a aceptar en el *anonymity-set* y largo máximo de los mensajes). Luego de esto la aplicación queda esperando que se conecten los participantes, para posteriormente enviar la dirección IP de todos los participantes al resto de ellos. Por otro lado, al ser Participante, es necesario primero conectarse a un Directorio, y luego de que el Directorio informa la dirección de cada otro participante, se establece un mensaje y se envía. El protocolo corre y los mensajes anónimos se van mostrando en pantalla a medida que van llegando.

Con esto, podemos argumentar que la API implementada sirve para poder ser utilizar por otra aplicación que necesite contar con mensajería anónima entre un grupo de usuarios en particular.

5 Experimentos

La API se puso a prueba simulando una red local utilizando el software *Common Open Research Emulator (CORE)*[11], que permite simular redes de manera simple y visual. La red se configuró para soportar 23 nodos participantes y 1 nodos directorio, conectados a través de una red local (latencia y ancho de banda *standard* de red local).

Se realizaron 2 tipos de experimentos:

1. Tamaño del *anonymity-set* creciente: se fue aumentando la cantidad de nodos participantes involucrados, además de que el 100% de los nodos enviaron mensajes.
2. Tamaño del *anonymity-set* fijo: siempre se mantuvieron 23 nodos participantes, pero se iba variando la cantidad de mensajes que se enviaban.

En cada uno de los 2 experimentos se midieron 3 variables: tiempo total de la ejecución (hasta que se envían todos los mensajes involucrados), tiempo que demora en llegar el primer mensaje y tiempo promedio por mensaje. Los resultados se muestran y analizan en la próxima sección.

6 Análisis de los Resultados

En la Figura 1 vemos que a medida que el anonymity-set va creciendo (y donde todos los nodos envían un mensaje), el tiempo total va aumentando paulatinamente de manera exponencial. Esto se debe a que como existen conexiones punto a punto entre los nodos deben realizarse n^2 conexiones distintas. Lo rescatable en este punto es que tanto el tiempo de llegada del primer mensaje como el tiempo promedio por mensaje no aumentan considerablemente, otorgando una cierta homogeneidad al protocolo.

Por otro lado, en la Figura 2 se ve que teniendo un tamaño fijo del anonymity-set (23 en este caso), el tiempo total a medida que más nodos van enviando mensajes va aumentando de manera lineal. Este escenario hace más próspero el uso del protocolo, ya que sería algo más común tener una cierta cantidad de participantes fijos, y habrán veces que algunos transmitan y otras veces no.

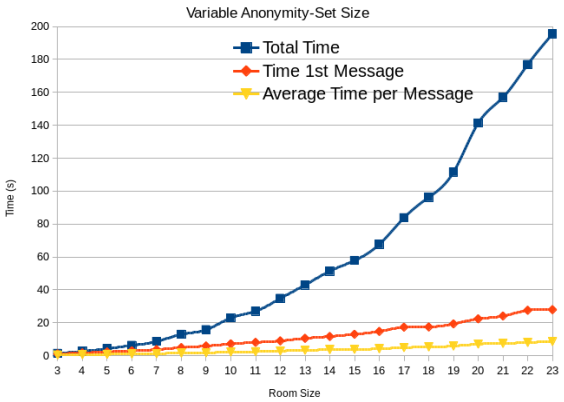

Figure 1: Tamaño de Anonymity-Set Variable

7 Conclusiones y Trabajo Futuro

El trabajo detallado hay que entenderlo como un trabajo en curso, por lo que aun queda mucho tramo por recorrer. Claramente queda como trabajo futuro

[11] `http://www.nrl.navy.mil/itd/ncs/products/core`

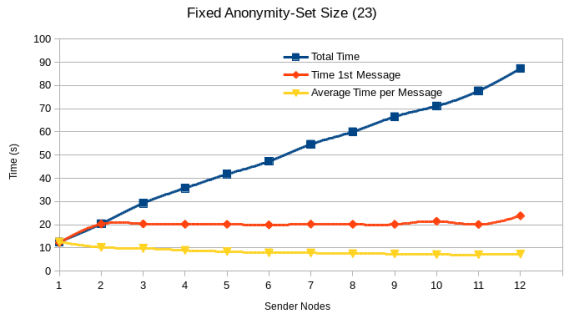

Figure 2: Tamaño de Anonymity-Set Fijo

(una vez diseñado e implementado todo el protocolo) realizar un análisis de seguridad a la API implementada, tratar de atacarla y buscar posibles *leaks* de información que comprometan el anonimato, ya que el protocolo puede que funcione correctamente, pero la implementación de éste requiere otro tipo de ataques y pruebas para ser considerada correcta. Otro aspecto importante a realizar es poder hacer un *profiling* a la implementación, encontrar zonas de ejecución que estén tomando mucho tiempo y realizarle mejoras para poder disminuir los tiempos. Por último sería importante realizar pruebas de la implementación en escenarios reales (comunicación a través de Internet, no en una red local) y observar como se comporta en dicho escenario.

Los tiempos obtenidos en las pruebas son tiempos razonables para pensar en una aplicación de foro anónimo (*bulletin-board*), donde la inmediatez de los mensajes no es un aspecto crucial, como lo sería un sistema de mensajería instantánea (sala de chat). Lo importante a destacar del trabajo es que es uno de los primeros que implementa y dejaría a la comunidad un sistema basado en DC-NET, el cual si bien ya lleva bastantes años desde su aparición, ha sido tomado en cuenta con mucha más importancia últimamente debido a los cuestionamientos y ataques descubiertos al protocolo por excelencia en anonimato, *onion-routing*. La necesidad por anonimato siempre estará, y es responsabilidad de la academia dotar de protocolos seguros que permitan a los usuarios ejercer su derecho a la privacidad.

Referencias

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[CS97] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical report, Citeseer, 1997.

[FvdG14] Christian Franck and Jeroen van de Graaf. Dining cryptographers are practical. *arXiv preprint arXiv:1402.2269*, 2014.

[GRS96] David M Goldschlag, Michael G Reed, and Paul F Syverson. Hiding routing information. In *International Workshop on Information Hiding*, pages 137–150. Springer, 1996.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[Wai89] Michael Waidner. Unconditional sender and recipient untraceability in spite of active attacks. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 302–319. Springer, 1989.